Package oracle.jdbc

Interface OracleConnection

All Superinterfaces:

`AutoCloseable`, `Connection`, `Wrapper`

All Known Implementing Classes:

`OracleConnectionWrapper`

```
public interface OracleConnection
extends Connection
```

This interface defines the Oracle extensions to the standard JDBC interface `java.sql.Connection`. You can use the `java.sql.Connection` interface in your application, where you do not make use of the Oracle extensions. However, when your application uses the Oracle extensions to `java.sql.Connection` you must use `oracle.jdbc.OracleConnection`.

`OracleConnection` extends standard JDBC connection functionality to create and return Oracle statement objects, set flags and options for Oracle performance extensions, and support type maps for Oracle objects.

Basic example : Once you connect to the database and, in the process, create your `Connection` object (see OracleDriver), the next step is to create a `Statement` object.
The `createStatement` method of your JDBC `Connection` object returns an object of the JDBC Statement type.
Following is an example of how to create the `Statement` object (`conn` being your connection object):

```
Statement stmt = conn.createStatement();
```

Note that there is nothing Oracle-specific about the preceding example; it follows standard JDBC syntax.

Connection Properties

This interface declares connection properties that are grouped into the following categories:

1. Database Authentication
2. Network Connection
3. Transport Layer Security (TLS/SSL)
4. High Availability
5. Tracing
6. Database Resident Connection Pooling (DRCP)
7. Sharding
8. Performance
9. API Behavior
10. Lightweight Directory Access Protocol (LDAP)

All connection properties can be specified as a URL parameter. Most connection properties can be specified as a system property, as a `Properties` object entry, or as a properties file entry.

See OracleDriver for more information about how to use the different options for specifying properties.

The `CONNECTION_PROPERTY_..._ACCESSMODE` constant fields of this interface express which options for specifying a property are supported. These access mode constant values are a bitwise OR of `ACCESSMODE_JAVAPROP`, `ACCESSMODE_SYSTEMPROP`, and/or `ACCESSMODE_FILEPROP`. To illustrate, consider the access mode of `CONNECTION_PROPERTY_PASSWORD`:

```
    boolean isJavaPropSupported = // evaluates to true
      0 != (ACCESSMODE_JAVAPROP & CONNECTION_PROPERTY_PASSWORD_ACCESSMODE);
    boolean isSystemPropSupported = // evaluates to false
      0 != (ACCESSMODE_SYSTEMPROP & CONNECTION_PROPERTY_PASSWORD_ACCESSMODE);
    boolean isFilePropSupported = // evaluates to true
      0 != (ACCESSMODE_FILEPROP & CONNECTION_PROPERTY_PASSWORD_ACCESSMODE);
```

The boolean evaluations would indicate that a password can be set with a `Properties` object or file, but can not be set as a system property.

Database Authentication Properties

The following properties effect how the driver authenticates with a database:

Name	Short Description	Default Value
oracle.jdbc.user	Specifies the user name when connecting to the database	null
oracle.jdbc.password	Specifies the password when connecting to the database	null
oracle.jdbc.loginTimeout	Specifies the timeout for opening a JDBC connection	0
oracle.net.authentication_services	Enables authentication with RADIUS, KERBEROS, or TCPS (that is: SSL/TLS)	null
oracle.jdbc.proxyClientName	Specifies the user name for proxy authentication	null
oracle.net.kerberos5_mutual_authentication	Enables Kerberos mutual authentication	null
oracle.net.kerberos5_cc_name	Specifies the location of the Kerberos credential cache	null
oracle.net.KerberosRealm	Specifies the realm used for Kerberos authentication	null
oracle.jdbc.newPassword	Specifies the new password during connection creation	null
internal_logon	Specifies the administrative user for authentication, such as SYSDBA	null
prelim_auth	Enables PRELIM_AUTH mode	false
oracle.jdbc.allowedLogonVersion	Specifies the minimum authentication protocol version	8

Network Connection Properties

The following properties effect how the driver establishes a network connection to a database:

Name	Short Description	Default Value
oracle.net.tns_admin	Specifies the file system path of the TNS ADMIN directory	null
oracle.net.CONNECT_TIMEOUT	Specifies the timeout when connecting a socket to the database listener	0
oracle.net.OUTBOUND_CONNECT_TIMEOUT	Specifies the timeout when negotiating a session with the database listener	0

oracle.net.TCP_KEEPINTERVAL	Specifies the frequency at which keep alive probes are retransmitted	-1
oracle.net.TCP_KEEPCOUNT	Specifies the maximum number of keep alive probes to send before terminating the connection	-1
oracle.jdbc.TcpNoDelay	Enables TCP_NODELAY	false
oracle.net.disableOob	Disables Out of Band (OOB) data	false
oracle.jdbc.thinForceDNSLoadBalancing	Enables load balancing when a hostname resolves to multiple IP addresses	false
oracle.net.DOWN_HOSTS_TIMEOUT	Specifies the duration for which a server address remains in the down hosts cache	600
oracle.net.encryption_client	Specifies the level of security for the encryption service	null
oracle.net.encryption_types_client	Specifies the list of encryption algorithms that you want to activate	null
oracle.net.crypto_seed	Specifies the encryption seed	null
oracle.net.useJCEAPI	Disables the use of JDK Crypto (JCE) APIs for encryption and decryption	true
oracle.net.crypto_checksum_client	Specifies the level of security for the integrity service	null
oracle.net.crypto_checksum_types_client	Specifies the list of integrity algorithms that you want to activate	null
oracle.net.networkCompression	Enables compression of network packets	off
oracle.net.networkCompressionLevels	Specifies the acceptable levels of network packet compression	(high)
oracle.net.networkCompressionThreshold	Specifies the minimum size that an uncompressed network packet must have before being compressed	1024
oracle.net.httpsProxyHost	Specifies the hostname or address of an https proxy server	null
oracle.net.httpsProxyPort	Specifies the port of an https proxy server	0
oracle.net.websocketUser	Specifies the webserver username when using Secure Websocket protocol (WSS)	null
oracle.net.websocketPassword	Specifies the webserver password when using Secure Websocket protocol (WSS)	null
oracle.net.socksProxyHost	Specifies the host name of a SOCKS proxy server	null
oracle.net.socksProxyPort	Specifies the port number of a SOCKS proxy server	1080
oracle.net.socksRemoteDNS	Enables remote DNS lookup of the database host when connecting through a SOCKS proxy server	false

Transport Layer Security (TLS/SSL) Properties

The following properties effect how the driver communicates with a database using TLS:

Name	Short Description	Default Value
oracle.net.wallet_location	Specifies the file system path to a wallet used when connecting with TLS/SSL	null
oracle.net.wallet_password	Specifies the wallet password to use when connecting with TLS/SSL	null
javax.net.ssl.keyStore	Specifies the file system path to a key store	null
javax.net.ssl.keyStoreType	Specifies the type of a key store, such as SSO, JKS, or PKCS12	null
javax.net.ssl.keyStorePassword	Specifies the password of a key store	null
javax.net.ssl.trustStore	Specifies the file system path to a trust store	null
javax.net.ssl.trustStoreType	Specifies the type of a trust store, such as SSO, JKS, or PKCS12	null
javax.net.ssl.trustStorePassword	Specifies the password of a trust store	null
oracle.net.ssl_certificate_alias	Specifies the keystore certificate alias	null
oracle.net.ssl_server_dn_match	Disables authentication of the Distinguished Name (DN) given by the database server's certificate	null
oracle.net.ssl_server_cert_dn	Specifies the Distinguished Name (DN) used to authenticate the database server's certificate	null
oracle.net.ssl_version	Restricts the versions of TLS/SSL that can be used	null
oracle.net.ssl_cipher_suites	Restricts the cipher suites that can be used for TLS/SSL	null
ssl.keyManagerFactory.algorithm	Specifies the javax.net.ssl.KeyManagerFactory algorithm	null
ssl.trustManagerFactory.algorithm	Specifies the javax.net.ssl.TrustManagerFactory algorithm	null
oracle.net.ssl_context_protocol	Specifies the javax.net.ssl.SSLContext protocol	TLS

High Availability Properties

The following properties configure the High Availablity functions of the driver:

Name	Short Description	Default Value
oracle.jdbc.fanEnabled	Disables Fast Application Notification (FAN)	true
oracle.jdbc.enableACSupport	Disables Application Continuity (AC)	true
oracle.jdbc.enableTGSupport	Enables Transaction Guard (TG)	false
oracle.jdbc.enableImplicitRequests	Disables implicit request boundaries for Application Continuity (AC)	true
oracle.jdbc.replay.protectedRequestSizeLimit	Specifies the maximum number of calls that can occur in a replayed request	2147483647
oracle.jdbc.ons.walletfile	Specifies the wallet file for ONS	null
oracle.jdbc.ons.walletpassword	Specifies the wallet password for ONS	null
oracle.jdbc.ons.protocol	Specifies the network protocol for ONS	TCP

Tracing Properties

The following properties configure values that are used for tracing:

Name	Short Description	Default Value
v$session.terminal	Specifies the value of v$session.terminal	unknown
v$session.machine	Specifies the value of v$session.machine	Local host name or "jdbcclient"
v$session.osuser	Specifies the value of v$session.osuser	user.name system property or "jdbcuser"
v$session.program	Specifies the value of v$session.program	JDBC Thin Client
v$session.process	Specifies the value of v$session.process	1234
oracle.jdbc.driverNameAttribute	Specifies the value of v$session_connect_info.client_driver	jdbcthin or jdbcoci
oracle.net.connectionIdPrefix	Specifies the a net connection id prefix	null
oracle.jdbc.DMSStatementMetrics	Enables DMS Statement metrics	false

Database Resident Connection Pooling (DRCP) Properties

The following properties configure how the driver interacts with DRCP:

Name	Short Description	Default Value

oracle.jdbc.UseDRCPMultipletag	Enables multiple DRCP tag names	false
oracle.jdbc.DRCPPLSQLCallback	Specifies the DRCP callback	null
oracle.jdbc.DRCPMultiplexingInRequestAPIs	Enables automatic DRCP session attachment and detachment with beginRequest and endRequest	false

Sharding Properties

The following properties configure how the driver interacts with a sharded database:

Name	Short Description	Default Value
oracle.jdbc.readOnlyInstanceAllowed	Enables connection to a read-only sharded database instance	false
oracle.jdbc.useShardingDriverConnection	Enables the sharding driver	false

Performance Properties

The following properties configure the driver in ways that effect performance:

Name	Short Description	Default Value
defaultRowPrefetch	Specifies the default number of rows to prefetch	10
useFetchSizeWithLongColumn	Enables prefetch of rows with a LONG or LONG RAW column	false
oracle.jdbc.defaultLobPrefetchSize	Specifies the default LOB prefetch size	32768
oracle.net.useZeroCopyIO	Disables zero-copy IO for SecureFile Lobs	true
oracle.jdbc.enableTempLobRefCnt	Disables tracking of references to temporary LOBs	true
oracle.jdbc.useThreadLocalBufferCache	Enables thread local buffer caching	false
oracle.jdbc.maxCachedBufferSize	Specifies the maximum size of a cached data buffer	30
SetFloatAndDoubleUseBinary	When set to true, instructs the driver to transport Java float and Java double values to the database in SQL BINARY_FLOAT and SQL BINARY_DOUBLE representation respectively	false
oracle.jdbc.enableQueryResultCache	Disables ResultSet caching	true
oracle.jdbc.useNio	Instructs the jdbc:oci driver to copy data using native I/O with java.nio.ByteBuffer APIs	false

API Behavior Properties

The following properties configure the driver in ways that effect the behavior of its APIs:

Name	Short Description	Default Value
oracle.jdbc.JDBCStandardBehavior	Enables strict compliance with the JDBC specification	false
oracle.jdbc.LobStreamPosStandardCompliant	Enables JDBC Specification compliant LOB positions	false
autoCommit	Specifies the the default value of the auto-commit mode	true
oracle.jdbc.autoCommitSpecCompliant	Disables JDBC Specification compliant behavior of the auto-commit mode	true
oracle.jdbc.commitOption	Specifies the default commit option	null
oracle.jdbc.continueBatchOnError	Enables batch updates to continue with the remaining rows, after an error occurs during the execution of a batch	false
oracle.jdbc.defaultConnectionValidation	Specifies the default level of effort for connection validation	NETWORK
fixedString	Instructs the driver to use FIXED CHAR padding when calling the setObject method	false
oracle.jdbc.strictASCIIConversion	Enables replacement characters when converting to ASCII	false
defaultNChar	Set NCHAR as the default mode for all character data columns	false
oracle.jdbc.convertNcharLiterals	Disables NCHAR literal conversion	true
processEscapes	Instructs the driver to disable escape processing by default	true
oracle.jdbc.mapDateToTimestamp	When set to false, instructs the driver to map column values of Oracle's non-standard DATE type to java.sql.Date	true
oracle.jdbc.use1900AsYearForTime	Sets the date component to 01 January 1900, when the setTime method is called	false
oracle.jdbc.timestampTzInGmt	Disables adjusting of TIMESTAMP WITH TIME ZONE data to GMT	true
oracle.jdbc.timezoneAsRegion	Disables the use of JVM default timezone rather than convert to a GMT offset	true
oracle.jdbc.createDescriptorUseCurrentSchemaForSchemaName	Qualify Abstract Data Type (ADT) names with the CURRENT_USER as the type owner	false
includeSynonyms	Instructs the driver to include synonyms when getting information about a column	false
restrictGetTables	Restricts the values returned by the DatabaseMetaData.getTables() method	false
oracle.jdbc.sqlTranslationProfile	Specifies the SQL translation profile	null
oracle.jdbc.sqlErrorTranslationFile	Specifies the error code translation file	null
protocol	Specifies the type of driver, whether it is thin, OCI, or KPRB	null
oracle.jdbc.editionName	Specifies the "session edition" name	null
oracle.jdbc.backwardCompatibileUpdateableResultSet	Enables backward compatibility with 12.1.0.2.0 for updatable ResultSets	false
oracle.jdbc.RetainV9LongBindBehavior	Enables backward compatibility with 9.0.1.0 for bind values with a length that exceeds the maximum length of the VARCHAR type	false
disableDefineColumnType	Disables the OracleStatement.defineColumnType method	false

Lightweight Directory Access Protocol (LDAP) Properties

The following properties configure how the driver interacts with an LDAP server:

Name	Short Description	Default Value
oracle.net.ldap.security.authentication	Specifies the authentication mechanism to be used by the LDAP service provider in the JDK	null
oracle.net.ldap.security.principal	Specifies the value of the principal that is used for LDAP authentication	null
oracle.net.ldap.security.credentials	Specifies the credentials used for LDAP authentication	null
com.sun.jndi.ldap.connect.timeout	Specifies the timeout when connecting to an LDAP server	null
com.sun.jndi.ldap.read.timeout	Specifies the timeout when reading a response from an LDAP server	null
oracle.net.ldap.ssl.walletLocation	Specifies the file system path to a wallet used when connecting to an LDAP server	null
oracle.net.ldap.ssl.walletPassword	Specifies the wallet password to use when connecting to an LDAP server	null

oracle.net.ldap.ssl.trustStoreType	Specifies the type of a trust store to use when connecting to an LDAP server such as SSO, JKS, or PKCS12	null
oracle.net.ldap.ssl.trustStore	Specifies the file system path to a trust store to use when connecting to an LDAP server	null
oracle.net.ldap.ssl.trustStorePassword	Specifies the password of a trust store to use when connecting to an LDAP server	null
oracle.net.ldap.ssl.supportedCiphers	Restricts the cipher suites that can be used when connecting to an LDAP server	null
oracle.net.ldap.ssl.supportedVersions	Restricts the versions of TLS/SSL that can be used	null
oracle.net.ldap.ssl.keyManagerFactory.algorithm	Specifies the key manager factory algorithm to use when connecting to an LDAP server	null
oracle.net.ldap.ssl.trustManagerFactory.algorithm	Specifies the trust manager factory algorithm to use when connecting to an LDAP server	null
oracle.net.ldap.ssl.ssl_context_protocol	Specifies the `protocol` of `SSLContext` objects used when connecting to an LDAP server	null

Since:

8.1.7

Nested Class Summary

Nested Classes

Modifier and Type	Interface	Description
`static class`	`OracleConnection.CommitOption`	
`static class`	`OracleConnection.ConnectionValidation`	Specifiers for how much effort to put into validating a `Connection`.
`static class`	`OracleConnection.DatabaseShutdownMode`	
`static class`	`OracleConnection.DatabaseStartupMode`	
`static class`	`OracleConnection.DRCPState`	

Field Summary

Fields

Modifier and Type	Field	Description
`static int`	`ABANDONED_CONNECTION_CALLBACK`	
`static byte`	`ACCESSMODE_BOTH`	Bitmask which can be applied to the `CONNECTION_PROPERTY_{name}_ACCESSMODE` constants defined in this interface.
`static byte`	`ACCESSMODE_FILEPROP`	Bitmask which can be applied to the `CONNECTION_PROPERTY_{name}_ACCESSMODE` constants defined in this interface.
`static byte`	`ACCESSMODE_JAVAPROP`	Bitmask which can be applied to the `CONNECTION_PROPERTY_{name}_ACCESSMODE` constants defined in this interface.
`static byte`	`ACCESSMODE_SYSTEMPROP`	Bitmask which can be applied to the `CONNECTION_PROPERTY_{name}_ACCESSMODE` constants defined in this interface.
`static int`	`ALL_CONNECTION_CALLBACKS`	
`static String`	`AQ_USE_HOST_CONNECTION_ADDR_INFO`	Set the value of AQ_USE_HOST_CONNECTION_ADDR_INFO to 'false' to use the address info returned by the server for establishing the client initiated Connection for JMS Message Listener .
`static int`	`CACHE_SIZE_NOT_SET`	
`static String`	`CLIENT_INFO_KEY_SEPARATOR`	Separate the namespace from the key the name of a client info.
`static String`	`CONNECTION_PROPERTY_ACCESS_TOKEN`	This property configures an access token that Oracle JDBC uses for authentication with Oracle Database.
`static byte`	`CONNECTION_PROPERTY_ACCESS_TOKEN_ACCESSMODE`	
`static String`	`CONNECTION_PROPERTY_ACCESS_TOKEN_DEFAULT`	
`static String`	`CONNECTION_PROPERTY_ACCUMULATE_BATCH_RESULT`	When using Oracle style batching, JDBC determines when to flush a batch to the database.
`static byte`	`CONNECTION_PROPERTY_ACCUMULATE_BATCH_RESULT_ACCESSMODE`	
`static String`	`CONNECTION_PROPERTY_ACCUMULATE_BATCH_RESULT_DEFAULT`	

static String	CONNECTION_PROPERTY_ALLOWED_LOGON_VERSION_DEFAULT	
static String	CONNECTION_PROPERTY_AUTO_COMMIT_SPEC_COMPLIANT	Alters the auto-commit behavior of the driver.
static byte	CONNECTION_PROPERTY_AUTO_COMMIT_SPEC_COMPLIANT_ACCESSMODE	
static String	CONNECTION_PROPERTY_AUTO_COMMIT_SPEC_COMPLIANT_DEFAULT	
static String	CONNECTION_PROPERTY_AUTOCOMMIT	Use this connection property to change the default value of autoCommit.
static byte	CONNECTION_PROPERTY_AUTOCOMMIT_ACCESSMODE	
static String	CONNECTION_PROPERTY_AUTOCOMMIT_DEFAULT	
static String	CONNECTION_PROPERTY_BACKWARD_COMPATIBLE_UPDATEABLE_RESULTSET	If set to true, use the old, pre 12.1.0.2.0, updateable ResultSet behavior.
static byte	CONNECTION_PROPERTY_BACKWARD_COMPATIBLE_UPDATEABLE_RESULTSET_ACCESSMODE	
static String	CONNECTION_PROPERTY_BACKWARD_COMPATIBLE_UPDATEABLE_RESULTSET_DEFAULT	
static String	CONNECTION_PROPERTY_COMMIT_OPTION	This connection property lets you define a default commit option that will be used when calling `connection.commit();`.
static byte	CONNECTION_PROPERTY_COMMIT_OPTION_ACCESSMODE	
static String	CONNECTION_PROPERTY_COMMIT_OPTION_DEFAULT	
static String	CONNECTION_PROPERTY_CONFIG_FILE	This property provides the location of one or more properties files.
static byte	CONNECTION_PROPERTY_CONFIG_FILE_ACCESSMODE	
static String	CONNECTION_PROPERTY_CONFIG_FILE_DEFAULT	
static String	CONNECTION_PROPERTY_CONNECTION_CLASS	Specify the connection class name for Database Resident Connection Pool (DRCP).
static byte	CONNECTION_PROPERTY_CONNECTION_CLASS_ACCESSMODE	
static String	CONNECTION_PROPERTY_CONNECTION_CLASS_DEFAULT	
static String	CONNECTION_PROPERTY_CONNECTION_PURITY	Specify the connection purity for a Database Resident Connection Pool (DRCP) connection.
static byte	CONNECTION_PROPERTY_CONNECTION_PURITY_ACCESSMODE	
static String	CONNECTION_PROPERTY_CONNECTION_PURITY_DEFAULT	
static String	CONNECTION_PROPERTY_CONTINUE_BATCH_ON_ERROR	This connection property specifies whether to continue batch execution when server encounters an erroneous row in the batch.
static byte	CONNECTION_PROPERTY_CONTINUE_BATCH_ON_ERROR_ACCESSMODE	
static String	CONNECTION_PROPERTY_CONTINUE_BATCH_ON_ERROR_DEFAULT	
static String	CONNECTION_PROPERTY_CONVERT_NCHAR_LITERALS	Convert NCHAR literals to Unicode literals when equal "true".
static byte	CONNECTION_PROPERTY_CONVERT_NCHAR_LITERALS_ACCESSMODE	
static String	CONNECTION_PROPERTY_CONVERT_NCHAR_LITERALS_DEFAULT	
static String	CONNECTION_PROPERTY_CREATE_DESCRIPTOR_USE_CURRENT_SCHEMA_FOR_SCHEMA_NAME	The user has to provide fully qualified ADT name ([username].[adt name]) for all ADT operations.
static byte	CONNECTION_PROPERTY_CREATE_DESCRIPTOR_USE_CURRENT_SCHEMA_FOR_SCHEMA_NAME_ACCESSMODE	
static String	CONNECTION_PROPERTY_CREATE_DESCRIPTOR_USE_CURRENT_SCHEMA_FOR_SCHEMA_NAME_DEFAULT	

static String	CONNECTION_PROPERTY_DATABASE_DEFAULT	
static String	CONNECTION_PROPERTY_DEFAULT_CONNECTION_VALIDATION	This connection property is used to specify how much effort to put into validating a `Connection`. This property controls what isValid() does. The possible values for this property are - "NONE", "LOCAL", "SOCKET", "NETWORK", "SERVER" and "COMPLETE". The values are case-sensitive, setting any other value throws exception. The default value of this property is "NETWORK".
static byte	CONNECTION_PROPERTY_DEFAULT_CONNECTION_VALIDATION_ACCESSMODE	
static String	CONNECTION_PROPERTY_DEFAULT_CONNECTION_VALIDATION_DEFAULT	
static String	CONNECTION_PROPERTY_DEFAULT_EXECUTE_BATCH	**Deprecated.** Oracle-Style batching is desupported.
static byte	CONNECTION_PROPERTY_DEFAULT_EXECUTE_BATCH_ACCESSMODE	
static String	CONNECTION_PROPERTY_DEFAULT_EXECUTE_BATCH_DEFAULT	
static String	CONNECTION_PROPERTY_DEFAULT_LOB_PREFETCH_SIZE	The value of this property is used as the default LOB prefetch size for this connection.
static byte	CONNECTION_PROPERTY_DEFAULT_LOB_PREFETCH_SIZE_ACCESSMODE	
static String	CONNECTION_PROPERTY_DEFAULT_LOB_PREFETCH_SIZE_DEFAULT	
static String	CONNECTION_PROPERTY_DEFAULT_ROW_PREFETCH	The value of this property is used as the default number of rows to prefetch.
static byte	CONNECTION_PROPERTY_DEFAULT_ROW_PREFETCH_ACCESSMODE	
static String	CONNECTION_PROPERTY_DEFAULT_ROW_PREFETCH_DEFAULT	
static String	CONNECTION_PROPERTY_DEFAULT_USE_NIO	In case of Jdbc-OCI drivers the data is copied from C layer to Java using Jni array copy api.
static byte	CONNECTION_PROPERTY_DEFAULT_USE_NIO_ACCESSMODE	
static String	CONNECTION_PROPERTY_DEFAULT_USE_NIO_DEFAULT	
static String	CONNECTION_PROPERTY_DEFAULTNCHAR	If the value of this property is "true", the default mode for all character data columns will be NCHAR.
static byte	CONNECTION_PROPERTY_DEFAULTNCHAR_ACCESSMODE	
static String	CONNECTION_PROPERTY_DEFAULTNCHAR_DEFAULT	
static String	CONNECTION_PROPERTY_DISABLE_DEFINECOLUMNTYPE	Disable the method OracleStatement.defineColumnType when equal "true".
static byte	CONNECTION_PROPERTY_DISABLE_DEFINECOLUMNTYPE_ACCESSMODE	
static String	CONNECTION_PROPERTY_DISABLE_DEFINECOLUMNTYPE_DEFAULT	
static String	CONNECTION_PROPERTY_DMS_PARENT_NAME	Override the default DMS parent name.
static byte	CONNECTION_PROPERTY_DMS_PARENT_NAME_ACCESSMODE	
static String	CONNECTION_PROPERTY_DMS_PARENT_NAME_DEFAULT	
static String	CONNECTION_PROPERTY_DMS_PARENT_TYPE	Override the default DMS parent type.
static byte	CONNECTION_PROPERTY_DMS_PARENT_TYPE_ACCESSMODE	
static String	CONNECTION_PROPERTY_DMS_PARENT_TYPE_DEFAULT	
static String	CONNECTION_PROPERTY_DMS_STMT_CACHING_METRICS	**Deprecated.**
static byte	CONNECTION_PROPERTY_DMS_STMT_CACHING_METRICS_ACCESSMODE	

static byte	CONNECTION_PROPERTY_DMS_STMT_METRICS_ACCESSMODE	
static String	CONNECTION_PROPERTY_DMS_STMT_METRICS_DEFAULT	
static String	CONNECTION_PROPERTY_DOWN_HOSTS_TIMEOUT	To specify the amount of time in seconds that information about the down state of server hosts is kept in driver's cache.
static byte	CONNECTION_PROPERTY_DOWN_HOSTS_TIMEOUT_ACCESSMODE	
static String	CONNECTION_PROPERTY_DOWN_HOSTS_TIMEOUT_DEFAULT	
static String	CONNECTION_PROPERTY_DRCP_MULTIPLEXING_IN_REQUEST_APIS	Specifies whether to enable DRCP-attach in beginRequest and DRCP-detach in endRequest.
static byte	CONNECTION_PROPERTY_DRCP_MULTIPLEXING_IN_REQUEST_APIS_ACCESSMODE	
static String	CONNECTION_PROPERTY_DRCP_MULTIPLEXING_IN_REQUEST_APIS_DEFAULT	
static String	CONNECTION_PROPERTY_DRCP_PLSQL_CALLBACK	This is PL/SQL `"fix-up"` callback name which is provided by the application, and it is used to transform a session checked out from the pool to the desired state requested by the application. `"fix-up"` callback can provide performance improvements to applications by running the `"session state fix-up"` logic on the server, thereby eliminating application round-trips to the database to run the `"fix-up"` logic. This is an optional configuration. This property is valid only for thin driver.
static byte	CONNECTION_PROPERTY_DRCP_PLSQL_CALLBACK_ACCESSMODE	
static String	CONNECTION_PROPERTY_DRCP_PLSQL_CALLBACK_DEFAULT	
static String	CONNECTION_PROPERTY_DRCP_TAG_NAME	This is the tag name that for Database Resident Connection Pool (DRCP).
static byte	CONNECTION_PROPERTY_DRCP_TAG_NAME_ACCESSMODE	
static String	CONNECTION_PROPERTY_DRCP_TAG_NAME_DEFAULT	
static String	CONNECTION_PROPERTY_DRIVER_NAME_ATTRIBUTE	The value passed to the server for the OCI_ATTR_DRIVER_NAME.
static byte	CONNECTION_PROPERTY_DRIVER_NAME_ATTRIBUTE_ACCESSMODE	
static String	CONNECTION_PROPERTY_DRIVER_NAME_ATTRIBUTE_DEFAULT	
static String	CONNECTION_PROPERTY_EDITION_NAME	This connection property can be used to specify a name for the "session edition".
static byte	CONNECTION_PROPERTY_EDITION_NAME_ACCESSMODE	
static String	CONNECTION_PROPERTY_EDITION_NAME_DEFAULT	
static String	CONNECTION_PROPERTY_ENABLE_AC_SUPPORT	Specifies whether driver support for Application Continuity (AC) is enabled.
static byte	CONNECTION_PROPERTY_ENABLE_AC_SUPPORT_ACCESSMODE	
static String	CONNECTION_PROPERTY_ENABLE_AC_SUPPORT_DEFAULT	
static String	CONNECTION_PROPERTY_ENABLE_DATA_IN_LOCATOR	The value of this property is used to control the use of the Data in Locator feature of the server.
static byte	CONNECTION_PROPERTY_ENABLE_DATA_IN_LOCATOR_ACCESSMODE	
static String	CONNECTION_PROPERTY_ENABLE_DATA_IN_LOCATOR_DEFAULT	
static String	CONNECTION_PROPERTY_ENABLE_IMPLICIT_REQUESTS	Specifies whether to enable implicit request boundary support for Application Continuity (AC).
static byte	CONNECTION_PROPERTY_ENABLE_IMPLICIT_REQUESTS_ACCESSMODE	

static byte	CONNECTION_PROPERTY_ENABLE_QUERY_RESULT_CACHE_ACCESSMODE	
static String	CONNECTION_PROPERTY_ENABLE_QUERY_RESULT_CACHE_DEFAULT	
static String	CONNECTION_PROPERTY_ENABLE_READ_DATA_IN_LOCATOR	The value of this property is used to control the use of the Data in Locator feature by the client.
static byte	CONNECTION_PROPERTY_ENABLE_READ_DATA_IN_LOCATOR_ACCESSMODE	
static String	CONNECTION_PROPERTY_ENABLE_READ_DATA_IN_LOCATOR_DEFAULT	
static String	CONNECTION_PROPERTY_ENABLE_RESULTSET_CACHE	This property is ignored in 18c.
static byte	CONNECTION_PROPERTY_ENABLE_RESULTSET_CACHE_ACCESSMODE	
static String	CONNECTION_PROPERTY_ENABLE_RESULTSET_CACHE_DEFAULT	
static String	CONNECTION_PROPERTY_ENABLE_TEMP_LOB_REF_COUNT	By default the JDBC thin driver counts the temp LOB references and only closes them on the server when this count is down to zero.
static byte	CONNECTION_PROPERTY_ENABLE_TEMP_LOB_REF_COUNT_ACCESSMODE	
static String	CONNECTION_PROPERTY_ENABLE_TEMP_LOB_REF_COUNT_DEFAULT	
static String	CONNECTION_PROPERTY_ENABLE_TG_SUPPORT	Specifies whether driver support for Transaction Guard (TG) is enabled.
static byte	CONNECTION_PROPERTY_ENABLE_TG_SUPPORT_ACCESSMODE	
static String	CONNECTION_PROPERTY_ENABLE_TG_SUPPORT_DEFAULT	
static String	CONNECTION_PROPERTY_FAN_ENABLED	Specifies whether driver High Availability (HA) or FAN (Fast Application Notification) is enabled.
static byte	CONNECTION_PROPERTY_FAN_ENABLED_ACCESSMODE	
static String	CONNECTION_PROPERTY_FAN_ENABLED_DEFAULT	
static String	CONNECTION_PROPERTY_FIXED_STRING	If the value of this property is "true", JDBC will use FIXED CHAR semantic when setObject is called with a String argument.
static byte	CONNECTION_PROPERTY_FIXED_STRING_ACCESSMODE	
static String	CONNECTION_PROPERTY_FIXED_STRING_DEFAULT	
static String	CONNECTION_PROPERTY_IMPLICIT_STATEMENT_CACHE_SIZE	The maximum number of statements that will be stored in this connection's statement cache.
static byte	CONNECTION_PROPERTY_IMPLICIT_STATEMENT_CACHE_SIZE_ACCESSMODE	
static String	CONNECTION_PROPERTY_IMPLICIT_STATEMENT_CACHE_SIZE_DEFAULT	
static String	CONNECTION_PROPERTY_IN_BAND_NOTIFICATION	Specifies whether driver in-band notification support is enabled.
static byte	CONNECTION_PROPERTY_IN_BAND_NOTIFICATION_ACCESSMODE	
static String	CONNECTION_PROPERTY_IN_BAND_NOTIFICATION_DEFAULT	
static String	CONNECTION_PROPERTY_INCLUDE_SYNONYMS	If the value of this property is "true", JDBC will include synonyms when getting information about a column.
static byte	CONNECTION_PROPERTY_INCLUDE_SYNONYMS_ACCESSMODE	
static String	CONNECTION_PROPERTY_INCLUDE_SYNONYMS_DEFAULT	
static String	CONNECTION_PROPERTY_INTERNAL_LOGON	The value of this property is used as the user name when performing an internal logon.
static byte	CONNECTION_PROPERTY_INTERNAL_LOGON_ACCESSMODE	

String		This property could be removed in the future and the default will be `true`.
static byte	CONNECTION_PROPERTY_J2EE13_COMPLIANT_ACCESSMODE	
static String	CONNECTION_PROPERTY_J2EE13_COMPLIANT_DEFAULT	
static String	CONNECTION_PROPERTY_JDBC_STANDARD_BEHAVIOR	Ensures the driver is in strict compliance with the JDBC specification.
static byte	CONNECTION_PROPERTY_JDBC_STANDARD_BEHAVIOR_ACCESSMODE	
static String	CONNECTION_PROPERTY_JDBC_STANDARD_BEHAVIOR_DEFAULT	
static String	CONNECTION_PROPERTY_LOB_STREAM_POS_STANDARD_COMPLIANT	Previous releases allowed the value of 0L to be set for the position parameter of Blob.setBinaryStream and Clob.setAsciiStream and setCharacterStream which is not correct in the specification.
static byte	CONNECTION_PROPERTY_LOB_STREAM_POS_STANDARD_COMPLIANT_ACCESSMODE	
static String	CONNECTION_PROPERTY_LOB_STREAM_POS_STANDARD_COMPLIANT_DEFAULT	
static String	CONNECTION_PROPERTY_LOGIN_TIMEOUT	Configures a timeout for creating a new connection.
static byte	CONNECTION_PROPERTY_LOGIN_TIMEOUT_ACCESSMODE	
static String	CONNECTION_PROPERTY_LOGIN_TIMEOUT_DEFAULT	
static String	CONNECTION_PROPERTY_MAP_DATE_TO_TIMESTAMP	This connection property lets you define how the driver will map SQL DATE values in the database to Java types.
static byte	CONNECTION_PROPERTY_MAP_DATE_TO_TIMESTAMP_ACCESSMODE	
static String	CONNECTION_PROPERTY_MAP_DATE_TO_TIMESTAMP_DEFAULT	
static String	CONNECTION_PROPERTY_MAX_CACHED_BUFFER_SIZE	The log base 2 of the size of the largest internal char or byte data buffer that the driver should cache.
static byte	CONNECTION_PROPERTY_MAX_CACHED_BUFFER_SIZE_ACCESSMODE	
static String	CONNECTION_PROPERTY_MAX_CACHED_BUFFER_SIZE_DEFAULT	
static String	CONNECTION_PROPERTY_NET_KEEPALIVE	Enables TCP keep alive on the network connection.
static byte	CONNECTION_PROPERTY_NET_KEEPALIVE_ACCESSMODE	
static String	CONNECTION_PROPERTY_NET_KEEPALIVE_DEFAULT	
static String	CONNECTION_PROPERTY_NETWORK_COMPRESSION	Enables compression of the protocol data sent over network.
static byte	CONNECTION_PROPERTY_NETWORK_COMPRESSION_ACCESSMODE	
static String	CONNECTION_PROPERTY_NETWORK_COMPRESSION_DEFAULT	
static String	CONNECTION_PROPERTY_NETWORK_COMPRESSION_LEVELS	The value is a comma separated list of supported levels in the user preference order surrounded by brackets.
static byte	CONNECTION_PROPERTY_NETWORK_COMPRESSION_LEVELS_ACCESSMODE	
static String	CONNECTION_PROPERTY_NETWORK_COMPRESSION_LEVELS_DEFAULT	
static String	CONNECTION_PROPERTY_NETWORK_COMPRESSION_THRESHOLD	Minimum size of data in packet required to perform compression.
static byte	CONNECTION_PROPERTY_NETWORK_COMPRESSION_THRESHOLD_ACCESSMODE	
static String	CONNECTION_PROPERTY_NETWORK_COMPRESSION_THRESHOLD_DEFAULT	
static String	CONNECTION_PROPERTY_NEW_PASSWORD	This property enables users to set a new password during connection creation if user's password got expired.
static byte	CONNECTION_PROPERTY_NEW_PASSWORD_ACCESSMODE	

