Oracle® NoSQL Database

C# Driver Developer's Guide

Release 20.1
E87793-10
June 2020

Oracle NoSQL Database C# Driver Developer's Guide, Release 20.1

E87793-10

Contents

Preface

This document describes how to get started with Oracle NoSQL Database C# Driver for Tables.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Accessible Access to Oracle Support

Oracle customers who have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Introduction

This document provides a quick introduction to the Oracle NoSQL Database C# driver. This driver provides native C# applications access to data stored in Oracle NoSQL Database tables. It can be used to perform basic database operations such as `get`, `put`, and `search`. `Search` operations can be executed in synchronous or asynchronous manner.

The driver is *thin* and *asynchronous.*

It is termed *thin* as it requires use of a proxy server which translates network activity between the C# client and the Oracle NoSQL Database store. The proxy is written in Java, and can run on any machine that is network accessible by both your C# client code and the Oracle NoSQL Database store. However, for performance and security reasons, Oracle recommends that you run the proxy on the same local host as your driver, and the proxy be used in a 1:1 configuration with your drivers (that is, each instance of the proxy should be used with just a single driver instance). The driver does not use any caching while iterating over potentially large datasets.

It is termed *asynchronous* as it has the capability to call driver operations in a non-blocking manner and receive results of the operation via *asynchronous* callback functions.

This quick start guide assumes that you have read and understood the concepts described in the *Java Direct Driver Developer's Guide*. The entirety of the API used by the C# driver is described in the *C# Driver API Reference*.

Installation

Both the C# driver and the proxy are available in a packaged assembly in *NuGet Package Manager*. The C# driver communicates with the proxy using Thrift protocol and the proxy further communicates with the Oracle NoSQL Database KVStore using Java RMI (Remote Method Invocation) protocol.

To install the C# driver into your project, download and install the `Oracle.nosql.driver.4.x.x` by using the NuGet Package Manager window or console. The package includes the following:

- All jar files required to run the proxy server in the `lib/java` directory
- The `driver.dll` file in the `lib/net46` directory
- Documentation on driver usage in the `doc` directory

To use the proxy, you must have an Oracle NoSQL Database server installation.

> **See Also:**
>
> "NuGet Package Manager"

Connecting to the Store

To perform any store operations, you must establish a network connection between your client code and the store. There are two pieces of information that you must provide:

1. Identify the host name, port number, and store name of any machine hosting a node in the store. (Because the store is comprised of many hosts, there should be multiple host/port pairs for you to choose from.)

   You create the `KVDriver` object by using `KVDriver.Create()`. Then you get the store information by using `driver.GetStore()` and store it in the `KVStore` variable.

2. Identify the path of your proxy server. By default, it is located inside the C# driver package in the `lib/java` directory.

   The `PROXY_CLASSPATH` must point to the location of the jar files. A `Dictionary` variable is created to store this value and it is used in `KVDriver.Create()` as a parameter.

For example, suppose you have an Oracle NoSQL Database store named *"MyNoSQLStore"* and it has a node running on *n1.example.org* at port *5000.* Then you would connect to the store in the following way:

```
using oracle.kv.client;
using oracle.kv.client.config;
using oracle.kv.client.option;
...
static void Main(string[] args) {
    Dictionary<Option, object> dict = new Dictionary<Option, object>();
    //the relative path to the jar files.
    //In this example, 4.5.12 is used for driver version. Replace it with
the version you are using.
    dict.Add(Options.PROXY_CLASSPATH, "..\\..\\..\\packages\
\Oracle.nosql.driver.4.5.12\\lib\\java\\*");

    //Connecting to the NoSQL Database
    IKVDriver driver = KVDriver.Create("nosql://n1.example.org:5000/
MyNoSQLStore", dict);
    //If you store is running on the localhost, the host name, port
number, and store name need not be specified. Below is an example:
    //IKVDriver driver = KVDriver.Create(dict);
    //Fetch store details from the NoSQL Database
    IKVStore store = driver.GetStore();
}
```

`PROXY_MANAGED` is set to true by default. The default value can be overridden by adding the `PROXY_MANAGED` value in the `Dictionary` variable. In the above example, `PROXY_MANAGED` is not specified and the C# Driver automatically manages the proxy server. It starts the proxy server when the application starts and stops it when the application stops.