String		is set to "OCI_TOKEN", this property specifies the Oracle Cloud ID (OCID) of the compartment for the database identified by CONNECTION_PROPERTY_OCI_DATABASE.
static byte	CONNECTION_PROPERTY_OCI_COMPARTMENT_ACCESSMODE	
static String	CONNECTION_PROPERTY_OCI_COMPARTMENT_DEFAULT	
static String	CONNECTION_PROPERTY_OCI_DATABASE	When CONNECTION_PROPERTY_PASSWORD_AUTHENTICATION is set to "OCI_TOKEN", this property specifies the Oracle Cloud ID (OCID) of the database that JDBC requests access to.
static byte	CONNECTION_PROPERTY_OCI_DATABASE_ACCESSMODE	
static String	CONNECTION_PROPERTY_OCI_DATABASE_DEFAULT	
static String	CONNECTION_PROPERTY_OCI_DRIVER_CHARSET	
static byte	CONNECTION_PROPERTY_OCI_DRIVER_CHARSET_ACCESSMODE	
static String	CONNECTION_PROPERTY_OCI_DRIVER_CHARSET_DEFAULT	
static String	CONNECTION_PROPERTY_OCI_ENV_HANDLE	
static byte	CONNECTION_PROPERTY_OCI_ENV_HANDLE_ACCESSMODE	
static String	CONNECTION_PROPERTY_OCI_ENV_HANDLE_DEFAULT	
static String	CONNECTION_PROPERTY_OCI_ERR_HANDLE	
static byte	CONNECTION_PROPERTY_OCI_ERR_HANDLE_ACCESSMODE	
static String	CONNECTION_PROPERTY_OCI_ERR_HANDLE_DEFAULT	
static String	CONNECTION_PROPERTY_OCI_IAM_URL	When CONNECTION_PROPERTY_PASSWORD_AUTHENTICATION is set to "OCI_TOKEN", this property must specify the full path of the Identity and Access Management (IAM) endpoint that Oracle JDBC authenticates with, as in:
static byte	CONNECTION_PROPERTY_OCI_IAM_URL_ACCESSMODE	
static String	CONNECTION_PROPERTY_OCI_IAM_URL_DEFAULT	
static String	CONNECTION_PROPERTY_OCI_SVC_CTX_HANDLE	
static byte	CONNECTION_PROPERTY_OCI_SVC_CTX_HANDLE_ACCESSMODE	
static String	CONNECTION_PROPERTY_OCI_SVC_CTX_HANDLE_DEFAULT	
static String	CONNECTION_PROPERTY_OCI_TENANCY	When CONNECTION_PROPERTY_PASSWORD_AUTHENTICATION is set to "OCI_TOKEN", this property must specify the Oracle Cloud ID (OCID) of the cloud tenant for the user that Oracle JDBC authenticates as.
static byte	CONNECTION_PROPERTY_OCI_TENANCY_ACCESSMODE	
static String	CONNECTION_PROPERTY_OCI_TENANCY_DEFAULT	
static String	CONNECTION_PROPERTY_ONS_PROTOCOL	Use this property to specify the ONS connection protocol, as either "TCP" or "TCPS".
static byte	CONNECTION_PROPERTY_ONS_PROTOCOL_ACCESSMODE	
static String	CONNECTION_PROPERTY_ONS_PROTOCOL_DEFAULT	
static String	CONNECTION_PROPERTY_ONS_WALLET_FILE	Use this property to specify the ONS wallet file, when you need Oracle Fast Application Notification (FAN).
static byte	CONNECTION_PROPERTY_ONS_WALLET_FILE_ACCESSMODE	
static String	CONNECTION_PROPERTY_ONS_WALLET_FILE_DEFAULT	

static byte	CONNECTION_PROPERTY_ONS_WALLET_PASSWORD_ACCESSMODE	
static String	CONNECTION_PROPERTY_ONS_WALLET_PASSWORD_DEFAULT	
static String	CONNECTION_PROPERTY_PASSWORD	The value of this property is used as the password when connecting to the database.
static byte	CONNECTION_PROPERTY_PASSWORD_ACCESSMODE	
static String	CONNECTION_PROPERTY_PASSWORD_AUTHENTICATION	Configures how Oracle JDBC performs authentication with a user name and password.
static byte	CONNECTION_PROPERTY_PASSWORD_AUTHENTICATION_ACCESSMODE	
static String	CONNECTION_PROPERTY_PASSWORD_AUTHENTICATION_DEFAULT	
static String	CONNECTION_PROPERTY_PASSWORD_DEFAULT	
static String	CONNECTION_PROPERTY_PRELIM_AUTH	If this property is set to "true", JDBC drivers connect in PRELIM_AUTH mode, which is the only mode that is permitted when the database is down.
static byte	CONNECTION_PROPERTY_PRELIM_AUTH_ACCESSMODE	
static String	CONNECTION_PROPERTY_PRELIM_AUTH_DEFAULT	
static String	CONNECTION_PROPERTY_PROCESS_ESCAPES	If the value of this property is "false" then the default setting for Statement.setEscapeProccessing is false.
static byte	CONNECTION_PROPERTY_PROCESS_ESCAPES_ACCESSMODE	
static String	CONNECTION_PROPERTY_PROCESS_ESCAPES_DEFAULT	
static String	CONNECTION_PROPERTY_PROTOCOL	
static byte	CONNECTION_PROPERTY_PROTOCOL_ACCESSMODE	
static String	CONNECTION_PROPERTY_PROTOCOL_DEFAULT	
static String	CONNECTION_PROPERTY_PROXY_CLIENT_NAME	The value of this property is used to specify the name of the proxy client during proxy authentication.
static byte	CONNECTION_PROPERTY_PROXY_CLIENT_NAME_ACCESSMODE	
static String	CONNECTION_PROPERTY_PROXY_CLIENT_NAME_DEFAULT	
static String	CONNECTION_PROPERTY_READONLY_INSTANCE_ALLOWED	This property allows connection creation to a read-only instance if the value is set to true.
static byte	CONNECTION_PROPERTY_READONLY_INSTANCE_ALLOWED_ACCESSMODE	
static String	CONNECTION_PROPERTY_READONLY_INSTANCE_ALLOWED_DEFAULT	
static String	CONNECTION_PROPERTY_REPORT_REMARKS	If the value of this property is "true", OracleDatabaseMetaData will include remarks in the meta data.
static byte	CONNECTION_PROPERTY_REPORT_REMARKS_ACCESSMODE	
static String	CONNECTION_PROPERTY_REPORT_REMARKS_DEFAULT	
static String	CONNECTION_PROPERTY_REQUEST_SIZE_LIMIT	Specifies the maximum request size, in terms of number of JDBC calls, beyond which AC replay will be disabled.
static byte	CONNECTION_PROPERTY_REQUEST_SIZE_LIMIT_ACCESSMODE	
static String	CONNECTION_PROPERTY_REQUEST_SIZE_LIMIT_DEFAULT	
static String	CONNECTION_PROPERTY_RESOURCE_MANAGER_ID	
static byte	CONNECTION_PROPERTY_RESOURCE_MANAGER_ID_ACCESSMODE	
static String	CONNECTION_PROPERTY_RESOURCE_MANAGER_ID_DEFAULT	

byte		
static String	CONNECTION_PROPERTY_RESTRICT_GETTABLES_DEFAULT	
static String	CONNECTION_PROPERTY_RETAIN_V9_BIND_BEHAVIOR	This is applicable only for the thin driver.
static byte	CONNECTION_PROPERTY_RETAIN_V9_BIND_BEHAVIOR_ACCESSMODE	
static String	CONNECTION_PROPERTY_RETAIN_V9_BIND_BEHAVIOR_DEFAULT	
static String	CONNECTION_PROPERTY_SERVER	
static byte	CONNECTION_PROPERTY_SERVER_ACCESSMODE	
static String	CONNECTION_PROPERTY_SERVER_DEFAULT	
static String	CONNECTION_PROPERTY_SET_FLOAT_AND_DOUBLE_USE_BINARY	If the value of this property is "true", the JDBC PreparedStatement setFloat and setDouble API's convert float and double values to the internal binary format for BINARY_FLOAT or BINARY_DOUBLE before sending to the database.
static byte	CONNECTION_PROPERTY_SET_FLOAT_AND_DOUBLE_USE_BINARY_ACCESSMODE	
static String	CONNECTION_PROPERTY_SET_FLOAT_AND_DOUBLE_USE_BINARY_DEFAULT	
static String	CONNECTION_PROPERTY_SET_NEW_PASSWORD	**Deprecated.**
static byte	CONNECTION_PROPERTY_SET_NEW_PASSWORD_ACCESSMODE	
static String	CONNECTION_PROPERTY_SET_NEW_PASSWORD_DEFAULT	
static String	CONNECTION_PROPERTY_SOCKS_PROXY_HOST	This connection property is used to configure the host name of the SOCKS proxy server.
static byte	CONNECTION_PROPERTY_SOCKS_PROXY_HOST_ACCESSMODE	
static String	CONNECTION_PROPERTY_SOCKS_PROXY_HOST_DEFAULT	
static String	CONNECTION_PROPERTY_SOCKS_PROXY_PORT	This connection property is used to configure the port value of the SOCKS proxy server.
static byte	CONNECTION_PROPERTY_SOCKS_PROXY_PORT_ACCESSMODE	
static String	CONNECTION_PROPERTY_SOCKS_PROXY_PORT_DEFAULT	
static String	CONNECTION_PROPERTY_SOCKS_REMOTE_DNS	This connection property is used to specify whether the DNS lookup for the DB Host should be performed locally or remotely when a SOCKS5 Proxy is being used.
static byte	CONNECTION_PROPERTY_SOCKS_REMOTE_DNS_ACCESSMODE	
static String	CONNECTION_PROPERTY_SOCKS_REMOTE_DNS_DEFAULT	
static String	CONNECTION_PROPERTY_SQL_ERROR_TRANSLATION_FILE	Path to an xml file which provides the Error code translations for those errors which occur if a connection can not be established to the server.
static byte	CONNECTION_PROPERTY_SQL_ERROR_TRANSLATION_FILE_ACCESSMODE	
static String	CONNECTION_PROPERTY_SQL_ERROR_TRANSLATION_FILE_DEFAULT	
static String	CONNECTION_PROPERTY_SQL_TRANSLATION_PROFILE	The string identifier for the translation profile or the translator to be used.
static byte	CONNECTION_PROPERTY_SQL_TRANSLATION_PROFILE_ACCESSMODE	
static String	CONNECTION_PROPERTY_SQL_TRANSLATION_PROFILE_DEFAULT	
static String	CONNECTION_PROPERTY_SSL_CONTEXT_PROTOCOL	Specifies a `protocol` name for the driver to use when obtaining an instance of `SSLContext` from `SSLContext.getInstance(String)` for a TLS enabled database connection.
static	CONNECTION_PROPERTY_SSL_CONTEXT_PROTOCOL_ACCESSMODE	

static byte	`CONNECTION_PROPERTY_STREAM_CHUNK_SIZE_ACCESSMODE`	
static String	`CONNECTION_PROPERTY_STREAM_CHUNK_SIZE_DEFAULT`	
static String	`CONNECTION_PROPERTY_STRICT_ASCII_CONVERSION`	The Oracle JDBC has been doing quick ASCII conversion (use only the low bytes) in different areas for the sake of performance.
static byte	`CONNECTION_PROPERTY_STRICT_ASCII_CONVERSION_ACCESSMODE`	
static String	`CONNECTION_PROPERTY_STRICT_ASCII_CONVERSION_DEFAULT`	
static String	`CONNECTION_PROPERTY_TCP_KEEPCOUNT`	Specifies a maximum number of keep alive probes to be sent before a connection is considered broken.
static byte	`CONNECTION_PROPERTY_TCP_KEEPCOUNT_ACCESSMODE`	
static String	`CONNECTION_PROPERTY_TCP_KEEPCOUNT_DEFAULT`	
static String	`CONNECTION_PROPERTY_TCP_KEEPIDLE`	Specifies a number of seconds for a network connection to remain idle before initiating a keep alive probe.
static byte	`CONNECTION_PROPERTY_TCP_KEEPIDLE_ACCESSMODE`	
static String	`CONNECTION_PROPERTY_TCP_KEEPIDLE_DEFAULT`	
static String	`CONNECTION_PROPERTY_TCP_KEEPINTERVAL`	Specifies a number of seconds to wait before retransmitting a keep alive probe.
static byte	`CONNECTION_PROPERTY_TCP_KEEPINTERVAL_ACCESSMODE`	
static String	`CONNECTION_PROPERTY_TCP_KEEPINTERVAL_DEFAULT`	
static String	`CONNECTION_PROPERTY_THIN_FORCE_DNS_LOAD_BALANCING`	When a hostname resolves to multiple addresses, the JDBC thin driver retrieves an array of addresses by calling "InetAddress.getAllByName()" and attempts to connect to first address in the array.
static byte	`CONNECTION_PROPERTY_THIN_FORCE_DNS_LOAD_BALANCING_ACCESSMODE`	
static String	`CONNECTION_PROPERTY_THIN_FORCE_DNS_LOAD_BALANCING_DEFAULT`	
static String	`CONNECTION_PROPERTY_THIN_HTTPS_PROXY_HOST`	Use this property to set the hostname or address of the https proxy server.
static byte	`CONNECTION_PROPERTY_THIN_HTTPS_PROXY_HOST_ACCESSMODE`	
static String	`CONNECTION_PROPERTY_THIN_HTTPS_PROXY_HOST_DEFAULT`	
static String	`CONNECTION_PROPERTY_THIN_HTTPS_PROXY_PORT`	Use this property to set the port of the https proxy server.
static byte	`CONNECTION_PROPERTY_THIN_HTTPS_PROXY_PORT_ACCESSMODE`	
static String	`CONNECTION_PROPERTY_THIN_HTTPS_PROXY_PORT_DEFAULT`	
static String	`CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORE`	Specify the file system path of a key store file which contains private keys and certificates used for TLS/SSL/TCPS authentication with a database.
static byte	`CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORE_ACCESSMODE`	
static String	`CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORE_DEFAULT`	
static String	`CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTOREPASSWORD`	Specify the password of a key store file specified by `CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORE`.
static byte	`CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTOREPASSWORD_ACCESSMODE`	
static String	`CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTOREPASSWORD_DEFAULT`	
static String	`CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORETYPE`	Specify the format of a key store file specified by `CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORE`.
static byte	`CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORETYPE_ACCESSMODE`	

String		contains certificate authorities that can be trusted when authenticating a database's certificate.
static byte	CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORE_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORE_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTOREPASSWORD	Specify the password of a trust store file specified by `CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORE`.
static byte	CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTOREPASSWORD_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTOREPASSWORD_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORETYPE	Specify the format of a trust store file specified by `CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORE`.
static byte	CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORETYPE_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORETYPE_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_JNDI_LDAP_CONNECT_TIMEOUT	Specify a timeout, in milliseconds, to apply when establishing a connection to an LDAP server.
static byte	CONNECTION_PROPERTY_THIN_JNDI_LDAP_CONNECT_TIMEOUT_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_JNDI_LDAP_CONNECT_TIMEOUT_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_JNDI_LDAP_READ_TIMEOUT	Specify a timeout, in milliseconds, to apply when reading a response from an LDAP server.
static byte	CONNECTION_PROPERTY_THIN_JNDI_LDAP_READ_TIMEOUT_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_JNDI_LDAP_READ_TIMEOUT_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_LDAP_SECURITY_AUTHENTICATION	Specifies the authentication mechanism to be used by the LDAP service provider in the JDK.
static byte	CONNECTION_PROPERTY_THIN_LDAP_SECURITY_AUTHENTICATION_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_LDAP_SECURITY_AUTHENTICATION_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_LDAP_SECURITY_CREDENTIALS	Use this property to configure the password which will be used while authenticating with the LDAP server.
static byte	CONNECTION_PROPERTY_THIN_LDAP_SECURITY_CREDENTIALS_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_LDAP_SECURITY_CREDENTIALS_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_LDAP_SECURITY_PRINCIPAL	Use this property to specify the value of the username(DN) which will be used while authenticating with the LDAP server.
static byte	CONNECTION_PROPERTY_THIN_LDAP_SECURITY_PRINCIPAL_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_LDAP_SECURITY_PRINCIPAL_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_CIPHER_SUITES	Use this property to specify the set of cipher suites to be used while SSL negotiation with LDAP server.
static byte	CONNECTION_PROPERTY_THIN_LDAP_SSL_CIPHER_SUITES_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_CIPHER_SUITES_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_CONTEXT_PROTOCOL	Specifies a `protocol` name for the driver to use when obtaining an instance of `SSLContext` from `SSLContext.getInstance(String)` for a TLS enabled LDAP connection.
static byte	CONNECTION_PROPERTY_THIN_LDAP_SSL_CONTEXT_PROTOCOL_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_CONTEXT_PROTOCOL_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYMANAGER_FACTORY_ALGORITHM	Use this property to override the default algorithm used by KeyManagerFactory.

static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE	This property is used to specify the path to the `KeyStore` file which will be used while SSL negotiation with LDAP server.
static byte	CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_PASSWORD	This property is used to specify the password for the KeyStore file which will be used while SSL negotiation with LDAP server.
static byte	CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_PASSWORD_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_PASSWORD_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_TYPE	Use this property to specify type of the configured `KeyStore` file to be used while SSL negotiation with LDAP Server.
static byte	CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_TYPE_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_TYPE_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTMANAGER_FACTORY_ALGORITHM	Use this property to override the default algorithm used by TrustManagerFactory.
static byte	CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTMANAGER_FACTORY_ALGORITHM_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTMANAGER_FACTORY_ALGORITHM_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE	This property is used to specify the path to the `TrustStore` file which will be used while SSL negotiation with LDAP server.
static byte	CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_PASSWORD	This property is used to specify the password for the TrustStore file which will be used while SSL negotiation with LDAP server.
static byte	CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_PASSWORD_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_PASSWORD_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_TYPE	Use this property to specify type of the configured `TrustStore` file to be used while SSL negotiation with LDAP Server.
static byte	CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_TYPE_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_TYPE_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_VERSIONS	Use this property to specify the valid SSL protocol version(s) used while SSL negotiation with LDAP Server.
static byte	CONNECTION_PROPERTY_THIN_LDAP_SSL_VERSIONS_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_VERSIONS_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_WALLET_LOCATION	Use this property to specify the wallet location.
static byte	CONNECTION_PROPERTY_THIN_LDAP_SSL_WALLET_LOCATION_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_WALLET_LOCATION_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_LDAP_SSL_WALLET_PASSWORD	Use this property to specify the password of the wallet file which will be used while SSL negotiation with LDAP server.
static byte	CONNECTION_PROPERTY_THIN_LDAP_SSL_WALLET_PASSWORD_ACCESSMODE	

String		strong session keys and weak ciphers can be disabled when *Native Network Encryption* is used.
static byte	CONNECTION_PROPERTY_THIN_NET_ALLOW_WEAK_CRYPTO_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_NET_ALLOW_WEAK_CRYPTO_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB_JAAS_LOGIN_MODULE	Use this connection property to specify the name of the JAAS login module which will be used for initializing `javax.security.auth.login.LoginContext`.
static byte	CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB_JAAS_LOGIN_MODULE_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB_JAAS_LOGIN_MODULE_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB_REALM	Use this connection property to specify the realm used for Kerberos authentication.
static byte	CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB_REALM_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB_REALM_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB5_CC_NAME	Use this connection property to specify the location of the Kerberos credential cache.
static byte	CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB5_CC_NAME_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB5_CC_NAME_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB5_MUTUAL	To turn on Kerberos mutual authentication, set this property to "true".
static byte	CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB5_MUTUAL_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB5_MUTUAL_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES	Use this connection property to turn on the authentication adaptors.
static byte	CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL	Use this property to specify the level of security for the integrity service.
static byte	CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES	Use this connection property to specify the list of integrity algorithms that you want to activate.
static byte	CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_NET_CONNECT_TIMEOUT	The connect timeout controls how much time is allowed to connect the socket to the database.
static byte	CONNECTION_PROPERTY_THIN_NET_CONNECT_TIMEOUT_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_NET_CONNECT_TIMEOUT_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_NET_CONNECTIONID_PREFIX	This property `oracle.net.connectionIdPrefix` can be used to customize the first 8 characters of the Net connection id that's sent to the listener during connection establishment for tracing purposes.
static byte	CONNECTION_PROPERTY_THIN_NET_CONNECTIONID_PREFIX_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_NET_CONNECTIONID_PREFIX_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_NET_CRYPTO_SEED	Use this connection property to specify the encryption seed (between 10 and 70 random characters).

static String	CONNECTION_PROPERTY_THIN_NET_DISABLE_OUT_OF_BAND_BREAK	Thin uses out of band breaks by default from 11g onwards.
static byte	CONNECTION_PROPERTY_THIN_NET_DISABLE_OUT_OF_BAND_BREAK_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_NET_DISABLE_OUT_OF_BAND_BREAK_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL	Use this property to specify the level of security for the encryption service.
static byte	CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES	Use this connection property to specify the list of encryption algorithms that you want to activate.
static byte	CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_NET_OLDSYNTAX	
static byte	CONNECTION_PROPERTY_THIN_NET_OLDSYNTAX_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_NET_OLDSYNTAX_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_NET_PROFILE	**Deprecated.**
static byte	CONNECTION_PROPERTY_THIN_NET_PROFILE_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_NET_PROFILE_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_NET_SET_FIPS_MODE	Use this connection property to enable `FIPS140-2` mode for native network encryption.
static byte	CONNECTION_PROPERTY_THIN_NET_SET_FIPS_MODE_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_NET_SET_FIPS_MODE_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_NET_USE_ZERO_COPY_IO	The thin driver uses the zero-copy IO codepath for SecureFile Lobs by default from 11gR2 onwards.
static byte	CONNECTION_PROPERTY_THIN_NET_USE_ZERO_COPY_IO_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_NET_USE_ZERO_COPY_IO_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_OUTBOUND_CONNECT_TIMEOUT	The outbound connect timeout controls the time allowed to connect the socket, let the server accept the connection to the desired service, negotiate the NS protocol as well as complete the ASO negotiation.
static byte	CONNECTION_PROPERTY_THIN_OUTBOUND_CONNECT_TIMEOUT_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_OUTBOUND_CONNECT_TIMEOUT_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_READ_TIMEOUT	Read timeout while reading from the socket.
static byte	CONNECTION_PROPERTY_THIN_READ_TIMEOUT_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_READ_TIMEOUT_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_SSL_CERTIFICATE_ALIAS	When a keystore (either wallet or jks file) contains multiple certificates, this property can be used to specify the alias name of the certificate to be used by the driver during the client authentication part of the SSL handshake.
static byte	CONNECTION_PROPERTY_THIN_SSL_CERTIFICATE_ALIAS_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_SSL_CERTIFICATE_ALIAS_DEFAULT	

byte		
static String	CONNECTION_PROPERTY_THIN_SSL_CIPHER_SUITES_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_SSL_KEYMANAGERFACTORY_ALGORITHM	Specify the algorithm to use when managing the key meterial of the key store file specified by CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORE.
static byte	CONNECTION_PROPERTY_THIN_SSL_KEYMANAGERFACTORY_ALGORITHM_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_SSL_KEYMANAGERFACTORY_ALGORITHM_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_SSL_SERVER_CERT_DN	Use this connection property to specify the distinguished name (DN) of the server used during the SSL handshake to authenticate the server.
static byte	CONNECTION_PROPERTY_THIN_SSL_SERVER_CERT_DN_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_SSL_SERVER_CERT_DN_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_SSL_SERVER_DN_MATCH	Use this connection property to enable or disable the authentication of the server during the SSL handshake.
static byte	CONNECTION_PROPERTY_THIN_SSL_SERVER_DN_MATCH_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_SSL_SERVER_DN_MATCH_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_SSL_TRUSTMANAGERFACTORY_ALGORITHM	Specify the algorithm to use when managing the meterial of the trust store file specified by CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORE.
static byte	CONNECTION_PROPERTY_THIN_SSL_TRUSTMANAGERFACTORY_ALGORITHM_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_SSL_TRUSTMANAGERFACTORY_ALGORITHM_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_SSL_VERSION	Sets the SSL version which will be used for SSL protocol negotiation. This is an optional property.
static byte	CONNECTION_PROPERTY_THIN_SSL_VERSION_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_SSL_VERSION_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_TCP_NO_DELAY	If the value of this property is "true", the TCP_NODELAY property is set on the socket when using the Thin driver.
static byte	CONNECTION_PROPERTY_THIN_TCP_NO_DELAY_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_TCP_NO_DELAY_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_USE_JCE_API	If the value of this property is "`true`" and Data Encryption service is enabled, then JDK Crypto(JCE) APIs are used for encryption and decryption of the data between the JDBC client and the Oracle Server, otherwise the built-in crypto implementation is used.
static byte	CONNECTION_PROPERTY_THIN_USE_JCE_API_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_USE_JCE_API_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_VSESSION_ENAME	
static byte	CONNECTION_PROPERTY_THIN_VSESSION_ENAME_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_VSESSION_ENAME_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_VSESSION_INAME	
static byte	CONNECTION_PROPERTY_THIN_VSESSION_INAME_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_VSESSION_INAME_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_VSESSION_MACHINE	Use this connection property to change the value that will show up in the "machine" column of the "v$session" table for this connection.

static String	CONNECTION_PROPERTY_THIN_VSESSION_OSUSER	By default, the driver retrieves the OS username from the "user.name" system property which is set by the JVM.
static byte	CONNECTION_PROPERTY_THIN_VSESSION_OSUSER_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_VSESSION_OSUSER_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_VSESSION_PROCESS	Use this connection property to change the value that will show up in the "process" column of the "v$session" table for this connection.
static byte	CONNECTION_PROPERTY_THIN_VSESSION_PROCESS_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_VSESSION_PROCESS_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_VSESSION_PROGRAM	Use this connection property to change the value that will show up in the "program" column of the "v$session" table for this connection.
static byte	CONNECTION_PROPERTY_THIN_VSESSION_PROGRAM_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_VSESSION_PROGRAM_DEFAULT	
static String	CONNECTION_PROPERTY_THIN_VSESSION_TERMINAL	Use this connection property to change the value that will show up in the "terminal" column of the "v$session" table for this connection.
static byte	CONNECTION_PROPERTY_THIN_VSESSION_TERMINAL_ACCESSMODE	
static String	CONNECTION_PROPERTY_THIN_VSESSION_TERMINAL_DEFAULT	
static String	CONNECTION_PROPERTY_TIMESTAMPTZ_IN_GMT	Obtain TIMESTAMP WITH TIME ZONE value in GMT than adjusting the same to local time zone.
static byte	CONNECTION_PROPERTY_TIMESTAMPTZ_IN_GMT_ACCESSMODE	
static String	CONNECTION_PROPERTY_TIMESTAMPTZ_IN_GMT_DEFAULT	
static String	CONNECTION_PROPERTY_TIMEZONE_AS_REGION	Use JVM default timezone as specified rather than convert to a GMT offset.
static byte	CONNECTION_PROPERTY_TIMEZONE_AS_REGION_ACCESSMODE	
static String	CONNECTION_PROPERTY_TIMEZONE_AS_REGION_DEFAULT	
static String	CONNECTION_PROPERTY_TNS_ADMIN	This property is used for setting the TNS Admin path.
static byte	CONNECTION_PROPERTY_TNS_ADMIN_ACCESSMODE	
static String	CONNECTION_PROPERTY_TNS_ADMIN_DEFAULT	
static String	CONNECTION_PROPERTY_TOKEN_AUTHENTICATION	Enables the use of access tokens that are stored in a file system location when authenticating with Oracle Database.
static byte	CONNECTION_PROPERTY_TOKEN_AUTHENTICATION_ACCESSMODE	
static String	CONNECTION_PROPERTY_TOKEN_AUTHENTICATION_DEFAULT	
static String	CONNECTION_PROPERTY_TOKEN_LOCATION	When `CONNECTION_PROPERTY_TOKEN_AUTHENTICATION` is set to "OCI_TOKEN" or "OAUTH", this property specifies the file system path to obtain access tokens from.
static byte	CONNECTION_PROPERTY_TOKEN_LOCATION_ACCESSMODE	
static String	CONNECTION_PROPERTY_TOKEN_LOCATION_DEFAULT	
static String	CONNECTION_PROPERTY_USE_1900_AS_YEAR_FOR_TIME	setTime used to set the date component to 01 Jan, 1900 by default in earlier versions (version < 10g).
static byte	CONNECTION_PROPERTY_USE_1900_AS_YEAR_FOR_TIME_ACCESSMODE	
static String	CONNECTION_PROPERTY_USE_1900_AS_YEAR_FOR_TIME_DEFAULT	

static String	CONNECTION_PROPERTY_USE_DRCP_MULTIPLE_TAG_DEFAULT	
static String	CONNECTION_PROPERTY_USE_FETCH_SIZE_WITH_LONG_COLUMN	If the value of this property is "true", then JDBC will prefetch rows even though there is a LONG or LONG RAW column in the result.
static byte	CONNECTION_PROPERTY_USE_FETCH_SIZE_WITH_LONG_COLUMN_ACCESSMODE	
static String	CONNECTION_PROPERTY_USE_FETCH_SIZE_WITH_LONG_COLUMN_DEFAULT	
static String	CONNECTION_PROPERTY_USE_SHARDING_DRIVER_CONNECTION	This is applicable only for the thin driver.
static byte	CONNECTION_PROPERTY_USE_SHARDING_DRIVER_CONNECTION_ACCESSMODE	
static String	CONNECTION_PROPERTY_USE_SHARDING_DRIVER_CONNECTION_DEFAULT	
static String	CONNECTION_PROPERTY_USE_THREADLOCAL_BUFFER_CACHE	If true, the statement data buffers are cached on a per thread basis.
static byte	CONNECTION_PROPERTY_USE_THREADLOCAL_BUFFER_CACHE_ACCESSMODE	
static String	CONNECTION_PROPERTY_USE_THREADLOCAL_BUFFER_CACHE_DEFAULT	
static String	CONNECTION_PROPERTY_USER_NAME	The value of this property is used as the user name when connecting to the database.
static byte	CONNECTION_PROPERTY_USER_NAME_ACCESSMODE	
static String	CONNECTION_PROPERTY_USER_NAME_DEFAULT	
static String	CONNECTION_PROPERTY_WALLET_LOCATION	Use this property to specify the wallet location.
static byte	CONNECTION_PROPERTY_WALLET_LOCATION_ACCESSMODE	
static String	CONNECTION_PROPERTY_WALLET_LOCATION_DEFAULT	
static String	CONNECTION_PROPERTY_WALLET_PASSWORD	Use this property to set the wallet password which is only required if you don't enable enable auto-login in the wallet.
static byte	CONNECTION_PROPERTY_WALLET_PASSWORD_ACCESSMODE	
static String	CONNECTION_PROPERTY_WALLET_PASSWORD_DEFAULT	
static String	CONNECTION_PROPERTY_WEBSOCKET_PASSWORD	This connection property is used to configure the password of the webserver, when the JDBC Thin driver is configured to connect to a webserver using the `Secure Websocket protocol (WSS)`. The webserver acts as a reverse proxy for the Oracle Database. The default value of this property is `null`.
static byte	CONNECTION_PROPERTY_WEBSOCKET_PASSWORD_ACCESSMODE	
static String	CONNECTION_PROPERTY_WEBSOCKET_PASSWORD_DEFAULT	
static String	CONNECTION_PROPERTY_WEBSOCKET_USER	This connection property is used to configure the username of the webserver, when the JDBC Thin driver is configured to connect to a webserver using the `Secure Websocket protocol (WSS)`. The webserver acts as a reverse proxy for the Oracle Database. The default value of this property is `null`.
static byte	CONNECTION_PROPERTY_WEBSOCKET_USER_ACCESSMODE	
static String	CONNECTION_PROPERTY_WEBSOCKET_USER_DEFAULT	
static int	CONNECTION_RELEASE_HIGH	
static int	CONNECTION_RELEASE_LOCKED	
static int	CONNECTION_RELEASE_LOW	

int		database connection is not closed but the database is not reachable.
static int	DATABASE_OK	Define return values for pingDatabase api The physical database connection is not closed and the database is reachable.
static int	DATABASE_TIMEOUT	Define return values for pingDatabase api The call timed out before any positive or negative acknowledgement was received.
static String	DCN_BEST_EFFORT	
static String	DCN_CLIENT_INIT_CONNECTION	Set the value of DCN_CLIENT_INIT_CONNECTION to 'true' for using the Client initiated DCN connection.
static String	DCN_IGNORE_DELETEOP	
static String	DCN_IGNORE_INSERTOP	
static String	DCN_IGNORE_UPDATEOP	
static String	DCN_NOTIFY_CHANGELAG	
static String	DCN_NOTIFY_ROWIDS	
static String	DCN_QUERY_CHANGE_NOTIFICATION	
static String	DCN_USE_HOST_CONNECTION_ADDR_INFO	Set the value of DCN_USE_HOST_CONNECTION_ADDR_INFO to 'false' to use the address info returned by the server for establishing the client initiated DCN Connection.
static int	END_TO_END_ACTION_INDEX	
static int	END_TO_END_CLIENTID_INDEX	
static int	END_TO_END_ECID_INDEX	
static int	END_TO_END_MODULE_INDEX	
static int	END_TO_END_STATE_INDEX_MAX	
static int	INVALID_CONNECTION	Values used for close(int).
static String	NETWORK_COMPRESSION_AUTO	
static String	NETWORK_COMPRESSION_LEVEL_HIGH	
static int	NETWORK_COMPRESSION_LEVEL_HIGH_VALUE	
static String	NETWORK_COMPRESSION_LEVEL_LOW	
static int	NETWORK_COMPRESSION_LEVEL_LOW_VALUE	
static String	NETWORK_COMPRESSION_OFF	
static String	NETWORK_COMPRESSION_ON	
static int	NETWORK_COMPRESSION_THRESHOLD_MIN	Minimum value supported by the connection property `CONNECTION_PROPERTY_NETWORK_COMPRESSION_THRESHOLD`.
static String	NTF_AQ_PAYLOAD	
static String	NTF_ASYNC_DEQ	
static int	NTF_DEFAULT_TCP_PORT	
static String	NTF_GROUPING_CLASS	
static String	NTF_GROUPING_CLASS_NONE	

static String	NTF_GROUPING_REPEAT_TIME	
static String	NTF_GROUPING_START_TIME	
static String	NTF_GROUPING_TYPE	
static String	NTF_GROUPING_TYPE_LAST	
static String	NTF_GROUPING_TYPE_SUMMARY	
static String	NTF_GROUPING_VALUE	
static String	NTF_LOCAL_HOST	
static String	NTF_LOCAL_TCP_PORT	
static String	NTF_QOS_AUTO_ACK	
static String	NTF_QOS_PURGE_ON_NTFN	
static String	NTF_QOS_RELIABLE	
static String	NTF_QOS_SECURE	
static String	NTF_QOS_TX_ACK	
static String	NTF_TIMEOUT	
static String	NTF_USE_SSL	
static String	OCSID_ACTION_KEY	
static String	OCSID_CLIENT_INFO_KEY	
static String	OCSID_CLIENTID_KEY	
static String	OCSID_DBOP_KEY	
static String	OCSID_ECID_KEY	
static String	OCSID_MODULE_KEY	
static String	OCSID_NAMESPACE	Special namespace for sending end-to-end metrics.
static String	OCSID_SEQUENCE_NUMBER_KEY	
static String	PROXY_CERTIFICATE	
static String	PROXY_DISTINGUISHED_NAME	
static String	PROXY_ROLES	
static int	PROXY_SESSION	Values used for close(int).
static String	PROXY_TYPE	
static String	PROXY_USER_NAME	
static String	PROXY_USER_PASSWORD	
static int	PROXYTYPE_CERTIFICATE	
static int	PROXYTYPE_DISTINGUISHED_NAME	
static int	PROXYTYPE_USER_NAME	

Fields inherited from interface java.sql.Connection

`TRANSACTION_NONE`, `TRANSACTION_READ_COMMITTED`, `TRANSACTION_READ_UNCOMMITTED`, `TRANSACTION_REPEATABLE_READ`, `TRANSACTION_SERIALIZABLE`

Method Summary

All Methods Instance Methods Abstract Methods Default Methods Deprecated Methods

Modifier and Type	Method	Description
`Connection`	`_getPC()`	Return the underlying physical connection if this is a logical connection.
`void`	`abort()`	Calling abort() on an open connection does the following: marks the connection as closed, closes any sockets or other primitive connections to the database, and insures that any thread that is currently accessing the connection will either progress to completion of the JDBC call or throw an exception.
`void`	`addLogicalTransactionIdEventListener (LogicalTransactionIdEventListener listener)`	Registers a listener to Logical Transaction Id events.
`void`	`addLogicalTransactionIdEventListener (LogicalTransactionIdEventListener listener, Executor executor)`	This flavor of `addLogicalTransactionIdEventListener` can be used to register a listener with an executor.
`default void`	`applyConnectionAttributes(Properties connAttr)`	**Deprecated.** The Implicit Connection Cache (ICC) has been desupported since 12.1.
`void`	`archive(int mode, int aseq, String acstext)`	**Deprecated.** This method will be removed in a future version.
`boolean`	`attachServerConnection()`	This method needs to be called before using a DRCP connection.
`void`	`beginRequest()`	Declares that a request to the server is starting on this connection.
`void`	`cancel()`	Performs an immediate (asynchronous) termination of any currently executing operation on this connection.
`void`	`clearAllApplicationContext(String nameSpace)`	**Deprecated.** This is deprecated since 12.1 in favor of the standard JDBC API setClientInfo().
`void`	`close(int opt)`	If opt is OracleConnection.INVALID_CONNECTION : Closes the given Logical connection, as well the underlying PooledConnection without returning the connection to the cache when called with the parameter INVALID_CONNECTION.
`default void`	`close(Properties connAttr)`	**Deprecated.** The Implicit Connection Cache (ICC) has been desupported since 12.1.
`default Flow.Publisher<Void>`	`closeAsyncOracle()`	Releases this Connection object's database and JDBC resources immediately.
`void`	`commit (EnumSet<OracleConnection.CommitOption> options)`	Commits the transaction with the given options.
`default Flow.Publisher<Void>`	`commitAsyncOracle()`	Asynchronously make all changes made since the previous commit/rollback permanent and releases any database locks currently held by this `Connection` object.
`ARRAY`	`createARRAY(String typeName, Object elements)`	Creates an ARRAY object with the given type name and elements.
`BINARY_DOUBLE`	`createBINARY_DOUBLE(double value)`	Creates a BINARY_DOUBLE that has the given value.
`BINARY_FLOAT`	`createBINARY_FLOAT(float value)`	Creates a BINARY_FLOAT that has the given value.
`DATE`	`createDATE(String value)`	Creates a DATE that has the given value.
`DATE`	`createDATE(Date value)`	Creates a DATE that has the given value.
`DATE`	`createDATE(Date value, Calendar cal)`	Creates a DATE that has the given value.
`DATE`	`createDATE(Time value)`	Creates a DATE that has the given value.
`DATE`	`createDATE(Time value, Calendar cal)`	Creates a DATE that has the given value.
`DATE`	`createDATE(Timestamp value)`	Creates a DATE that has the given value.
`DATE`	`createDATE(Timestamp value, Calendar cal)`	Creates a DATE that has the given value.
`INTERVALDS`	`createINTERVALDS(String value)`	Creates an INTERVALDS that has the given value.

NUMBER	createNUMBER(byte value)	Creates a new NUMBER that has the given value.
NUMBER	createNUMBER(double value)	Creates a new NUMBER that has the given value.
NUMBER	createNUMBER(float value)	Creates a new NUMBER that has the given value.
NUMBER	createNUMBER(int value)	Creates a new NUMBER that has the given value.
NUMBER	createNUMBER(long value)	Creates a new NUMBER that has the given value.
NUMBER	createNUMBER(short value)	Creates a new NUMBER that has the given value.
NUMBER	createNUMBER(String value, int scale)	Creates a new NUMBER that has the given value and scale.
NUMBER	createNUMBER(BigDecimal value)	Creates a new NUMBER that has the given value.
NUMBER	createNUMBER(BigInteger value)	Creates a new NUMBER that has the given value.
Array	createOracleArray(String arrayTypeName, Object elements)	Creates an Array object with the given type name and elements.
TIMESTAMP	createTIMESTAMP(String value)	Creates a new TIMESTAMP with the given value.
TIMESTAMP	createTIMESTAMP(Date value)	Creates a new TIMESTAMP with the given value.
TIMESTAMP	createTIMESTAMP(Time value)	Creates a new TIMESTAMP with the given value.
TIMESTAMP	createTIMESTAMP(Timestamp value)	Creates a new TIMESTAMP with the given value.
TIMESTAMP	createTIMESTAMP(Timestamp value, Calendar cal)	Creates a new TIMESTAMP with the given value.
TIMESTAMP	createTIMESTAMP(DATE value)	Creates a new TIMESTAMP with the given value.
TIMESTAMPLTZ	createTIMESTAMPLTZ(String value, Calendar cal)	Creates a new TIMESTAMPLTZ with the given value.
TIMESTAMPLTZ	createTIMESTAMPLTZ(Date value, Calendar cal)	Creates a new TIMESTAMPLTZ with the given value.
TIMESTAMPLTZ	createTIMESTAMPLTZ(Time value, Calendar cal)	Creates a new TIMESTAMPLTZ with the given value.
TIMESTAMPLTZ	createTIMESTAMPLTZ(Timestamp value, Calendar cal)	Creates a new TIMESTAMPLTZ with the given value.
TIMESTAMPLTZ	createTIMESTAMPLTZ(DATE value, Calendar cal)	Creates a new TIMESTAMPLTZ with the given value.
TIMESTAMPTZ	createTIMESTAMPTZ(String value)	Creates a new TIMESTAMPTZ with the given value.
TIMESTAMPTZ	createTIMESTAMPTZ(String value, Calendar cal)	Creates a new TIMESTAMPTZ with the given value.
TIMESTAMPTZ	createTIMESTAMPTZ(Date value)	Creates a new TIMESTAMPTZ with the given value.
TIMESTAMPTZ	createTIMESTAMPTZ(Date value, Calendar cal)	Creates a new TIMESTAMPTZ with the given value.
TIMESTAMPTZ	createTIMESTAMPTZ(Time value)	Creates a new TIMESTAMPTZ with the given value.
TIMESTAMPTZ	createTIMESTAMPTZ(Time value, Calendar cal)	Creates a new TIMESTAMPTZ with the given value.
TIMESTAMPTZ	createTIMESTAMPTZ(Timestamp value)	Creates a new TIMESTAMPTZ with the given value.
TIMESTAMPTZ	createTIMESTAMPTZ(Timestamp value, java.time.ZoneId tzid)	Creates a new TIMESTAMPTZ with the given value.
TIMESTAMPTZ	createTIMESTAMPTZ(Timestamp value, Calendar cal)	Creates a new TIMESTAMPTZ with the given value.
TIMESTAMPTZ	createTIMESTAMPTZ(DATE value)	
AQMessage	dequeue(String queueName, AQDequeueOptions opt, byte[] tdo)	Dequeues an AQ message from the queue specified by its name.
AQMessage	dequeue(String queueName, AQDequeueOptions opt, byte[] tdo, int version)	Dequeues an AQ message from the queue specified by its name.
AQMessage[]	dequeue(String queueName, AQDequeueOptions opt, byte[] tdo, int version, int deqsize)	Dequeues an array of AQ messages from the queue specified by its name.
AQMessage	dequeue(String queueName, AQDequeueOptions opt, String typeName)	Dequeues an AQ message from the queue specified by its name.
AQMessage[]	dequeue(String queueName, AQDequeueOptions opt, String typeName, int deqsize)	Dequeues an array of AQ messages from the queue specified by its name.
void	detachServerConnection(String tag)	Notify the server that this connection will not be used.
void	disableLogging()	Disables the logging for the connection.
void	dumpLog()	Dumps the log for the connection to the configured dump location.
void	enableLogging()	Enables logging for the connection.
void	endRequest()	Declares that the request that was in progress on this connection has completed.

	AQMessage[] mesgs)	specified by its name.
TypeDescriptor[]	getAllTypeDescriptorsInCurrentSchema()	Obtain all the type descriptors associated with object types or array in the schema of this connection.
String	getAuthenticationAdaptorName()	Returns the name of the adaptor that is used for authentication by the thin driver.
boolean	getAutoClose()	The driver is always in auto-close mode.
CallableStatement	getCallWithKey(String key)	getCallWithKey Searches the explicit cache for a match on key.
String	getChecksumProviderName()	If network integrity service is enabled, returns the name of the checksum provider, otherwise returns `null`.
default Properties	getConnectionAttributes()	**Deprecated.** The Implicit Connection Cache (ICC) has been desupported since 12.1.
default int	getConnectionReleasePriority()	**Deprecated.** The Implicit Connection Cache (ICC) has been desupported since 12.1.
boolean	getCreateStatementAsRefCursor()	Retrieves the current setting of the `createStatementAsRefCursor` flag which you can set with the `setCreateStatementAsRefCursor` method.
String	getCurrentSchema()	Obtains the current schema of the current connection.
DatabaseChangeRegistration	getDatabaseChangeRegistration(int regid)	Maps an existing registration identified by its ID 'regid' with a new DatabaseChangeRegistration object.
String	getDataIntegrityAlgorithmName()	Returns the name of the algorithm that is used for data integrity checking by the thin driver on the network.
int	getDefaultExecuteBatch()	**Deprecated.** As of 12.1 all APIs related to oracle-style statement batching are deprecated in favor of standard JDBC batching.
int	getDefaultRowPrefetch()	Retrieves the value of row prefetch for all statements associated with this connection and created after this value was set.
TimeZone	getDefaultTimeZone()	Returns the TimeZone set through setDefaultTimeZone.
Object	getDescriptor(String sql_name)	Gets a Descriptor object corresponding to a sql type.
String	getDRCPPLSQLCallbackName()	Returns the PL/SQL Fix-up callback name if configured, otherwise returns `Null`
String	getDRCPReturnTag()	Returns the tag associated with this DRCP pooled server.
OracleConnection.DRCPState	getDRCPState()	Returns an enum indicating if the connection is attached to a DRCP server process.
String	getEncryptionAlgorithmName()	Returns the name of the algorithm that is used for data encryption by the thin driver on the network.
String	getEncryptionProviderName()	If network encryption service is enabled, returns the name of the encryption provider, otherwise returns `null`.
short	getEndToEndECIDSequenceNumber()	**Deprecated.** This is deprecated since 12.1 in favor of getClientInfo().
String[]	getEndToEndMetrics()	**Deprecated.** This has been deprecated since 12.1 in favor of getClientInfo().
boolean	getExplicitCachingEnabled()	getExplicitCachingEnabled Returns true if the explicit cache is currently enabled, false otherwise.
boolean	getImplicitCachingEnabled()	getImplicitCachingEnabled Returns true if the implicit cache is currently enabled, false otherwise.
boolean	getIncludeSynonyms()	Checks whether or not synonyms information is included in `DatabaseMetaData.getColumns`.
Object	getJavaObject(String sql_name)	**Deprecated.**
oracle.jdbc.diagnostics.SecuredLogger	getLogger()	Returns the SecuredLogger instance of the OracleConnection.
LogicalTransactionId	getLogicalTransactionId()	Gets the current Logical Transaction Id which are sent by the server in a piggy back message and hence this method call doesn't make a roundtrip.
String	getNetConnectionId()	Returns the Net Connection ID associated with this connection.

		REMARKS column.
boolean	**getRestrictGetTables**()	Gets the restriction status of the returned data in DatabaseMetaData.getTables.
String	**getSessionTimeZone**()	Obtain Oracle session time zone region name.
String	**getSessionTimeZoneOffset**()	Obtain the time zone offset in hours of the current database session.
String	**getSQLType**(**Object** obj)	**Deprecated.**
int	**getStatementCacheSize**()	getStatementCacheSize Returns the current size of the application cache.
PreparedStatement	**getStatementWithKey**(**String** key)	getStatementWithKey Searches the explicit cache for a match on key.
int	**getStmtCacheSize**()	**Deprecated.** Use getStatementCacheSize() instead.
short	**getStructAttrCsId**()	Obtain the Oracle identifier of the character set used in STRUCT attributes.
TypeDescriptor[]	**getTypeDescriptorsFromList** (**String**[][] schemaAndTypeNamePairs)	Obtain the type descriptors associated with object types or arrays from an array of scheama and type names.
TypeDescriptor[]	**getTypeDescriptorsFromListInCurrentSchema** (**String**[] typeNames)	Obtain the type descriptors associated with object types or array in a schema from an array of type names.
default **Properties**	**getUnMatchedConnectionAttributes**()	**Deprecated.** The Implicit Connection Cache (ICC) has been desupported since 12.1.
String	**getUserName**()	Gets the user name of the current connection.
boolean	**getUsingXAFlag**()	**Deprecated.**
boolean	**getXAErrorFlag**()	**Deprecated.**
boolean	**isDRCPEnabled**()	Returns true if the connection is participating in DRCP.
boolean	**isDRCPMultitagEnabled**()	Returns true if multiple tags are allowed with DRCP Connection.
boolean	**isLogicalConnection**()	Method that returns a boolean indicating whether its a logical connection or not.
boolean	**isProxySession**()	Returns true if the current session associated with this connection is a proxy session.
boolean	**isUsable**()	Identifies whether this connection is still usable for JDBC operations.
boolean	**isValid** (**OracleConnection.ConnectionValidation** effort, int timeout)	Returns true if this connection was working properly to the extent specified by effort at the instant during this call it was checked.
boolean	**needToPurgeStatementCache**()	Returns if the client side Statement cache has to be purged.
void	**openProxySession**(int type, **Properties** prop)	Opens a new proxy session with the username provided in the prop argument and switches to this new session. This feature is supported for both thin and oci driver. Three proxy types are supported : OracleConnection.PROXYTYPE_USER_NAME : In this type PROXY_USER_NAME needs to be provided in prop.
void	**oracleReleaseSavepoint**(**OracleSavepoint** savepoint)	Removes the given OracleSavepoint object from the current transaction.
void	**oracleRollback**(**OracleSavepoint** savepoint)	Undoes all changes made after the given OracleSavepoint object was set.
OracleSavepoint	**oracleSetSavepoint**()	**Deprecated.**
OracleSavepoint	**oracleSetSavepoint**(**String** name)	Creates a savepoint with the given name in the current transaction and returns the new OracleSavepoint object that represents it.
int	**pingDatabase**()	Ping Database server to see if both database and the connection are actively up.
int	**pingDatabase**(int timeOut)	**Deprecated.**

		cached with this KEY.
`PreparedStatement`	`prepareStatementWithKey(String key)`	**Deprecated.** This is same as prepareStatement, except if a Prepared Statement with the given KEY exists in the Cache, then the statement is returned AS IT IS when it was closed and cached with this KEY.
`void`	`purgeExplicitCache()`	purgeExplicitCache Removes all existing statements from the explicit cache, after which it will be empty.
`void`	`purgeImplicitCache()`	purgeImplicitCache Removes all existing statements from the implicit cache, after which it will be empty.
`void`	`putDescriptor(String sql_name, Object desc)`	Store the Object Descriptor for later usage.
`AQNotificationRegistration[]`	`registerAQNotification(String[] name, Properties[] options, Properties globaloptions)`	Registers your interest into being notified when a message is enqueued in a particular queue (or array of queues).
`default void`	`registerConnectionCacheCallback(OracleConnectionCacheCallback occc, Object userObj, int cbkFlag)`	**Deprecated.** The Implicit Connection Cache (ICC) has been desupported since 12.1.
`DatabaseChangeRegistration`	`registerDatabaseChangeNotification(Properties options)`	/ Creates a new database change registration.
`void`	`registerSQLType(String sql_name, Class<?> java_class)`	**Deprecated.**
`void`	`registerSQLType(String sql_name, String java_class_name)`	**Deprecated.**
`void`	`registerTAFCallback(OracleOCIFailover cbk, Object obj)`	Register an application TAF Callback instance that will be called when an application failover occurs.
`void`	`removeLogicalTransactionIdEventListener(LogicalTransactionIdEventListener listener)`	Deregisters the Logical Transaction Id event listener.
`default Flow.Publisher<Void>`	`rollbackAsyncOracle()`	Undoes all changes made in the current transaction and releases any database locks currently held by this Connection object.
`void`	`setApplicationContext(String nameSpace, String attribute, String value)`	**Deprecated.** This has been deprecated since 12.1 in favour of setClientInfo().
`void`	`setAutoClose(boolean autoClose)`	set auto-close mode.
`default void`	`setConnectionReleasePriority(int priority)`	**Deprecated.** The Implicit Connection Cache (ICC) has been desupported since 12.1.
`void`	`setCreateStatementAsRefCursor(boolean value)`	When this is set to `true`, any new statements created from this connection will be created as a `REF CURSOR`.
`void`	`setDefaultExecuteBatch(int batch)`	**Deprecated.** As of 12.1 all APIs related to oracle-style statement batching are deprecated in favor of standard JDBC batching.
`void`	`setDefaultRowPrefetch(int value)`	Sets the value of row prefetch for all statements associated with this connection and created after this value was set.
`void`	`setDefaultTimeZone(TimeZone tz)`	The TimeZone to be used while creating java.sql.Date, java.sql.Time & java.sql.Timestamp.
`void`	`setEndToEndMetrics(String[] metrics, short sequenceNumber)`	**Deprecated.** It has been deprecated since 12.1 in favor of setClientInfo().
`void`	`setExplicitCachingEnabled(boolean cache)`	setExplicitCachingEnabled Enables or disables the explicit cache.
`void`	`setImplicitCachingEnabled(boolean cache)`	setImplicitCachingEnabled Enables or disables the implicit cache.
`void`	`setIncludeSynonyms(boolean synonyms)`	Turns on or off retrieval of synonym information in DatabaseMetaData.
`void`	`setPlsqlWarnings(String setting)`	Enable/Disable PLSQL Compiler Warnings
`void`	`setRemarksReporting(boolean reportRemarks)`	Turns on or off the reporting of the REMARKS columns by the `getTables` and `getColumns` calls of the `DatabaseMetaData` interface.
`void`	`setRestrictGetTables(boolean restrict)`	Turns on or off the restriction of the returned data in `DatabaseMetaData.getTables`.
`void`	`setSessionTimeZone(String regionName)`	Set the session time zone.

boolean	**setShardingKeyIfValid** (**OracleShardingKey** shardingKey, int timeout)	Checks the validity of the connection and also checks if the sharding key passed to this method is valid for the connection.If the sharding key is valid, it will be set on the connection.
boolean	**setShardingKeyIfValid** (**OracleShardingKey** shardingKey, **OracleShardingKey** superShardingKey, int timeout)	Checks the validity of the connection and also checks if the sharding keys passed to this method are valid for the connection.If the sharding keys are valid, it will be set on the connection.
void	**setStatementCacheSize**(int size)	setStatementCacheSize Specifies the size of the size of the application cache (which will be used by both implicit and explicit caching).
void	**setStmtCacheSize**(int size)	**Deprecated.** Use setStatementCacheSize() instead.
void	**setStmtCacheSize**(int size, boolean clearMetaData)	**Deprecated.** Use setStatementCacheSize() instead.
void	**setUsingXAFlag**(boolean value)	**Deprecated.**
void	**setWrapper**(**OracleConnection** wrapper)	Set the wrapping object.
void	**setXAErrorFlag**(boolean value)	**Deprecated.**
void	**shutdown** (**OracleConnection.DatabaseShutdownMode** mode)	Shuts the database server down.
void	**startup**(**String** startup_str, int mode)	**Deprecated.** This method will be removed in a future version.
void	**startup**(**OracleConnection.DatabaseStartupMode** mode)	Starts the database server up.
void	**startup**(**OracleConnection.DatabaseStartupMode** mode, **String** pfileName)	Starts the database server up.
void	**unregisterAQNotification** (**AQNotificationRegistration** registration)	Deletes a given AQ registration.
void	**unregisterDatabaseChangeNotification** (int registrationId)	**Deprecated.**
void	**unregisterDatabaseChangeNotification** (int registrationId, **String** host, int tcpport)	**Deprecated.**
void	**unregisterDatabaseChangeNotification** (long registrationId, **String** callback)	Deletes a given database change registration in the server.
void	**unregisterDatabaseChangeNotification** (**DatabaseChangeRegistration** registration)	Deletes a given database change registration.
OracleConnection	**unwrap**()	Return the wrapped object if any else null.

Methods inherited from interface java.sql.Connection

`abort, clearWarnings, close, commit, createArrayOf, createBlob, createClob, createNClob, createSQLXML, createStatement, createStatement, createStatement, createStruct, getAutoCommit, getCatalog, getClientInfo, getClientInfo, getHoldability, getMetaData, getNetworkTimeout, getSchema, getTransactionIsolation, getTypeMap, getWarnings, isClosed, isReadOnly, isValid, nativeSQL, prepareCall, prepareCall, prepareCall, prepareStatement, prepareStatement, prepareStatement, prepareStatement, prepareStatement, prepareStatement, releaseSavepoint, rollback, rollback, setAutoCommit, setCatalog, setClientInfo, setClientInfo, setHoldability, setNetworkTimeout, setReadOnly, setSavepoint, setSavepoint, setSchema, setShardingKey, setShardingKey, setShardingKeyIfValid, setShardingKeyIfValid, setTransactionIsolation, setTypeMap`

Methods inherited from interface java.sql.Wrapper

`isWrapperFor, unwrap`

Field Detail

ACCESSMODE_JAVAPROP

```
static final byte ACCESSMODE_JAVAPROP
```

Bitmask which can be applied to the `CONNECTION_PROPERTY_{name}_ACCESSMODE` constants defined in this interface.
Indicates that the driver will read a connection property from a Properties object.

See Also:

Constant Field Values

ACCESSMODE_SYSTEMPROP

```
static final byte ACCESSMODE_SYSTEMPROP
```

ACCESSMODE_BOTH

```
static final byte ACCESSMODE_BOTH
```

Bitmask which can be applied to the `CONNECTION_PROPERTY_{name}_ACCESSMODE` constants defined in this interface.
Indicates that the driver will read a connection property from either a Properties object or the JVM's system properties. If a property is defined in both sources, the value given by the Properties object takes precedence.

See Also:
Constant Field Values

ACCESSMODE_FILEPROP

```
static final byte ACCESSMODE_FILEPROP
```

Bitmask which can be applied to the `CONNECTION_PROPERTY_{name}_ACCESSMODE` constants defined in this interface.
Indicates that the driver will read a connection property from a properties file.

See Also:
`CONNECTION_PROPERTY_CONFIG_FILE`, Constant Field Values

CONNECTION_PROPERTY_RETAIN_V9_BIND_BEHAVIOR

```
static final String CONNECTION_PROPERTY_RETAIN_V9_BIND_BEHAVIOR
```

This is applicable only for the thin driver. Pass "true" to retain the V9 bind behavior for Long and potential long binds. "false" is the default behavior which would emulate the same behavior as in OCI driver.

See Also:
Constant Field Values

CONNECTION_PROPERTY_RETAIN_V9_BIND_BEHAVIOR_DEFAULT

```
static final String CONNECTION_PROPERTY_RETAIN_V9_BIND_BEHAVIOR_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_RETAIN_V9_BIND_BEHAVIOR_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_RETAIN_V9_BIND_BEHAVIOR_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_USER_NAME

```
static final String CONNECTION_PROPERTY_USER_NAME
```

The value of this property is used as the user name when connecting to the database. Note that there are other ways to set the username: in the URL or in a wallet but the value of this property overwrites any other value.

See Also:
Constant Field Values

CONNECTION_PROPERTY_USER_NAME_DEFAULT

```
static final String CONNECTION_PROPERTY_USER_NAME_DEFAULT
```

CONNECTION_PROPERTY_USER_NAME_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_USER_NAME_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_NEW_PASSWORD

```
static final String CONNECTION_PROPERTY_NEW_PASSWORD
```

CONNECTION_PROPERTY_NEW_PASSWORD_DEFAULT

```
static final String CONNECTION_PROPERTY_NEW_PASSWORD_DEFAULT
```

CONNECTION_PROPERTY_NEW_PASSWORD_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_NEW_PASSWORD_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_DATABASE

```
static final String CONNECTION_PROPERTY_DATABASE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_DATABASE_DEFAULT

```
static final String CONNECTION_PROPERTY_DATABASE_DEFAULT
```

CONNECTION_PROPERTY_DATABASE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_DATABASE_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_AUTOCOMMIT

```
static final String CONNECTION_PROPERTY_AUTOCOMMIT
```

Use this connection property to change the default value of autoCommit.

See Also:
Constant Field Values

CONNECTION_PROPERTY_AUTOCOMMIT_DEFAULT

```
static final String CONNECTION_PROPERTY_AUTOCOMMIT_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_AUTOCOMMIT_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_AUTOCOMMIT_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_PROTOCOL

```
static final String CONNECTION_PROPERTY_PROTOCOL
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_PROTOCOL_DEFAULT

```
static final String CONNECTION_PROPERTY_PROTOCOL_DEFAULT
```

Constant Field Values

CONNECTION_PROPERTY_STREAM_CHUNK_SIZE

```
static final String CONNECTION_PROPERTY_STREAM_CHUNK_SIZE
```

Deprecated.

Stream chunk size for input streams. This property is deprecated since 12.1 and is no longer used internally.

See Also:
Constant Field Values

CONNECTION_PROPERTY_STREAM_CHUNK_SIZE_DEFAULT

```
static final String CONNECTION_PROPERTY_STREAM_CHUNK_SIZE_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_STREAM_CHUNK_SIZE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_STREAM_CHUNK_SIZE_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_SET_FLOAT_AND_DOUBLE_USE_BINARY

```
static final String CONNECTION_PROPERTY_SET_FLOAT_AND_DOUBLE_USE_BINARY
```

If the value of this property is "true", the JDBC PreparedStatement setFloat and setDouble API's convert float and double values to the internal binary format for BINARY_FLOAT or BINARY_DOUBLE before sending to the database. If the property is not set or set to other than "true", the setFloat and setDouble API's convert float and double values to the internal format for NUMBER. See the JavaDoc for setBinaryFloat in oracle.jdbc.PreparedStatement. Use only for 10g or later databases.

See Also:
Constant Field Values

CONNECTION_PROPERTY_SET_FLOAT_AND_DOUBLE_USE_BINARY_DEFAULT

```
static final String CONNECTION_PROPERTY_SET_FLOAT_AND_DOUBLE_USE_BINARY_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_SET_FLOAT_AND_DOUBLE_USE_BINARY_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_SET_FLOAT_AND_DOUBLE_USE_BINARY_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_VSESSION_TERMINAL

```
static final String CONNECTION_PROPERTY_THIN_VSESSION_TERMINAL
```

Use this connection property to change the value that will show up in the "terminal" column of the "v$session" table for this connection. Note that this setting only applies to the thin driver.

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_VSESSION_TERMINAL_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_VSESSION_TERMINAL_DEFAULT
```

See Also:
Constant Field Values

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_VSESSION_MACHINE

```
static final String CONNECTION_PROPERTY_THIN_VSESSION_MACHINE
```

Use this connection property to change the value that will show up in the "machine" column of the "v$session" table for this connection. Note that this setting only applies to the thin driver.

If you don't specify a value, by default, the driver will attempt to retrieve your host name. If the attempt fails, it will use "jdbcclient".

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_VSESSION_MACHINE_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_VSESSION_MACHINE_DEFAULT
```

CONNECTION_PROPERTY_THIN_VSESSION_MACHINE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_VSESSION_MACHINE_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_VSESSION_OSUSER

```
static final String CONNECTION_PROPERTY_THIN_VSESSION_OSUSER
```

By default, the driver retrieves the OS username from the "user.name" system property which is set by the JVM. You can however override this value by using this connection property (thin driver only).

The OS username will show up in the "osuser" column of the "v$session" table for this connection.

If you don't specify any value and if the JVM's "user.name" system property is null, the value will be set to "jdbcuser".

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_VSESSION_OSUSER_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_VSESSION_OSUSER_DEFAULT
```

CONNECTION_PROPERTY_THIN_VSESSION_OSUSER_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_VSESSION_OSUSER_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_VSESSION_PROGRAM

```
static final String CONNECTION_PROPERTY_THIN_VSESSION_PROGRAM
```

Use this connection property to change the value that will show up in the "program" column of the "v$session" table for this connection. Note that this setting only applies to the thin driver.

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_VSESSION_PROGRAM_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_VSESSION_PROGRAM_DEFAULT
```

See Also:
Constant Field Values

Constant Field Values

CONNECTION_PROPERTY_THIN_VSESSION_PROCESS

`static final String CONNECTION_PROPERTY_THIN_VSESSION_PROCESS`

Use this connection property to change the value that will show up in the "process" column of the "v$session" table for this connection. Note that this setting only applies to the thin driver.

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_VSESSION_PROCESS_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_VSESSION_PROCESS_DEFAULT`

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_VSESSION_PROCESS_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_VSESSION_PROCESS_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_VSESSION_INAME

`static final String CONNECTION_PROPERTY_THIN_VSESSION_INAME`

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_VSESSION_INAME_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_VSESSION_INAME_DEFAULT`

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_VSESSION_INAME_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_VSESSION_INAME_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_VSESSION_ENAME

`static final String CONNECTION_PROPERTY_THIN_VSESSION_ENAME`

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_VSESSION_ENAME_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_VSESSION_ENAME_DEFAULT`

CONNECTION_PROPERTY_THIN_VSESSION_ENAME_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_VSESSION_ENAME_ACCESSMODE`

See Also:
Constant Field Values

Deprecated.

This property no longer has any effect.

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_NET_PROFILE_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_NET_PROFILE_DEFAULT
```

CONNECTION_PROPERTY_THIN_NET_PROFILE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_NET_PROFILE_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES

```
static final String CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES
```

Use this connection property to turn on the authentication adaptors. The adaptors are "RADIUS", "KERBEROS" or "TCPS" which is SSL authentication.

For example, to turn on KERBEROS authentication:

```
    Properties prop = new Properties();
    prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES,
                     "( KERBEROS )");
    prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB5_MUTUAL,
                     "true");
    // Specify where my krb5 configuration file is because JSSE can't find it:
    System.setProperty("java.security.krb5.conf","C:\\WINDOWS\\krb5.ini");
```

Or to turn on RADIUS:

```
    prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES,
                     "( RADIUS)");
```

Or to turn on RADIUS, KERBEROS and SSL authentication adaptors:

```
    prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES,
                     "( RADIUS, KERBEROS, SSL)");
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES_DEFAULT
```

CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB5_MUTUAL

```
static final String CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB5_MUTUAL
```

To turn on Kerberos mutual authentication, set this property to "true".

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB5_MUTUAL_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB5_MUTUAL_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB5_CC_NAME

```
static final String CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB5_CC_NAME
```

Use this connection property to specify the location of the Kerberos credential cache.

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB5_CC_NAME_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB5_CC_NAME_DEFAULT
```

CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB5_CC_NAME_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB5_CC_NAME_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB_REALM

```
static final String CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB_REALM
```

Use this connection property to specify the realm used for Kerberos authentication.

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB_REALM_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB_REALM_DEFAULT
```

CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB_REALM_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB_REALM_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB_JAAS_LOGIN_MODULE

```
static final String CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB_JAAS_LOGIN_MODULE
```

Use this connection property to specify the name of the JAAS login module which will be used for initializing `javax.security.auth.login.LoginContext`. This property works in conjunction with the JAAS configuration. Please refer JDK documentation to know about configuring the JAAS config file. By default the Thin driver uses `com.sun.security.auth.module.Krb5LoginModule` as LoginModule and the Kerberos Credential Cache configured via `CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB5_CC_NAME` is used for retreiving the TGT. The default value is `null`. The JAAS configuration is used only if this property is configured with login module name.

See Also:
`CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB5_CC_NAME`, `CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_SERVICES`, Constant Field Values

CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB_JAAS_LOGIN_MODULE_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB_JAAS_LOGIN_MODULE_DEFAULT
```

CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB_JAAS_LOGIN_MODULE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_NET_AUTHENTICATION_KRB_JAAS_LOGIN_MODULE_ACCESSMODE
```

See Also:
Constant Field Values

Use this connection property to enable `FIPS140-2` mode for native network encryption. The default key pair generation mechanism is not `FIPS140-2` compliant and fails to generate a Diffie-Hellman key pair while negotiating the network encryption. Set the value to `true` to enable `FIPS-140-2` mode. The default value is `false` and `FIPS140-2` mode is disabled.

Since:

21c

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_NET_SET_FIPS_MODE_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_NET_SET_FIPS_MODE_DEFAULT
```

CONNECTION_PROPERTY_THIN_NET_SET_FIPS_MODE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_NET_SET_FIPS_MODE_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_NET_ALLOW_WEAK_CRYPTO

```
static final String CONNECTION_PROPERTY_THIN_NET_ALLOW_WEAK_CRYPTO
```

Starting in release 23, the driver can be configured to use strong session keys and weak ciphers can be disabled when *Native Network Encryption* is used. But this breaks compatibility with older versions of the server which may not have the same capability. An exception with error code `12268` is thrown when server does not have this capabilty. By default, in order to allow compatibility with such servers, and at the cost of security, this property is set to "true". When this property is configured to "false" then strong session keys are used and the following algorithms are disabled. Disabled Encryption algorithms : RC4_40, DES40, DES, RC4_56, RC4_128, 3DES112, 3DES168 and RC4_256. Disabled Checksum algorithms : MD5. This property can also be configured through URL. The value configured through URL has higher priority than the value configured through connection properties. Please see the below examples to know how to configure this property through URL.
EzConnectPlus Format:
jdbc:oracle:thin:@//host:port/servicename?allow_weak_crypto=false
TNS Long URL Format:
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=host)(PORT=5521))(CONNECT_DATA= (SERVICE_NAME=servicename))(Security=(ALLOW_WEAK_CRYPTO=false)))

Since:

23c

See Also:

`CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL`, `CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES`, `CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES`, `CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL`, Constant Field Values

CONNECTION_PROPERTY_THIN_NET_ALLOW_WEAK_CRYPTO_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_NET_ALLOW_WEAK_CRYPTO_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_NET_ALLOW_WEAK_CRYPTO_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_NET_ALLOW_WEAK_CRYPTO_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL

```
static final String CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL
```

Use this property to specify the level of security for the encryption service. In order of increasing security the parameter can be (note that the default is "ACCEPTED"):<bt>

- "REJECTED": Select this value if you do not elect to enable the encryption service, even if required by the other side. In this scenario, this side of the connection specifies that the encryption service is not permitted. If the other side is set to REQUIRED, the connection terminates with error message ORA-12650. If the other side is set to REQUESTED, ACCEPTED, or REJECTED, the connection continues without error and without the encryption service enabled.
- "ACCEPTED": Select this value to enable the encryption service if required or requested by the other side. In this scenario, this side of the connection does not require the encryption service, but it is enabled if the other side is set to REQUIRED or REQUESTED. If the other side is set to REQUIRED or REQUESTED, and an encryption algorithm match is found, the connection continues without error and with the encryption service enabled. If the other side is set to REQUIRED and no algorithm match is found, the connection terminates with error message ORA-12650. If the other side is set to REQUESTED and no algorithm match is found, or if the other side is set to ACCEPTED or REJECTED, the connection continues without error and without the security service enabled.
- "REQUESTED": Select this value to enable the encryption service if the other side permits it. In this scenario, this side of the connection specifies that the

This property can also be configured through URL. The value configured through URL has higher priority than the value configured through connection properties.
Please see the below examples to know how to configure this property through URL.
EzConnectPlus Format:
jdbc:oracle:thin:@//host:port/servicename?encryption_client=REQUIRED
TNS Long URL Format:
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=host)(PORT=5521))(CONNECT_DATA= (SERVICE_NAME=servicename))(Security=(ENCRYPTION_CLIENT=REQUESTED)))

See Also:

`CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES`, `CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL`, Constant Field Values

CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL_DEFAULT
```

CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES

```
static final String CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES
```

Use this connection property to specify the list of encryption algorithms that you want to activate.

The supported algorithms are:

- "AES256": AES 256-bit key
- "AES192": AES 192-bit key
- "AES128": AES 128-bit key

The following weak algorithms are disabled by default:

- "3DES168": 3-key 3DES
- "3DES112": 2-key 3DES
- "DES56C": DES 56-bit key CBC
- "DES40C": DES 40-bit key CBC
- "RC4_256": RC4 256-bit key
- "RC4_128": RC4 128-bit key
- "RC4_56": RC4 56-bit key
- "RC4_40": RC4 40-bit key

For example, if you require the connection to be encrypted with either AES256 or AES192, you would set the following properties:

```
    prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES,
                     "( AES256, AES192 )");
    prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL,
                     "REQUIRED");
```

This property can also be configured through URL. The value configured through URL has higher priority than the value configured through connection properties.
Please see the below examples to know how to configure this property through URL.
EzConnectPlus Format:
jdbc:oracle:thin:@//host:port/servicename?encryption_client=required&encryption_types_client=AES128,AES192
TNS Long URL Format:
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=host)(PORT=5521))(CONNECT_DATA= (SERVICE_NAME=servicename))(Security=(ENCRYPTION_CLIENT=REQUESTED)(ENCRYPTION_TYPES_CLIENT=AES128, AES192)))

See Also:

`CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL`, `CONNECTION_PROPERTY_THIN_NET_ALLOW_WEAK_CRYPTO`, Constant Field Values

CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES_DEFAULT
```

CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL

configure this property through URL.
EzConnectPlus Format:
jdbc:oracle:thin:@//host:port/servicename?crypto_checksum_client=required
TNS Long URL Format:
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=host)(PORT=5521))(CONNECT_DATA= (SERVICE_NAME=servicename))(Security=(CRYPTO_CHECKSUM_CLIENT=REQUESTED)))

See Also:

`CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL`, `CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES`, Constant Field Values

CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL_DEFAULT
```

CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES

```
static final String CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES
```

Use this connection property to specify the list of integrity algorithms that you want to activate.

The supported algorithms are:

- "SHA1"
- "SHA256"
- "SHA384"
- "SHA512"
- "MD5" (disabled by default)

For example, if you require checksumming to be turned on and you want either MD5, SHA1, SHA256, SHA384 or SHA512:

```
prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES,
                 "( MD5, SHA1, SHA256, SHA384 or SHA512 )");
prop.setProperty(OracleConnection.CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL,
                 "REQUIRED");
```

This property can also be configured through URL. The value configured through URL has higher priority than the value configured through the connection properties. Please see the below examples to know how to configure this property through URL.
EzConnectPlus Format:
jdbc:oracle:thin:@//host:port/servicename?crypto_checksum_client=required&crypto_checksum_types_client=sha256,sha1
TNS Long URL Format:
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=host)(PORT=5521))(CONNECT_DATA= (SERVICE_NAME=servicename))(Security=(CRYPTO_CHECKSUM_CLIENT=REQUESTED)(CRYPTO_CHECKSUM_TYPES_CLIENT=SHA256,SHA1)))

See Also:

`CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL`, `CONNECTION_PROPERTY_THIN_NET_ALLOW_WEAK_CRYPTO`, Constant Field Values

CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES_DEFAULT
```

CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_NET_CRYPTO_SEED

```
static final String CONNECTION_PROPERTY_THIN_NET_CRYPTO_SEED
```

Use this connection property to specify the encryption seed (between 10 and 70 random characters). The encryption seed for the client should not be the same as that for the server. Note that you don't have to specify a seed on the client.

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_NET_CRYPTO_SEED_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_NET_CRYPTO_SEED_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_USE_JCE_API

```
static final String CONNECTION_PROPERTY_THIN_USE_JCE_API
```

If the value of this property is `"true"` and Data Encryption service is enabled, then JDK Crypto(JCE) APIs are used for encryption and decryption of the data between the JDBC client and the Oracle Server, otherwise the built-in crypto implementation is used. Since 19.1, the default value is `true` for the Thin driver. If the JVM version is older than 1.8.0_u151, then you need to change the JVM security policy to allow unlimited key sizes. This is done by downloading and replacing the files found in `$JAVA_HOME/lib/security` (`local_policy.jar` and `US_export_policy.jar`). Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files can be downloaded from Oracle website. When this property value is set to `"true"` and the Unlimited Crypto Strength is not available then the Thin driver will automatically change this property value to `"false"` and will use the built-in crypto implementation.

See Also:

`CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_TYPES`, `CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL`, `CONNECTION_PROPERTY_THIN_NET_CHECKSUM_TYPES`, `CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL`, Constant Field Values

CONNECTION_PROPERTY_THIN_USE_JCE_API_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_USE_JCE_API_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_USE_JCE_API_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_USE_JCE_API_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_TCP_NO_DELAY

```
static final String CONNECTION_PROPERTY_THIN_TCP_NO_DELAY
```

If the value of this property is "true", the TCP_NODELAY property is set on the socket when using the Thin driver. See java.net.SocketOptions.TCP_NODELAY. Can be either a system property or a connection property

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_TCP_NO_DELAY_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_TCP_NO_DELAY_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_TCP_NO_DELAY_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_TCP_NO_DELAY_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_READ_TIMEOUT

```
static final String CONNECTION_PROPERTY_THIN_READ_TIMEOUT
```

Read timeout while reading from the socket. This affects only the thin driver. The value is in milliseconds. Starting from 12.2 this timeout value is set right after the socket establishment and this read timeout will be applicable to the initial NS Protocol negotiation as well. Since 18.1 this value can be followed by 'ms', 'sec' or 'min' (case not sensitive) to indicate 'milliseconds', 'seconds' or 'minutes'.

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_READ_TIMEOUT_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_READ_TIMEOUT_ACCESSMODE`

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_OUTBOUND_CONNECT_TIMEOUT

`static final String CONNECTION_PROPERTY_THIN_OUTBOUND_CONNECT_TIMEOUT`

The outbound connect timeout controls the time allowed to connect the socket, let the server accept the connection to the desired service, negotiate the NS protocol as well as complete the ASO negotiation. It doesn't include the user authentication.
This value is assumed in seconds if no explicit unit is indicated (both in the URL and in the connection's properties).
Default value is `"0"` (no timeout).
This affects only the thin driver.
This property can also be set through connection URL. The outbound timeout value set in the connection URL overrides the value set using connection properties.
Since 18.1 this value can be followed by 'ms', 'sec' or 'min' (case not sensitive) to indicate 'milliseconds', 'seconds' or 'minutes'. Here is an example of how to set the outbound connect timeout to 10 seconds through the URL:

```
jdbc:oracle:thin:@(DESCRIPTION=(CONNECT_TIMEOUT=10)
(ADDRESS_LIST=(ADDRESS=(HOST=myhost)(PORT=5521)(PROTOCOL=tcp)))(CONNECT_DATA=(SERVICE_NAME=myService)))
```

Since:

12.2

See Also:

`CONNECTION_PROPERTY_THIN_NET_CONNECT_TIMEOUT`, Constant Field Values

CONNECTION_PROPERTY_THIN_OUTBOUND_CONNECT_TIMEOUT_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_OUTBOUND_CONNECT_TIMEOUT_DEFAULT`

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_OUTBOUND_CONNECT_TIMEOUT_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_OUTBOUND_CONNECT_TIMEOUT_ACCESSMODE`

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_NET_CONNECT_TIMEOUT

`static final String CONNECTION_PROPERTY_THIN_NET_CONNECT_TIMEOUT`

The connect timeout controls how much time is allowed to connect the socket to the database. Successfully connecting the socket doesn't necessarily mean that the database service is up but it means that the listener is accepting connections.
This value is assumed in seconds if set in the URL with no explicit units and in milliseconds if set in the connection's properties.
Default value is `"0"` (no timeout). This affects only the thin driver. The connect timeout can also be set through the connection URL using TRANSPORT_CONNECT_TIMEOUT like in the example below. The value set in the URL overrides the value set in this property.
Since 18.1 this value can be followed by 'ms', 'sec' or 'min' (case not sensitive) to indicate 'milliseconds', 'seconds' or 'minutes'.
Here is an example of how to set the connect timeout to 5 seconds through the URL:

```
jdbc:oracle:thin:@(DESCRIPTION=(TRANSPORT_CONNECT_TIMEOUT=5)
(ADDRESS_LIST=(ADDRESS=(HOST=myhost)(PORT=5521)(PROTOCOL=tcp)))(CONNECT_DATA=(SERVICE_NAME=myService)))
```

See Also:

`CONNECTION_PROPERTY_THIN_OUTBOUND_CONNECT_TIMEOUT`, Constant Field Values

CONNECTION_PROPERTY_THIN_NET_CONNECT_TIMEOUT_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_NET_CONNECT_TIMEOUT_DEFAULT`

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_NET_CONNECT_TIMEOUT_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_NET_CONNECT_TIMEOUT_ACCESSMODE`

See Also:

Constant Field Values

Thin uses out of band breaks by default from 11g onwards. If the user prefers to use inband breaks instead of Out of Band ones then this property could be set to true.

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_NET_DISABLE_OUT_OF_BAND_BREAK_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_NET_DISABLE_OUT_OF_BAND_BREAK_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_NET_DISABLE_OUT_OF_BAND_BREAK_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_NET_DISABLE_OUT_OF_BAND_BREAK_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_NET_USE_ZERO_COPY_IO

```
static final String CONNECTION_PROPERTY_THIN_NET_USE_ZERO_COPY_IO
```

The thin driver uses the zero-copy IO codepath for SecureFile Lobs by default from 11gR2 onwards. To use the regular codepath, set this property to "false".