The proxy listens to a port randomly selected between `8000` and `9000`. The default port range can be overridden by adding the `PROXY_PORT_RANGE_START` and `PROXY_PORT_RANGE_END` values in the `Dictionary` variable.

> **Note:**
>
> In the above example, the proxy server is managed by the application. If you want to start a proxy server which is independent of the driver, see Using the Proxy Server. To connect to an independently running proxy server, see Connecting to the Proxy Server.

If you are using a secure store then the configuration must also include the user name to login to the secured store. This can be specified by either of the following methods:

1. `dict.Add(Options.STORE_SECURITY_FILE, “path_to_the_security_file”);`

   `“path_to_the_security_file”` identifies the security file used to log into the store.

2. `dict.Add(Options.STORE_USER_NAME, “user_name_to_login”);`

   `“user_name_to_login”` specifies the user name used when authenticating to the store.

Creating a Table

Before you can write data to tables in your store, you must define your tables using table DDL statements. You can also use DDL statements to define indexes.

If you want to submit table DDL statements to the store from your C# client code, use `store.ExecuteSQL()`. The table DDL is described in detail in the *Java Direct Driver Developer's Guide*.

> **Note:**
>
> C# Driver supports DDL statements only, so you can create and modify tables, but cannot use SQL queries to manage data.

Here is an example of how to create a table:

```
public static void createTable(IKVStore store, String tableName)
{
    string sql = @"CREATE TABLE IF NOT EXISTS " + tableName
    + " (id INTEGER"
    + ", loginId STRING"
    + ", password STRING"
    + ", PRIMARY KEY(loginId))";
    if (store.ExecuteSQL(sql))
        Console.WriteLine("Table created successfully!");
}
```

Creating a Table with an IDENTITY Column

You can create a table with an IDENTITY column using C# client code.

The following is an example of how to create a table with IDENTITY Column (the store connection is skipped for brevity):

```
public static void createTableWithIdentity(IKVStore store, String
tableName)
{
    string sql = @"CREATE TABLE IF NOT EXISTS " + tableName
    + " (id INTEGER GENERATED ALWAYS AS IDENTITY " +
    + " (START WITH 1 INCREMENT BY 1 NO CYCLE)" +
    + ", name STRING"
    + ", PRIMARY KEY(id))";

    if (store.ExecuteSQL(sql))
        Console.WriteLine("Table created successfully!");
}
```

For complete details on IDENTITY column, see Creating Tables With an IDENTITY Column.

Writing to a Table Row

Once you have defined a table in the store, use `store.CreateRow()` to create an empty table row. Populate the values that you wish to write in the empty table row, and then use the `store.Put()` method to populate the row in the table.

For example, for a table designed like this:

```
CREATE TABLE users (id INTEGER,
    loginId STRING,
    password STRING,
    PRIMARY KEY(loginId)
)
```

You can write a row of table data in the following fashion (the store connection is skipped for brevity):

```
public static void putRow(IKVStore store, String tableName, int id, String
loginId, String password)
{
    IRow insertedRow = null;
    var row = store.CreateRow(tableName);
    row["id"] = id;
    row["loginId"] = loginId;
    row["password"] = password;
    insertedRow = store.Put(row);
    Console.WriteLine(insertedRow); //prints the inserted row in JSON
format
}
```

There are other versions of `store.Put()` that allow you to provide options and version information, such as:

- `store.PutIfAbsent()`
- `store.PutIfPresent()`
- `store.PutIfVersion()`

Writing Rows to a Table with an IDENTITY Column

You can write new rows into a table that has an IDENTITY Column using C# client code.

To create a table with an IDENTITY column, see Creating a Table with an IDENTITY Column.

Use `store.CreateRow()` to create an empty table row. Use the `store.Put()` method to populate the row in the table. Then, use `row.Get()` method to get the generated identity value.

The example below explains how to write data into a table with an IDENTITY column using the C# driver code (the store connection is skipped for brevity):

```
public static void putIdentityRow(IKVStore store, String tableName, String
name) {
    IRow row = store.CreateRow(tableName);
    row["name"] = name;
    store.Put(row);
    int returnid = row.Get("id", Convert.ToInt32);
    Console.WriteLine("Identity Row: " + returnid);
}
```

> **Note:**
>
> A column of NUMBER type in Oracle NoSQL database is mapped to C# DECIMAL type. So in C# driver, identity values of NUMBER type are limited to the range that a C# DECIMAL could represent, which is approximately: 1.0 x 10^-28 to 7.9228 x 10^28.

Deleting a Table Row

Use `store.CreateRow()` to create an empty table row. Populate the empty row with at least the primary key(s) of the rows you wish to delete. Then use `store.Delete()` to delete the row in the table.

```
public static void deleteRow(IKVStore store, String tableName, String
loginId)
{
    IRow deletedRow = null;
    var row = store.CreateRow(tableName);
    row["loginId"] = loginId;
    deletedRow = store.Delete(row, null);
    Console.WriteLine(deletedRow); //prints the deleted row in JSON
format
}
```

There are other version of `store.Delete()` that allows you to provide options and version information, such as:

- `store.DeleteAll()`
- `store.DeleteIfVersion()`

Reading a Single Table Row

To read a single table row, use `store.CreateRow()` to create an empty table row. Populate the empty table row with at least the primary key(s) of the row you wish to read. Then, create an `IRow` variable that you will use to hold the retrieved row. The row is then retrieved using the `store.Get()` method and stored in the `IRow` variable.

For example, to retrieve a table row that uses the primary key `'loginId'`:

```
public static void getRow(IKVStore store, String tableName, String loginId)
{
    IRow fetchedRow = null;
    var row = store.CreateRow(tableName);
    row["loginId"] = loginId;
    fetchedRow = store.Get(row);
    Console.WriteLine(fetchedRow); //prints the fetched row in JSON format
}
```

`store.GetAll()` is a version of `store.Get()` that can be used for reading multiple table rows.

Reading Multiple Table Rows

Use `store.GetAll()` or `store.Search()` to read multiple rows from a table at a time. These functions require you to create an empty row by using `store.CreateRow()` that serves as the lookup key. Different restrictions apply to the key you provide, depending on the function that you use.

The example provided here uses `Store.GetAll()` which requires that the provided key at least contains all the table's shard keys. If all of the shard keys are not present, then the function returns an exception. `store.GetAll()` populates a `List`, which you iterate to retrieve the row available for each position in the result set.

For example, suppose you design a table like this:

```
CREATE TABLE university_data (
        university STRING,
        course STRING,
        studentID STRING,
        studentName STRING,
        studentAddress STRING,
        studentEmail STRING,
        PRIMARY KEY (SHARD(university, course), studentID)
)
```

And you populate it with data like this:

```
using oracle.kv.client;
using oracle.kv.client.config;
...
static void Main(string[] args) {
    ... //connecting to the store and creating the table is skipped for
brevity
    putRow(store, "XYZ University", "Science", "14SC123", "John Doe",
"US", "john.doe@example.com");
    putRow(store, "XYZ University", "Science", "14SC124", "Mike Ruben",
"China", "mike.ruben@example.com");
    putRow(store, "ABC University", "Arts", "14AR101", "Ram Paul",
"India", "ram.paul@example.com");
    putRow(store, "ABC University", "Arts", "14AR102", "Edward Snow",
"UK", "edward.snow@example.com");
    getMultiRows(store, "XYZ University", "Science")
    getAllRows(store, "university_data");
}

public static void putRow(IKVStore store, String university, String
course, String studentID, String studentName, String studentAddress,
String studentEmail)
{
    String tableName = "university_data";
    var row = store.CreateRow(tableName);
    IRow insertedRow = null;
    row["university"] = university;
```

```
    row["course"] = course;
    row["studentID"] = studentID;
    row["studentName"] = studentName;
    row["studentAddress"] = studentAddress;
    row["studentEmail"] = studentEmail;
    insertedRow = store.Put(row);
    Console.WriteLine(insertedRow); //prints the inserted row in JSON
format
}
```

Now you can retrieve data of all students studying `Science` at `XYZ University` by providing just the shard keys.

```
public static void getMultiRows(IKVStore store, String university, String
course)
{
    String tableName = "university_data";
    List<IRow> fetchedRow = null;
    var row = store.CreateRow(tableName);
    row["university"] = university;
    row["course"] = course;
    fetchedRow = store.GetAll(row, null);
    fetchedRow.ForEach(Console.WriteLine); //prints the fetched row in
JSON format
}
```

`store.GetAllKeys()` is a version of `store.GetAll()` that can be used for displaying only the keys of the fetched rows, in this example the `university, course,` and `studentID` fields.