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_NET_USE_ZERO_COPY_IO_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_NET_USE_ZERO_COPY_IO_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_NET_USE_ZERO_COPY_IO_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_NET_USE_ZERO_COPY_IO_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_USE_1900_AS_YEAR_FOR_TIME

```
static final String CONNECTION_PROPERTY_USE_1900_AS_YEAR_FOR_TIME
```

setTime used to set the date component to 01 Jan, 1900 by default in earlier versions (version < 10g). However, this changed after 10.1 where the time date component in the time was also honored by Jdbc. This flag is introduced to retain the old behavior (as in 9.2)

See Also:
Constant Field Values

CONNECTION_PROPERTY_USE_1900_AS_YEAR_FOR_TIME_DEFAULT

```
static final String CONNECTION_PROPERTY_USE_1900_AS_YEAR_FOR_TIME_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_USE_1900_AS_YEAR_FOR_TIME_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_USE_1900_AS_YEAR_FOR_TIME_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_TIMESTAMPTZ_IN_GMT

```
static final String CONNECTION_PROPERTY_TIMESTAMPTZ_IN_GMT
```

CONNECTION_PROPERTY_TIMESTAMPTZ_IN_GMT_DEFAULT

`static final String CONNECTION_PROPERTY_TIMESTAMPTZ_IN_GMT_DEFAULT`

See Also:
Constant Field Values

CONNECTION_PROPERTY_TIMESTAMPTZ_IN_GMT_ACCESSMODE

`static final byte CONNECTION_PROPERTY_TIMESTAMPTZ_IN_GMT_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_TIMEZONE_AS_REGION

`static final String CONNECTION_PROPERTY_TIMEZONE_AS_REGION`

Use JVM default timezone as specified rather than convert to a GMT offset. Default is true.

See Also:
Constant Field Values

CONNECTION_PROPERTY_TIMEZONE_AS_REGION_DEFAULT

`static final String CONNECTION_PROPERTY_TIMEZONE_AS_REGION_DEFAULT`

See Also:
Constant Field Values

CONNECTION_PROPERTY_TIMEZONE_AS_REGION_ACCESSMODE

`static final byte CONNECTION_PROPERTY_TIMEZONE_AS_REGION_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_SSL_CERTIFICATE_ALIAS

`static final String CONNECTION_PROPERTY_THIN_SSL_CERTIFICATE_ALIAS`

When a keystore (either wallet or jks file) contains multiple certificates, this property can be used to specify the alias name of the certificate to be used by the driver during the client authentication part of the SSL handshake.

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_SSL_CERTIFICATE_ALIAS_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_SSL_CERTIFICATE_ALIAS_DEFAULT`

CONNECTION_PROPERTY_THIN_SSL_CERTIFICATE_ALIAS_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_SSL_CERTIFICATE_ALIAS_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_SSL_SERVER_DN_MATCH

`static final String CONNECTION_PROPERTY_THIN_SSL_SERVER_DN_MATCH`

Use this connection property to enable or disable the authentication of the server during the SSL handshake. Authenticating the server means that the driver will verify that the Distinguished Name (DN) of the server's certificate matches the one that's specified either in the connection string using "ssl_server_cert_dn" or through the connection property "oracle.net.ssl_server_cert_dn". Starting in 18.0, the driver will automatically authenticate the server if its DN is specified. You can use this property to disable authentication by setting the value to "false".

See Also:
`CONNECTION_PROPERTY_THIN_SSL_SERVER_CERT_DN`, Constant Field Values

```
static final String CONNECTION_PROPERTY_THIN_SSL_SERVER_DN_MATCH_DEFAULT
```

CONNECTION_PROPERTY_THIN_SSL_SERVER_DN_MATCH_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_SSL_SERVER_DN_MATCH_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_SSL_SERVER_CERT_DN

```
static final String CONNECTION_PROPERTY_THIN_SSL_SERVER_CERT_DN
```

Use this connection property to specify the distinguished name (DN) of the server used during the SSL handshake to authenticate the server. This value can also be configured in the URL using the parameter "ssl_server_cert_dn". The value set in the URL overrides the value set in this property.

Since:

18.0

See Also:

`CONNECTION_PROPERTY_THIN_SSL_SERVER_DN_MATCH`, Constant Field Values

CONNECTION_PROPERTY_THIN_SSL_SERVER_CERT_DN_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_SSL_SERVER_CERT_DN_DEFAULT
```

CONNECTION_PROPERTY_THIN_SSL_SERVER_CERT_DN_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_SSL_SERVER_CERT_DN_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_SSL_VERSION

```
static final String CONNECTION_PROPERTY_THIN_SSL_VERSION
```

Sets the SSL version which will be used for SSL protocol negotiation.
This is an optional property. By default the JDBC thin driver uses all the protocols supported by JVM out of which SSLv3 and SSLv2Hello are excluded since they are not supported by the Oracle Database starting in 12.2.
Following are the valid values and the corresponding protocol versions used during negotiation.
`"0"` - TLSv1.0 or TLSv1.1 or TLSv1.2
`"undetermined"` - TLSv1.0 or TLSv1.1 or TLSv1.2
`"2"` - SSLv2Hello
`"2.0"` - SSLv2Hello
`"version 2"` - SSLv2Hello
`"3"` - SSLv3
`"3.0"` - SSLv3
`"version 3 only"` - SSLv3
`"1"` - TLSv1.0
`"1.0"` - TLSv1.0
`"version 1 only"` - TLSv1.0
`"1 or 3"` - TLSv1.0 or SSLv3
`"1.0 or 3.0"` - TLSv1.0 or SSLv3
`"version 1 or version 3"` - TLSv1.0 or SSLv3
`"1.1"` - TLSv1.1
`"1.2"` - TLSv1.2
`"1.1 or 3.0"` - TLSv1.1 or SSLv3
`"1.2 or 3.0"` - TLSv1.2 or SSLv3
`"1.1 or 1.0"` - TLSv1.1 or TLSv1.0
`"1.2 or 1.0"` - TLSv1.2 or TLSv1.0
`"1.2 or 1.1"` - TLSv1.2 or TLSv1.1
`"1.1 or 1.0 or 3.0"` - TLSv1.1 or TLSv1.0 or SSLv3
`"1.2 or 1.0 or 3.0"` - TLSv1.2 or TLSv1.0 or SSLv3
`"1.2 or 1.1 or 1.0"` - TLSv1.2 or TLSv1.1 or TLSv1.0
`"1.2 or 1.1 or 3.0"` - TLSv1.2 or TLSv1.1 or SSLv3
`"1.2 or 1.1 or 1.0 or 3.0"` - TLSv1.2 or TLSv1.1 or TLSv1.0 or SSLv3

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_SSL_VERSION_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_SSL_VERSION_DEFAULT
```

Constant Field Values

CONNECTION_PROPERTY_THIN_SSL_CIPHER_SUITES

`static final String CONNECTION_PROPERTY_THIN_SSL_CIPHER_SUITES`

Specify a comma separated list of cipher suites to enable for TLS communications with a database. The list must include at least one cipher suite that is also enabled for the database. If the driver and the database do not share a common cipher, then connection establishment will result in a TLS handshake failure.

Note that the standard list of cipher suite names may be found in the JSSE Cipher Suite Names section of the Java Cryptography Architecture Standard Algorithm Name Documentation. Providers may support cipher suite names not found in this list or might not use the recommended name for a certain cipher suite.

If no value is set for this property, the driver will use the set of cipher suites which your JSSE Security Provider has enabled by default.

This property is only supported by the jdbc:oracle:thin driver. This property can also be configured through the EZConnect Plus URL format using the parameter `SSL_CIPHERS`. The value configured via EZConnect URL has higher priority. Please note that the JDBC Thin driver does not support configuring this property through the parameter `SSL_CIPHER_SUITES` in long TNS URL format.

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_SSL_CIPHER_SUITES_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_SSL_CIPHER_SUITES_DEFAULT`

CONNECTION_PROPERTY_THIN_SSL_CIPHER_SUITES_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_SSL_CIPHER_SUITES_ACCESSMODE`

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORE

`static final String CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORE`

Specify the file system path of a key store file which contains private keys and certificates used for TLS/SSL/TCPS authentication with a database.

This property has no effect if `CONNECTION_PROPERTY_WALLET_LOCATION` has also been set.

This property is only supported by the jdbc:oracle:thin driver.

See Also:

`CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORETYPE`, `CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTOREPASSWORD`, Constant Field Values

CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORE_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORE_DEFAULT`

CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORE_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORE_ACCESSMODE`

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORETYPE

`static final String CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORETYPE`

Specify the format of a key store file specified by `CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORE`. Examples of commonly used formats are "SSO", "PKCS12", and "JKS".

This property has no effect if `CONNECTION_PROPERTY_WALLET_LOCATION` has also been set.

If this property is not set, the JDBC driver will attempt to automatically recognize the key store type based on the file extension of the key store.

Automatic Type Recognition

Recognized Type	File Extension(s)
SSO	.sso
PKCS12	.p12 or .pfx
JKS	.jks

For example, if the key store file is named "MyKeyStore.jks", and a type is not specified using this property, then the type is automatically recognized as JKS.

CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORETYPE_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORETYPE_DEFAULT`

CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORETYPE_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORETYPE_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTOREPASSWORD

`static final String CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTOREPASSWORD`

Specify the password of a key store file specified by `CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORE`.

This property has no effect if `CONNECTION_PROPERTY_WALLET_LOCATION` has also been set.

If this property is not set, the JDBC driver will attempt to access the key store without using a password.

This property is only supported by the jdbc:oracle:thin driver.

See Also:
`CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORE`, Constant Field Values

CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTOREPASSWORD_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTOREPASSWORD_DEFAULT`

CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTOREPASSWORD_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTOREPASSWORD_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORE

`static final String CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORE`

Specify the file system path of a trust store file which contains certificate authorities that can be trusted when authenticating a database's certificate.

If no value is set for this property, the driver will use the `cacerts` file included with your JDK installation.

If a database's certificate can not be authenticated, then connection establishment will result in a TLS handshake failure. The failure message may describe a problem with building or finding a verification *path*.

This property has no effect if `CONNECTION_PROPERTY_WALLET_LOCATION` has also been set.

This property is only supported by the jdbc:oracle:thin driver.

See Also:
`CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORETYPE`, `CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTOREPASSWORD`, Constant Field Values

CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORE_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORE_DEFAULT`

CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORE_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORE_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORETYPE

`static final String CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORETYPE`

If this property is not set, the JDBC driver will attempt to automatically recognize the trust store type based on the file extension of the trust store.

Automatic Type Recognition

Recognized Type	File Extension(s)
SSO	.sso
PKCS12	.p12 or .pfx
JKS	.jks

For example, if the trust store file is named "MyTrustStore.jks", and a type is not specified using this property, then the type is automatically recognized as JKS.

This property is only supported by the jdbc:oracle:thin driver.

See Also:
`CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORE`, Constant Field Values

CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORETYPE_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORETYPE_DEFAULT
```

CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORETYPE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORETYPE_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTOREPASSWORD

```
static final String CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTOREPASSWORD
```

Specify the password of a trust store file specified by `CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORE`.

This property has no effect if `CONNECTION_PROPERTY_WALLET_LOCATION` has also been set.

If this property is not set, the JDBC driver will attempt to access the trust store without using a password.

This property is only supported by the jdbc:oracle:thin driver.

See Also:
`CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORE`, Constant Field Values

CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTOREPASSWORD_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTOREPASSWORD_DEFAULT
```

CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTOREPASSWORD_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTOREPASSWORD_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_SSL_KEYMANAGERFACTORY_ALGORITHM

```
static final String CONNECTION_PROPERTY_THIN_SSL_KEYMANAGERFACTORY_ALGORITHM
```

Specify the algorithm to use when managing the key meterial of the key store file specified by `CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORE`.

If this property is not set, the JDBC driver will use your JDK's default algorithm, as returned by `KeyManagerFactory.getDefaultAlgorithm()`.

This property is only supported by the jdbc:oracle:thin driver.

See Also:
`CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_KEYSTORE`, Constant Field Values

CONNECTION_PROPERTY_THIN_SSL_KEYMANAGERFACTORY_ALGORITHM_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_SSL_KEYMANAGERFACTORY_ALGORITHM_DEFAULT
```

CONNECTION_PROPERTY_THIN_SSL_KEYMANAGERFACTORY_ALGORITHM_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_SSL_KEYMANAGERFACTORY_ALGORITHM_ACCESSMODE
```

CONNECTION_PROPERTY_THIN_SSL_TRUSTMANAGERFACTORY_ALGORITHM

`static final String CONNECTION_PROPERTY_THIN_SSL_TRUSTMANAGERFACTORY_ALGORITHM`

Specify the algorithm to use when managing the meterial of the trust store file specified by `CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORE`.

If this property is not set, the JDBC driver will use your JDK's default algorithm, as returned by `TrustManagerFactory.getDefaultAlgorithm()`.

This property is only supported by the jdbc:oracle:thin driver.

See Also:
`CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORE`, Constant Field Values

CONNECTION_PROPERTY_THIN_SSL_TRUSTMANAGERFACTORY_ALGORITHM_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_SSL_TRUSTMANAGERFACTORY_ALGORITHM_DEFAULT`

CONNECTION_PROPERTY_THIN_SSL_TRUSTMANAGERFACTORY_ALGORITHM_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_SSL_TRUSTMANAGERFACTORY_ALGORITHM_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_NET_OLDSYNTAX

`static final String CONNECTION_PROPERTY_THIN_NET_OLDSYNTAX`

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_NET_OLDSYNTAX_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_NET_OLDSYNTAX_DEFAULT`

CONNECTION_PROPERTY_THIN_NET_OLDSYNTAX_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_NET_OLDSYNTAX_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_JNDI_LDAP_CONNECT_TIMEOUT

`static final String CONNECTION_PROPERTY_THIN_JNDI_LDAP_CONNECT_TIMEOUT`

Specify a timeout, in milliseconds, to apply when establishing a connection to an LDAP server.

See Also:
`CONNECTION_PROPERTY_THIN_JNDI_LDAP_READ_TIMEOUT`, Constant Field Values

CONNECTION_PROPERTY_THIN_JNDI_LDAP_CONNECT_TIMEOUT_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_JNDI_LDAP_CONNECT_TIMEOUT_DEFAULT`

CONNECTION_PROPERTY_THIN_JNDI_LDAP_CONNECT_TIMEOUT_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_JNDI_LDAP_CONNECT_TIMEOUT_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_JNDI_LDAP_READ_TIMEOUT

`static final String CONNECTION_PROPERTY_THIN_JNDI_LDAP_READ_TIMEOUT`

Specify a timeout, in milliseconds, to apply when reading a response from an LDAP server.

CONNECTION_PROPERTY_THIN_JNDI_LDAP_READ_TIMEOUT_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_JNDI_LDAP_READ_TIMEOUT_DEFAULT
```

CONNECTION_PROPERTY_THIN_JNDI_LDAP_READ_TIMEOUT_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_JNDI_LDAP_READ_TIMEOUT_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_WALLET_LOCATION

```
static final String CONNECTION_PROPERTY_WALLET_LOCATION
```

Use this property to specify the wallet location. The driver will use this wallet to:

- Retrieve the username and password which can be stored in a wallet. The driver attempts to retrieve the username and password from the wallet unless they are specified in the JDBC URL or in the properties (in order, it will look in the properties first, then in the URL and then in the wallet). The "mkstore" utility can be used to store the username and password in an existing wallet. For example, if the wallet is in the "client_wallet" directory: `mkstore -wrl ./client_wallet -createCredential \(DESCRIPTION=\(ADDRESS=\(PROTOCOL=tcp\)\(HOST=servername\)\( PORT=5560\)\)\(CONNECT_DATA=\(SERVICE_NAME=service_name\)\)\) scott tiger`
- Create an SSL connection if the TCPS protocol is enabled. The wallet is used for both the truststore and the keystore and overwrites the JSSE properties.

The wallet location can be set in two formats:

- `file:/path/ewallet.sso" or "file:/path/cwallet.p12" or "file:/path/to/directory/`
- `(SOURCE=(METHOD=FILE)(METHOD_DATA=(DIRECTORY=/path/to/directory)))`

Note that if you don't use SSO wallets but PKCS12 wallets, you must provide the wallet password through the "oracle.net.wallet_password" property.

See Also:
Constant Field Values

CONNECTION_PROPERTY_WALLET_LOCATION_DEFAULT

```
static final String CONNECTION_PROPERTY_WALLET_LOCATION_DEFAULT
```

CONNECTION_PROPERTY_WALLET_LOCATION_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_WALLET_LOCATION_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_WALLET_PASSWORD

```
static final String CONNECTION_PROPERTY_WALLET_PASSWORD
```

Use this property to set the wallet password which is only required if you don't enable enable auto-login in the wallet. In this case "ewallet.p12" will be used instead of "cwallet.sso".

See Also:
Constant Field Values

CONNECTION_PROPERTY_WALLET_PASSWORD_DEFAULT

```
static final String CONNECTION_PROPERTY_WALLET_PASSWORD_DEFAULT
```

CONNECTION_PROPERTY_WALLET_PASSWORD_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_WALLET_PASSWORD_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_CIPHER_SUITES

```
static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_CIPHER_SUITES
```

Use this property to specify the set of cipher suites to be used while SSL negotiation with LDAP server. Multiple cipher suite names can be seperated by comma. This is an optional property and by default the JDBC Thin driver uses all the cipher suites supported by the JVM.

CONNECTION_PROPERTY_THIN_LDAP_SSL_CIPHER_SUITES_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_CIPHER_SUITES_DEFAULT
```

CONNECTION_PROPERTY_THIN_LDAP_SSL_CIPHER_SUITES_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_LDAP_SSL_CIPHER_SUITES_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_VERSIONS

```
static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_VERSIONS
```

Use this property to specify the valid SSL protocol version(s) used while SSL negotiation with LDAP Server. Multiple protocol versions can be seperated by comma. The default value is `null`. The valid protocol versions are
`TLSv1.2`
`TLSv1.1`
`TLSv1`
`SSLv3`
`SSLv2Hello`

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_VERSIONS_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_VERSIONS_DEFAULT
```

CONNECTION_PROPERTY_THIN_LDAP_SSL_VERSIONS_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_LDAP_SSL_VERSIONS_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE

```
static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE
```

This property is used to specify the path to the `KeyStore` file which will be used while SSL negotiation with LDAP server. The default value is `null`. When no value is specified, the default keystore of the JVM is used.

See Also:
`CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_PASSWORD`, `CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_TYPE`, Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_DEFAULT
```

CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_TYPE

```
static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_TYPE
```

Use this property to specify type of the configured `KeyStore` file to be used while SSL negotiation with LDAP Server. This is an optional property and the JDBC Thin driver will try to resolve the key store type automatically using the file extension. JVM's default `KeyStoreType` is used if the type is not configured and the driver is not able to resolve the type automatically.

See Also:
`CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE`, Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_TYPE_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_TYPE_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_PASSWORD

`static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_PASSWORD`

This property is used to specify the password for the KeyStore file which will be used while SSL negotiation with LDAP server. The default value is `null`.

See Also:
`CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE`, Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_PASSWORD_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_PASSWORD_DEFAULT`

CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_PASSWORD_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE_PASSWORD_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYMANAGER_FACTORY_ALGORITHM

`static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYMANAGER_FACTORY_ALGORITHM`

Use this property to override the default algorithm used by KeyManagerFactory. The default value is `null`.

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYMANAGER_FACTORY_ALGORITHM_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYMANAGER_FACTORY_ALGORITHM_DEFAULT`

CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYMANAGER_FACTORY_ALGORITHM_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYMANAGER_FACTORY_ALGORITHM_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE

`static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE`

This property is used to specify the path to the `TrustStore` file which will be used while SSL negotiation with LDAP server. The default value is `null`. When no value is specified, the default truststore of the JVM is used.

See Also:
`CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_TYPE`, `CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_PASSWORD`, `CONNECTION_PROPERTY_THIN_LDAP_SSL_KEYSTORE`, Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_DEFAULT`

CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_ACCESSMODE`

See Also:

CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_TYPE

```
static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_TYPE
```

Use this property to specify type of the configured `TrustStore` file to be used while SSL negotiation with LDAP Server. This is an optional property and the JDBC Thin driver will try to resolve the trust store type automatically using the file extension. JVM's default `KeyStoreType` is used if the type is not configured and the driver is not able to resolve the type automatically.

See Also:
`CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE`, Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_TYPE_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_TYPE_DEFAULT
```

CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_TYPE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_TYPE_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_PASSWORD

```
static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_PASSWORD
```

This property is used to specify the password for the TrustStore file which will be used while SSL negotiation with LDAP server. The default value is `null`.

See Also:
`CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE`, Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_PASSWORD_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_PASSWORD_DEFAULT
```

CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_PASSWORD_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTSTORE_PASSWORD_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTMANAGER_FACTORY_ALGORITHM

```
static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTMANAGER_FACTORY_ALGORITHM
```

Use this property to override the default algorithm used by TrustManagerFactory. The default value is `null`.

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTMANAGER_FACTORY_ALGORITHM_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTMANAGER_FACTORY_ALGORITHM_DEFAULT
```

CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTMANAGER_FACTORY_ALGORITHM_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_LDAP_SSL_TRUSTMANAGER_FACTORY_ALGORITHM_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_WALLET_LOCATION

```
static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_WALLET_LOCATION
```

Use this property to specify the wallet location. The driver will use this wallet while SSL negotiation with LDAP server. The default value is `null`. Username(DN) and password to be used for authenticating with the LDAP server can be added to the wallet secret store using the key names `oracle.ldap.client.dn` and `oracle.ldap.client.password` respectively. The above authentication credentials can also be configured through connection properties.

CONNECTION_PROPERTY_THIN_LDAP_SECURITY_CREDENTIALS, CONNECTION_PROPERTY_THIN_LDAP_SECURITY_AUTHENTICATION, Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_WALLET_LOCATION_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_WALLET_LOCATION_DEFAULT
```

CONNECTION_PROPERTY_THIN_LDAP_SSL_WALLET_LOCATION_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_LDAP_SSL_WALLET_LOCATION_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_WALLET_PASSWORD

```
static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_WALLET_PASSWORD
```

Use this property to specify the password of the wallet file which will be used while SSL negotiation with LDAP server. The default value is `null` and no password is required for accessing the wallet.

Since:

21c

See Also:

CONNECTION_PROPERTY_THIN_LDAP_SSL_WALLET_LOCATION, Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_WALLET_PASSWORD_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_WALLET_PASSWORD_DEFAULT
```

CONNECTION_PROPERTY_THIN_LDAP_SSL_WALLET_PASSWORD_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_LDAP_SSL_WALLET_PASSWORD_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SECURITY_AUTHENTICATION

```
static final String CONNECTION_PROPERTY_THIN_LDAP_SECURITY_AUTHENTICATION
```

Specifies the authentication mechanism to be used by the LDAP service provider in the JDK.
This can be one of the following strings: `none` or `simple`. The default value is `none` and LDAP authentication is disabled. If it is configured with the value `simple` then the LDAP Server authentication details must to be set through either wallet secret store or connection properties.

Since:

21c

See Also:

CONNECTION_PROPERTY_THIN_LDAP_SECURITY_PRINCIPAL, CONNECTION_PROPERTY_THIN_LDAP_SECURITY_CREDENTIALS, Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SECURITY_AUTHENTICATION_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_LDAP_SECURITY_AUTHENTICATION_DEFAULT
```

CONNECTION_PROPERTY_THIN_LDAP_SECURITY_AUTHENTICATION_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_LDAP_SECURITY_AUTHENTICATION_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SECURITY_PRINCIPAL

```
static final String CONNECTION_PROPERTY_THIN_LDAP_SECURITY_PRINCIPAL
```

Use this property to specify the value of the username(DN) which will be used while authenticating with the LDAP server.
The default value is `null`. This property can also be configured via wallet secret store entry `oracle.ldap.client.dn`. The value configured in connection property has higher priority over the value configured in the wallet secret store. This property is used only when the LDAP authentication is set to `simple`.

CONNECTION_PROPERTY_THIN_LDAP_SECURITY_PRINCIPAL_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_LDAP_SECURITY_PRINCIPAL_DEFAULT`

CONNECTION_PROPERTY_THIN_LDAP_SECURITY_PRINCIPAL_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_LDAP_SECURITY_PRINCIPAL_ACCESSMODE`

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SECURITY_CREDENTIALS

`static final String CONNECTION_PROPERTY_THIN_LDAP_SECURITY_CREDENTIALS`

Use this property to configure the password which will be used while authenticating with the LDAP server.
The default value is `null`. This property can also be configured via wallet secret store entry `oracle.ldap.client.password`. The value configured in connection property has higher priority over the value configured in the wallet secret store. This property is used only when the LDAP authentication is set to `simple`.

Since:

21c

See Also:

`CONNECTION_PROPERTY_THIN_LDAP_SECURITY_AUTHENTICATION`, `CONNECTION_PROPERTY_THIN_LDAP_SECURITY_PRINCIPAL`, Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SECURITY_CREDENTIALS_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_LDAP_SECURITY_CREDENTIALS_DEFAULT`

CONNECTION_PROPERTY_THIN_LDAP_SECURITY_CREDENTIALS_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_LDAP_SECURITY_CREDENTIALS_ACCESSMODE`

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_CONTEXT_PROTOCOL

`static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_CONTEXT_PROTOCOL`

Specifies a `protocol` name for the driver to use when obtaining an instance of `SSLContext` from `SSLContext.getInstance(String)` for a TLS enabled LDAP connection.

This property has no effect on which versions of SSL or TLS will be accepted during handshakes with the LDAP server. To configure the set of protocol versions accepted during handshakes, use `CONNECTION_PROPERTY_THIN_LDAP_SSL_VERSIONS`.

If this property is not specified, the driver will use "TLS" by default.

This property is only supported by the Type 4 driver (ie: jdbc:oracle:thin).

Since:

21c

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_LDAP_SSL_CONTEXT_PROTOCOL_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_LDAP_SSL_CONTEXT_PROTOCOL_DEFAULT`

CONNECTION_PROPERTY_THIN_LDAP_SSL_CONTEXT_PROTOCOL_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_LDAP_SSL_CONTEXT_PROTOCOL_ACCESSMODE`

See Also:

Constant Field Values

CONNECTION_PROPERTY_PROXY_CLIENT_NAME

Note that this connection property can be used to obtain a proxy client connection from scratch and in this model there is only one database session involved instead of two when you first create a regular connection as "proxy" and then call `openProxySession(...)` to obtain a proxy client session. This is called the single-session proxy model. This property is only supported for connections to database versions of 10.2 and higher. There is no support for the single-session proxy model in earlier database versions.

See Also:

Constant Field Values

CONNECTION_PROPERTY_PROXY_CLIENT_NAME_DEFAULT

```
static final String CONNECTION_PROPERTY_PROXY_CLIENT_NAME_DEFAULT
```

CONNECTION_PROPERTY_PROXY_CLIENT_NAME_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_PROXY_CLIENT_NAME_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_DEFAULT_USE_NIO

```
static final String CONNECTION_PROPERTY_DEFAULT_USE_NIO
```

In case of Jdbc-OCI drivers the data is copied from C layer to Java using Jni array copy api. Alternatively by setting this property to "true" the user can instruct the driver to copy data using NIO. Note that the feature would enabled if the underlying JVM supports NIO in JNI layer. The flag is turned off by default (set to false).

See Also:

Constant Field Values

CONNECTION_PROPERTY_DEFAULT_USE_NIO_DEFAULT

```
static final String CONNECTION_PROPERTY_DEFAULT_USE_NIO_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_DEFAULT_USE_NIO_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_DEFAULT_USE_NIO_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_HTTPS_PROXY_HOST

```
static final String CONNECTION_PROPERTY_THIN_HTTPS_PROXY_HOST
```

Use this property to set the hostname or address of the https proxy server. This can also be set via the URL. The value set through the URL has higher priority than the value set using this property. Here is an example of how to set the proxy host name through the URL:

```
jdbc:oracle:thin:@(DESCRIPTION=
(ADDRESS_LIST= (ADDRESS=(HTTPS_PROXY=myproxyserver)(HTTPS_PROXY_PORT=8080)(HOST=myhost)(PORT=5521)(PROTOCOL=tcp))) (CONNECT_DATA=
(SERVICE_NAME=myService)))
```

Since:

19.3

See Also:

`CONNECTION_PROPERTY_THIN_HTTPS_PROXY_PORT`, Constant Field Values

CONNECTION_PROPERTY_THIN_HTTPS_PROXY_HOST_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_HTTPS_PROXY_HOST_DEFAULT
```

CONNECTION_PROPERTY_THIN_HTTPS_PROXY_HOST_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_THIN_HTTPS_PROXY_HOST_ACCESSMODE
```

See Also:

Constant Field Values

Use this property to set the port of the https proxy server. This can also be set via the URL. The value set through the URL has higher priority than the value set using this property. Here is an example of how to set the proxy port through the URL:
`jdbc:oracle:thin:@(DESCRIPTION=`
`(ADDRESS_LIST= (ADDRESS=(HTTPS_PROXY=myproxyserver)(HTTPS_PROXY_PORT=8080)(HOST=myhost)(PORT=5521)(PROTOCOL=tcp))) (CONNECT_DATA=`
`(SERVICE_NAME=myService)))`

Since:

19.3

See Also:

`CONNECTION_PROPERTY_THIN_HTTPS_PROXY_HOST`, Constant Field Values

CONNECTION_PROPERTY_THIN_HTTPS_PROXY_PORT_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_HTTPS_PROXY_PORT_DEFAULT`

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_HTTPS_PROXY_PORT_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_HTTPS_PROXY_PORT_ACCESSMODE`

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_NET_CONNECTIONID_PREFIX

`static final String CONNECTION_PROPERTY_THIN_NET_CONNECTIONID_PREFIX`

This property `oracle.net.connectionIdPrefix` can be used to customize the first 8 characters of the Net connection id that's sent to the listener during connection establishment for tracing purposes. Its value is a 8 character long string (can contain only `alphabets`, `numbers` and `_`). For example the value can be set to `App_231` (which contains only supported characters) to identify connections coming from a particular application.
This can also be configured via the connection URL using the `CONNECT_DATA` parameter `CONNECTION_ID_PREFIX`.
The value set using the connection property has higher precedence over the value set using URL.

Since:

21c

See Also:

Constant Field Values

CONNECTION_PROPERTY_THIN_NET_CONNECTIONID_PREFIX_DEFAULT

`static final String CONNECTION_PROPERTY_THIN_NET_CONNECTIONID_PREFIX_DEFAULT`

CONNECTION_PROPERTY_THIN_NET_CONNECTIONID_PREFIX_ACCESSMODE

`static final byte CONNECTION_PROPERTY_THIN_NET_CONNECTIONID_PREFIX_ACCESSMODE`

See Also:

Constant Field Values

CONNECTION_PROPERTY_OCI_DRIVER_CHARSET

`static final String CONNECTION_PROPERTY_OCI_DRIVER_CHARSET`

See Also:

Constant Field Values

CONNECTION_PROPERTY_OCI_DRIVER_CHARSET_DEFAULT

`static final String CONNECTION_PROPERTY_OCI_DRIVER_CHARSET_DEFAULT`

CONNECTION_PROPERTY_OCI_DRIVER_CHARSET_ACCESSMODE

`static final byte CONNECTION_PROPERTY_OCI_DRIVER_CHARSET_ACCESSMODE`

See Also:

Constant Field Values

The following SQL query:

```
SELECT sys_context('USERENV', 'CURRENT_EDITION_NAME')
    FROM dual
```

will return the same value.

Note that this property can also be set as a system property.

By default, if you don't set this property, the "session edition" will be set to the database default edition (for example "ORA$BASE").

See Also:
Constant Field Values

CONNECTION_PROPERTY_EDITION_NAME_DEFAULT

`static final String CONNECTION_PROPERTY_EDITION_NAME_DEFAULT`

CONNECTION_PROPERTY_EDITION_NAME_ACCESSMODE

`static final byte CONNECTION_PROPERTY_EDITION_NAME_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_INTERNAL_LOGON

`static final String CONNECTION_PROPERTY_INTERNAL_LOGON`

The value of this property is used as the user name when performing an internal logon. Usually this will be "SYS" or "SYSDBA".

As of 12.1 server and driver, "SYSBACKUP", "SYSDG" and "SYSKM" are also supported.

See Also:
Constant Field Values

CONNECTION_PROPERTY_INTERNAL_LOGON_DEFAULT

`static final String CONNECTION_PROPERTY_INTERNAL_LOGON_DEFAULT`

CONNECTION_PROPERTY_INTERNAL_LOGON_ACCESSMODE

`static final byte CONNECTION_PROPERTY_INTERNAL_LOGON_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_CREATE_DESCRIPTOR_USE_CURRENT_SCHEMA_FOR_SCHEMA_NAME

`static final String CONNECTION_PROPERTY_CREATE_DESCRIPTOR_USE_CURRENT_SCHEMA_FOR_SCHEMA_NAME`

The user has to provide fully qualified ADT name ([username].[adt name]) for all ADT operations. However, if the user does not provide fully qualified name, the user name provided during login is appended to the ADT name to obtain fully qualified name. This is also the behavior when this flag is set to `false`.

The user also has an option to append the CURRENT_USER value to the ADT name to obtain fully qualified name by setting this property to `true`. Note that it takes a network round trip to fetch the CURRENT_SCHEMA value.

The default value of this flag is `false` which means that the driver appends the user name used to login as the user name to append to the ADT name.

See Also:
Constant Field Values

CONNECTION_PROPERTY_CREATE_DESCRIPTOR_USE_CURRENT_SCHEMA_FOR_SCHEMA_NAME_DEFAULT

`static final String CONNECTION_PROPERTY_CREATE_DESCRIPTOR_USE_CURRENT_SCHEMA_FOR_SCHEMA_NAME_DEFAULT`

See Also:
Constant Field Values

CONNECTION_PROPERTY_CREATE_DESCRIPTOR_USE_CURRENT_SCHEMA_FOR_SCHEMA_NAME_ACCESSMODE

CONNECTION_PROPERTY_OCI_SVC_CTX_HANDLE

```
static final String CONNECTION_PROPERTY_OCI_SVC_CTX_HANDLE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_OCI_SVC_CTX_HANDLE_DEFAULT

```
static final String CONNECTION_PROPERTY_OCI_SVC_CTX_HANDLE_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_OCI_SVC_CTX_HANDLE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_OCI_SVC_CTX_HANDLE_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_OCI_ENV_HANDLE

```
static final String CONNECTION_PROPERTY_OCI_ENV_HANDLE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_OCI_ENV_HANDLE_DEFAULT

```
static final String CONNECTION_PROPERTY_OCI_ENV_HANDLE_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_OCI_ENV_HANDLE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_OCI_ENV_HANDLE_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_OCI_ERR_HANDLE

```
static final String CONNECTION_PROPERTY_OCI_ERR_HANDLE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_OCI_ERR_HANDLE_DEFAULT

```
static final String CONNECTION_PROPERTY_OCI_ERR_HANDLE_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_OCI_ERR_HANDLE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_OCI_ERR_HANDLE_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_PRELIM_AUTH

```
static final String CONNECTION_PROPERTY_PRELIM_AUTH
```

If this property is set to "true", JDBC drivers connect in PRELIM_AUTH mode, which is the only mode that is permitted when the database is down

CONNECTION_PROPERTY_PRELIM_AUTH_DEFAULT

```
static final String CONNECTION_PROPERTY_PRELIM_AUTH_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_PRELIM_AUTH_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_PRELIM_AUTH_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_SET_NEW_PASSWORD

```
static final String CONNECTION_PROPERTY_SET_NEW_PASSWORD
```

Deprecated.

Property which sets enables the user to set a new password during connection. This property is deprecated since 12.2 and should not be used. Use oracle.jdbc.newPassword.

See Also:

Constant Field Values

CONNECTION_PROPERTY_SET_NEW_PASSWORD_DEFAULT

```
static final String CONNECTION_PROPERTY_SET_NEW_PASSWORD_DEFAULT
```

CONNECTION_PROPERTY_SET_NEW_PASSWORD_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_SET_NEW_PASSWORD_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_DEFAULT_EXECUTE_BATCH

```
static final String CONNECTION_PROPERTY_DEFAULT_EXECUTE_BATCH
```

Deprecated.
Oracle-Style batching is desupported. Standard JDBC batch execution is recommended instead.

The value of this property is ignored since the 20c release. Prior to 20c, the value of this property would configure the default batch size when using Oracle-Style batching.

See Also:

Constant Field Values

CONNECTION_PROPERTY_DEFAULT_EXECUTE_BATCH_DEFAULT

```
static final String CONNECTION_PROPERTY_DEFAULT_EXECUTE_BATCH_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_DEFAULT_EXECUTE_BATCH_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_DEFAULT_EXECUTE_BATCH_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_DEFAULT_ROW_PREFETCH

```
static final String CONNECTION_PROPERTY_DEFAULT_ROW_PREFETCH
```

The value of this property is used as the default number of rows to prefetch.

CONNECTION_PROPERTY_DEFAULT_ROW_PREFETCH_DEFAULT

```
static final String CONNECTION_PROPERTY_DEFAULT_ROW_PREFETCH_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_DEFAULT_ROW_PREFETCH_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_DEFAULT_ROW_PREFETCH_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_DEFAULT_LOB_PREFETCH_SIZE

```
static final String CONNECTION_PROPERTY_DEFAULT_LOB_PREFETCH_SIZE
```

The value of this property is used as the default LOB prefetch size for this connection.

The LOB prefetch size can be overriden at the statement level through the `setLobPrefetchSize(int)` which is defined in `oracle.jdbc.OracleStatement`. The statement level LOB prefetch size can be overriden at the column level through the `defineColumnType` method.

The value can be "-1" to disable LOB prefetch for this connection, "0" to enable LOB prefetch for meta-data only or any value greater than 0 which represents a number of bytes for BLOBs and chars for CLOBs to be prefetched along with the locator during fetch operations. The default value for this property is "32768".

Since:

11.2

See Also:

`OracleStatement.setLobPrefetchSize`, Constant Field Values

CONNECTION_PROPERTY_DEFAULT_LOB_PREFETCH_SIZE_DEFAULT

```
static final String CONNECTION_PROPERTY_DEFAULT_LOB_PREFETCH_SIZE_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_DEFAULT_LOB_PREFETCH_SIZE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_DEFAULT_LOB_PREFETCH_SIZE_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_ENABLE_DATA_IN_LOCATOR

```
static final String CONNECTION_PROPERTY_ENABLE_DATA_IN_LOCATOR
```

The value of this property is used to control the use of the Data in Locator feature of the server.

Data in Locator is a server side feature introduced in 10.2. For small lobs the actual data in included in the locator bytes shipped to the client. These may be shipped back and forth several times as the client accesses the lob using the lob APIs. For fast networks this actually increases performance because it greatly reduces server CPU consumption.

For slower networks there is a net slow down. Setting this property to false will disable this feature.

This property is currently only effective for the thin driver.

Data in Locator is automatically disabled when Lob Prefetch is enabled. Thus this property will most likely be used for 10.2 servers where lob prefetch is not available.

Since:

11.2

See Also:

Constant Field Values

CONNECTION_PROPERTY_ENABLE_DATA_IN_LOCATOR_DEFAULT

```
static final String CONNECTION_PROPERTY_ENABLE_DATA_IN_LOCATOR_DEFAULT
```

See Also:

Constant Field Values

Constant Field Values

CONNECTION_PROPERTY_ENABLE_READ_DATA_IN_LOCATOR

```
static final String CONNECTION_PROPERTY_ENABLE_READ_DATA_IN_LOCATOR
```

The value of this property is used to control the use of the Data in Locator feature by the client.

Data in Locator is a server side feature introduced in 10.2. The JDBC driver is enhanced to use this data directly. This saves a number of round trips which previously occurred when lob APIs were used to read the data.

This feature is only enabled for 10.2 servers. For earlier servers the Data in Locator feature did not exist and for later ones the lob prefetch functionality make this uunnecessary and the new lob storage types complicate the locator structure.

Since:

11.2

See Also:

Constant Field Values

CONNECTION_PROPERTY_ENABLE_READ_DATA_IN_LOCATOR_DEFAULT

```
static final String CONNECTION_PROPERTY_ENABLE_READ_DATA_IN_LOCATOR_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_ENABLE_READ_DATA_IN_LOCATOR_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_ENABLE_READ_DATA_IN_LOCATOR_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_REPORT_REMARKS

```
static final String CONNECTION_PROPERTY_REPORT_REMARKS
```

If the value of this property is "true", OracleDatabaseMetaData will include remarks in the meta data. This can result in a substantial reduction in performance.

See Also:

Constant Field Values

CONNECTION_PROPERTY_REPORT_REMARKS_DEFAULT

```
static final String CONNECTION_PROPERTY_REPORT_REMARKS_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_REPORT_REMARKS_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_REPORT_REMARKS_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_INCLUDE_SYNONYMS

```
static final String CONNECTION_PROPERTY_INCLUDE_SYNONYMS
```

If the value of this property is "true", JDBC will include synonyms when getting information about a column.