If you want to display all the rows in the table, you use `Store.Search()` with an empty row for the `key` parameter.

```
public static void getAllRows(IKVStore store, String tableName)
{
    var row = store.CreateRow(tableName);
    var fetchedRows = store.Search(row, null);
    foreach (IRow fetchedRow in fetchedRows)
        Console.WriteLine(fetchedRow);
}
```

`store.SearchAsync()` is a version of `store.Search()` that can be used for searching rows asynchronously.

Reading Using Indexes

Use `store.SearchByIndex()` to read table rows based on a specified index. To use this function, the index must first be created using the `CREATE INDEX` statement.

There are two ways to identify the index values you want the results set based on. The first way is to create a row using `store.CreateRow()` that represents the indexed field(s) and value(s) that you want retrieved. The second way is to create a `FetchOptions` object structure and setting `FieldRange.StartValue` and `FieldRange.EndValue` that identifies starting and ending index values that you want returned. The `store.CreateRow()` method and `FetchOptions` structure can be used together to restrict the return set values.

If both `store.CreateRow()` and `FetchOptions` values are `NULL`, then every row in the table is contained in the return set.

For example, suppose you have a table defined like this:

```
CREATE TABLE student_data (
    id STRING,
    firstName STRING,
    lastName STRING,
    email STRING,
    dateOfBirth STRING,
    PRIMARY KEY(SHARD(firstName, lastName), id)
)
```

With this index:

```
// Index is created with name "dob"
CREATE INDEX dob ON student_data (dateOfBirth)
```

And you populate the table with data like this:

```
using oracle.kv.client;
using oracle.kv.client.config;
...
static void Main(string[] args) {
    ... //connecting to the store and creating the table is skipped for
brevity
    putRow(store, "14SC123", "John", "Doe", "john.doe@example.com",
"1996-01-19");
    putRow(store, "14SC124", "Mike", "Ruben", "mike.ruben@example.com",
"1997-02-27");
    putRow(store, "14SC125", "Ram", "Paul", "ram.paul@example.com",
"1997-12-31");
    readUsingIndexes(store);
}
public static void putRow(IKVStore store, String id, String firstName,
String lastName, String email, String dateOfBirth)
{
    String tableName = "student_data";
```

```
    var row = store.CreateRow(tableName);
    IRow insertedRow = null;
    row["id"] = id;
    row["firstName"] = firstName;
    row["lastName"] = lastName;
    row["email"] = email;
    row["dateOfBirth"] = dateOfBirth;
    insertedRow = store.Put(row);
    Console.WriteLine(insertedRow); //prints the inserted row in JSON
format
}
```

Then you read data using the `dob` index using the following function. In the following example, `BLOCK 1` (see the comments in the code) is commented out, because its usage with `BLOCK 2` throws an exception. Comment both `BLOCK 1` and `BLOCK 2` in order to print the entire table.

```
public static void readUsingIndexes(IKVStore store)
{
    String tableName = "student_data";
    var row = store.CreateRow(tableName);
    FetchOptions fetchoptions = new FetchOptions();

    // BLOCK 1:
    // Uncomment this block to look up only table rows with a
    // dateOfBirth field set to "1997-02-27". If this
    // block and BLOCK 2 are both used, then the result set
    // will be empty.
    //row["dateOfBirth"] = "1997-02-27";

    // BLOCK 2:
    // This field range restricts the results set to only
    // those rows with a dateOfBirth field value between
    // "1996-01-01" and "1997-12-31", inclusive.
    fetchoptions.FieldRange.FieldName = "dateOfBirth";
    fetchoptions.FieldRange.StartValue = "1996-01-01";
    fetchoptions.FieldRange.StartIsInclusive = true;
    fetchoptions.FieldRange.EndValue = "1997-12-31";
    fetchoptions.FieldRange.EndIsInclusive = true;

    // "dob" is the name of the index
    var fetchedRows = store.SearchByIndex(row, "dob",
fetchoptions);
    foreach (IRow fetchedRow in fetchedRows)
        Console.WriteLine(fetchedRow);
}
```

`store.SearchByIndexAsync()` is a version of `store.SearchByIndex()` that can be used for searching rows asynchronously.

Reading Asynchronously

Use `store.SearchAsync()` or `store.SearchByIndexAsync()` for reading rows asynchronously. When you read asynchronously, the reading function returns the control immediately and permits other processing to continue, while a read operation is running in the background. The time between the initiation of asynchronous read operation and its completion can be used to do something useful.

This non-blocking mechanism can be achieved by implementing the `IObserver` interface in a class which provides a mechanism for receiving the output asynchronously. This class should override the following callback functions for receiving and handling the output:

- `void OnNext(IRow)` — This function is invoked everytime a row is read from the table.
- `void OnCompleted()` — This function is invoked after the asynchronous read operation is complete.
- `void OnError(Exception)` — If an exception is raised during the asynchronous read operation, this function is invoked.

The following code example shows basic implementation of a subscriber:

```
//the following class overrides the callback functions
class AsyncSubscriber : IObserver<IRow>
{
    // driver calls back for each result
    public void OnNext(IRow value)
    {
        Console.WriteLine("Received search result:" +
value.ToJSONString());
    }

    // driver calls back when search completes
    public void OnCompleted()
    {
        Console.WriteLine("Search complete...");
    }

    // driver calls back if an error occurs
    public void OnError(Exception error)
    {
        Console.WriteLine("Error receiving search results...");
        Console.WriteLine(error.StackTrace);
    }
}
```

To perform an asynchronous read operation, create an object of the class that implements the `IObserver` interface and overrides the callback functions. Then, use the callback class object as a parameter in `store.SearchAsync()` to invoke the asynchronous read operation.

For example, the code fragment shown in Reading Multiple Table Rows for reading all rows can be rewritten to use asynchronous read in the following way:

```
public static void getAllRowsAsync(IKVStore store, String tableName)
{
    var row = store.CreateRow(tableName);
    //an object of the class overriding the callback functions is created
    AsyncSubscriber asyncsubscriber = new AsyncSubscriber();
    //the callback class object is used as the parameter
    store.SearchAsync(row, null, asyncsubscriber);
}
```

The above example retrieves all the rows. To restrict the results, you must provide a row as a search key, similar to synchronous search. See Reading Multiple Table Rows.

Similarly, the code fragment shown for Reading Using Indexes can be rewritten to use asynchronous read in the following way:

```
public static void readUsingIndexesAsync(IKVStore store)
{
    String tableName = "student_data";
    var row = store.CreateRow(tableName);
    // an object of the class overriding the callback functions is created
    AsyncSubscriber asynsubscriber = new AsyncSubscriber();
    FetchOptions fetchoptions = new FetchOptions();

    // BLOCK 1:
    // Uncomment this block to look up only table rows with a
    // dateOfBirth field set to "1997-02-27". If this
    // block and BLOCK 2 are both used, then the result set
    // will be empty.
    //row["dateOfBirth"] = "1997-02-27";

    // BLOCK 2:
    // This field range restricts the results set to only
    // those rows with a dateOfBirth field value between
    // "1996-01-01" and "1997-12-31", inclusive.
    fetchoptions.FieldRange.FieldName = "dateOfBirth";
    fetchoptions.FieldRange.StartValue = "1996-01-01";
    fetchoptions.FieldRange.StartIsInclusive = true;
    fetchoptions.FieldRange.EndValue = "1997-12-31";
    fetchoptions.FieldRange.EndIsInclusive = true;

    // "dob" is the name of the index
    // the callback class object is used as the parameter
    store.SearchByIndexAsync(row, "dob", fetchoptions, AsyncSubscriber);
}
```

Setting Consistency Guarantees

By default, read operations are performed with a consistency guarantee of `NONE_REQUIRED`. Use one of the following to create a consistency guarantee that overrides this default:

1. `SimpleConsistency`
2. `VersionConsistency`
3. `TimeConsistency`

To set consistency guarantees for reading a single table row, create a `ReadOptions` object by using `new ReadOptions()`. Then, create a consistency object and include it in the `ReadOptions` object.