See Also:

Constant Field Values

CONNECTION_PROPERTY_INCLUDE_SYNONYMS_DEFAULT

```
static final String CONNECTION_PROPERTY_INCLUDE_SYNONYMS_DEFAULT
```

See Also:

Constant Field Values

```
static final byte CONNECTION_PROPERTY_INCLUDE_SYNONYMS_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_RESTRICT_GETTABLES

```
static final String CONNECTION_PROPERTY_RESTRICT_GETTABLES
```

If the value of this property is "true", JDBC will return a more refined value for DatabaseMeta.getTables. By default JDBC will return things that are not accessible tables. These can be non-table objects or accessible synonymns for inaccessible tables. If this property is true JDBC will return only accessible tables. This has a substantial performance penalty.

See Also:
Constant Field Values

CONNECTION_PROPERTY_RESTRICT_GETTABLES_DEFAULT

```
static final String CONNECTION_PROPERTY_RESTRICT_GETTABLES_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_RESTRICT_GETTABLES_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_RESTRICT_GETTABLES_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_ACCUMULATE_BATCH_RESULT

```
static final String CONNECTION_PROPERTY_ACCUMULATE_BATCH_RESULT
```

When using Oracle style batching, JDBC determines when to flush a batch to the database. If this property is true, then the number of modified rows accumulated across all batches flushed from a single statement. The default is to count each batch separately.

See Also:
Constant Field Values

CONNECTION_PROPERTY_ACCUMULATE_BATCH_RESULT_DEFAULT

```
static final String CONNECTION_PROPERTY_ACCUMULATE_BATCH_RESULT_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_ACCUMULATE_BATCH_RESULT_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_ACCUMULATE_BATCH_RESULT_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_USE_FETCH_SIZE_WITH_LONG_COLUMN

```
static final String CONNECTION_PROPERTY_USE_FETCH_SIZE_WITH_LONG_COLUMN
```

If the value of this property is "true", then JDBC will prefetch rows even though there is a LONG or LONG RAW column in the result. By default JDBC fetches only one row at a time if there are LONG or LONG RAW columns in the result. Setting this property to "true" can improve performance but can also cause SQLExceptions if the results are too big.

See Also:
Constant Field Values

CONNECTION_PROPERTY_USE_FETCH_SIZE_WITH_LONG_COLUMN_DEFAULT

```
static final String CONNECTION_PROPERTY_USE_FETCH_SIZE_WITH_LONG_COLUMN_DEFAULT
```

See Also:
Constant Field Values

See Also:
Constant Field Values

CONNECTION_PROPERTY_PROCESS_ESCAPES

```
static final String CONNECTION_PROPERTY_PROCESS_ESCAPES
```

If the value of this property is "false" then the default setting for Statement.setEscapeProccessing is false.

See Also:
Constant Field Values

CONNECTION_PROPERTY_PROCESS_ESCAPES_DEFAULT

```
static final String CONNECTION_PROPERTY_PROCESS_ESCAPES_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_PROCESS_ESCAPES_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_PROCESS_ESCAPES_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_FIXED_STRING

```
static final String CONNECTION_PROPERTY_FIXED_STRING
```

If the value of this property is "true", JDBC will use FIXED CHAR semantic when setObject is called with a String argument. By default JDBC uses VARCHAR semantics. The difference is in blank padding. With the default there is no blank padding so, for example, 'a' does not equal 'a ' in a CHAR(4). If true these two will be equal.

See Also:
Constant Field Values

CONNECTION_PROPERTY_FIXED_STRING_DEFAULT

```
static final String CONNECTION_PROPERTY_FIXED_STRING_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_FIXED_STRING_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_FIXED_STRING_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_DEFAULTNCHAR

```
static final String CONNECTION_PROPERTY_DEFAULTNCHAR
```

If the value of this property is "true", the default mode for all character data columns will be NCHAR.

See Also:
Constant Field Values

CONNECTION_PROPERTY_DEFAULTNCHAR_DEFAULT

```
static final String CONNECTION_PROPERTY_DEFAULTNCHAR_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_DEFAULTNCHAR_ACCESSMODE

CONNECTION_PROPERTY_RESOURCE_MANAGER_ID

```
static final String CONNECTION_PROPERTY_RESOURCE_MANAGER_ID
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_RESOURCE_MANAGER_ID_DEFAULT

```
static final String CONNECTION_PROPERTY_RESOURCE_MANAGER_ID_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_RESOURCE_MANAGER_ID_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_RESOURCE_MANAGER_ID_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_DISABLE_DEFINECOLUMNTYPE

```
static final String CONNECTION_PROPERTY_DISABLE_DEFINECOLUMNTYPE
```

Disable the method OracleStatement.defineColumnType when equal "true". When this connection property has the value true, the method defineColumnType is has no effect. This is highly recommended when using the Thin driver, especially when the database character set contains four byte characters that expand to two UCS2 surrogate characters, e.g. AL32UTF8. The method defineColumnType provides no performance benefit (or any other benefit) when used with the 10g Thin driver. This property is provided so that you do not have to remove the calls from your code. This is especially valuable if you use the same code with Thin driver and either the OCI or Server Internal driver.

See Also:
Constant Field Values

CONNECTION_PROPERTY_DISABLE_DEFINECOLUMNTYPE_DEFAULT

```
static final String CONNECTION_PROPERTY_DISABLE_DEFINECOLUMNTYPE_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_DISABLE_DEFINECOLUMNTYPE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_DISABLE_DEFINECOLUMNTYPE_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_CONVERT_NCHAR_LITERALS

```
static final String CONNECTION_PROPERTY_CONVERT_NCHAR_LITERALS
```

Convert NCHAR literals to Unicode literals when equal "true". The default is true.

See Also:
Constant Field Values

CONNECTION_PROPERTY_CONVERT_NCHAR_LITERALS_DEFAULT

```
static final String CONNECTION_PROPERTY_CONVERT_NCHAR_LITERALS_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_CONVERT_NCHAR_LITERALS_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_CONVERT_NCHAR_LITERALS_ACCESSMODE
```

See Also:

CONNECTION_PROPERTY_AUTO_COMMIT_SPEC_COMPLIANT

```
static final String CONNECTION_PROPERTY_AUTO_COMMIT_SPEC_COMPLIANT
```

Alters the auto-commit behavior of the driver. By default the driver complies with JDBC specification. User can choose to alter the behavior by changing the value of this flag.

Oracle JDBC 12.1 drivers comply with JDBC specification 4.1 and will:

- throw `SQLException` when `Connection.commit()` or `Connection.rollback()` is invoked when auto-commit is `true`.
- issue an implicit commit of the local transaction when `Connection.setAutoCommit(boolean)` is called and the mode is changed from `false` to `true`.

Because the standard behavior may break existing applications, this flag is provided as a convenience and can be set to `false`. Most applications may never need to set this flag. Users are encouraged to modify their applications to support the specification instead of using this flag.

Since:

12.1

See Also:

`Connection.commit()`, `Connection.rollback()`, `Connection.setAutoCommit(boolean)`, Constant Field Values

CONNECTION_PROPERTY_AUTO_COMMIT_SPEC_COMPLIANT_DEFAULT

```
static final String CONNECTION_PROPERTY_AUTO_COMMIT_SPEC_COMPLIANT_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_AUTO_COMMIT_SPEC_COMPLIANT_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_AUTO_COMMIT_SPEC_COMPLIANT_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_JDBC_STANDARD_BEHAVIOR

```
static final String CONNECTION_PROPERTY_JDBC_STANDARD_BEHAVIOR
```

Ensures the driver is in strict compliance with the JDBC specification. When "false" (the default), previous Oracle specific non-standard deviations from the standard are maintained. When "true", this flag will override the behavior of all other compatibility flags. For complete backwards compatibility with previous drivers which differed from the JDBC standards, you should leave this flag set to "false". If you require compliance with the JDBC standard, then " set to "true"; this ensures ALL non-standard behavior is removed from the driver. Note that backwards compatibility and standards compliance is often contradictory and you can't have both.

Since:

12.2

See Also:

Constant Field Values

CONNECTION_PROPERTY_JDBC_STANDARD_BEHAVIOR_DEFAULT

```
static final String CONNECTION_PROPERTY_JDBC_STANDARD_BEHAVIOR_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_JDBC_STANDARD_BEHAVIOR_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_JDBC_STANDARD_BEHAVIOR_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_J2EE13_COMPLIANT

```
static final String CONNECTION_PROPERTY_J2EE13_COMPLIANT
```

Deprecated.
This property could be removed in the future and the default will be `true`. So if this property is used as workaround to turn off compliant behavior, consider changing the application.

If the value of this property is "true", JDBC uses strict compliance for some edge cases. In general Oracle's JDBC drivers will allow some operations that are not permitted in the strict interpretation of J2EE 1.3. Setting this property to true will cause those cases to throw SQLExceptions. There are some other edge cases where Oracle's JDBC drivers have slightly different behavior than defined in J2EE 1.3. This results from Oracle having defined the behavior prior to the J2EE 1.3 specification and the resultant need for compatibility with existing customer code. Setting this property will result in full J2EE 1.3 compliance at the cost of

- getObject on a NUMBER column with precision 0 and scale unconstrained returns Double
- ResultSetMetaData.getColumnType on a NUMBER column with precision 0 and scale unconstrained returns FLOAT
- ResultSetMetaData.getColumnClassName on a NUMBER column with precision 0 and scale unconstrained returns java.lang.Double
- getObject on a TIMESTAMP column returns an object of type java.sql.Timestamp
- ResultSetMetaData.getColumnClassName on a TIMESTAMP column returns java.sql.Timestamp

The property is `false` by default for all drivers while using regular JDBC library. The value of this property is `true` by default in DMS jar file.

See Also:
Constant Field Values

CONNECTION_PROPERTY_J2EE13_COMPLIANT_DEFAULT

```
static final String CONNECTION_PROPERTY_J2EE13_COMPLIANT_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_J2EE13_COMPLIANT_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_J2EE13_COMPLIANT_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_DMS_PARENT_NAME

```
static final String CONNECTION_PROPERTY_DMS_PARENT_NAME
```

Override the default DMS parent name. This property should only be set if it is absolutely necessary to do so. For most cases, the default name should be used.

See Also:
Constant Field Values

CONNECTION_PROPERTY_DMS_PARENT_NAME_DEFAULT

```
static final String CONNECTION_PROPERTY_DMS_PARENT_NAME_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_DMS_PARENT_NAME_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_DMS_PARENT_NAME_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_DMS_PARENT_TYPE

```
static final String CONNECTION_PROPERTY_DMS_PARENT_TYPE
```

Override the default DMS parent type. This property should only be set if it is absolutely necessary to do so. For most cases, the default type should be used.

See Also:
Constant Field Values

CONNECTION_PROPERTY_DMS_PARENT_TYPE_DEFAULT

```
static final String CONNECTION_PROPERTY_DMS_PARENT_TYPE_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_DMS_PARENT_TYPE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_DMS_PARENT_TYPE_ACCESSMODE
```

See Also:
Constant Field Values

Property which enables/disables DMS Statement metrics

See Also:

Constant Field Values

CONNECTION_PROPERTY_DMS_STMT_METRICS_DEFAULT

```
static final String CONNECTION_PROPERTY_DMS_STMT_METRICS_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_DMS_STMT_METRICS_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_DMS_STMT_METRICS_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_DMS_STMT_CACHING_METRICS

```
static final String CONNECTION_PROPERTY_DMS_STMT_CACHING_METRICS
```

Deprecated.

This property no longer has any effect.

See Also:

Constant Field Values

CONNECTION_PROPERTY_DMS_STMT_CACHING_METRICS_DEFAULT

```
static final String CONNECTION_PROPERTY_DMS_STMT_CACHING_METRICS_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_DMS_STMT_CACHING_METRICS_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_DMS_STMT_CACHING_METRICS_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_MAP_DATE_TO_TIMESTAMP

```
static final String CONNECTION_PROPERTY_MAP_DATE_TO_TIMESTAMP
```

This connection property lets you define how the driver will map SQL DATE values in the database to Java types. Since Oracle SQL DATE includes a time component and java.sql.Date does not, mapping DATE to java.sql.Date looses information. It is more appropriate to map DATE to java.sql.Timestamp and that is the default behavior.

The 9i and 10g drivers mistakenly mapped DATE to java.sql.Date by default. Setting this property to false will cause the driver to map SQL DATE to java.util.Date with the corresponding loss of time information in each DATE value. This is for backwards compatibility only.

See Also:

Constant Field Values

CONNECTION_PROPERTY_MAP_DATE_TO_TIMESTAMP_DEFAULT

```
static final String CONNECTION_PROPERTY_MAP_DATE_TO_TIMESTAMP_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_MAP_DATE_TO_TIMESTAMP_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_MAP_DATE_TO_TIMESTAMP_ACCESSMODE
```

See Also:

Constant Field Values

If true, the statement data buffers are cached on a per thread basis. If false, the data buffers are cached on a per connection basis. These buffers can be quite large. It is important that the number of them be minimized.

In most cases you should use the per connection cache. However if your app has many more idle connections than active connections at any given moment then using the thread local cache may reduce the total JDBC driver memory footprint. If you are not having a problem with Java heap size, leave this alone.

See Also:
Constant Field Values

CONNECTION_PROPERTY_USE_THREADLOCAL_BUFFER_CACHE_DEFAULT

```
static final String CONNECTION_PROPERTY_USE_THREADLOCAL_BUFFER_CACHE_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_USE_THREADLOCAL_BUFFER_CACHE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_USE_THREADLOCAL_BUFFER_CACHE_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_DRIVER_NAME_ATTRIBUTE

```
static final String CONNECTION_PROPERTY_DRIVER_NAME_ATTRIBUTE
```

The value passed to the server for the OCI_ATTR_DRIVER_NAME. This property is supported with both the OCI and Thin drivers. The attribute aids in diagnosability. In most cases you should not need to set it. The value is limited to a maximum of 8 printable 7-bit ASCII characters. The default value depends on which driver is used. The default value for the THIN driver is "jdbcthin" and the default value for the OCI driver is "jdbcoci".

See Also:
Constant Field Values

CONNECTION_PROPERTY_DRIVER_NAME_ATTRIBUTE_DEFAULT

```
static final String CONNECTION_PROPERTY_DRIVER_NAME_ATTRIBUTE_DEFAULT
```

CONNECTION_PROPERTY_DRIVER_NAME_ATTRIBUTE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_DRIVER_NAME_ATTRIBUTE_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_MAX_CACHED_BUFFER_SIZE

```
static final String CONNECTION_PROPERTY_MAX_CACHED_BUFFER_SIZE
```

The log base 2 of the size of the largest internal char or byte data buffer that the driver should cache. The default is 30 which means the largest cached buffer is 1 giga(byte/char). Values greater than 30 are treated as the actual buffer size. Values less than 12 will effectively disable the buffer cache. Rather than setting the value less than 16, you should increase the amount of heap available to your application by setting -Xmx and -Xms.

The driver uses char and byte buffers to retrieve query results. These buffers can be quite large and are cached. If some queries create particularly large buffers, the driver will attempt to cache those large buffers. This may possibly reduce performance. The The best way to approach this problem is to reduce buffer size of those queries by setting their fetch_size to a smaller value, but if that is impractical, you can set this property to prevent these large buffers from being cached. The appropriate value depends on the heap size, number of connections open, number of statements open at once, and fraction of the heap that can be allocated to JDBC. A reasonable starting point for a middle-tier application server might be 21 (2MB). It bears repating that you are better off setting the fetch size of the problematic statements, if possible.

See Also:
`USE_THREAD_LOCAL_BUFFER_CACHE`, Constant Field Values

CONNECTION_PROPERTY_MAX_CACHED_BUFFER_SIZE_DEFAULT

```
static final String CONNECTION_PROPERTY_MAX_CACHED_BUFFER_SIZE_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_MAX_CACHED_BUFFER_SIZE_ACCESSMODE

CONNECTION_PROPERTY_IMPLICIT_STATEMENT_CACHE_SIZE

`static final String CONNECTION_PROPERTY_IMPLICIT_STATEMENT_CACHE_SIZE`

The maximum number of statements that will be stored in this connection's statement cache. The default is 0 which disables the statement cache. If set to a value greater than 0, the implicit statement cache is enabled. Calls to setStatementCacheSize and setImplicitCachingEnabled override this.

See Also:

Constant Field Values

CONNECTION_PROPERTY_IMPLICIT_STATEMENT_CACHE_SIZE_DEFAULT

`static final String CONNECTION_PROPERTY_IMPLICIT_STATEMENT_CACHE_SIZE_DEFAULT`

See Also:

Constant Field Values

CONNECTION_PROPERTY_IMPLICIT_STATEMENT_CACHE_SIZE_ACCESSMODE

`static final byte CONNECTION_PROPERTY_IMPLICIT_STATEMENT_CACHE_SIZE_ACCESSMODE`

See Also:

Constant Field Values

CONNECTION_PROPERTY_LOB_STREAM_POS_STANDARD_COMPLIANT

`static final String CONNECTION_PROPERTY_LOB_STREAM_POS_STANDARD_COMPLIANT`

Previous releases allowed the value of 0L to be set for the position parameter of Blob.setBinaryStream and Clob.setAsciiStream and setCharacterStream which is not correct in the specification. It had the same effect as setting 1L. This was a legacy of the orginal Oracle proprietary APiS. If this switch is set false the old incorrect behavior is retained for compatibility

See Also:

Constant Field Values

CONNECTION_PROPERTY_LOB_STREAM_POS_STANDARD_COMPLIANT_DEFAULT

`static final String CONNECTION_PROPERTY_LOB_STREAM_POS_STANDARD_COMPLIANT_DEFAULT`

See Also:

Constant Field Values

CONNECTION_PROPERTY_LOB_STREAM_POS_STANDARD_COMPLIANT_ACCESSMODE

`static final byte CONNECTION_PROPERTY_LOB_STREAM_POS_STANDARD_COMPLIANT_ACCESSMODE`

See Also:

Constant Field Values

CONNECTION_PROPERTY_STRICT_ASCII_CONVERSION

`static final String CONNECTION_PROPERTY_STRICT_ASCII_CONVERSION`

The Oracle JDBC has been doing quick ASCII conversion (use only the low bytes) in different areas for the sake of performance. However, when the input characters are not pure ASCII, they need to be converted to the corresponding ASCII replacement characters. To accomodate this need, Oracle JDBC implements this flag. This flag is default to false, in which no characters will be converted, quick ASCII conversion is done for good performance. When this flag is set to true, Oracle JDBC will check for non-ASCII characters and convert them with replacement characters. This flag controls all areas where ASCII conversion is done.

See Also:

Constant Field Values

CONNECTION_PROPERTY_STRICT_ASCII_CONVERSION_DEFAULT

`static final String CONNECTION_PROPERTY_STRICT_ASCII_CONVERSION_DEFAULT`

See Also:

Constant Field Values

CONNECTION_PROPERTY_STRICT_ASCII_CONVERSION_ACCESSMODE

CONNECTION_PROPERTY_CONNECTION_CLASS

```
static final String CONNECTION_PROPERTY_CONNECTION_CLASS
```

Specify the connection class name for Database Resident Connection Pool (DRCP). Connection class must be provided to enable DRCP. Along with the connection class the URL must be altered to include `(SERVER=POOLED)` in long URL form. In thin the short URL form should be modified to append `:POOLED`.

See Also:

Constant Field Values

CONNECTION_PROPERTY_CONNECTION_CLASS_DEFAULT

```
static final String CONNECTION_PROPERTY_CONNECTION_CLASS_DEFAULT
```

CONNECTION_PROPERTY_CONNECTION_CLASS_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_CONNECTION_CLASS_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_DRCP_TAG_NAME

```
static final String CONNECTION_PROPERTY_DRCP_TAG_NAME
```

This is the tag name that for Database Resident Connection Pool (DRCP). Tag name is provided during connect time. The server will make an attempt to obtain a server process of the same tag. If it succeeds the next `#attachConnection()` will return `true`.

See Also:

Constant Field Values

CONNECTION_PROPERTY_DRCP_TAG_NAME_DEFAULT

```
static final String CONNECTION_PROPERTY_DRCP_TAG_NAME_DEFAULT
```

CONNECTION_PROPERTY_DRCP_TAG_NAME_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_DRCP_TAG_NAME_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_CONNECTION_PURITY

```
static final String CONNECTION_PROPERTY_CONNECTION_PURITY
```

Specify the connection purity for a Database Resident Connection Pool (DRCP) connection. Session purity specifies whether the application wants a "brand new" session or whether the application logic is set up to reuse a "pooled" session. There are two possible values, `NEW` or `SELF`. The default is `SELF`.

See Also:

Constant Field Values

CONNECTION_PROPERTY_CONNECTION_PURITY_DEFAULT

```
static final String CONNECTION_PROPERTY_CONNECTION_PURITY_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_CONNECTION_PURITY_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_CONNECTION_PURITY_ACCESSMODE
```

See Also:

Constant Field Values

Enables multiple tagging feature in DRCP. Default value is false.
A valid tag has to be a key value pair separated by = character. Multiple tags are separated by ; character.
Value of `key` and `value` can not be null or empty. This property is valid only for thin driver.

See Also:
Constant Field Values

CONNECTION_PROPERTY_USE_DRCP_MULTIPLE_TAG_DEFAULT

```
static final String CONNECTION_PROPERTY_USE_DRCP_MULTIPLE_TAG_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_USE_DRCP_MULTIPLE_TAG_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_USE_DRCP_MULTIPLE_TAG_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_DRCP_PLSQL_CALLBACK

```
static final String CONNECTION_PROPERTY_DRCP_PLSQL_CALLBACK
```

This is PL/SQL `"fix-up"` callback name which is provided by the application, and it is used to transform a session checked out from the pool to the desired state requested by the application.
`"fix-up"` callback can provide performance improvements to applications by running the `"session state fix-up"` logic on the server, thereby eliminating application round-trips to the database to run the `"fix-up"` logic.
This is an optional configuration.
This property is valid only for thin driver.

See Also:
Constant Field Values

CONNECTION_PROPERTY_DRCP_PLSQL_CALLBACK_DEFAULT

```
static final String CONNECTION_PROPERTY_DRCP_PLSQL_CALLBACK_DEFAULT
```

CONNECTION_PROPERTY_DRCP_PLSQL_CALLBACK_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_DRCP_PLSQL_CALLBACK_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_FORCE_DNS_LOAD_BALANCING

```
static final String CONNECTION_PROPERTY_THIN_FORCE_DNS_LOAD_BALANCING
```

When a hostname resolves to multiple addresses, the JDBC thin driver retrieves an array of addresses by calling "InetAddress.getAllByName()" and attempts to connect to first address in the array. If the connection fails, it tries to connect to the second address and so on.

By default, because "InetAddress.getAllByName()" always returns the addresses in the same order, the first connection attempt will always be made to the same IP address. This defeats the goal of SCAN (Single Client Access Name which is a 11.2 RAC feature). In order to force the the driver to make the first connection attempt to a different IP address each time, you can set this property to "true". The default value is "false".

When this connection is set to "true", the array of IP addresses that a hostname resolves to, will be rotated by one for each new JDBC connection. As a result, DNS load balancing will happen properly.

This is a JDBC thin driver property only.

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_FORCE_DNS_LOAD_BALANCING_DEFAULT

```
static final String CONNECTION_PROPERTY_THIN_FORCE_DNS_LOAD_BALANCING_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_THIN_FORCE_DNS_LOAD_BALANCING_ACCESSMODE

CONNECTION_PROPERTY_ENABLE_TEMP_LOB_REF_COUNT

```
static final String CONNECTION_PROPERTY_ENABLE_TEMP_LOB_REF_COUNT
```

By default the JDBC thin driver counts the temp LOB references and only closes them on the server when this count is down to zero. For example it may happen that two instances of CLOB (or OracleClob) A and B point to the same temp lob. At this point thin's temp lob ref count is 2. If you close A, no roundtrip will be issued because B is still holding on the temp LOB (counts is 1). Thin will send a close to the database only when B is also closed.

The JDBC Thin driver will tell the server that it's counting temp lob references so that the server's temp lob ref count is always 1 as long as a close hasn't been issued.

If you're running into `ORA-22922: NONEXISTENT LOB VALUE` errors in your application you should make sure that you haven't accidentally set this property to *false*.

This property applies to the JDBC Thin driver only. It's new in 11.2.0.3.

See Also:
Constant Field Values

CONNECTION_PROPERTY_ENABLE_TEMP_LOB_REF_COUNT_DEFAULT

```
static final String CONNECTION_PROPERTY_ENABLE_TEMP_LOB_REF_COUNT_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_ENABLE_TEMP_LOB_REF_COUNT_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_ENABLE_TEMP_LOB_REF_COUNT_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_NET_KEEPALIVE

```
static final String CONNECTION_PROPERTY_NET_KEEPALIVE
```

Enables TCP keep alive on the network connection.

Valid values for this property are "true" or "false". If this property is not set, the default value is "false".

When this property is set to "true", a TCP keep alive probe will be initiated when the network connection has been idle for some period of time.

The behavior of the keep alive probe can be configured using additional connection properties:

- `CONNECTION_PROPERTY_TCP_KEEPIDLE`
- `CONNECTION_PROPERTY_TCP_KEEPINTERVAL`
- `CONNECTION_PROPERTY_TCP_KEEPCOUNT`

See Also:
`StandardSocketOptions.SO_KEEPALIVE`, Constant Field Values

CONNECTION_PROPERTY_NET_KEEPALIVE_DEFAULT

```
static final String CONNECTION_PROPERTY_NET_KEEPALIVE_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_NET_KEEPALIVE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_NET_KEEPALIVE_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_SQL_TRANSLATION_PROFILE

```
static final String CONNECTION_PROPERTY_SQL_TRANSLATION_PROFILE
```

The string identifier for the translation profile or the translator to be used. Presence of this property activates the support for SQL Translation and is thus mandatory if SQL Translation feature is required.

See Also:
Constant Field Values

CONNECTION_PROPERTY_SQL_TRANSLATION_PROFILE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_SQL_TRANSLATION_PROFILE_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_SQL_ERROR_TRANSLATION_FILE

```
static final String CONNECTION_PROPERTY_SQL_ERROR_TRANSLATION_FILE
```

Path to an xml file which provides the Error code translations for those errors which occur if a connection can not be established to the server. The XML file is to be provided by the user and must conform to the following DTD. This is an optional property and if not provided then untranslated exceptions with oracle error codes are thrown. This property only affects the exceptions which happen when a connection to the server cannot be established. Once connection is established the translation happens on the server bypassing the local error translation file.

```
<!DOCTYPE LocalTranslationProfile[
<!ELEMENT LocalTranslationProfile (Exception+)>
<!ELEMENT Exception (ORAError, ErrorCode, SQLState )>
<!ELEMENT ORAError (#PCDATA)>
<!ELEMENT ErrorCode (#PCDATA)>
<!ELEMENT SQLState (#PCDATA)>
]>
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_SQL_ERROR_TRANSLATION_FILE_DEFAULT

```
static final String CONNECTION_PROPERTY_SQL_ERROR_TRANSLATION_FILE_DEFAULT
```

CONNECTION_PROPERTY_SQL_ERROR_TRANSLATION_FILE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_SQL_ERROR_TRANSLATION_FILE_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_READONLY_INSTANCE_ALLOWED

```
static final String CONNECTION_PROPERTY_READONLY_INSTANCE_ALLOWED
```

This property allows connection creation to a read-only instance if the value is set to true. It is applicable for sharded database only. The default value is false. This property is only supported by Thin driver.

Since:

21c

See Also:

Constant Field Values

CONNECTION_PROPERTY_READONLY_INSTANCE_ALLOWED_DEFAULT

```
static final String CONNECTION_PROPERTY_READONLY_INSTANCE_ALLOWED_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_READONLY_INSTANCE_ALLOWED_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_READONLY_INSTANCE_ALLOWED_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_ENABLE_RESULTSET_CACHE

```
static final String CONNECTION_PROPERTY_ENABLE_RESULTSET_CACHE
```

This property is ignored in 18c.

CONNECTION_PROPERTY_ENABLE_RESULTSET_CACHE_DEFAULT

```
static final String CONNECTION_PROPERTY_ENABLE_RESULTSET_CACHE_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_ENABLE_RESULTSET_CACHE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_ENABLE_RESULTSET_CACHE_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_ENABLE_QUERY_RESULT_CACHE

```
static final String CONNECTION_PROPERTY_ENABLE_QUERY_RESULT_CACHE
```

This property is introduced in 18c. If set to false, this property turns off the ResultSet Cache feature of the JDBC Thin driver. To use this feature the server-side initialization parameter CLIENT_RESULT_CACHE_SIZE also has to be configured to a non zero value. This value controls how much memory the Thin driver can use for its cache.
A read-only or read-mostly table can then be annoted (`RESULT_CACHE(MODE FORCE)` for example) for its data to be cached on the driver. You can also use a SQL hint `/*+result_cache */` to identify queries that are worth being cached.

See Also:
Constant Field Values

CONNECTION_PROPERTY_ENABLE_QUERY_RESULT_CACHE_DEFAULT

```
static final String CONNECTION_PROPERTY_ENABLE_QUERY_RESULT_CACHE_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_ENABLE_QUERY_RESULT_CACHE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_ENABLE_QUERY_RESULT_CACHE_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_BACKWARD_COMPATIBLE_UPDATEABLE_RESULTSET

```
static final String CONNECTION_PROPERTY_BACKWARD_COMPATIBLE_UPDATEABLE_RESULTSET
```

If set to true, use the old, pre 12.1.0.2.0, updateable ResultSet behavior. If false use JDBC standard compliant updateable ResultSet behavior.

See Also:
Constant Field Values

CONNECTION_PROPERTY_BACKWARD_COMPATIBLE_UPDATEABLE_RESULTSET_DEFAULT

```
static final String CONNECTION_PROPERTY_BACKWARD_COMPATIBLE_UPDATEABLE_RESULTSET_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_BACKWARD_COMPATIBLE_UPDATEABLE_RESULTSET_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_BACKWARD_COMPATIBLE_UPDATEABLE_RESULTSET_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_ALLOWED_LOGON_VERSION

```
static final String CONNECTION_PROPERTY_ALLOWED_LOGON_VERSION
```

Minimum authentication protocol required by the client. The term VERSION in the parameter name refers to the version of the authentication protocol, not the Oracle Database release. If the database doesn't meet or exceed the value defined by this parameter, then JDBC throws ORA-17292 : No valid logon method found.
Allowed values :

Thus if client sets the allowedLogonVersion to be 10. Then connection would fail if the database server doesn support authentication protocol 10 or above ie 11, 12 and 12a. And similarly for other values. Default logon version for JDBC thin is 8. These values are JDBC thin counterpart of SQLNET.ALLOWED_LOGON_VERSION_CLIENT. For more information on the logon values and authentication protocol, read the documenation for SQLNET.ALLOWED_LOGON_VERSION_CLIENT and SQLNET.ALLOWED_LOGON_VERSION_SERVER.

See Also:

Constant Field Values

CONNECTION_PROPERTY_ALLOWED_LOGON_VERSION_DEFAULT

`static final String CONNECTION_PROPERTY_ALLOWED_LOGON_VERSION_DEFAULT`

See Also:

Constant Field Values

CONNECTION_PROPERTY_ALLOWED_LOGON_VERSION_ACCESSMODE

`static final byte CONNECTION_PROPERTY_ALLOWED_LOGON_VERSION_ACCESSMODE`

See Also:

Constant Field Values

CONNECTION_PROPERTY_COMMIT_OPTION

`static final String CONNECTION_PROPERTY_COMMIT_OPTION`

This connection property lets you define a default commit option that will be used when calling `connection.commit();`. This can be useful in cases where you don't want to rewrite your application to specify a commit option at the call level, such as `connection.commit(myCommitOption);`.

This property can be set at the system level (all connections will use it) or at the connection level (only that particular connection will be affected). A call level commit option will override the default value.

Note that by default, if you don't set this property, the commit option is '0', zero, and the Oracle server's defaults apply. These defaults are: IMMEDIATE and WAIT (IO operations are done immediately and the call waits until the operation has completed to return).

See Also:

Constant Field Values

CONNECTION_PROPERTY_COMMIT_OPTION_DEFAULT

`static final String CONNECTION_PROPERTY_COMMIT_OPTION_DEFAULT`

CONNECTION_PROPERTY_COMMIT_OPTION_ACCESSMODE

`static final byte CONNECTION_PROPERTY_COMMIT_OPTION_ACCESSMODE`

See Also:

Constant Field Values

CONNECTION_PROPERTY_DOWN_HOSTS_TIMEOUT

`static final String CONNECTION_PROPERTY_DOWN_HOSTS_TIMEOUT`

To specify the amount of time in seconds that information about the `down` state of server hosts is kept in driver's cache.

The driver discovers the `down` state of server hosts when attempting connections. When a connection attempt fails, the information about the `down` state of the server host is added to the driver's cache. Subsequent connection attempts moves the `down` hosts to the end of the address list, thereby reducing the priority of such hosts. When the time specified by the `oracle.net.DOWN_HOSTS_TIMEOUT` parameter has passed, the host is purged from the driver's cache, and its priority in the address list is restored.

Default value is 600 seconds.

See Also:

Constant Field Values

CONNECTION_PROPERTY_DOWN_HOSTS_TIMEOUT_DEFAULT

`static final String CONNECTION_PROPERTY_DOWN_HOSTS_TIMEOUT_DEFAULT`

See Also:

Constant Field Values

Constant Field Values

CONNECTION_PROPERTY_FAN_ENABLED

`static final String CONNECTION_PROPERTY_FAN_ENABLED`

Specifies whether driver High Availability (HA) or FAN (Fast Application Notification) is enabled.

This property can be set at the system level (which applies to all connections), or at the connection level (which applies to a particular connection). The property is applicable to THIN and JDBC-OCI drivers only.

By default, if you don't set this property, FAN/HA is enabled. The primary use of this property is to disable driver HA/FAN.

See Also:
Constant Field Values

CONNECTION_PROPERTY_FAN_ENABLED_DEFAULT

`static final String CONNECTION_PROPERTY_FAN_ENABLED_DEFAULT`

See Also:
Constant Field Values

CONNECTION_PROPERTY_FAN_ENABLED_ACCESSMODE

`static final byte CONNECTION_PROPERTY_FAN_ENABLED_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_TNS_ADMIN

`static final String CONNECTION_PROPERTY_TNS_ADMIN`

This property is used for setting the TNS Admin path. When using TNS Names with JDBC Thin driver this property has to be set to the directory which contains the `tnsnames.ora` file.
This property can also be used to set the path of a properties file. When set, the driver will look for a file named `ojdbc.properties` in the TNS Admin directory.

See Also:
`CONNECTION_PROPERTY_CONFIG_FILE`, Constant Field Values

CONNECTION_PROPERTY_TNS_ADMIN_DEFAULT

`static final String CONNECTION_PROPERTY_TNS_ADMIN_DEFAULT`

CONNECTION_PROPERTY_TNS_ADMIN_ACCESSMODE

`static final byte CONNECTION_PROPERTY_TNS_ADMIN_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_NETWORK_COMPRESSION

`static final String CONNECTION_PROPERTY_NETWORK_COMPRESSION`

Enables compression of the protocol data sent over network. The value can be either `"on"` , `"off"` or `"auto"`. The default value is `"off"`.

See Also:
Constant Field Values

CONNECTION_PROPERTY_NETWORK_COMPRESSION_DEFAULT

`static final String CONNECTION_PROPERTY_NETWORK_COMPRESSION_DEFAULT`

See Also:
Constant Field Values

CONNECTION_PROPERTY_NETWORK_COMPRESSION_ACCESSMODE

CONNECTION_PROPERTY_NETWORK_COMPRESSION_LEVELS

```
static final String CONNECTION_PROPERTY_NETWORK_COMPRESSION_LEVELS
```

The value is a comma separated list of supported levels in the user preference order surrounded by brackets. The value is used at the time of negotiation to check what levels can be supported by both the client and the server and decide on the first common match. Thin driver only supports `("high")` compression level and it is the default value, so setting this property is optional. Please note that the server should be configured to support `high` compression level. By default the server supports only `low` compression level.

See Also:

Constant Field Values

CONNECTION_PROPERTY_NETWORK_COMPRESSION_LEVELS_DEFAULT

```
static final String CONNECTION_PROPERTY_NETWORK_COMPRESSION_LEVELS_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_NETWORK_COMPRESSION_LEVELS_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_NETWORK_COMPRESSION_LEVELS_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_NETWORK_COMPRESSION_THRESHOLD

```
static final String CONNECTION_PROPERTY_NETWORK_COMPRESSION_THRESHOLD
```

Minimum size of data in packet required to perform compression. Packet compression will not be done if size of data to be sent in the packet is less than specified value. The default is `"1024"`. The value cannot be less than `"200"`. The value is in bytes.

See Also:

Constant Field Values

CONNECTION_PROPERTY_NETWORK_COMPRESSION_THRESHOLD_DEFAULT

```
static final String CONNECTION_PROPERTY_NETWORK_COMPRESSION_THRESHOLD_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_NETWORK_COMPRESSION_THRESHOLD_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_NETWORK_COMPRESSION_THRESHOLD_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_CONFIG_FILE

```
static final String CONNECTION_PROPERTY_CONFIG_FILE
```

This property provides the location of one or more properties files. When a connection is opened, the driver will read connection properties from these files. This feature is not supported for connections made by the server-side internal driver.
A file location can be a simple path: "path/to/my/file", or a platform neutral file URI: "file:///path/to/my/file".
Multiple files can be given as a comma delimited list: "fileOne, fileTwo, fileThree". The list may contain the value "default", which will be resolved as the default locations described below. Higher precedence is given to files which appear later in the list. In the example, a property defined in fileThree would override fileTwo's definition of that property. Likewise, fileTwo's definitions can override fileOne's.
This feature can be explictly disabled by setting this property to an empty string.

Default File Locations
If oracle.jdbc.config.file is not set, the driver will attempt to read from a default location: $TNS_ADMIN/ojdbc.properties. Here, $TNS_ADMIN can refer to "TNS_ADMIN" set as a system property or environment variable. It can also refer to the connection property `CONNECTION_PROPERTY_TNS_ADMIN`. If the connection property is set, the driver will use that value rather than the system property or environment variable. If the connection property is not set, "TNS_ADMIN" as a system property will override the environment variable.
If a tnsnames alias is used to connect, the driver will also attempt to read from $TNS_ADMIN/ojdbc_<alias>.properties, where <alias> is the name given in the connection string. For example: A DataSource configured with "jdbc:oracle:thin:@*orcl*" would read connection properties from $TNS_ADMIN/ojdbc_*orcl*.properties, in addition to $TNS_ADMIN/ojdbc.properties. A property defined in ojdbc_orcl.properties would override ojdbc.properties' definition of that property. If ojdbc.properties does not exist, then properties will only be read from ojdbc_<alias>.properties.
There is no requirement that any of the default files actually exist. If the driver can not locate one of the default files, or it can not resolve the value of $TNS_ADMIN, it will still attempt to connect with any properties provided by alternative sources such as system propeties or a Properties object.

1. "${xyz}" is replaced with a system property or environment variable named "xyz". The environment variable is used only if the system property is not defined. If neither are defined, the driver will throw a SQLException.
2. "?" is replaced with the value of the ORACLE_HOME environment variable. If the environment variable is not defined, the driver will throw a SQLException.
3. "${/}" is replaced with the file system's path separator. "path${/}to${/}" is replaced with "path/to/" on Linux, and "path\to\" on Windows.
4. Two consecutive special characters will escape evaluation.
   - "$${xyz}" is replaced with "${xyz}"
   - "??abc" is replaced with "?abc"

Precedence
An order of precedence is applied when connection properties are defined by multiple sources. A property's value will always be resolved by the source which has the highest precedence. The sources of connection properties, ranked from highest to lowest precedence are:

1. Properties set in the connection URL.
2. A Properties object passed to OracleDataSource, DriverManager, etc.
3. The jar-internal properties file: defaultConnectionProperties.properties in the oracle.jdbc package
4. Java system properties
5. An external properties file: A default under $TNS_ADMIN, or one given by oracle.jdbc.config.file
6. The default value of a property, as specified by the CONNECTION_PROPERTY_{name}_DEFAULT constants

See Also:

`Properties.load(Reader)`, `ACCESSMODE_FILEPROP`, Constant Field Values

CONNECTION_PROPERTY_CONFIG_FILE_DEFAULT

```
static final String CONNECTION_PROPERTY_CONFIG_FILE_DEFAULT
```

CONNECTION_PROPERTY_CONFIG_FILE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_CONFIG_FILE_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_WEBSOCKET_USER

```
static final String CONNECTION_PROPERTY_WEBSOCKET_USER
```

This connection property is used to configure the username of the webserver, when the JDBC Thin driver is configured to connect to a webserver using the `Secure Websocket protocol (WSS)`.
The webserver acts as a reverse proxy for the Oracle Database.
The default value of this property is `null`.