Finally, use `store.Get()`, with the `ReadOptions` object as the parameter, to perform single table row read operation in the store.

For example, the code fragment shown in Reading a Single Table Row can be rewritten to use a consistency policy in the following way:

```
public static void getRow(IKVStore store, String tableName, String loginId)
{
    IRow fetchedRow = null;
    ReadOptions readoptions = new ReadOptions(SimpleConsistency.ABSOLUTE,
500L);
    var row = store.CreateRow(tableName);
    row["loginId"] = loginId;
    fetchedRow = store.Get(row, readoptions);
    Console.WriteLine(fetchedRow); //prints the fetched row in JSON format
}
```

To set consistency guarantees for reading multiple table rows, you must use `FetchOptions`.

For example, the code fragment shown in Reading Multiple Table Rows for reading all rows can be rewritten to use a consistency policy in the following way:

```
public static void getAllRows(IKVStore store, String tableName)
{
    FetchOptions fetchoptions = new FetchOptions();
    TimeConsistency timeconsistency = new TimeConsistency(10, 400L);
    fetchoptions.ReadOptions.Consistency = timeconsistency;
    var row = store.CreateRow(tableName);
    var fetchedRows = store.Search(row, fetchoptions);
    foreach (IRow fetchedRow in fetchedRows)
        Console.WriteLine(fetchedRow);
}
```

To set consistency guarantees for all single and multiple rows read operations, add the following in the code fragment for Connecting to the Store (before `KVDriver.Create()`):

```
ReadOptions readoptions = new ReadOptions(SimpleConsistency.ABSOLUTE,
300L); //for reading single table row
FetchOptions fetchoptions = new FetchOptions(Read); //for reading multiple
table rows
fetchoptions.ReadOptions = readoptions;
dict.Add(Options.OPTIONS_FETCH_DEFAULT, fetchoptions);
dict.Add(Options.OPTIONS_READ_DEFAULT, readoptions);
```

The driver uses `ReadOptions`, `FetchOptions`, and `WriteOptions` for read, search, and write operations respectively. When `null` is specified for these options, it implies default values. The default value for read, fetch, and write options can also be obtained by using `IKVDriver.DefaultReadOptions`, `IKVDriver.DefaultFetchOptions`, and `IKVDriver.DefaultWriteOptions` respectively.

For example, if you want to use the default value for `ReadOptions`, but want to use a non-default value for a particular operation, you could do the following:

```
//Connecting to the store is skipped for brevity
//Similar approach can be followed for FetchOptions and WriteOptions
ReadOptions readoptions = new ReadOptions();
readoptions = driver.DefaultReadOptions;
readoptions.Consistency = new TimeConsistency(100, 10);
```

The read options when modified (as in the above example), does not modify the default value for the driver.

Setting Durability Guarantees

By default, write operations are performed with a durability guarantee of `COMMIT_NO_SYNC`. You can override this by creating and using a durability guarantee.

Use `new Durability()` to create a `Durability` object. Create a `WriteOptions` object by using `new WriteOptions()` and then include the durability policy in it.

Finally, use the `store.Put()` method, with the `WriteOptions` object as the parameter, to perform a write operation in the store.

For example, the code fragment shown in Writing to a Table Row can be rewritten to use a durability policy in the following way:

```
public void putRow(IKVStore store, String tableName, int id, String
loginId, String password)
{
    Durability durability = new Durability(
            SyncPolicy.SYNC, //Master sync
            SyncPolicy.NO_SYNC, //Replica sync
            ReplicaAckPolicy.SIMPLE_MAJORITY //Ack policy
            );
    WriteOptions writeoptions = new WriteOptions(durability,
            3 //0 is the default timeout
            );
    IRow insertedRow = null;
    var row = store.CreateRow(tableName);
    row["id"] = id;
    row["loginId"] = loginId;
    row["password"] = password;
    insertedRow = store.Put(row, writeoptions);
    Console.WriteLine(insertedRow);
}
```

To set durability guarantees for all `Put` transactions, add the following in the code fragment for Connecting to the Store (before `KVDriver.Create()`):

```
Durability durability = new Durability(
        SyncPolicy.SYNC, //Master sync
        SyncPolicy.NO_SYNC, //Replica sync
        ReplicaAckPolicy.SIMPLE_MAJORITY //Ack policy
);
WriteOptions writeoptions = new WriteOptions(durability,
        3 //0 is the default timeout
);
dict.Add(Options.OPTIONS_WRITE_DEFAULT, writeoptions);
```

Using the Proxy Server

The proxy server is a Java application that accepts network traffic from the Table C# API, translates it into requests that the Oracle NoSQL Database store can understand, and then forwards the translated request to the store. The proxy also provides the reverse translation service by interpreting store responses and forwarding them to the client.

The proxy server can be managed or unmanaged:

- **Managed Proxy Server:** When the proxy server is set to managed, the driver itself manages the proxy server based on the configuration specified in the application. A user does not need to start or configure the proxy server. By default, the proxy server is managed. See Connecting to the Store.
- **Unmanaged Proxy Server:** In unmanaged, the proxy server must be running on any network-accessible machine before your C# client can access the store. It has minimal resource requirements and, in many cases, can run on the same machine as the client code is running.

  It requires a set of jar files to be in its class path, either by using the `java -cp` command line option, or by using the `CLASSPATH` environment variable. The jar files are included with the driver and can be found in the following location:

  – *PathToYourC#ProjectFolder*\packages\Oracle.nosql.driver.4.x.x\lib\java*

  The proxy server itself is started using the `oracle.kv.proxy.KVProxy` command. At a minimum, the following information is required when you start the proxy server:

  – `-helper-hosts`

    This is a list of one or more `host:port` pairs representing Oracle NoSQL Database storage nodes that the proxy server can use to connect to the store.

  – `-port`

    The port on which the store listens to the proxy server.

  – `-store`

    The name of the store to which the proxy server is connecting.

  A range of other command line options are available. In particular, if you are using the proxy server with a secure store, you must provide authentication information to the proxy server. In addition, you will probably have to identify a store name to the proxy server. For a complete description of the proxy server and its command line options, see Proxy Server Reference in the *Python Driver Developer's Guide*.

  The simple examples provided in this quick start guide were written to work with a proxy server that is connected to a `kvlite` instance which was started with `secure-config disabled` and all other values as default. The location of the jar files were provided using a `CLASSPATH` environment variable. The command line call used to start the proxy server was:

```
java oracle.kv.proxy.KVProxy -port 7010 -helper-hosts localhost:5000 -
store kvstore
```

Connecting to the Proxy Server

The C# driver can connect to an independently running proxy server. Since the proxy is not managed by the driver, it is termed Unmanaged Proxy Server.

When the proxy is set to unmanaged, the following options and their respective object values should be added to the `Dictionary` variable:

- `PROXY_MANAGED`: This should be set to `false`.
- `PROXY_HOST`: The name or IP of the node running the proxy server.
- `PROXY_PORT`: The port on which the proxy server is listening.

For example, suppose you have a proxy server running on *n1.example.org* at port *7010*. Further, suppose you have an Oracle NoSQL Database store named "*MyNoSQLStore*" running on the same node at port *5000*. Then you would modify the code fragment shown in Connecting to the Store to use an unmanaged proxy service in the following way:

```
using oracle.kv.client;
using oracle.kv.client.config;
...
static void Main(string[] args) {
    //Setting Proxy Parameters. In this case the proxy server is running
independently.
    Dictionary<Option, object> dict = new Dictionary<Option, object>();
    //proxy server is not managed by the application
    dict.Add(Options.PROXY_MANAGED, false);
    dict.Add(Options.PROXY_HOST, "n1.example.org");
    dict.Add(Options.PROXY_PORT, Convert.ToInt32(7010));

    //The host name, port number, and store name need not be specified as
the proxy server and the store is running on the same node.
    IKVDriver driver = KVDriver.Create(dict); //Connecting to the store
    //Fetch store details from the NoSQL Database
    IKVStore store = (KVStore)driver.GetStore();
}
```

> **Note:**
>
> It is recommended to have the proxy server and the store deployed on the same node.