See Also:

`CONNECTION_PROPERTY_WEBSOCKET_PASSWORD`, Constant Field Values

CONNECTION_PROPERTY_WEBSOCKET_USER_DEFAULT

```
static final String CONNECTION_PROPERTY_WEBSOCKET_USER_DEFAULT
```

CONNECTION_PROPERTY_WEBSOCKET_USER_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_WEBSOCKET_USER_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_WEBSOCKET_PASSWORD

```
static final String CONNECTION_PROPERTY_WEBSOCKET_PASSWORD
```

This connection property is used to configure the password of the webserver, when the JDBC Thin driver is configured to connect to a webserver using the `Secure Websocket protocol (WSS)`.
The webserver acts as a reverse proxy for the Oracle Database.
The default value of this property is `null`.

See Also:

`CONNECTION_PROPERTY_WEBSOCKET_USER`, Constant Field Values

CONNECTION_PROPERTY_WEBSOCKET_PASSWORD_DEFAULT

```
static final String CONNECTION_PROPERTY_WEBSOCKET_PASSWORD_DEFAULT
```

Constant Field Values

CONNECTION_PROPERTY_SOCKS_PROXY_HOST

```
static final String CONNECTION_PROPERTY_SOCKS_PROXY_HOST
```

This connection property is used to configure the host name of the SOCKS proxy server. When this property is configured the connection to the Oracle Database Server is attempted via this SOCKS proxy server. The default value of this property is `null`. The HTTPS proxy configuration has higher precedence over SOCKS proxy. If both are configured then HTTPS proxy is used for establishing connection to the Oracle Database Server.

Since:

20c

See Also:

`CONNECTION_PROPERTY_SOCKS_PROXY_PORT`, `CONNECTION_PROPERTY_THIN_HTTPS_PROXY_HOST`, `CONNECTION_PROPERTY_THIN_HTTPS_PROXY_PORT`, Constant Field Values

CONNECTION_PROPERTY_SOCKS_PROXY_HOST_DEFAULT

```
static final String CONNECTION_PROPERTY_SOCKS_PROXY_HOST_DEFAULT
```

CONNECTION_PROPERTY_SOCKS_PROXY_HOST_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_SOCKS_PROXY_HOST_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_SOCKS_PROXY_PORT

```
static final String CONNECTION_PROPERTY_SOCKS_PROXY_PORT
```

This connection property is used to configure the port value of the SOCKS proxy server. The default value of this property is `1080`.

Since:

20c

See Also:

`CONNECTION_PROPERTY_SOCKS_PROXY_HOST`, `CONNECTION_PROPERTY_THIN_HTTPS_PROXY_HOST`, `CONNECTION_PROPERTY_THIN_HTTPS_PROXY_PORT`, Constant Field Values

CONNECTION_PROPERTY_SOCKS_PROXY_PORT_DEFAULT

```
static final String CONNECTION_PROPERTY_SOCKS_PROXY_PORT_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_SOCKS_PROXY_PORT_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_SOCKS_PROXY_PORT_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_SOCKS_REMOTE_DNS

```
static final String CONNECTION_PROPERTY_SOCKS_REMOTE_DNS
```

This connection property is used to specify whether the DNS lookup for the DB Host should be performed locally or remotely when a SOCKS5 Proxy is being used.
The default value of this property is `false` and the DNS lookup is performed locally.
Please note that when this property is set to `true` the DNS load balancing is disabled.

Since:

21c

See Also:

`CONNECTION_PROPERTY_SOCKS_PROXY_HOST`, `CONNECTION_PROPERTY_THIN_HTTPS_PROXY_HOST`, `CONNECTION_PROPERTY_THIN_HTTPS_PROXY_PORT`, `#THIN_FORCE_DNS_LOAD_BALANCING`, Constant Field Values

CONNECTION_PROPERTY_SOCKS_REMOTE_DNS_DEFAULT

```
static final String CONNECTION_PROPERTY_SOCKS_REMOTE_DNS_DEFAULT
```

CONNECTION_PROPERTY_SOCKS_REMOTE_DNS_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_SOCKS_REMOTE_DNS_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_DEFAULT_CONNECTION_VALIDATION

```
static final String CONNECTION_PROPERTY_DEFAULT_CONNECTION_VALIDATION
```

This connection property is used to specify how much effort to put into validating a `Connection`.
This property controls what isValid() does.
The possible values for this property are - "NONE", "LOCAL", "SOCKET", "NETWORK", "SERVER" and "COMPLETE".
The values are case-sensitive, setting any other value throws exception.
The default value of this property is "NETWORK".

See Also:
Constant Field Values

CONNECTION_PROPERTY_DEFAULT_CONNECTION_VALIDATION_DEFAULT

```
static final String CONNECTION_PROPERTY_DEFAULT_CONNECTION_VALIDATION_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_DEFAULT_CONNECTION_VALIDATION_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_DEFAULT_CONNECTION_VALIDATION_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_ENABLE_AC_SUPPORT

```
static final String CONNECTION_PROPERTY_ENABLE_AC_SUPPORT
```

Specifies whether driver support for Application Continuity (AC) is enabled.

This property can be set at the system level (which applies to all connections), or at the connection level (which applies to a particular connection). The property is applicable to THIN driver only.

By default, if you don't set this property, AC support is enabled on the JDBC driver data sources. The primary use of this property is to disable AC on the data sources.

Note that when this property is set to true, whether AC is actually active depends on other factors like server AC configuration.

See Also:
Constant Field Values

CONNECTION_PROPERTY_ENABLE_AC_SUPPORT_DEFAULT

```
static final String CONNECTION_PROPERTY_ENABLE_AC_SUPPORT_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_ENABLE_AC_SUPPORT_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_ENABLE_AC_SUPPORT_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_ENABLE_TG_SUPPORT

```
static final String CONNECTION_PROPERTY_ENABLE_TG_SUPPORT
```

Specifies whether driver support for Transaction Guard (TG) is enabled.

This property can be set at the system level (which applies to all connections), or at the connection level (which applies to a particular connection). The property is applicable to THIN driver only.

By default, if you don't set this property, TG support is disabled on the JDBC driver data sources, unless Application Continuity (AC) is enabled. The primary use of this property is to enable TG on the data sources, when AC is not enabled.

CONNECTION_PROPERTY_ENABLE_TG_SUPPORT_DEFAULT

```
static final String CONNECTION_PROPERTY_ENABLE_TG_SUPPORT_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_ENABLE_TG_SUPPORT_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_ENABLE_TG_SUPPORT_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_ENABLE_IMPLICIT_REQUESTS

```
static final String CONNECTION_PROPERTY_ENABLE_IMPLICIT_REQUESTS
```

Specifies whether to enable implicit request boundary support for Application Continuity (AC).

Implicit request support helps to reduce application failover recovery time. This AC optimization should be used with caution for applications that change server session states during a request. For more details, please consult the JDBC and RAC documentations on Auto-AC.

This property can be set at the system level (which applies to all connections), or at the connection level (which applies to a particular connection). The property is applicable to THIN driver only.

By default, the value of this property is "true", which means that implicit request support is enabled.

See Also:
Constant Field Values

CONNECTION_PROPERTY_ENABLE_IMPLICIT_REQUESTS_DEFAULT

```
static final String CONNECTION_PROPERTY_ENABLE_IMPLICIT_REQUESTS_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_ENABLE_IMPLICIT_REQUESTS_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_ENABLE_IMPLICIT_REQUESTS_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_DRCP_MULTIPLEXING_IN_REQUEST_APIS

```
static final String CONNECTION_PROPERTY_DRCP_MULTIPLEXING_IN_REQUEST_APIS
```

Specifies whether to enable DRCP-attach in beginRequest and DRCP-detach in endRequest.

Enabling this makes DRCP transparent to connection pools that call the request-APIs at pool check-out and check-in.

This property can be set at the system level (which applies to all connections), or at the connection level (which applies to a particular connection).

By default, the value of this property is "false", which means that the support is disabled.

See Also:
Constant Field Values

CONNECTION_PROPERTY_DRCP_MULTIPLEXING_IN_REQUEST_APIS_DEFAULT

```
static final String CONNECTION_PROPERTY_DRCP_MULTIPLEXING_IN_REQUEST_APIS_DEFAULT
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_DRCP_MULTIPLEXING_IN_REQUEST_APIS_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_DRCP_MULTIPLEXING_IN_REQUEST_APIS_ACCESSMODE
```

See Also:

CONNECTION_PROPERTY_CONTINUE_BATCH_ON_ERROR

`static final String CONNECTION_PROPERTY_CONTINUE_BATCH_ON_ERROR`

This connection property specifies whether to continue batch execution when server encounters an erroneous row in the batch. If this property value is set to "true", server skips the erroneous row in the batch and continues processing rest of the rows. BatchUpdateException.getLargeUpdateCounts() method can be used to know which row in the batch failed.

This property is applicable to the THIN driver only. The default value of this property is "false".

See Also:
Constant Field Values

CONNECTION_PROPERTY_CONTINUE_BATCH_ON_ERROR_DEFAULT

`static final String CONNECTION_PROPERTY_CONTINUE_BATCH_ON_ERROR_DEFAULT`

See Also:
Constant Field Values

CONNECTION_PROPERTY_CONTINUE_BATCH_ON_ERROR_ACCESSMODE

`static final byte CONNECTION_PROPERTY_CONTINUE_BATCH_ON_ERROR_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_TCP_KEEPIDLE

`static final String CONNECTION_PROPERTY_TCP_KEEPIDLE`

Specifies a number of seconds for a network connection to remain idle before initiating a keep alive probe. If this property is set to a value other than -1, the property value will override any other value set by an EXPIRE_TIME parameter in a connect descriptor URL.

This property is applicable to the THIN driver only.

`CONNECTION_PROPERTY_NET_KEEPALIVE` must be set to "true" to enable TCP keep alive.

The default value is system dependent. If this property is not set, or if it is set to -1, the driver will use the system dependent default value.

Since:
20

See Also:
`ExtendedSocketOptions.TCP_KEEPIDLE`, Constant Field Values

CONNECTION_PROPERTY_TCP_KEEPIDLE_DEFAULT

`static final String CONNECTION_PROPERTY_TCP_KEEPIDLE_DEFAULT`

See Also:
Constant Field Values

CONNECTION_PROPERTY_TCP_KEEPIDLE_ACCESSMODE

`static final byte CONNECTION_PROPERTY_TCP_KEEPIDLE_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_TCP_KEEPINTERVAL

`static final String CONNECTION_PROPERTY_TCP_KEEPINTERVAL`

Specifies a number of seconds to wait before retransmitting a keep alive probe.

This property is applicable to the THIN driver only.

`CONNECTION_PROPERTY_NET_KEEPALIVE` must be set to "true" to enable TCP keep alive.

The default value is system dependent. If this property is not set, or if it is set to -1, the driver will use the system dependent default value.

Since:
20

See Also:
`ExtendedSocketOptions.TCP_KEEPINTERVAL`, Constant Field Values

Constant Field Values

CONNECTION_PROPERTY_TCP_KEEPINTERVAL_ACCESSMODE

`static final byte CONNECTION_PROPERTY_TCP_KEEPINTERVAL_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_USE_SHARDING_DRIVER_CONNECTION

`static final String CONNECTION_PROPERTY_USE_SHARDING_DRIVER_CONNECTION`

This is applicable only for the thin driver. Pass "true" to use the sharding driver connection. The sharding driver connection derives a sharding key from a SQL command. "false" is the default behavior which would give a Thin driver connection.

Since:
20

See Also:
Constant Field Values

CONNECTION_PROPERTY_USE_SHARDING_DRIVER_CONNECTION_DEFAULT

`static final String CONNECTION_PROPERTY_USE_SHARDING_DRIVER_CONNECTION_DEFAULT`

See Also:
Constant Field Values

CONNECTION_PROPERTY_USE_SHARDING_DRIVER_CONNECTION_ACCESSMODE

`static final byte CONNECTION_PROPERTY_USE_SHARDING_DRIVER_CONNECTION_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_TCP_KEEPCOUNT

`static final String CONNECTION_PROPERTY_TCP_KEEPCOUNT`

Specifies a maximum number of keep alive probes to be sent before a connection is considered broken.

This property is applicable to the THIN driver only.

`CONNECTION_PROPERTY_NET_KEEPALIVE` must be set to "true" to enable TCP keep alive.

The default value is system dependent. If this property is not set, or if it is set to -1, the driver will use the system dependent default value.

Since:
20

See Also:
`ExtendedSocketOptions.TCP_KEEPCOUNT`, Constant Field Values

CONNECTION_PROPERTY_TCP_KEEPCOUNT_DEFAULT

`static final String CONNECTION_PROPERTY_TCP_KEEPCOUNT_DEFAULT`

See Also:
Constant Field Values

CONNECTION_PROPERTY_TCP_KEEPCOUNT_ACCESSMODE

`static final byte CONNECTION_PROPERTY_TCP_KEEPCOUNT_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_REQUEST_SIZE_LIMIT

`static final String CONNECTION_PROPERTY_REQUEST_SIZE_LIMIT`

Specifies the maximum request size, in terms of number of JDBC calls, beyond which AC replay will be disabled. For example, if the property value is set to 100, it

Replay disabling applies only to any request that exceeds the limit. Replay will be reenabled at the next beginRequest on the same JDBC physical connection.

This property can be set at the system level (which applies to all connections), or at the connection level (which applies to a particular connection). The property is applicable to THIN driver only.

There is a request-size histogram in the AC statistics that users can use to get an idea of the request size distribution of their application. It can be obtained by calling `getRequestSizeHistogram()` on `oracle.jdbc.replay.ReplayStatistics`.

The histogram is also dumped into the Oracle JDBC driver logs, when the driver detects significant memory pressure.

Users can use the histogram and AC statistics to determine the majority of their request sizes, and set the request-size limit to be slightly above those. For small number of long requests above the limit, users could consider alternatives such as more frequent connection pool checkout/checkin's, deploying Transparent Application Continuity (TAC) to reduce request size, or to allow replay being disabled for requests with only long queries.

By default, the value of this property is "2147483647", which is Integer.MAX_VALUE that means replay is practically enabled for all requests.

Since:

20.1

See Also:

Constant Field Values

CONNECTION_PROPERTY_REQUEST_SIZE_LIMIT_DEFAULT

`static final String CONNECTION_PROPERTY_REQUEST_SIZE_LIMIT_DEFAULT`

See Also:

Constant Field Values

CONNECTION_PROPERTY_REQUEST_SIZE_LIMIT_ACCESSMODE

`static final byte CONNECTION_PROPERTY_REQUEST_SIZE_LIMIT_ACCESSMODE`

See Also:

Constant Field Values

CONNECTION_PROPERTY_ONS_WALLET_FILE

`static final String CONNECTION_PROPERTY_ONS_WALLET_FILE`

Use this property to specify the ONS wallet file, when you need Oracle Fast Application Notification (FAN).

The Oracle JDBC driver just passes this wallet file to the lower layer. The file can be specified on a global or per-data source basis.

The supported wallet file specification syntax is the same as for `oracle.net.wallet_location`

If the `oracle.jdbc.ons.walletfile`property is not specified but the `oracle.net.wallet_location` property is, and if the `oracle.jdbc.ons.protocol` property is set to "TCPS", the driver will use `oracle.net.wallet_location` property's value as the ONS wallet file. In that case, both the JDBC connection and the ONS connection share the same Oracle wallet.

Since:

20.1

See Also:

`CONNECTION_PROPERTY_WALLET_LOCATION`, Constant Field Values

CONNECTION_PROPERTY_ONS_WALLET_FILE_DEFAULT

`static final String CONNECTION_PROPERTY_ONS_WALLET_FILE_DEFAULT`

CONNECTION_PROPERTY_ONS_WALLET_FILE_ACCESSMODE

`static final byte CONNECTION_PROPERTY_ONS_WALLET_FILE_ACCESSMODE`

See Also:

Constant Field Values

CONNECTION_PROPERTY_ONS_WALLET_PASSWORD

`static final String CONNECTION_PROPERTY_ONS_WALLET_PASSWORD`

Use this property to specify the ONS wallet password, which is only required if you don't enable auto-login in the ONS wallet. In this case "ewallet.p12" will be used instead of "cwallet.sso".

Since:

20.1

See Also:

`CONNECTION_PROPERTY_WALLET_PASSWORD`, Constant Field Values `static final String CONNECTION_PROPERTY_ONS_WALLET_PASSWORD_DEFAULT`

CONNECTION_PROPERTY_ONS_WALLET_PASSWORD_ACCESSMODE

`static final byte CONNECTION_PROPERTY_ONS_WALLET_PASSWORD_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_ONS_PROTOCOL

`static final String CONNECTION_PROPERTY_ONS_PROTOCOL`

Use this property to specify the ONS connection protocol, as either "TCP" or "TCPS". The default is "TCP".

When this property is "TCPS" and `oracle.jdbc.ons.walletfile` is not specified, any JDBC wallet configured via the connection property `oracle.net.wallet_location` will also be used as the ONS wallet for ONS connections.

If `oracle.jdbc.ons.walletfile` is specified, it will be used as the ONS wallet and the ONS connection protocol is assumed to be TCPS.

Since:
20.1

See Also:
Constant Field Values

CONNECTION_PROPERTY_ONS_PROTOCOL_DEFAULT

`static final String CONNECTION_PROPERTY_ONS_PROTOCOL_DEFAULT`

See Also:
Constant Field Values

CONNECTION_PROPERTY_ONS_PROTOCOL_ACCESSMODE

`static final byte CONNECTION_PROPERTY_ONS_PROTOCOL_ACCESSMODE`

See Also:
Constant Field Values

CONNECTION_PROPERTY_LOGIN_TIMEOUT

`static final String CONNECTION_PROPERTY_LOGIN_TIMEOUT`

Configures a timeout for creating a new connection. The value of this property is parsed as an integer number of seconds. A value of 0 configures the driver to not use a timeout. The default value is 0. Values which are less than 0 are invalid.

When specified, the timeout is applied to any method call which opens a new connection, such as `DataSource.getConnection()` or `ConnectionBuilder.build()`. If the timeout expires, these method calls will throw a `SQLException` with error code 18714.

A value specified for this property can be overriden by a value set with `CommonDataSource.setLoginTimeout(int)`.

This property is only supported by the Type 4 driver (ie: jdbc:oracle:thin).

Since:
20

See Also:
Constant Field Values

CONNECTION_PROPERTY_LOGIN_TIMEOUT_DEFAULT

`static final String CONNECTION_PROPERTY_LOGIN_TIMEOUT_DEFAULT`

See Also:
Constant Field Values

CONNECTION_PROPERTY_LOGIN_TIMEOUT_ACCESSMODE

`static final byte CONNECTION_PROPERTY_LOGIN_TIMEOUT_ACCESSMODE`

See Also:
Constant Field Values

Autonomous Database (ADB) and Real Application Clusters (RAC) planned maintenance.

This property can be set at the system level (which applies to all connections), or at the connection level (which applies to a particular connection). The property is applicable to THIN and JDBC-OCI drivers only.

By default, if you don't set this property, in-band notification is always enabled. The primary use of this property is to disable this feature, which might be necessary in case in-band notification interferes with similar features in upper-stacks (like a connection pool or application container), or malfunctions.

//_end_m4_ifInServer

See Also:

Constant Field Values

CONNECTION_PROPERTY_IN_BAND_NOTIFICATION_DEFAULT

```
static final String CONNECTION_PROPERTY_IN_BAND_NOTIFICATION_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_IN_BAND_NOTIFICATION_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_IN_BAND_NOTIFICATION_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_SSL_CONTEXT_PROTOCOL

```
static final String CONNECTION_PROPERTY_SSL_CONTEXT_PROTOCOL
```

Specifies a `protocol` name for the driver to use when obtaining an instance of `SSLContext` from `SSLContext.getInstance(String)` for a TLS enabled database connection.

This property has no effect on which versions of SSL or TLS will be accepted during handshakes with the database server. To configure the set of protocol versions accepted during handshakes, use `CONNECTION_PROPERTY_THIN_SSL_VERSION`.

If this property is not specified, the driver will use "TLS" by default.

This property is only supported by the Type 4 driver (ie: jdbc:oracle:thin).

Since:

20.3

See Also:

Constant Field Values

CONNECTION_PROPERTY_SSL_CONTEXT_PROTOCOL_DEFAULT

```
static final String CONNECTION_PROPERTY_SSL_CONTEXT_PROTOCOL_DEFAULT
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_SSL_CONTEXT_PROTOCOL_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_SSL_CONTEXT_PROTOCOL_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_TOKEN_AUTHENTICATION

```
static final String CONNECTION_PROPERTY_TOKEN_AUTHENTICATION
```

Enables the use of access tokens that are stored in a file system location when authenticating with Oracle Database.

In this release of Oracle JDBC, "OCI_TOKEN" and "OAUTH" are the only accepted values for this property. Setting this property to "OCI_TOKEN" or "OAUTH" configures Oracle JDBC to obtain tokens from the file system as described in the JavaDoc of `CONNECTION_PROPERTY_TOKEN_LOCATION`.

If an Oracle Net Descriptor style URL includes the TOKEN_AUTH parameter then the value of that parameter takes precedence over a value defined by this property.

If a username and password are provided, then Oracle JDBC will use them to authenticate with the database, and this property is ignored.

If a token is configured using `CONNECTION_PROPERTY_ACCESS_TOKEN`, then Oracle JDBC will use it to authenticate with the database, and this property is ignored.

Since:

23

CONNECTION_PROPERTY_TOKEN_AUTHENTICATION_DEFAULT

```
static final String CONNECTION_PROPERTY_TOKEN_AUTHENTICATION_DEFAULT
```

CONNECTION_PROPERTY_TOKEN_AUTHENTICATION_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_TOKEN_AUTHENTICATION_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_TOKEN_LOCATION

```
static final String CONNECTION_PROPERTY_TOKEN_LOCATION
```

When `CONNECTION_PROPERTY_TOKEN_AUTHENTICATION` is set to "OCI_TOKEN" or "OAUTH", this property specifies the file system path to obtain access tokens from. The path specified by this property must be a directory containing a file named "token", and the token file must contain a JSON Web Token (JWT) on a single line of UTF-8 encoded text. The JWT format is specified by RFC 7519.

- If the value of `CONNECTION_PROPERTY_TOKEN_AUTHENTICATION` is "OCI_TOKEN" then the path given by this property must be a directory containing a both a token file, and a file named "oci_db_key.pem" that stores the proof-of-possession key for the JWT token. The private key file must use the PEM format and must contain the base64 encoding of an RSA private key in the PKCS#8 format. The private key encoding must appear between the tags "-----BEGIN PRIVATE KEY---" and "-----END PRIVATE KEY-----". Oracle JDBC uses the private key to demonstrate proof of possession. Proof of possession is specified by RFC 7800.

  Note that the OCI CLI tool may be used to generate both the token and private key files. By default, the OCI CLI tool will write these files to the location of `$HOME/.oci/db-token/`. If the value of `CONNECTION_PROPERTY_TOKEN_AUTHENTICATION` is "OCI_TOKEN", and no location is configured by this property, then Oracle JDBC will read the token and private key files from the default location of `$HOME/.oci/db-token/`.

- If the value of `CONNECTION_PROPERTY_TOKEN_AUTHENTICATION` is "OAUTH", then only the token file is required, and the path given by this property may locate either a file or directory. If a file is located, then the JWT is read from it. Otherwise, if a directory is located, then the JWT is read from a file named "token" in that directory.

If an Oracle Net Descriptor style URL includes the TOKEN_LOCATION parameter then the value of that parameter takes precedence over a value defined by this property.

Since:

23

See Also:

Constant Field Values

CONNECTION_PROPERTY_TOKEN_LOCATION_DEFAULT

```
static final String CONNECTION_PROPERTY_TOKEN_LOCATION_DEFAULT
```

CONNECTION_PROPERTY_TOKEN_LOCATION_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_TOKEN_LOCATION_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_PASSWORD_AUTHENTICATION

```
static final String CONNECTION_PROPERTY_PASSWORD_AUTHENTICATION
```

Configures how Oracle JDBC performs authentication with a user name and password.

In this release of Oracle JDBC, "PASSWORD_VERIFIER" and "OCI_TOKEN" are the only accepted values for this property.

If the value is "PASSWORD_VERIFIER", then database authentication is performed.

If the value is "OCI_TOKEN", then authentication is performed with the Oracle Identity Cloud Service as described in `CONNECTION_PROPERTY_OCI_IAM_URL`

The default value of this property is "PASSWORD_VERIFIER".

If an Oracle Net Descriptor style URL includes the PASSWORD_AUTH parameter then the value of that parameter takes precedence over a value defined by this property.

Since:

23

See Also:

Constant Field Values

CONNECTION_PROPERTY_PASSWORD_AUTHENTICATION_DEFAULT

CONNECTION_PROPERTY_PASSWORD_AUTHENTICATION_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_PASSWORD_AUTHENTICATION_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_OCI_IAM_URL

```
static final String CONNECTION_PROPERTY_OCI_IAM_URL
```

When `CONNECTION_PROPERTY_PASSWORD_AUTHENTICATION` is set to "OCI_TOKEN", this property must specify the full path of the Identity and Access Management (IAM) endpoint that Oracle JDBC authenticates with, as in:

```
    https://<{iam-endpoint}/{version}/dbBearerToken
```

Oracle JDBC must be configured to trust the certificate of the server located by this URL. Certificates signed by an authority included in the JDK's default cacerts file will be trusted. Additional trusted certificates may be configued with `CONNECTION_PROPERTY_WALLET_LOCATION`, `CONNECTION_PROPERTY_THIN_JAVAX_NET_SSL_TRUSTSTORE`.

If an Oracle Net Descriptor style URL includes the OCI_IAM_URL parameter, then the value of that parameter takes precedence over a value defined by this property.

Since:

23

See Also:

Constant Field Values

CONNECTION_PROPERTY_OCI_IAM_URL_DEFAULT

```
static final String CONNECTION_PROPERTY_OCI_IAM_URL_DEFAULT
```

CONNECTION_PROPERTY_OCI_IAM_URL_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_OCI_IAM_URL_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_OCI_TENANCY

```
static final String CONNECTION_PROPERTY_OCI_TENANCY
```

When `CONNECTION_PROPERTY_PASSWORD_AUTHENTICATION` is set to "OCI_TOKEN", this property must specify the Oracle Cloud ID (OCID) of the cloud tenant for the user that Oracle JDBC authenticates as.

If an Oracle Net Descriptor style URL includes the OCI_TENANCY parameter, then the value of that parameter takes precedence over a value defined by this property.

Since:

23

See Also:

Constant Field Values

CONNECTION_PROPERTY_OCI_TENANCY_DEFAULT

```
static final String CONNECTION_PROPERTY_OCI_TENANCY_DEFAULT
```

CONNECTION_PROPERTY_OCI_TENANCY_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_OCI_TENANCY_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_OCI_COMPARTMENT

```
static final String CONNECTION_PROPERTY_OCI_COMPARTMENT
```

When `CONNECTION_PROPERTY_PASSWORD_AUTHENTICATION` is set to "OCI_TOKEN", this property specifies the Oracle Cloud ID (OCID) of the compartment for the database identified by `CONNECTION_PROPERTY_OCI_DATABASE`. If this property is not set, then Oracle JDBC requests access to all databases within the tenancy

Since:

23

See Also:

Constant Field Values

CONNECTION_PROPERTY_OCI_COMPARTMENT_DEFAULT

```
static final String CONNECTION_PROPERTY_OCI_COMPARTMENT_DEFAULT
```

CONNECTION_PROPERTY_OCI_COMPARTMENT_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_OCI_COMPARTMENT_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_OCI_DATABASE

```
static final String CONNECTION_PROPERTY_OCI_DATABASE
```

When `CONNECTION_PROPERTY_PASSWORD_AUTHENTICATION` is set to "OCI_TOKEN", this property specifies the Oracle Cloud ID (OCID) of the database that JDBC requests access to. If this property is not set, then Oracle JDBC requests access to all databases within the compartment specified by `CONNECTION_PROPERTY_OCI_COMPARTMENT`.

If an Oracle Net Descriptor style URL includes the OCI_DATABASE parameter, then the value of that parameter takes precedence over a value defined by this property.

Since:

23

See Also:

Constant Field Values

CONNECTION_PROPERTY_OCI_DATABASE_DEFAULT

```
static final String CONNECTION_PROPERTY_OCI_DATABASE_DEFAULT
```

CONNECTION_PROPERTY_OCI_DATABASE_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_OCI_DATABASE_ACCESSMODE
```

See Also:

Constant Field Values

CONNECTION_PROPERTY_ACCESS_TOKEN

```
static final String CONNECTION_PROPERTY_ACCESS_TOKEN
```

This property configures an access token that Oracle JDBC uses for authentication with Oracle Database. An access token configured with this property will only be used if neither a user name nor password has been provided when creating a new connection. If a user name or a password is provided, via connection properties, programmatic APIs, or by any other means, then the value of this property is ignored.

If a value is configured for this property, then Oracle JDBC will ignore any token that is specified by `CONNECTION_PROPERTY_TOKEN_AUTHENTICATION` and `CONNECTION_PROPERTY_TOKEN_LOCATION`.

The value of this property must be a JSON Web Token (JWT). The JWT format is specified by RFC 7519.

The database instance that Oracle JDBC connects to must be configured to validate the token with the service that issued it. The Oracle Database Security Guide specifies how to configure the database for token based authentication.

On systems where access tokens are stored in environment variables, it may be useful to note that an ojdbc.properties file can include expressions that resolve to the value of an environment variable. For example, this line in ojdbc.properties would configure this property as the value of an environment variable named "DATABASE_ACCESS_TOKEN":

```
    oracle.jdbc.accessToken=${DATABASE_ACCESS_TOKEN}
```

Since:

23

See Also:

Constant Field Values

CONNECTION_PROPERTY_ACCESS_TOKEN_DEFAULT

CONNECTION_PROPERTY_ACCESS_TOKEN_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_ACCESS_TOKEN_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_PASSWORD

```
static final String CONNECTION_PROPERTY_PASSWORD
```

The value of this property is used as the password when connecting to the database.

See Also:
Constant Field Values

CONNECTION_PROPERTY_PASSWORD_DEFAULT

```
static final String CONNECTION_PROPERTY_PASSWORD_DEFAULT
```

CONNECTION_PROPERTY_PASSWORD_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_PASSWORD_ACCESSMODE
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_SERVER

```
static final String CONNECTION_PROPERTY_SERVER
```

See Also:
Constant Field Values

CONNECTION_PROPERTY_SERVER_DEFAULT

```
static final String CONNECTION_PROPERTY_SERVER_DEFAULT
```

CONNECTION_PROPERTY_SERVER_ACCESSMODE

```
static final byte CONNECTION_PROPERTY_SERVER_ACCESSMODE
```

See Also:
Constant Field Values

DATABASE_OK

```
static final int DATABASE_OK
```

Define return values for pingDatabase api The physical database connection is not closed and the database is reachable. SQL requests my succeed.

See Also:
Constant Field Values

DATABASE_CLOSED

```
static final int DATABASE_CLOSED
```

Define return values for pingDatabase api The physical database connection is closed. SQL requests will fail.

See Also:
Constant Field Values

DATABASE_NOTOK

```
static final int DATABASE_NOTOK
```

Define return values for pingDatabase api The physical database connection is not closed but the database is not reachable. SQL requests will fail.

See Also:

DATABASE_TIMEOUT

```
static final int DATABASE_TIMEOUT
```

Define return values for pingDatabase api The call timed out before any positive or negative acknowledgement was received. SQL requests may or may not succeed.

See Also:
Constant Field Values

INVALID_CONNECTION

```
static final int INVALID_CONNECTION
```

Values used for close(int). The connection is no longer useable.

See Also:
Constant Field Values

PROXY_SESSION

```
static final int PROXY_SESSION
```

Values used for close(int). Close the proxy session, not the entire connection

See Also:
Constant Field Values

ABANDONED_CONNECTION_CALLBACK

```
static final int ABANDONED_CONNECTION_CALLBACK
```

See Also:
Constant Field Values

RELEASE_CONNECTION_CALLBACK

```
static final int RELEASE_CONNECTION_CALLBACK
```

See Also:
Constant Field Values

ALL_CONNECTION_CALLBACKS

```
static final int ALL_CONNECTION_CALLBACKS
```

See Also:
Constant Field Values

CONNECTION_RELEASE_LOCKED

```
static final int CONNECTION_RELEASE_LOCKED
```

See Also:
Constant Field Values

CONNECTION_RELEASE_LOW

```
static final int CONNECTION_RELEASE_LOW
```

See Also:
Constant Field Values

CONNECTION_RELEASE_HIGH

```
static final int CONNECTION_RELEASE_HIGH
```

See Also:
Constant Field Values

Constant Field Values

PROXYTYPE_DISTINGUISHED_NAME

```
static final int PROXYTYPE_DISTINGUISHED_NAME
```

See Also:
Constant Field Values

PROXYTYPE_CERTIFICATE

```
static final int PROXYTYPE_CERTIFICATE
```

See Also:
Constant Field Values

PROXY_TYPE

```
static final String PROXY_TYPE
```

See Also:
Constant Field Values

PROXY_USER_NAME

```
static final String PROXY_USER_NAME
```

See Also:
Constant Field Values

PROXY_USER_PASSWORD

```
static final String PROXY_USER_PASSWORD
```

See Also:
Constant Field Values

PROXY_DISTINGUISHED_NAME

```
static final String PROXY_DISTINGUISHED_NAME
```

See Also:
Constant Field Values

PROXY_CERTIFICATE

```
static final String PROXY_CERTIFICATE
```

See Also:
Constant Field Values

PROXY_ROLES

```
static final String PROXY_ROLES
```

See Also:
Constant Field Values

CLIENT_INFO_KEY_SEPARATOR

```
static final String CLIENT_INFO_KEY_SEPARATOR
```

Separate the namespace from the key the name of a client info. All Oracle client info names are of the form <namespace>.<key>.

See Also:
Constant Field Values

Special namespace for sending end to end metrics. There are two namespaces for sending end to end metrics values. This one uses the OCSID mechanism to send the values as opposed to the mechanism used for all other client info values.

See Also:
Constant Field Values

OCSID_ACTION_KEY

```
static final String OCSID_ACTION_KEY
```

See Also:
Constant Field Values

OCSID_CLIENTID_KEY

```
static final String OCSID_CLIENTID_KEY
```

See Also:
Constant Field Values

OCSID_ECID_KEY

```
static final String OCSID_ECID_KEY
```

See Also:
Constant Field Values

OCSID_MODULE_KEY

```
static final String OCSID_MODULE_KEY
```

See Also:
Constant Field Values

OCSID_DBOP_KEY

```
static final String OCSID_DBOP_KEY
```

See Also:
Constant Field Values

OCSID_SEQUENCE_NUMBER_KEY

```
static final String OCSID_SEQUENCE_NUMBER_KEY
```

See Also:
Constant Field Values

OCSID_CLIENT_INFO_KEY

```
static final String OCSID_CLIENT_INFO_KEY
```

See Also:
Constant Field Values

END_TO_END_ACTION_INDEX

```
static final int END_TO_END_ACTION_INDEX
```

See Also:
Constant Field Values

END_TO_END_CLIENTID_INDEX

```
static final int END_TO_END_CLIENTID_INDEX
```

See Also:
Constant Field Values

See Also:
Constant Field Values

END_TO_END_MODULE_INDEX

```
static final int END_TO_END_MODULE_INDEX
```

See Also:
Constant Field Values

END_TO_END_STATE_INDEX_MAX

```
static final int END_TO_END_STATE_INDEX_MAX
```

See Also:
Constant Field Values

NETWORK_COMPRESSION_OFF

```
static final String NETWORK_COMPRESSION_OFF
```

See Also:
Constant Field Values

NETWORK_COMPRESSION_ON

```
static final String NETWORK_COMPRESSION_ON
```

See Also:
Constant Field Values

NETWORK_COMPRESSION_AUTO

```
static final String NETWORK_COMPRESSION_AUTO
```

See Also:
Constant Field Values

NETWORK_COMPRESSION_LEVEL_LOW

```
static final String NETWORK_COMPRESSION_LEVEL_LOW
```

See Also:
Constant Field Values

NETWORK_COMPRESSION_LEVEL_LOW_VALUE

```
static final int NETWORK_COMPRESSION_LEVEL_LOW_VALUE
```

See Also:
Constant Field Values

NETWORK_COMPRESSION_LEVEL_HIGH

```
static final String NETWORK_COMPRESSION_LEVEL_HIGH
```

See Also:
Constant Field Values

NETWORK_COMPRESSION_LEVEL_HIGH_VALUE

```
static final int NETWORK_COMPRESSION_LEVEL_HIGH_VALUE
```

See Also:
Constant Field Values

See Also:
`CONNECTION_PROPERTY_NETWORK_COMPRESSION_THRESHOLD`, Constant Field Values

CACHE_SIZE_NOT_SET

```
static final int CACHE_SIZE_NOT_SET
```

See Also:
Constant Field Values

NTF_TIMEOUT

```
static final String NTF_TIMEOUT
```

See Also:
Constant Field Values

NTF_QOS_PURGE_ON_NTFN

```
static final String NTF_QOS_PURGE_ON_NTFN
```

See Also:
Constant Field Values

NTF_QOS_RELIABLE

```
static final String NTF_QOS_RELIABLE
```

See Also:
Constant Field Values

NTF_QOS_SECURE

```
static final String NTF_QOS_SECURE
```

See Also:
Constant Field Values

NTF_ASYNC_DEQ

```
static final String NTF_ASYNC_DEQ
```

See Also:
Constant Field Values

NTF_AQ_PAYLOAD

```
static final String NTF_AQ_PAYLOAD
```

See Also:
Constant Field Values

NTF_USE_SSL

```
static final String NTF_USE_SSL
```

See Also:
Constant Field Values

NTF_QOS_TX_ACK

```
static final String NTF_QOS_TX_ACK
```

See Also:
Constant Field Values

Constant Field Values

NTF_LOCAL_TCP_PORT

```
static final String NTF_LOCAL_TCP_PORT
```

See Also:
Constant Field Values

NTF_DEFAULT_TCP_PORT

```
static final int NTF_DEFAULT_TCP_PORT
```

See Also:
Constant Field Values

NTF_LOCAL_HOST

```
static final String NTF_LOCAL_HOST
```

See Also:
Constant Field Values

NTF_GROUPING_CLASS

```
static final String NTF_GROUPING_CLASS
```

See Also:
Constant Field Values

NTF_GROUPING_CLASS_NONE

```
static final String NTF_GROUPING_CLASS_NONE
```

See Also:
Constant Field Values

NTF_GROUPING_CLASS_TIME

```
static final String NTF_GROUPING_CLASS_TIME
```

See Also:
Constant Field Values

NTF_GROUPING_VALUE

```
static final String NTF_GROUPING_VALUE
```

See Also:
Constant Field Values

NTF_GROUPING_TYPE

```
static final String NTF_GROUPING_TYPE
```

See Also:
Constant Field Values

NTF_GROUPING_TYPE_SUMMARY

```
static final String NTF_GROUPING_TYPE_SUMMARY
```

See Also:
Constant Field Values

Constant Field Values

NTF_GROUPING_START_TIME

```
static final String NTF_GROUPING_START_TIME
```

See Also:
Constant Field Values

NTF_GROUPING_REPEAT_TIME

```
static final String NTF_GROUPING_REPEAT_TIME
```

See Also:
Constant Field Values

NTF_GROUPING_REPEAT_FOREVER

```
static final String NTF_GROUPING_REPEAT_FOREVER
```

See Also:
Constant Field Values

DCN_NOTIFY_ROWIDS

```
static final String DCN_NOTIFY_ROWIDS
```

See Also:
Constant Field Values

DCN_IGNORE_INSERTOP

```
static final String DCN_IGNORE_INSERTOP
```

See Also:
Constant Field Values

DCN_IGNORE_UPDATEOP

```
static final String DCN_IGNORE_UPDATEOP
```

See Also:
Constant Field Values

DCN_IGNORE_DELETEOP

```
static final String DCN_IGNORE_DELETEOP
```

See Also:
Constant Field Values

DCN_NOTIFY_CHANGELAG

```
static final String DCN_NOTIFY_CHANGELAG
```

See Also:
Constant Field Values

DCN_QUERY_CHANGE_NOTIFICATION

```
static final String DCN_QUERY_CHANGE_NOTIFICATION
```

See Also:
Constant Field Values

Constant Field Values

DCN_CLIENT_INIT_CONNECTION

```
static final String DCN_CLIENT_INIT_CONNECTION
```

Set the value of DCN_CLIENT_INIT_CONNECTION to 'true' for using the Client initiated DCN connection. By default the value is 'false' and the DCN Connection is initiated by the Server.

See Also:

Constant Field Values

DCN_USE_HOST_CONNECTION_ADDR_INFO

```
static final String DCN_USE_HOST_CONNECTION_ADDR_INFO
```

Set the value of DCN_USE_HOST_CONNECTION_ADDR_INFO to 'false' to use the address info returned by the server for establishing the client initiated DCN Connection. Default value is 'true' and Host connection's connection string is used for establishing the Client initiated DCN connection.

See Also:

Constant Field Values

AQ_USE_HOST_CONNECTION_ADDR_INFO

```
static final String AQ_USE_HOST_CONNECTION_ADDR_INFO
```

Set the value of AQ_USE_HOST_CONNECTION_ADDR_INFO to 'false' to use the address info returned by the server for establishing the client initiated Connection for JMS Message Listener . Default value is 'true' and Host connection's connection string is used for establishing the Client initiated JMS Messsage Listener connection.

See Also:

Constant Field Values

Method Detail

commit

```
void commit(EnumSet<OracleConnection.CommitOption> options) throws SQLException
```

Commits the transaction with the given options.

Parameters:

`flags` - commit options

Throws:

`SQLException`

archive

```
void archive(int mode, int aseq, String acstext) throws SQLException
```

Deprecated.
This method will be removed in a future version.

Not implemented.

Throws:

`SQLException`

openProxySession

```
void openProxySession(int type, Properties prop) throws SQLException
```

Opens a new proxy session with the username provided in the prop argument and switches to this new session.

This feature is supported for both thin and oci driver.

Three proxy types are supported :

- OracleConnection.PROXYTYPE_USER_NAME : In this type PROXY_USER_NAME needs to be provided in prop. The value should be a java.lang.String;
- OracleConnection.PROXYTYPE_DISTINGUISHED_NAME : In this type PROXY_DISTINGUISHED_NAME has to be set in prop. The value is a java.lang.String object;
- OracleConnection.PROXYTYPE_CERTIFICATE : In this type PROXY_CERTIFICATE has to be set in prop. The value is a bytep[] which contains the certificate.

Roles can also be provided in the property argument. The key is OracleConnection.PROXY_ROLES. The value is a String[] which contains the roles.

SQLException

getAutoClose

```
boolean getAutoClose() throws SQLException
```

The driver is always in auto-close mode.

Returns:

should always return `true`

Throws:

`SQLException` - should never been raised

See Also:

`setAutoClose`

getDefaultExecuteBatch

```
int getDefaultExecuteBatch()
```

Deprecated.
As of 12.1 all APIs related to oracle-style statement batching are deprecated in favor of standard JDBC batching. We recommend using the standard model going forward as it is spec compliant and provides more information and control to the application.

Executions are not batched but sent immediately. Oracle style of batching has been deprecated in 12.1 and made a no-op in 12.2. We recommend using the standard model of batching.

Returns:

the batch value, always 1.

See Also:

`OraclePreparedStatement.setExecuteBatch`, `setDefaultExecuteBatch`

getDefaultRowPrefetch

```
int getDefaultRowPrefetch()
```

Retrieves the value of row prefetch for all statements associated with this connection and created after this value was set.

The row-prefetching feature associates an integer row-prefetch setting with a given statement object. JDBC fetches that number of rows at a time from the database during the query. That is, JDBC will fetch N rows that match the query criteria and bring them all back to the client at once, where N is the prefetch setting. Then, once your `next` calls have run through those N rows, JDBC will go back to fetch the next N rows that match the criteria.

You can set the number of rows to prefetch for a particular Oracle statement (any type of statement). You can also reset the default number of rows that will be prefetched for all statements in your connection with the `setDefaultRowPrefetch` method. Therefore, the row prefetch value returned by this `getDefaultRowPrefetch` entrypoint is valid for statements for which you have not defined a different row prefetch value.

The default number of rows to prefetch to the client is 10.

Example where `conn` is your connection object:

```
//Get the default row-prefetch setting for this connection
int defRowPref = ((OracleConnection)conn).getDefaultRowPrefetch();
```

Returns:

the row prefetch value

See Also:

`OracleStatement.setRowPrefetch`, `setDefaultRowPrefetch`

getDescriptor

```
Object getDescriptor(String sql_name)
```

Gets a Descriptor object corresponding to a sql type.

Parameters:

`sql_name` - the sql type

Returns:

the Descriptor Object that matches the sql type

See Also:

`putDescriptor`, `oracle.sql.TypeDescriptor`

getEndToEndMetrics

```
String[] getEndToEndMetrics() throws SQLException
```

return the metrics set via DMS, not those set via setEndToEndMetrics. The DMS metric override the metrics set by setEndToEndMetrics.

Returns:

a String[]. The indices are the END_TO_END_XXX_INDEX constants. The values are the values of the corresponding metrics.

Throws:

`SQLException` - if an error occurs

See Also:

`setEndToEndMetrics`, `Connection.getClientInfo(java.lang.String)`, `Connection.getClientInfo()`

getEndToEndECIDSequenceNumber

`short getEndToEndECIDSequenceNumber() throws SQLException`

Deprecated.
This is deprecated since 12.1 in favor of getClientInfo(). It is not recommended to use this API intermingled with get/setClientInfo APIs.

Gets the current end to end tracing context id sequence number. This could be any of the following values: the value passed in the most recent call to setEndToEndMetrics the value returned by the database after the most recent statement execution the value incremented by JDBC diagnostic messages the value JDBC retrieved from DMS (only in a DMS environment)

Returns:

the current ECID sequence number

Throws:

`SQLException` - if an error occurs

See Also:

`Connection.getClientInfo(java.lang.String)`, `Connection.getClientInfo()`

getIncludeSynonyms

`boolean getIncludeSynonyms()`

Checks whether or not synonyms information is included in `DatabaseMetaData.getColumns`. By default and for performance reasons it won't but you can change this with the `setIncludeSynonyms` method.

Returns:

`true` if `DatabaseMetaData.getColumns` will report information if a table synonym is passed in, and `false` otherwise

See Also:

`setIncludeSynonyms`

getRestrictGetTables

`boolean getRestrictGetTables()`

Gets the restriction status of the returned data in `DatabaseMetaData.getTables`.

The default behavior is to return information about all synonyms, including those which do not point to accessible tables or views. But you can change this with the `setRestrictGetTables` method.

Returns:

`true` if the information returned by `DatabaseMetaData.getTables` is restricted, and `false` otherwise

See Also:

`setRestrictGetTables`

getJavaObject

`Object getJavaObject(String sql_name) throws SQLException`

Deprecated.

Throws:

`SQLException`

By default and for performance reasons it won't (it will return `null`) but you can change this with the `setRemarksReporting` method.

Returns:

`true` if the `DatabaseMetaData` calls `getTables` and `getColumns` will report the REMARKS column and `false` otherwise

See Also:

`setRemarksReporting`

getSQLType

`String getSQLType(Object obj) throws SQLException`

Deprecated.

Throws:

`SQLException`

getStmtCacheSize

`int getStmtCacheSize()`

Deprecated.
Use getStatementCacheSize() instead.

getStructAttrCsId

`short getStructAttrCsId() throws SQLException`

Obtain the Oracle identifier of the character set used in STRUCT attributes. Note that the network transport layer always send structure attributes in the database character set.

Returns:

the Oracle identifier of the character set.

Throws:

`SQLException` - if Conversion is `null`

See Also:

oracle.sql.CharacterSet for the set of constants defined for the identifiers."

getUserName

`String getUserName() throws SQLException`

Gets the user name of the current connection.

Example where `conn` is your connection object:
`String UserName = ((OracleConnection)conn).getUserName();`

Returns:

the user name

Throws:

`SQLException` - if the logical connection is closed

getCurrentSchema

`String getCurrentSchema() throws SQLException`

Obtains the current schema of the current connection.

Returns:

current_schema value

Throws:

`SQLException` - If there was an error while fetching the results

getUsingXAFlag

`boolean getUsingXAFlag()`

Deprecated.

Gets the value of the `UsingXA` flag which the driver sets to `true` when using XA to manage distributed transactions. If you are not using distributed transactions with the XA library, the value of the `UsingXA` flag will be `false`.

getXAErrorFlag

`boolean getXAErrorFlag()`

Deprecated.

Gets the value of the `XAError` flag which is used with distributed transactions.

When using distributed transactions with an XA library, you can ask the driver to raise exception when doing anything that might require a transaction. To do so, set the value of the `XAError` flag to `true` with the method `setXAErrorFlag`.

The default value is `false`.

Returns:

`false` is the normal JDBC usage. `true` means that the driver will raise an exception when doing anything that might require a transaction.

See Also:

`setXAErrorFlag`

pingDatabase

`int pingDatabase() throws SQLException`

Ping Database server to see if both database and the connection are actively up.

Returns:

DATABASE_OK if the database server is up, and DATABASE_CLOSED if any error occurs.

Throws:

`SQLException`

pingDatabase

`int pingDatabase(int timeOut) throws SQLException`

Deprecated.

ping Database

Parameters:

`timeOut` -

Returns:

Throws:

`SQLException`

putDescriptor

`void putDescriptor(String sql_name, Object desc) throws SQLException`

Store the Object Descriptor for later usage.

Parameters:

`sql_name` - the sql type

`desc` - the Object Descriptor associated

Throws:

`SQLException` - if sql_name or desc is `null`

See Also:

`getDescriptor`, `oracle.sql.TypeDescriptor`

registerSQLType

`void registerSQLType(String sql_name, Class<?> java_class) throws SQLException`

Deprecated.

Throws:

`SQLException`

registerSQLType

`void registerSQLType(String sql_name, String java_class_name) throws SQLException`

SQLException

setAutoClose

`void setAutoClose(boolean autoClose) throws SQLException`

set auto-close mode. Only true is accepted.

Parameters:

`autoClose` - the boolean value

Throws:

`SQLException` - when the argument autoClose is `false`

See Also:

`getAutoClose`

setDefaultExecuteBatch

`void setDefaultExecuteBatch(int batch) throws SQLException`

Deprecated.
As of 12.1 all APIs related to oracle-style statement batching are deprecated in favor of standard JDBC batching. We recommend using the standard model going forward as it is spec compliant and provides more information and control to the application.

Executions are not batched but sent immediately. Oracle style of batching has been deprecated in 12.1 and made a no-op in 12.2. We recommend using the standard model of batching.

Parameters:

`batch` - value is discarded.

Throws:

`SQLException` - never thrown.

See Also:

`OraclePreparedStatement.setExecuteBatch`, `getDefaultExecuteBatch`

setDefaultRowPrefetch

`void setDefaultRowPrefetch(int value) throws SQLException`

Sets the value of row prefetch for all statements associated with this connection and created after this value was set.

The row-prefetching feature associates an integer row-prefetch setting with a given statement object. JDBC fetches that number of rows at a time from the database during the query. That is, JDBC will fetch N rows that match the query criteria and bring them all back to the client at once, where N is the prefetch setting. Then, once your `next` calls have run through those N rows, JDBC will go back to fetch the next N rows that match the criteria.

You can set the number of rows to prefetch for a particular Oracle statement (any type of statement) but this method allows you to reset the default number of rows that will be prefetched for all statements in your connection. The default number of rows to prefetch to the client is 10.

Use the `setDefaultRowPrefetch` method to set the default number of rows to prefetch, passing in an integer that specifies the desired default. If you want to check the current setting of the default, then use the `getDefaultRowPrefetch` method. This method returns an integer.

Example where `conn` is your connection object:
```
//Set the default row-prefetch setting for this connection to 7
((OracleConnection)conn).setDefaultRowPrefetch(7);
```

Note 1 : A statement object receives the default row-prefetch setting from the associated connection at the time the statement object is created. Subsequent changes to the connection's default row-prefetch setting have no effect on the statement's row-prefetch setting.

Note 2 : If a column of a result set is of datatype LONG or LONG RAW (that is, the streaming types), JDBC changes the statement's row-prefetch setting to 1, even if you never actually read a value of either of those types.

Note 3 : Do not mix the JDBC 2.0 fetch size API and the Oracle row-prefetching API in your application. You can use one or the other but not both.

Parameters:

`value` - the number of rows to prefetch

Throws:

`SQLException` - if the argument `value` is <=0

See Also:

`OracleStatement.setRowPrefetch`, `getDefaultRowPrefetch`

setEndToEndMetrics

`void setEndToEndMetrics(String[] metrics, short sequenceNumber) throws SQLException`

Deprecated.
It has been deprecated since 12.1 in favor of setClientInfo(). It is not recommended to use this API intermingled with get/setClientInfo APIs.

Sets the values of the end-to-end tracing metrics. The indices for the array are the END_TO_END_XXX_INDEX values defined in this class. The values set by this method are overridden by any values set via DMS if DMS is in use.

Parameters:

See Also:

getEndToEndMetrics, Connection.setClientInfo(java.util.Properties), Connection.setClientInfo(java.lang.String, java.lang.String)

setIncludeSynonyms

```
void setIncludeSynonyms(boolean synonyms)
```

Turns on or off retrieval of synonym information in DatabaseMetaData. getColumns.

Similar to `setRemarksReporting`, `getColumns` performs extremely slow if information about synonyms has to be included, because it neccessitates an outer join so, by default, the JDBC driver will not report information about synonyms.

You can get synonym information by passing true to this method, and turn it off by passing false. You can also control this behavior by passing a property named "includeSynonyms" as "true" to DriverManager.getConnection.

Parameters:

`synonyms` - `true` if you want to retrieve synonym information in DatabaseMetaData.getColumns and `false` otherwise.

See Also:

getIncludeSynonyms

setRemarksReporting

```
void setRemarksReporting(boolean reportRemarks)
```

Turns on or off the reporting of the REMARKS columns by the `getTables` and `getColumns` calls of the `DatabaseMetaData` interface.

The `DatabaseMetaData` calls `getTables` and `getColumns` are extremely slow if the REMARKS column has to be reported as this necessitates an expensive outer join so by default the JDBC driver does not report the REMARKS columns.

You can turn the reporting of REMARKS on by passing a `true` argument to this method. You turn it back off by passing a `false` argument.

Example where `conn` is your connection object:
`((OracleConnection)conn).setRemarksReporting(true);`

You can also control the reporting of REMARKS by passing a property named `remarksReporting` as `true` to the `DriverManager.getConnection` call.

Parameters:

`reportRemarks` - `true` if you want to turn on the reporting of the REMARKS columns and `false` otherwise.

See Also:

getRemarksReporting

setRestrictGetTables

```
void setRestrictGetTables(boolean restrict)
```

Turns on or off the restriction of the returned data in `DatabaseMetaData.getTables.`

`DatabaseMetaData.getTables` will return information about all accessible tables, views, and synonyms. There are two issues relating to synonyms which can affect the quality of the returned data:

1. Public synonyms can exist for tables to which you don't have access. Although the synonym itself is viewable, the underlying table is not.
2. Synonyms can exist for non-table objects, such as procedures, sequences, Java classes, etc.

As a result of the above issues, `getTables` can return rows containing objects that are not describable with `getColumns`, either because they are not accessible (issue 1) or because they are not tables or views (issue 2).

To remedy this, you can restrict the results of `getTables` to only those tables and views to which you have access. This is done by either passing `true` to this method, or by passing the `restrictGetTables` property as `true` to the `DriverManager.getConnection` call. The default behavior is to return information about all synonyms, including those which do not point to accessible tables or views.

Note that `getTables` can return more than one row for the same object, one for the object itself, and additional rows for any synonyms defined for that object. This is the case regardless of the setting for `restrictGetTables.`

The following code turns on the restriction:
`((OracleConnection)conn).setRestrictGetTables(true);`

Parameters:

`restrict` - `true` to turn on the restriction and `false` otherwise.

See Also:

getRestrictGetTables

setStmtCacheSize

```
void setStmtCacheSize(int size) throws SQLException
```

Deprecated.
Use setStatementCacheSize() instead.

Throws:

SQLException

Deprecated.
Use setStatementCacheSize() instead.

Throws:
`SQLException`

setStatementCacheSize

```
void setStatementCacheSize(int size) throws SQLException
```

setStatementCacheSize Specifies the size of the size of the application cache (which will be used by both implicit and explicit caching).

Parameters:
`size` - Requested size of the cache. If the existing cache size is less than size, statements will be purged to reduce the size.

Throws:
`SQLException` - if size < 0, or if called on a logical connection.

getStatementCacheSize

```
int getStatementCacheSize() throws SQLException
```

getStatementCacheSize Returns the current size of the application cache. This is valid on both physical and logical connections. If the statement cache has not been initialized with setStatementCacheSize(), then CACHE_SIZE_NOT_SET is returned.

Returns:
the cache size

Throws:
`SQLException`

setImplicitCachingEnabled

```
void setImplicitCachingEnabled(boolean cache) throws SQLException
```

setImplicitCachingEnabled Enables or disables the implicit cache. Note that this is independent of the cache size, set with setStatmentCacheSize().

Parameters:
`cache` - If true, then implicit caching will be enabled. If false, then any existing statements will be purged and the implicit cache will be disabled.

Throws:
`SQLException` - if called on a logical connection.

getImplicitCachingEnabled

```
boolean getImplicitCachingEnabled() throws SQLException
```

getImplicitCachingEnabled Returns true if the implicit cache is currently enabled, false otherwise. This method is valid on both logical and physical connections.

Returns:

Throws:
`SQLException`

setExplicitCachingEnabled

```
void setExplicitCachingEnabled(boolean cache) throws SQLException
```

setExplicitCachingEnabled Enables or disables the explicit cache. Note that this is independent of the cache size, set with setStatmentCacheSize().

Parameters:
`cache` - If true, then explicit caching will be enabled. If false, then any existing statements will be purged and the explicit cache will be disabled.

Throws:
`SQLException` - if called on a logical connection.

getExplicitCachingEnabled

```
boolean getExplicitCachingEnabled() throws SQLException
```

getExplicitCachingEnabled Returns true if the explicit cache is currently enabled, false otherwise. This method is valid on both logical and physical connections.

Returns:

Throws:
`SQLException`

cache, nor the enabled/disabled status.

Throws:

SQLException

purgeExplicitCache

`void purgeExplicitCache() throws SQLException`

purgeExplicitCache Removes all existing statements from the explicit cache, after which it will be empty. This method does not affect the size of the application cache, nor the enabled/disabled status.

Throws:

SQLException

getStatementWithKey

`PreparedStatement getStatementWithKey(String key) throws SQLException`

getStatementWithKey Searches the explicit cache for a match on key. If found, the statement is returned, with the paramater and define metadata identical to the last usage. If no match is found, or if explicit caching is not enabled, then null is returned (as opposed to throwing an exception).

Parameters:

key - Specified key to search for

Returns:

Throws:

SQLException

getCallWithKey

`CallableStatement getCallWithKey(String key) throws SQLException`

getCallWithKey Searches the explicit cache for a match on key. If found, the statement is returned, with the paramater and define metadata identical to the last usage. If no match is found, or if explicit caching is not enabled, then null is returned (as opposed to throwing an exception).

Parameters:

key - Specified key to search for

Returns:

Throws:

SQLException

setUsingXAFlag

`void setUsingXAFlag(boolean value)`

Deprecated.

When using distributed transactions with XA, you can set the value of the `UsingXA` flag.

XA is a general standard (not specific to Java) for distributed transactions. You should use this method only when using XA.

By default, when using distributed transactions with XA, the driver will set the `UsingXA` flag to `true` and exceptions will be raised when you want to do anything with your logical connection that might require a transaction. Otherwise the flag `UsingXA` is always `false`.

If you are actually using distributed transactions with XA and you dislike the default behavior, you can set the flag back to `false`.

Parameters:

`value` - the value of the `UsingXA` flag

See Also:

`getUsingXAFlag`

setXAErrorFlag

`void setXAErrorFlag(boolean value)`

Deprecated.

Sets the value of the `XAError` flag which is used with distributed transactions. When coexisting with an XA library, you can set the `XAError` flag to `true` and the driver will then raise an exception when doing anything that might require a transaction.

Parameters:

`value` - the value of the `XAError` flag

See Also:

`getXAErrorFlag`

Parameters:

`mode` - can be either

- `OracleConnection.DatabaseShutdownMode.CONNECT`
- `OracleConnection.DatabaseShutdownMode.TRANSACTIONAL`
- `OracleConnection.DatabaseShutdownMode.TRANSACTIONAL_LOCAL`
- `OracleConnection.DatabaseShutdownMode.IMMEDIATE`
- `OracleConnection.DatabaseShutdownMode.ABORT`
- `OracleConnection.DatabaseShutdownMode.FINAL`

Throws:

`SQLException`

startup

```
void startup(String startup_str, int mode) throws SQLException
```

Deprecated.
This method will be removed in a future version.

Not implemented

Throws:

`SQLException`

startup

```
void startup(OracleConnection.DatabaseStartupMode mode) throws SQLException
```

Starts the database server up. This method requires to be connected as either SYSOPER or SYSDBA in the PRELIM_AUTH mode which is the only mode permietted when the database is down (see the connection property CONNECTION_PROPERTY_PRELIM_AUTH).

Parameters:

`mode` - can be either

- `OracleConnection.DatabaseStartupMode.NO_RESTRICTION`
- `OracleConnection.DatabaseStartupMode.FORCE`
- `OracleConnection.DatabaseStartupMode.RESTRICT`

Throws:

`SQLException`

startup

```
void startup(OracleConnection.DatabaseStartupMode mode, String pfileName) throws SQLException
```

Starts the database server up. This method requires to be connected as either SYSOPER or SYSDBA in the PRELIM_AUTH mode which is the only mode permietted when the database is down (see the connection property CONNECTION_PROPERTY_PRELIM_AUTH).

Parameters:

`mode` - can be either

- `OracleConnection.DatabaseStartupMode.NO_RESTRICTION`
- `OracleConnection.DatabaseStartupMode.FORCE`
- `OracleConnection.DatabaseStartupMode.RESTRICT`

`pfileName` - : PFILE name. If client-side parameter file is null or doesn't exist, it will throw exception otherwise read the file and pass parameters to server.

Throws:

`SQLException`

prepareStatementWithKey

```
PreparedStatement prepareStatementWithKey(String key) throws SQLException
```

Deprecated.
This is same as prepareStatement, except if a Prepared Statement with the given KEY exists in the Cache, then the statement is returned AS IT IS when it was closed and cached with this KEY. An object returned from the Cache based on Key will have its state set to "KEYED". If no such Prepared Statement is found, a null is returned. Key cannot be null.

Parameters:

`key` - the key with which it was closed

Returns:

a OraclePreparedStatement object

Throws:

`SQLException` - if a database access error occurs

prepareCallWithKey

and cached with this KEY. An object returned from the Cache based on Key will have its state set to "KEYED". If no such Callable Statement is found, then null is returned. Key cannot be null.

Parameters:

`key` - the key with which it was closed

Returns:

a java.sql.CallableStatement object

Throws:

`SQLException` - if a database access error occurs

setCreateStatementAsRefCursor

```
void setCreateStatementAsRefCursor(boolean value)
```

When this is set to `true`, any new statements created from this connection will be created as a `REF CURSOR`. Only resultsets obtained from statements that are created as `REF CURSORS` can be returned from a Java Stored Procedure. This feature is supported by the server-side internal driver only, and is no-op in all other JDBC drivers.

Default value is `false`.

To use the `setCreateStatementAsRefCursor` entrypoint you have to cast the Connection object to the type `oracle.jdbc.OracleConnection`.

Parameters:

`value` - `true` if new statements should be created as `REF CURSORS`, `false` otherwise

See Also:

`getCreateStatementAsRefCursor`

getCreateStatementAsRefCursor

```
boolean getCreateStatementAsRefCursor()
```

Retrieves the current setting of the `createStatementAsRefCursor` flag which you can set with the `setCreateStatementAsRefCursor` method.

To use the `getCreateStatementAsRefCursor` entrypoint you have to cast the Connection object to the type `oracle.jdbc.OracleConnection`.

Returns:

the current setting of the createStatementAsRefCursor flag

See Also:

`setCreateStatementAsRefCursor`

setSessionTimeZone

```
void setSessionTimeZone(String regionName) throws SQLException
```

Set the session time zone.

This method is used to set the session time zone. This method must be invoked before accessing any TIMESTAMP WITH LOCAL TIME ZONE data. Upon invocation of this method, the Jdbc driver sets the session timezone of the connection and saves the session timezone so that any TSLTZ data accessed via Jdbc are adjusted using the session timezone.

Parameters:

`regionName` - Oracle session time zone region name.

Throws:

`SQLException` - if an error occurred.

Since:

9i

getSessionTimeZone

```
String getSessionTimeZone()
```

Obtain Oracle session time zone region name.

Returns:

Oracle session time zone region name.

Since:

9i

getSessionTimeZoneOffset

```
String getSessionTimeZoneOffset() throws SQLException
```

Obtain the time zone offset in hours of the current database session. The result will always be accurate. In other words, you can execute "`ALTER SESSION SET TIME_ZONE ...`" and then call this method, it will return the new value.

The value returned by this method is that same as the result of "`SELECT SESSIONTIMEZONE FROM DUAL;`". The drivers may use some performance optimization to

Throws:

`SQLException`

Since:

11.1

getProperties

`Properties getProperties()`

Determines the connection properties.

Returns:

_getPC

`Connection _getPC()`

Return the underlying physical connection if this is a logical connection. Returns null otherwise.

Returns:

Connection object if its a logical handle otherwise returns null

isLogicalConnection

`boolean isLogicalConnection()`

Method that returns a boolean indicating whether its a logical connection or not.

Returns:

boolean true if this is a logical connection

registerTAFCallback

`void registerTAFCallback(OracleOCIFailover cbk, Object obj) throws SQLException`

Register an application TAF Callback instance that will be called when an application failover occurs. The TAF feature is only available in the Jdbc OCI driver.

Parameters:

`cbk` - Callback instance.

`obj` - Context object in which any client's state can be stored and provided when the callback method is invoked.

Throws:

`SQLException` - if this method is invoked in drivers other than the Jdbc OCI driver.

Since:

9i

unwrap

`OracleConnection unwrap()`

Return the wrapped object if any else null. This method should not delegate to the wrapped object. Instead it should return the wrapped object.

Returns:

wrapped object which implements oracle.jdbc.OracleConnection if any else return null

Since:

9iRw

setWrapper

`void setWrapper(OracleConnection wrapper)`

Set the wrapping object. The argument is an object that wraps this object. Calling wrapper.unwrap() should return this.

Parameters:

`wrapper` - An object which implements oracle.jdbc.OracleConnection and which is a wrapper for this object # @since 9iR2

oracleSetSavepoint

`OracleSavepoint oracleSetSavepoint() throws SQLException`

Deprecated.

Creates an unnamed savepoint in the current transaction and returns the new `OracleSavepoint` object that represents it.

Returns:

9.0.2

See Also:

`OracleSavepoint`

oracleSetSavepoint

```
OracleSavepoint oracleSetSavepoint(String name) throws SQLException
```

Creates a savepoint with the given name in the current transaction and returns the new `OracleSavepoint` object that represents it.

Parameters:

`name` - a `String` containing the name of the savepoint

Returns:

the new `OracleSavepoint` object

Throws:

`SQLException` - if a database access error occurs or this `Connection` object is currently in auto-commit mode

Since:

9.0.2

See Also:

`OracleSavepoint`

oracleRollback

```
void oracleRollback(OracleSavepoint savepoint) throws SQLException
```

Undoes all changes made after the given `OracleSavepoint` object was set.

This method should be used only when auto-commit has been disabled.

Parameters:

`savepoint` - the `OracleSavepoint` object to roll back to

Throws:

`SQLException` - if a database access error occurs, the `OracleSavepoint` object is no longer valid, or this `Connection` object is currently in auto-commit mode

Since:

9.0.2

See Also:

`OracleSavepoint`

oracleReleaseSavepoint

```
void oracleReleaseSavepoint(OracleSavepoint savepoint) throws SQLException
```

Removes the given `OracleSavepoint` object from the current transaction. Any reference to the savepoint after it have been removed will cause an `SQLException` to be thrown.

Parameters:

`savepoint` - the `OracleSavepoint` object to be removed

Throws:

`SQLException` - if a database access error occurs or the given `OracleSavepoint` object is not a valid savepoint in the current transaction

Since:

9.0.2

See Also:

`OracleSavepoint`

close

```
@Deprecated default void close(Properties connAttr) throws SQLException
```

Deprecated.
The Implicit Connection Cache (ICC) has been desupported since 12.1. This method throws UnsupportedOperationException, and will be removed soon in the future.

Throws:

`SQLException`

close

```
void close(int opt) throws SQLException
```

If opt is OracleConnection.INVALID_CONNECTION : Closes the given Logical connection, as well the underlying PooledConnection without returning the connection to the cache when called with the parameter INVALID_CONNECTION. If this API is called on a physical connection, the supplied parameter has no `opt` - set to INVALID_CONNECTION to close the PooledConnection

Throws:

`SQLException` - if a database access error occurs

isProxySession

```
boolean isProxySession()
```

Returns true if the current session associated with this connection is a proxy session.

Returns:

applyConnectionAttributes

```
@Deprecated default void applyConnectionAttributes(Properties connAttr) throws SQLException
```

Deprecated.
The Implicit Connection Cache (ICC) has been desupported since 12.1. This method throws UnsupportedOperationException, and will be removed soon in the future.

Throws:

`SQLException`

getConnectionAttributes

```
@Deprecated default Properties getConnectionAttributes() throws SQLException
```

Deprecated.
The Implicit Connection Cache (ICC) has been desupported since 12.1. This method throws UnsupportedOperationException, and will be removed soon in the future.

Throws:

`SQLException`

getUnMatchedConnectionAttributes

```
@Deprecated default Properties getUnMatchedConnectionAttributes() throws SQLException
```

Deprecated.
The Implicit Connection Cache (ICC) has been desupported since 12.1. This method throws UnsupportedOperationException, and will be removed soon in the future.

Throws:

`SQLException`

registerConnectionCacheCallback

```
@Deprecated default void registerConnectionCacheCallback(OracleConnectionCacheCallback occc, Object userObj, int cbkFlag) throws
SQLException
```

Deprecated.
The Implicit Connection Cache (ICC) has been desupported since 12.1. This method throws UnsupportedOperationException, and will be removed soon in the future.

Throws:

`SQLException`

setConnectionReleasePriority

```
@Deprecated default void setConnectionReleasePriority(int priority) throws SQLException
```

Deprecated.
The Implicit Connection Cache (ICC) has been desupported since 12.1. This method throws UnsupportedOperationException, and will be removed soon in the future.

Throws:

`SQLException`

Deprecated.
The Implicit Connection Cache (ICC) has been desupported since 12.1. This method throws UnsupportedOperationException, and will be removed soon in the future.

Throws:

`SQLException`

setPlsqlWarnings

```
void setPlsqlWarnings(String setting) throws SQLException
```

Enable/Disable PLSQL Compiler Warnings

Parameters:

`setting` - Setting specified for ALTER SESSION SET PLSQL_WARNINGS. Sample values are: "'ENABLE:ALL'", "'DISABLE:ALL'", "'ENALBLE:INFORMATIONAL'", etc. Please refer to the SQL reference of ALTER SESSION SET PLSQL_WARNINGS for more information. If the setting is "'DISABLE:ALL'", jdbc drivers turn off PLSQL Compiler Warnings. Note: the quotes(') in the setting String are necessary.

Throws:

`SQLException` - if a database access error occurs

registerAQNotification

```
AQNotificationRegistration[] registerAQNotification(String[] name, Properties[] options, Properties globaloptions) throws SQLException
```

Registers your interest into being notified when a message is enqueued in a particular queue (or array of queues).

This method creates an array of new AQ registrations in the database server with the given options. It also opens a listening socket which will be used by the database to send notifications. Note that mutiple registrations can share the same listening socket.

Each registration will continue to live after this connection is closed. You need to explicitly unregister it to destroy it in the server and release the resources in the driver.

This method uses one roundtrip.

Parameters:

`name` - contains an array of queue names ("SCOTT.MY_QUEUE") for single consumer queues or queue names with the consumer name ("SCOTT.MY_QUEUE:RECEIVER") for multiple consumer queues.

`options` - Possible options are ([] means default):

- `OracleConnection.NTF_QOS_RELIABLE`: "true"/["false"]. Set this option to "true" to make the notifications persistent which comes at a performance cost.
- `OracleConnection.NTF_QOS_PURGE_ON_NTFN`: "true"/["false"]. Set this option to "true" and the registration will be expunged on the first notification event.
- `OracleConnection.NTF_TIMEOUT`: value in seconds "60"/["0"]. Specifies the time in seconds after which the registration is automatically expunged by the database. The default is "0": the registration lives until explicitly deregistered.
- `OracleConnection.NTF_AQ_PAYLOAD`: "true"/["false"]. Sets this to "true" to make the server send the payload within the notification. Note that this feature works only with "RAW" payloads.
- `OracleConnection.NTF_GROUPING_CLASS`: `OracleConnection.NTF_GROUPING_CLASS_TIME/[OracleConnection.NTF_GROUPING_CLASS_NONE]`. Notification Grouping Class, the criterion or dimension for grouping. As of 11.2 the only supported class is `OracleConnection.NTF_GROUPING_CLASS_TIME` meaning grouping by time, that is, the user specifies a time value and a single notification gets published at the end of that time. To use grouping at least this option must be specified to a value other than the default `OracleConnection.NTF_GROUPING_CLASS_NONE`, which is no grouping.
- `OracleConnection.NTF_GROUPING_VALUE`: "1200/["600"]. Notification Grouping Value, the value of the grouping class. The value must be an integer number. For the TIME grouping class, this value represents a number of seconds, meaning the time after which grouped notifications are sent. If not specified, it defaults to 600 sec.
- `OracleConnection.NTF_GROUPING_TYPE`: `OracleConnection.NTF_GROUPING_TYPE_LAST/[OracleConnection.NTF_GROUPING_TYPE_SUMMARY]`. Notification Grouping Type, the format of grouping notification. It can either contain the summary of all events (default) or the last event in the group.
- `OracleConnection.NTF_GROUPING_START_TIME`: When to start grouping? Notification grouping can start from a user-specified time that should a valid timestamp with time zone, that is an instance of `oracle.sql.TIMESTAMPTZ`. If this option is not specified when using grouping, it defaults to current system time. For example if `prop` was the option properties, and `conn` the connection object, you would call: `prop.put(OracleConnection.NTF_GROUPING_START_TIME,new TIMESTAMPTZ(conn,"2007-06-21 10:10:00.0"));`.
- `OracleConnection.NTF_GROUPING_REPEAT_TIME`: "100/[NTF_GROUPING_REPEAT_FOREVER]". How many times do grouping? Grouping notifications will be sent as many times as specified by the notification grouping repeat count and after that revert to regular notifications. If not specified, it will default to: `NTF_GROUPING_REPEAT_FOREVER` - keep sending grouping notifications forever.

`globaloptions` - Possible options are ([] means default):

- `OracleConnection.NTF_LOCAL_TCP_PORT`: "1234"/[NTF_DEFAULT_TCP_PORT]. This option lets you specify what TCP port the driver should use for the listening socket. If you don't specify a port, the driver will use NTF_DEFAULT_TCP_PORT and if it's already used, it will increment it by one until it finds one that is available.
- `OracleConnection.NTF_LOCAL_HOST`: example "212.121.134.12". Use this option to manually specify the IP address of the machine that will receive the notifications from the server. Use this option with caution: only specify the IP address of the local machine when the driver is unable to find it out on its own (it uses `InetAddress.getLocalHost()`). For example if the machine on which runs the JDBC driver is a VPN client, you may have to specify the IP address of the VPN client which the driver cannot find out on its own. This option should **not** be used to attempt to have a different remote host receive the notifications from the server.

Returns:

AQNotificationRegistration[]

Throws:

`SQLException`

Since:

11.1

unregisterAQNotification

```
void unregisterAQNotification(AQNotificationRegistration registration) throws SQLException
```

`SQLException`

Since:

11.1

dequeue

`AQMessage dequeue(String queueName, AQDequeueOptions opt, byte[] tdo) throws SQLException`

Dequeues an AQ message from the queue specified by its name.

Parameters:

`queueName` - name of the queue from which to dequeue.

`opt` - dequeue options

`tdo` - the Type Descriptor Object OID of the type of the queue.

Returns:

the AQMessage dequeued.

Throws:

`SQLException`

Since:

11.1

dequeue

`AQMessage dequeue(String queueName, AQDequeueOptions opt, byte[] tdo, int version) throws SQLException`

Dequeues an AQ message from the queue specified by its name.

Parameters:

`queueName` - name of the queue from which to dequeue.

`opt` - dequeue options

`tdo` - the Type Descriptor Object OID of the type of the queue.

`version` - the version of the type Descriptor

Returns:

the AQMessage dequeued.

Throws:

`SQLException`

Since:

11.1

dequeue

`AQMessage dequeue(String queueName, AQDequeueOptions opt, String typeName) throws SQLException`

Dequeues an AQ message from the queue specified by its name.

Parameters:

`queueName` - name of the queue from which to dequeue.

`opt` - dequeue options.

`typeName` - the name of the type of the queue. For example, it can be "RAW", "SYS.ANYDATA" or "SCOTT.MY_OBJECT_TYPE".

Returns:

the AQMessage dequeued.

Throws:

`SQLException`

Since:

11.1

enqueue

`void enqueue(String queueName, AQEnqueueOptions opt, AQMessage mesg) throws SQLException`

Enqueues the given AQ message to the queue specified by its name.

Parameters:

`queueName` - name of the queue where to enqueue.

`opt` - enqueue options.

`mesg` - the AQ message to enqueue.

Throws:

`SQLException`

Since:

enqueue

```
int enqueue(String queueName, AQEnqueueOptions opt, AQMessage[] mesgs) throws SQLException
```

Enqueues the given array of AQ messages to the queue specified by its name.

Parameters:

`queueName` - name of the queue where to enqueue.

`opt` - enqueue options.

`mesgs` - the array of AQ messages to enqueue.

Returns:

actual number of messages enqueued.

Throws:

`SQLException`

Since:

21.1

dequeue

```
AQMessage[] dequeue(String queueName, AQDequeueOptions opt, String typeName, int deqsize) throws SQLException
```

Dequeues an array of AQ messages from the queue specified by its name.

Parameters:

`queueName` - name of the queue from which to dequeue.

`opt` - dequeue options.

`typeName` - the name of the type of the queue. For example, it can be "RAW", "SYS.ANYDATA" or "SCOTT.MY_OBJECT_TYPE".

`deqsize` - dequeue number of messages

Returns:

the array of AQMessage dequeued.

Throws:

`SQLException`

Since:

21.1

dequeue

```
AQMessage[] dequeue(String queueName, AQDequeueOptions opt, byte[] tdo, int version, int deqsize) throws SQLException
```

Dequeues an array of AQ messages from the queue specified by its name.

Parameters:

`queueName` - name of the queue from which to dequeue.

`opt` - dequeue options

`tdo` - the Type Descriptor Object OID of the type of the queue.

`version` - the version of the type Descriptor

`deqsize` - dequeue number of messages

Returns:

the array of AQMessage dequeued.

Throws:

`SQLException`

Since:

11.1

registerDatabaseChangeNotification

```
DatabaseChangeRegistration registerDatabaseChangeNotification(Properties options) throws SQLException
```

/ Creates a new database change registration.

This method creates a new database change registration in the database server with the given options. It also opens a listening socket which will be used by the database to send notifications. Note that if there already is a listening socket (created by a different registration), then it will be used by this registration as well.

This method returns a DatabaseChangeRegistration object that can then be used to associate a statement with this registration.

The registration will continue to live after this connection is closed. You need to explicitly unregister it to destroy it in the server and release the resources in the driver.

This method uses one roundtrip.

Parameters:

`options` - Possible options are ([] means default):

- `OracleConnection.NTF_QOS_RELIABLE`: "true"/["false"]. Set this option to "true" to make the notifications persistent which comes at a performance cost.
- `OracleConnection.NTF_QOS_PURGE_ON_NTFN`: "true"/["false"]. Set this option to "true" and the registration will be expunged on the first notification event.
- `OracleConnection.NTF_TIMEOUT`: value in seconds "60"/["0"]. Specifies the time in seconds after which the registration is automatically expunged by the

notifications from the server. Use this option with caution: only specify the IP address of the local machine when the driver is unable to find it out on its own (it uses `InetAddress.getLocalHost()`). For example if the machine on which runs the JDBC driver is a VPN client, you may have to specify the IP address of the VPN client which the driver cannot find out on its own. This option should **not** be used to attempt to have a different remote host receive the notifications from the server.

- `OracleConnection.DCN_NOTIFY_ROWIDS`: "true"/["false"]. Use this option to have the server send the ROWIDs of the row that have changed within the notification event. By default this feature is turned off.
- `OracleConnection.DCN_IGNORE_INSERTOP`: "true"/["false"]. Use this option to tell the server to ignore INSERT operations.
- `OracleConnection.DCN_IGNORE_UPDATEOP`: "true"/["false"]. Use this options to tell the server to ignore UPDATE operations.
- `OracleConnection.DCN_IGNORE_DELETEOP`: "true"/["false"]. Use this options to tell the server to ignore DELETE operations.
- `OracleConnection.DCN_NOTIFY_CHANGELAG`: "30"/["0"]. This is an int value (specified as a String), that can be used to specify the number of transactions by which the client is willing to lag behind. This option can be used by the client as a throttling mechanism for database change events. When this option is chosen, ROWID level granularity of information will not be available in the events, even if the DCN_NOTIFY_ROWIDS option was set to "true".
- `OracleConnection.DCN_QUERY_CHANGE_NOTIFICATION`: "true"/["false"]. Use this option to activate query change notification instead of object change notification. Note that this option is only available in the database server starting in 11.1.
- `OracleConnection.DCN_BEST_EFFORT`: "true"/["false"]. If a query has been successfully registered, by default there will be no FALSE positives. If this option is selected during registrations, then registrations on complex queries may still be allowed but notifications may have some FALSE positives, because full pruning may not be performed if determined to be too expensive. In the worst case notifications will be generated in response to any DML/DDL changes to underlying objects. Note that this option is ignored if the `DCN_QUERY_CHANGE_NOTIFICATION` isn't turned on. As `DCN_QUERY_CHANGE_NOTIFICATION`, this option is only available in the database server starting in 11.1.
- `OracleConnection.NTF_GROUPING_CLASS`: `OracleConnection.NTF_GROUPING_CLASS_TIME`/[`OracleConnection.NTF_GROUPING_CLASS_NONE`]. Notification Grouping Class, the criterion or dimension for grouping. The only supported class is `OracleConnection.NTF_GROUPING_CLASS_TIME` meaning grouping by time, that is, the user specifies a time value and a single notification gets published at the end of that time. To use grouping at least this option must be specified to a value other than the default `OracleConnection.NTF_GROUPING_CLASS_NONE`, which is no grouping.
- `OracleConnection.NTF_GROUPING_VALUE`: "1200/["600"]. Notification Grouping Value, the value of the grouping class. The value must be an integer number. For the TIME grouping class, this value represents a number of seconds, meaning the time after which grouped notifications are sent. If not specified, it defaults to 600 sec.
- `OracleConnection.NTF_GROUPING_TYPE`: `OracleConnection.NTF_GROUPING_TYPE_LAST`/[`OracleConnection.NTF_GROUPING_TYPE_SUMMARY`]. Notification Grouping Type, the format of grouping notification. It can either contain the summary of all events (default) or the last event in the group.
- `OracleConnection.NTF_GROUPING_START_TIME`: When to start grouping? Notification grouping can start from a user-specified time that should a valid timestamp with time zone, that is an instance of `oracle.sql.TIMESTAMPTZ`. If this option is not specified when using grouping, it defaults to current system time. For example if `prop` was the option properties, and `conn` the connection object, you would call: `prop.put(OracleConnection.NTF_GROUPING_START_TIME,new TIMESTAMPTZ(conn,"2007-06-21 10:10:00.0"));`.
- `OracleConnection.NTF_GROUPING_REPEAT_TIME`: "100"/[`NTF_GROUPING_REPEAT_FOREVER`]. How many times do grouping? Grouping notifications will be sent as many times as specified by the notification grouping repeat count and after that revert to regular notifications. If not specified, it will default to: `NTF_GROUPING_REPEAT_FOREVER` - keep sending grouping notifications forever.
- `OracleConnection.DCN_CLIENT_INIT_CONNECTION`: "true"/["false"]. This can be configured to initiate a connection from the client instead of opening a listener socket for receiving the database change notifications. Set the value to 'true' for using the Client initiated DCN connection. By default the value is 'false' and opens a listening socket for receiving notifications from the server.
- `OracleConnection.DCN_USE_HOST_CONNECTION_ADDR_INFO`: ["true"]/"false". Set the value to 'false' to use the address info returned by the server for establishing the client initiated DCN Connection. Default value is 'true' and the database hostname and port information present in this database connection's connection string is used for establishing the client initiated DCN connection.

Returns:

DatabaseChangeRegistration

Throws:

`SQLException`

Since:

11.1

getDatabaseChangeRegistration

```
DatabaseChangeRegistration getDatabaseChangeRegistration(int regid) throws SQLException
```

Maps an existing registration identified by its ID 'regid' with a new DatabaseChangeRegistration object.

This method can be used if you create a registration through PLSQL and you want to associate a JDBC statement with it.

This method doesn't create a new listener on the JDBC driver side and DatabaseChangeEvent won't be created. Thus you won't be allowed to attach any listeners to this registration.

Note that this method doesn't generate any roundtrip to the database.

Parameters:

`regid` - The id of the registration

Returns:

DatabaseChangeRegistration A new instance that can be used to associate a statement with this registration

Throws:

`SQLException`

Since:

11.1

unregisterDatabaseChangeNotification

```
void unregisterDatabaseChangeNotification(DatabaseChangeRegistration registration) throws SQLException
```

Deletes a given database change registration. The registration will be destroyed in the server and in the driver (the network listener will be closed if it's not used anymore).

This method interrupts the notification thread and removes all listeners attached to this registration before closing it.

Parameters:

`registration` -

unregisterDatabaseChangeNotification(long,String)

unregisterDatabaseChangeNotification

`void unregisterDatabaseChangeNotification(int registrationId, String host, int tcpport) throws SQLException`

Deprecated.

Deletes a given database change registration in the server. This method doesn't free any resources in the drivers.

This method will throw an "ORA-24950: unregister failed, registration not found" if you don't provide the correct TCP port which can be extracted from the "callback" value in the "USER_CHANGE_NOTIFICATION_REGS" table.

Throws:

SQLException

Since:

11.1

See Also:

unregisterDatabaseChangeNotification(long,String)

unregisterDatabaseChangeNotification

`void unregisterDatabaseChangeNotification(int registrationId) throws SQLException`

Deprecated.

Deletes a given database change registration in the server. This method doesn't free any resources in the drivers.

Throws:

SQLException

Since:

11.1

See Also:

unregisterDatabaseChangeNotification(long, String)

unregisterDatabaseChangeNotification

`void unregisterDatabaseChangeNotification(long registrationId, String callback) throws SQLException`

Deletes a given database change registration in the server. This method doesn't free any resources in the drivers and should only be used to clean up a registration in the database that wasn't properly closed (in the case of JVM crash for example).

This flavor of `unregisterDatabaseChangeNotification` can be used to process the result of the following query: `select regid,callback from USER_CHANGE_NOTIFICATION_REGS;`.

For example to remove all registrations from the database you would execute the following code:

```
    Statement stmt= conn.createStatement();
    ResultSet rs = stmt.executeQuery("select regid,callback from USER_CHANGE_NOTIFICATION_REGS");
    while(rs.next())
    {
      long regid = rs.getLong(1);
      String callback = rs.getString(2);
      ((OracleConnection)conn).unregisterDatabaseChangeNotification(regid,callback);
    }
    rs.close();
    stmt.close();
```

Throws:

SQLException

See Also:

unregisterDatabaseChangeNotification(oracle.jdbc.dcn.DatabaseChangeRegistration)

createARRAY

`ARRAY createARRAY(String typeName, Object elements) throws SQLException`

Creates an ARRAY object with the given type name and elements.

Parameters:

`typeName` - the name of the SQL type of the created object

`elements` - the elements of the created object

Returns:

an ARRAY

Throws:

createOracleArray

```
Array createOracleArray(String arrayTypeName, Object elements) throws SQLException
```

Creates an Array object with the given type name and elements. The standard createArrayOf accepts the element type name. This method accepts the type of the array itself. Oracle does not support anonymous array types and so does not support the standard createArrayOf method.

Parameters:

`arrayTypeName` - the name of the SQL type of the created object

`elements` - the elements of the created object

Returns:

an ARRAY

Throws:

`SQLException` - if a database error occurs

Since:

11.2.0.5.0

createBINARY_DOUBLE

```
BINARY_DOUBLE createBINARY_DOUBLE(double value) throws SQLException
```

Creates a BINARY_DOUBLE that has the given value.

Parameters:

`value` - the value that the new object should represent

Returns:

a new BINARY_DOUBLE

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createBINARY_FLOAT

```
BINARY_FLOAT createBINARY_FLOAT(float value) throws SQLException
```

Creates a BINARY_FLOAT that has the given value.

Parameters:

`value` - the value that the new object should represent

Returns:

a new BINARY_FLOAT

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createDATE

```
DATE createDATE(Date value) throws SQLException
```

Creates a DATE that has the given value.

Parameters:

`value` - the value that the new object should repreesnt

Returns:

a new DATE

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createDATE

```
DATE createDATE(Time value) throws SQLException
```

Creates a DATE that has the given value.

Parameters:

`value` - the value that the new object should repreesnt

Returns:

createDATE

```
DATE createDATE(Timestamp value) throws SQLException
```

Creates a DATE that has the given value.

Parameters:

`value` - the value that the new object should repreesnt

Returns:

a new DATE

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createDATE

```
DATE createDATE(Date value, Calendar cal) throws SQLException
```

Creates a DATE that has the given value. The value is interpreted as being in the time zone represented by cal.

Parameters:

`value` - the value that the new object should repreesnt

`cal` - the timezone in which the value is interpreted

Returns:

a new DATE

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createDATE

```
DATE createDATE(Time value, Calendar cal) throws SQLException
```

Creates a DATE that has the given value. The value is interpreted as being in the time zone represented by cal.

Parameters:

`value` - the value that the new object should repreesnt

`cal` - the timezone in which the value is interpreted

Returns:

a new DATE

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createDATE

```
DATE createDATE(Timestamp value, Calendar cal) throws SQLException
```

Creates a DATE that has the given value. The value is interpreted as being in the time zone represented by cal.

Parameters:

`value` - the value that the new object should repreesnt

`cal` - the timezone in which the value is interpreted

Returns:

a new DATE

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createDATE

```
DATE createDATE(String value) throws SQLException
```

Creates a DATE that has the given value.

Parameters:

`SQLException` - if a database error occurs

Since:

11R1

createINTERVALDS

```
INTERVALDS createINTERVALDS(String value) throws SQLException
```

Creates an INTERVALDS that has the given value.

Parameters:

`value` - the value that the new object shoud represent

Returns:

a new INTERVALDS

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createINTERVALYM

```
INTERVALYM createINTERVALYM(String value) throws SQLException
```

Creates an INTERVALYM that has the given value.

Parameters:

`value` - the value that the new object shoud represent

Returns:

a new INTERVALYM

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createNUMBER

```
NUMBER createNUMBER(boolean value) throws SQLException
```

Creates a new NUMBER that has the given value.

Parameters:

`value` - the value that the new object should represent

Returns:

a new NUMBER

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createNUMBER

```
NUMBER createNUMBER(byte value) throws SQLException
```

Creates a new NUMBER that has the given value.

Parameters:

`value` - the value that the new object should represent

Returns:

a new NUMBER

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createNUMBER

```
NUMBER createNUMBER(short value) throws SQLException
```

Creates a new NUMBER that has the given value.

Parameters:

Since:

11R1

createNUMBER

`NUMBER createNUMBER(int value) throws SQLException`

Creates a new NUMBER that has the given value.

Parameters:

`value` - the value that the new object should represent

Returns:

a new NUMBER

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createNUMBER

`NUMBER createNUMBER(long value) throws SQLException`

Creates a new NUMBER that has the given value.

Parameters:

`value` - the value that the new object should represent

Returns:

a new NUMBER

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createNUMBER

`NUMBER createNUMBER(float value) throws SQLException`

Creates a new NUMBER that has the given value.

Parameters:

`value` - the value that the new object should represent

Returns:

a new NUMBER

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createNUMBER

`NUMBER createNUMBER(double value) throws SQLException`

Creates a new NUMBER that has the given value.

Parameters:

`value` - the value that the new object should represent

Returns:

a new NUMBER

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createNUMBER

`NUMBER createNUMBER(BigDecimal value) throws SQLException`

Creates a new NUMBER that has the given value.

Parameters:

`value` - the value that the new object should represent

11R1

createNUMBER

`NUMBER createNUMBER(BigInteger value) throws SQLException`

Creates a new NUMBER that has the given value.

Parameters:

`value` - the value that the new object should represent

Returns:

a new NUMBER

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createNUMBER

`NUMBER createNUMBER(String value, int scale) throws SQLException`

Creates a new NUMBER that has the given value and scale.

Parameters:

`value` - the value that the new object should represent

`scale` - the scale of the new object

Returns:

a new NUMBER

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createTIMESTAMP

`TIMESTAMP createTIMESTAMP(Date value) throws SQLException`

Creates a new TIMESTAMP with the given value.

Parameters:

`value` - the value that the new object should represent

Returns:

a new TIMESTAMP

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createTIMESTAMP

`TIMESTAMP createTIMESTAMP(DATE value) throws SQLException`

Creates a new TIMESTAMP with the given value.

Parameters:

`value` - the value that the new object should represent

Returns:

a new TIMESTAMP

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createTIMESTAMP

`TIMESTAMP createTIMESTAMP(Time value) throws SQLException`

Creates a new TIMESTAMP with the given value.

Parameters:

`value` - the value that the new object should represent

Since:

11R1

createTIMESTAMP

`TIMESTAMP createTIMESTAMP(Timestamp value) throws SQLException`

Creates a new TIMESTAMP with the given value.

Parameters:

`value` - the value that the new object should represent

Returns:

a new TIMESTAMP

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createTIMESTAMP

`TIMESTAMP createTIMESTAMP(Timestamp value, Calendar cal) throws SQLException`

Creates a new TIMESTAMP with the given value.

Parameters:

`value` - the value that the new object should represent

`cal` - the timezone of the value

Returns:

a new TIMESTAMP

Throws:

`SQLException` - if a database error occurs

Since:

12R2

createTIMESTAMP

`TIMESTAMP createTIMESTAMP(String value) throws SQLException`

Creates a new TIMESTAMP with the given value.

Parameters:

`value` - the value that the new object should represent

Returns:

a new TIMESTAMP

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createTIMESTAMPTZ

`TIMESTAMPTZ createTIMESTAMPTZ(Date value) throws SQLException`

Creates a new TIMESTAMPTZ with the given value.

Parameters:

`value` - the value that the new object should represent

Returns:

a new TIMESTAMPTZ

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createTIMESTAMPTZ

`TIMESTAMPTZ createTIMESTAMPTZ(Date value, Calendar cal) throws SQLException`

Creates a new TIMESTAMPTZ with the given value. The value is interpreted in the time zone of the calendar.

Parameters:

`SQLException` - if a database error occurs

Since:

11R1

createTIMESTAMPTZ

```
TIMESTAMPTZ createTIMESTAMPTZ(Time value) throws SQLException
```

Creates a new TIMESTAMPTZ with the given value.

Parameters:

`value` - the value that the new object should represent

Returns:

a new TIMESTAMPTZ

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createTIMESTAMPTZ

```
TIMESTAMPTZ createTIMESTAMPTZ(Time value, Calendar cal) throws SQLException
```

Creates a new TIMESTAMPTZ with the given value. The value is interpreted in the time zone of the calendar.

Parameters:

`value` - the value that the new object should represent

`cal` - the timezone of the value

Returns:

a new TIMESTAMPTZ

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createTIMESTAMPTZ

```
TIMESTAMPTZ createTIMESTAMPTZ(Timestamp value) throws SQLException
```

Creates a new TIMESTAMPTZ with the given value.

Parameters:

`value` - the value that the new object should represent

Returns:

a new TIMESTAMPTZ

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createTIMESTAMPTZ

```
TIMESTAMPTZ createTIMESTAMPTZ(Timestamp value, Calendar cal) throws SQLException
```

Creates a new TIMESTAMPTZ with the given value. The value is interpreted in the time zone of the calendar.

Parameters:

`value` - the value that the new object should represent

`cal` - the timezone of the value

Returns:

a new TIMESTAMPTZ

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createTIMESTAMPTZ

```
TIMESTAMPTZ createTIMESTAMPTZ(Timestamp value, java.time.ZoneId tzid) throws SQLException
```

Returns:

a new TIMESTAMPTZ

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createTIMESTAMPTZ

```
TIMESTAMPTZ createTIMESTAMPTZ(String value) throws SQLException
```

Creates a new TIMESTAMPTZ with the given value.

Parameters:

`value` - the value that the new object should represent

Returns:

a new TIMESTAMPTZ

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createTIMESTAMPTZ

```
TIMESTAMPTZ createTIMESTAMPTZ(String value, Calendar cal) throws SQLException
```

Creates a new TIMESTAMPTZ with the given value. The value is interpreted in the time zone of the calendar.

Parameters:

`value` - the value that the new object should represent

`cal` - the timezone of the value

Returns:

a new TIMESTAMPTZ

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createTIMESTAMPTZ

```
TIMESTAMPTZ createTIMESTAMPTZ(DATE value) throws SQLException
```

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createTIMESTAMPLTZ

```
TIMESTAMPLTZ createTIMESTAMPLTZ(Date value, Calendar cal) throws SQLException
```

Creates a new TIMESTAMPLTZ with the given value. The value is interpreted in the time zone of the calendar.

Parameters:

`value` - the value that the new object should represent

`cal` - the timezone of the value

Returns:

a new TIMESTAMPLTZ

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createTIMESTAMPLTZ

```
TIMESTAMPLTZ createTIMESTAMPLTZ(Time value, Calendar cal) throws SQLException
```

Creates a new TIMESTAMPLTZ with the given value. The value is interpreted in the time zone of the calendar.

Parameters:

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createTIMESTAMPLTZ

```
TIMESTAMPLTZ createTIMESTAMPLTZ(Timestamp value, Calendar cal) throws SQLException
```

Creates a new TIMESTAMPLTZ with the given value. The value is interpreted in the time zone of the calendar.

Parameters:

`value` - the value that the new object should represent

`cal` - the timezone of the value

Returns:

a new TIMESTAMPLTZ

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createTIMESTAMPLTZ

```
TIMESTAMPLTZ createTIMESTAMPLTZ(String value, Calendar cal) throws SQLException
```

Creates a new TIMESTAMPLTZ with the given value. The value is interpreted in the time zone of the calendar.

Parameters:

`value` - the value that the new object should represent

`cal` - the timezone of the value

Returns:

a new TIMESTAMPLTZ

Throws:

`SQLException` - if a database error occurs

Since:

11R1

createTIMESTAMPLTZ

```
TIMESTAMPLTZ createTIMESTAMPLTZ(DATE value, Calendar cal) throws SQLException
```

Creates a new TIMESTAMPLTZ with the given value. The value is interpreted in the time zone of the calendar.

Parameters:

`value` - the value that the new object should represent

`cal` - the timezone of the value

Returns:

a new TIMESTAMPLTZ

Throws:

`SQLException` - if a database error occurs

Since:

11R1

cancel

```
void cancel() throws SQLException
```

Performs an immediate (asynchronous) termination of any currently executing operation on this connection. It is normally used to stop a long-running JDBC call being processed on the server. It can be called by a user thread in multithreaded applications.

For example, in the context of AQ, it can be used to cancel a 'dequeue' call that is waiting for a new message to be enqueued.

Throws:

`SQLException` - if the cancel operation fails

abort

```
void abort() throws SQLException
```

Calling abort() on an open connection does the following: marks the connection as closed, closes any sockets or other primitive connections to the database, and insures that any thread that is currently accessing the connection will either progress to completion of the JDBC call or throw an exception. Calling abort() on a

Connection.close is synchronized it may hang briefly while any thread that has a lock on the connection completes and releases the lock. Recall that after calling abort any thread that is using the connection will be able to proceed to completion or will throw an exception. abort is very different from OracleConnection.cancel. cancel gracefully stops the execution of any SQL operation. It is a synchronous operation that communicates with the database. It leaves the connection and the statements in a well-known and usable state. In contrast abort tears down the client side network connection to the database, leaving the server to clean up as best it can whenever it discovers that the connection has been broken. abort does not clean up client side resources and leaves the connection and associated statements in an unknown and unusable state. The only valid thing to do with a connection after calling abort is to call close and then discard the connection object. After calling cancel the app can continue to use the connection and statements. If there is a security manager, its checkPermission method is called with an oracle.jdbc.OracleSQLPermission("callAbort") permission to see if the caller has permission to abort a connection. If the caller does not have permission, a SecurityException is thrown. See the ojdbc.policy file in the demo directory for help in granting the appropriate permissions when using a SecurityManager with Oracle JDBC. The best use of the abort() call is in the layer that manages connections such as the connection pool. Stale or invalid connections may often appear to hang when an application thread is blocked on a network call. Connection pools may implement a cleaner thread, that simply looks for such stale connections and issues the abort() call. This results in releasing all client handles and resources without expecting an acknowledgement from the database backend. There is no need for user code to call abort when using an Oracle connection pool such as the Implicit Connection Cache or the Universal Connection Pool as these connection pools will call abort when necessary.

Throws:

`SQLException` - -- Io Exception: Socket closed - ORA-17002 TNS:not connected - ORA-12153

`SecurityException` - if the caller does not have the necessary permission

Since:

11.0

getAllTypeDescriptorsInCurrentSchema

```
TypeDescriptor[] getAllTypeDescriptorsInCurrentSchema() throws SQLException
```

Obtain all the type descriptors associated with object types or array in the schema of this connection. Note that synonyms are not suportted. Requires an internal PL/SQL package that is present only in database 11 and above.

Returns:

An array of the appropriate descriptors for Arrays or Structs depending on the type names found.

Throws:

`SQLException` - If an error occurs.

Since:

11.1

getTypeDescriptorsFromListInCurrentSchema

```
TypeDescriptor[] getTypeDescriptorsFromListInCurrentSchema(String[] typeNames) throws SQLException
```

Obtain the type descriptors associated with object types or array in a schema from an array of type names. Note that synonyms are not suportted. Requires an internal PL/SQL package that is present only in database 11 and above.

Parameters:

An - array of Strings which are type names. Use upper case unless the type names are mixed case names.

Returns:

An array of the appropriate descriptors for Arrays or Structs depending on the type names found.

Throws:

`SQLException` - if the specified type does not exist, or if an error occurred.

Since:

11.1

getTypeDescriptorsFromList

```
TypeDescriptor[] getTypeDescriptorsFromList(String[][] schemaAndTypeNamePairs) throws SQLException
```

Obtain the type descriptors associated with object types or arrays from an array of scheama and type names. Note that synonyms are not suportted. Requires an internal PL/SQL package that is present only in database 11 and above.

Parameters:

An - array of arrays of Strings which are pairs of schema and type names.

Returns:

An array of the appropriate descriptors for Arrays or Structs depending on the type names found.

Throws:

`SQLException` - if any of the specified types does not exist, or if an error occurs.

Since:

11.1

getDataIntegrityAlgorithmName

```
String getDataIntegrityAlgorithmName() throws SQLException
```

Returns the name of the algorithm that is used for data integrity checking by the thin driver on the network. Returns "" when there is no data integrity checking.

Throws:

`SQLException`

Returns the name of the algorithm that is used for data encryption by the thin driver on the network. Returns "" when the data isn't encrypted on the network.

Throws:

`SQLException`

getAuthenticationAdaptorName

```
String getAuthenticationAdaptorName() throws SQLException
```

Returns the name of the adaptor that is used for authentication by the thin driver. Returns "" for basic user/password authenticatin.

Throws:

`SQLException`

isUsable

```
boolean isUsable()
```

Identifies whether this connection is still usable for JDBC operations.

Returns:

true if this connection is usable; false otherwise.

setDefaultTimeZone

```
void setDefaultTimeZone(TimeZone tz) throws SQLException
```

The TimeZone to be used while creating java.sql.Date, java.sql.Time & java.sql.Timestamp.

Parameters:

`Default` - TimeZone to be used for all Date, Time and Timestamp conversions.

Throws:

`SQLException` - if there is an issue while setting the TimeZone

getDefaultTimeZone

```
TimeZone getDefaultTimeZone() throws SQLException
```

Returns the TimeZone set through setDefaultTimeZone.

Returns:

TimeZone set through setDefaultTimeZone. Returns null if TimeZone if setDefaultTimeZone in not invoked with proper values.

Throws:

`SQLException` - If there is any issue while retrieving the TimeZone

setApplicationContext

```
void setApplicationContext(String nameSpace, String attribute, String value) throws SQLException
```

Deprecated.
This has been deprecated since 12.1 in favour of setClientInfo(). It is not recommended to use this API intermingled with get/setClientInfo APIs.

Sets a attribute/value pair in a particular namespace in the application context on the server. This application context is stored in the user session. Note that you can call this method multiple times to set more than one attribute/value pair in the application context. Please note that the only supported namespace is CLIENTCONTEXT. More may be added in a future release. This method does not require any additional roundtrip.

Parameters:

`nameSpace` - The namespace where this attribute/value pairs will be stored. The only supported namespace is CLIENTCONTEXT. The value cannot be null nor empty.

`attribute` - The attribute whose value needs to be set. If the value is null then a NullPointerException is thrown. If the value is an empty string (""), then the namespace will be cleared and the value is ignored.

`value` - The value of the attribute. If the value is null then a NullPointerException is thrown. If the value is an empty string (""), then the attribute will be cleared.

Throws:

`SQLException` - If an error occurs

See Also:

`clearAllApplicationContext(java.lang.String)`, `Connection.setClientInfo(java.lang.String, java.lang.String)`, `Connection.setClientInfo(java.util.Properties)`

clearAllApplicationContext

```
void clearAllApplicationContext(String nameSpace) throws SQLException
```

Parameters:

`nameSpace` - The namespace which will be cleared

Throws:

`SQLException`

See Also:

`setApplicationContext(java.lang.String,java.lang.String,java.lang.String)`, `Connection.setClientInfo(java.util.Properties)`

addLogicalTransactionIdEventListener

```
void addLogicalTransactionIdEventListener(LogicalTransactionIdEventListener listener) throws SQLException
```

Registers a listener to Logical Transaction Id events. The caller is responsible for writing an implementation of the `LogicalTransactionIdEventListener` interface. A listener instance must then be registered using this method.

Throws:

`SQLException`

addLogicalTransactionIdEventListener

```
void addLogicalTransactionIdEventListener(LogicalTransactionIdEventListener listener, Executor executor) throws SQLException
```

This flavor of `addLogicalTransactionIdEventListener` can be used to register a listener with an executor. When a Logical Transaction Id event is triggered the driver will use this executor to call the listener's `onLogicalTransactionIdEvent` method. Typically you would call this method if you want `onLogicalTransactionIdEvent` to be executed in a separate thread.

Throws:

`SQLException`

removeLogicalTransactionIdEventListener

```
void removeLogicalTransactionIdEventListener(LogicalTransactionIdEventListener listener) throws SQLException
```

Deregisters the Logical Transaction Id event listener.

Throws:

`SQLException`

getLogicalTransactionId

```
LogicalTransactionId getLogicalTransactionId() throws SQLException
```

Gets the current Logical Transaction Id which are sent by the server in a piggy back message and hence this method call doesn't make a roundtrip.

Throws:

`SQLException`

isDRCPEnabled

```
boolean isDRCPEnabled() throws SQLException
```

Returns `true` if the connection is participating in DRCP.

Returns:

true if DRCP is enabled

Throws:

`SQLException` - if there is an error while processing the request

See Also:

`attachServerConnection()`, `detachServerConnection(java.lang.String)`, `needToPurgeStatementCache()`

isDRCPMultitagEnabled

```
boolean isDRCPMultitagEnabled() throws SQLException
```

Returns true if multiple tags are allowed with DRCP Connection.

Returns:

true if DRCP multitagging is enabled.

Throws:

`SQLException`

See Also:

`CONNECTION_PROPERTY_USE_DRCP_MULTIPLE_TAG`

Returns the tag associated with this DRCP pooled server. If multiple tags are used, they will be separated by a semicolon. If tagging is not enabled or not available or if this connection is currently not attached to any DRCP pooled server then this method returns null.

Throws:

`SQLException`

See Also:

`detachServerConnection(java.lang.String)`, `CONNECTION_PROPERTY_USE_DRCP_MULTIPLE_TAG`

getDRCPPLSQLCallbackName

```
String getDRCPPLSQLCallbackName() throws SQLException
```

Returns the PL/SQL Fix-up callback name if configured, otherwise returns `Null`

Throws:

`SQLException`

See Also:

`CONNECTION_PROPERTY_DRCP_PLSQL_CALLBACK`, `CONNECTION_PROPERTY_USE_DRCP_MULTIPLE_TAG`

attachServerConnection

```
boolean attachServerConnection() throws SQLException
```

This method needs to be called before using a DRCP connection. This method is a local call when used without tagging and will return immediately. The server process from the specified connection class is obtained during the next roundtrip to the server. So the database roundtrip after attachServerConnection will take longer than usual. However when tagging is used this call makes a roundtrip to obtain a server process from the same connection class. So invoking this call will wait until a server process is available for this session.

Multiple invocation of this method with tagging will not make multiple roundtrips to the database instead return the status from the previous call.

Invoking this method on a non DRCP connection or invoking this method multiple times in a non-tagging case is a noop and will return `true`. Invoking this method multiple times in a tagging case will return the tag match status.

`needToPurgeStatementCache()` returns `true` if the underlying session has changed and if the local statement cache should be purged.

Returns:

true - If the tag matched. In cases without tagging it returns `true` by default

Throws:

`SQLException` - If there is an exception while obtaining server process

Since:

12.1

See Also:

`isDRCPEnabled()`, `detachServerConnection(java.lang.String)`, `needToPurgeStatementCache()`

detachServerConnection

```
void detachServerConnection(String tag) throws SQLException
```

Notify the server that this connection will not be used. The server can choose to reuse the server process if needed. The connection can be released with a tag to so that, upon the next invocation of `attachServerConnection()` on this connection, the server will attempt to pair this connection with the server process of the same tag.

This method makes a one way call to the database for performance reasons. However the call is two way when the connection is participating in a Transaction

Parameters:

`tag` - A string value that the connection will be associated in the the server. `null` is a valid argument when no tagging is required. An empty String will be treated the same as null.

Throws:

`SQLException` - If there was an exception while releasing

Since:

12.1

See Also:

`isDRCPEnabled()`, `attachServerConnection()`, `needToPurgeStatementCache()`

needToPurgeStatementCache

```
boolean needToPurgeStatementCache() throws SQLException
```

Returns if the client side Statement cache has to be purged. This method informs the connection managers if the local statement cache should be purged.

Returns:

`true` to purge the statement cache

Throws:

`SQLException` - if there is an exception while performing this operation.

Since:

12.1

See Also:

getDRCPState

`OracleConnection.DRCPState getDRCPState() throws SQLException`

Returns an enum indicating if the connection is attached to a DRCP server process.

Returns:

`OracleConnection.DRCPState.DETACHED` if the connection is detached. `OracleConnection.DRCPState.ATTACHED_IMPLICIT` or `OracleConnection.DRCPState.ATTACHED_EXPLICIT` if the connection is attached.

Throws:

`SQLException`

Since:

12.2

See Also:

`attachServerConnection()`, `detachServerConnection(java.lang.String)`, `isDRCPEnabled()`

beginRequest

`void beginRequest() throws SQLException`

Declares that a request to the server is starting on this connection. When called after another beginRequest() but before an endRequest(), this call is a no-op and does not throw any exception. Therefore, application is allowed to call beginRequest after a connection pool checkout, which implicitly calls beginRequest.

Specified by:

`beginRequest` in interface `Connection`

Throws:

`SQLException` - When called with an open transaction on this connection.

endRequest

`void endRequest() throws SQLException`

Declares that the request that was in progress on this connection has completed. Existing connection labels and state on the connection are not affected by this call. Calling endRequest() multiple times without beginRequest() in-between is allowed.

Specified by:

`endRequest` in interface `Connection`

Throws:

`SQLException` - When called with an open transaction on this connection.

setShardingKeyIfValid

`boolean setShardingKeyIfValid(OracleShardingKey shardingKey, OracleShardingKey superShardingKey, int timeout) throws SQLException`

Checks the validity of the connection and also checks if the sharding keys passed to this method are valid for the connection.If the sharding keys are valid, it will be set on the connection.

Parameters:

`shardingKey` - Sharding key to be validated and set against this connection

`superShardingKey` - Super Sharding key to be validated and set against this connection

`timeout` - Time in seconds before which the validation process is expected to be complete, else the validation process is aborted. The value of the timeout must be set to zero to disable the timeout during the validation.

Returns:

true if the connection is valid and the shard keys are valid to be set on this connection.

Throws:

`SQLException` - if there is any exception while performing this validation or if timeout value is less than 0.

setShardingKey

`void setShardingKey(OracleShardingKey shardingKey, OracleShardingKey superShardingKey) throws SQLException`

Sets the sharding key and the super sharding key on this connection.

Parameters:

`shardingKey` - Sharding key to be set on this connection

`superShardingKey` - Super Sharding key to be set on this connection

Throws:

`SQLException` - if there is an exception while setting the sharding keys on this connection.

setShardingKeyIfValid

`boolean setShardingKeyIfValid(OracleShardingKey shardingKey, int timeout) throws SQLException`

Checks the validity of the connection and also checks if the sharding key passed to this method is valid for the connection.If the sharding key is valid, it will be set

set to zero to disable the timeout during the validation.

Returns:

true if the connection is valid and the shard keys are valid to be set on this connection.

Throws:

`SQLException` - if there is any exception while performing this validation or if timeout value is less than 0.

setShardingKey

```
void setShardingKey(OracleShardingKey shardingKey) throws SQLException
```

Sets the sharding key on this connection.

Parameters:

`shardingKey` - Sharding key to be set on this connection

Throws:

`SQLException` - if there is an exception while setting the sharding keys on this connection.

isValid

```
boolean isValid(OracleConnection.ConnectionValidation effort, int timeout) throws SQLException
```

Returns true if this connection was working properly to the extent specified by `effort` at the instant during this call it was checked. It does not imply it is still working after the call returns, only it worked at some point during the call. Returns false if the connection is not working properly at the instant during the call when it is checked or if the timeout is exceeded while checking.

Parameters:

`timeout` - The time in seconds to wait for the validation action to complete. If the timeout expires before the action completes the method returns false. A value of 0 mean no limit.

`effort` - How much effort to expend checking the connection.

Returns:

true if the connection is valid, false otherwise.

Throws:

`SQLException` - if timeout < 0

getEncryptionProviderName

```
String getEncryptionProviderName() throws SQLException
```

If network encryption service is enabled, returns the name of the encryption provider, otherwise returns `null`.

Returns:

encryptionProviderName

Throws:

`SQLException`

See Also:

`CONNECTION_PROPERTY_THIN_NET_ENCRYPTION_LEVEL`

getChecksumProviderName

```
String getChecksumProviderName() throws SQLException
```

If network integrity service is enabled, returns the name of the checksum provider, otherwise returns `null`.

Returns:

checksumProviderName

Throws:

`SQLException`

See Also:

`CONNECTION_PROPERTY_THIN_NET_CHECKSUM_LEVEL`

getNetConnectionId

```
String getNetConnectionId() throws SQLException
```

Returns the Net Connection ID associated with this connection. In case of a connection failure, this ID will appear in the log. If connection ID prefix is configured then the Net Connection ID is prepended with the configured value. Note that this method can be called on a closed connection.

Throws:

`SQLException`

See Also:

`CONNECTION_PROPERTY_THIN_NET_CONNECTIONID_PREFIX`

Disables the logging for the connection. If the underlying OracleConnection does not support in-memory logging then it is a NO-OP call.

Throws:

`SQLException`

enableLogging

```
void enableLogging() throws SQLException
```

Enables logging for the connection. If the underlying OracleConnection does not support in-memory logging then it is a NO-OP call.

Throws:

`SQLException`

dumpLog

```
void dumpLog() throws SQLException
```

Dumps the log for the connection to the configured dump location. The log file content is encrypted using the configured public key certificate. If the underlying OracleConnection does not support in-memory logging then it is a NO-OP call.

Throws:

`SQLException`

getLogger

```
oracle.jdbc.diagnostics.SecuredLogger getLogger() throws SQLException
```

Returns the SecuredLogger instance of the OracleConnection. Returns null if the underlying connection implementation does not support logging.

Throws:

`SQLException`

commitAsyncOracle

```
default Flow.Publisher<Void> commitAsyncOracle() throws SQLException
```

Asynchronously make all changes made since the previous commit/rollback permanent and releases any database locks currently held by this `Connection` object. This method should be used only when auto-commit mode has been disabled.

Calling any method of this `Connection` except `cancel`, `abort`, `isClosed` or one defined by `Object` after this method is called will block until the returned `Publisher` calls `onComplete` or `onError`.

The returned publisher will only emit `onComplete` or `onError`; No items are emitted to `onNext`.

Asynchronous tasks initiated by this method will execute under the current `AccessControlContext` of the calling thread.

Returns:

a `Publisher` that emits `onComplete` when the database commit is completed.

Throws:

`SQLException` - if a database access error occurs, this method is called while participating in a distributed transaction, if this method is called on a closed connection or this Connection object is in auto-commit mode

Since:

20

rollbackAsyncOracle

```
default Flow.Publisher<Void> rollbackAsyncOracle() throws SQLException
```

Undoes all changes made in the current transaction and releases any database locks currently held by this Connection object. This method should be used only when auto-commit mode has been disabled.

Calling any method of this `Connection` except `cancel`, `abort`, `isClosed` or one defined by `Object` after this method is called will block until the returned `Publisher` calls `onComplete` or `onError`.

The returned publisher will only emit `onComplete` or `onError`; No items are emitted to `onNext`.

Asynchronous tasks initiated by this method will execute under the current `AccessControlContext` of the calling thread.

Returns:

a `Publisher` that emits `onComplete` when the database rollback is completed.

Throws:

`SQLException` - if a database access error occurs, this method is called while participating in a distributed transaction, if this method is called on a closed connection or this Connection object is in auto-commit mode

Since:

20

Releases this Connection object's database and JDBC resources immediately. Calling the method close on a Connection object that is already closed is a no-op.

Calling any method of this `Connection` except `cancel`, `abort`, `isClosed` or one defined by `Object` after this method is called will block until the returned `Publisher` calls `onComplete` or `onError`.

The returned publisher will only emit `onComplete` or `onError`; No items are emitted to `onNext`.

Asynchronous tasks initiated by this method will execute under the current `AccessControlContext` of the calling thread.

Returns:

a `Publisher` that emits `onComplete` when the `Connection` is closed.

Throws:

`SQLException` - if a database access error occurs

Since:

20