If you are using a secure store then the configuration must also include the user name to login to the secured store. This can be specified by either of the following methods:

1. `dict.Add(Options.STORE_SECURITY_FILE, “path_to_the_security_file”);`

   `“path_to_the_security_file”` identifies the security file used to log into the store.

2. `dict.Add(Options.STORE_USER_NAME, “user_name_to_login”);`

“`user_name_to_login`” specifies the user name used when authenticating to the store.

Driver Configuration

This section lists all the options supported by the driver along with it's data type, default values, and a short description.

Options

Option	Type	Default	Description
ITERATOR_EXPIRATION	Integer	300000	Timeout in millisecond to close an idle table iterator. In unmanaged proxy mode, the value should match the value used by the proxy service.
ITERATOR_MAX_BATCH_SIZE	Integer	100	Maximum number of results to fetch in a single iterator call. In unmanaged proxy mode, the value should match the value used by the proxy service.
ITERATOR_MAX_OPEN	Integer	10000	Maximum number of iterators that can be opened concurrently. In unmanaged proxy mode, the value should match the value used by the proxy service.
ITERATOR_MAX_RESULTS_BATCHES	Integer	0	The maximum number of result batches that can be held in the proxy per iterator. In unmanaged proxy mode, the value should match the value used by the proxy service.
LATENCY_TRACKING_CEILING	Integer	10000	Threshold for logging higher than expected latency in milliseconds per request. Logged at WARNING level. In unmanaged proxy mode, the value should match the value used by the proxy service.
LATENCY_TRACKING_THRESHOLD	Integer	10000	Maximum threshold for tracking latency in milliseconds. In unmanaged proxy mode, the value should match the value used by the proxy service.
LOGGING	String read-only	N.A.	Produces verbose logging message.
OPTIONS_FETCH_DEFAULT	FetchOptions	N.A.	Default options for fetch operations. Only used in managed proxy mode.
OPTIONS_READ_DEFAULT	ReadOptions	N.A.	Default options for read operations. Only used in managed proxy mode.
OPTIONS_WRITE_DEFAULT	WriteOptions	N.A.	Default options for write operations. Only used in managed proxy mode.
PERF_STATS	Boolean	False	Enable performance statistics into default logger. Statistics are logged at FINE level. In unmanaged proxy mode, the value should match the value used by the proxy service.
PROXY_CLASSPATH	String	C:\Program Files (x86)\kvproxy* or /usr/local/lib/kv.proxy/*	The classpath to start the proxy service in managed proxy mode. The classpath is used as-it-is to invoke a Java program in localhost. The wildcard can be used for Java classpath.

Option	Type	Default	Description
PROXY_EXECUTABLE	String	java	Java executable to start proxy service in managed proxy mode. The executable refers to a path in the same host where this driver is running.
PROXY_HOST	String	localhost	The host where driver would connect to the proxy service in unmanaged proxy mode. It is not used in managed proxy mode as managed proxy runs on localhost.
PROXY_MANAGED	Boolean	True	If true, manages own proxy service.
PROXY_PORT	Integer	5010	The port where driver would connect to a Proxy Service in unmanaged proxy mode. It is not used in managed proxy mode, a managed proxy listens to a randomly selected port.
PROXY_PORT_RANGE_END	Integer	9000	End of port range for managed proxy process. Only used in managed proxy mode.
PROXY_PORT_RANGE_START	Integer	8000	Start of port range for managed proxy process. Only used in managed proxy mode.
PROXY_START_ATTEMPT	Integer read-only	2	Number of attempts made to spawn a proxy process. Only used in managed proxy mode.
PROXY_STARTUP_WAIT_TIME_MS	Integer	5000	Wait time in millisecond for a managed proxy service to start. Used only in managed proxy mode.
REQUEST_MAX_ACTIVE	Integer	100	Maximum number of active requests to data store. In unmanaged proxy mode, the value should match the value used by the proxy service.
REQUEST_PERCENT_LIMIT_PER_NODE	Integer	80	Limit on the number of requests per node, as a percentage of requested maximum active requests. In unmanaged proxy mode, the value should match the value used by the proxy service.
REQUEST_PERCENT_THRESHOLD	Integer	90	Threshold for activating request throttling, as a percentage of the requested maximum active requests. In unmanaged proxy mode, the value should match the value used by the proxy service.
REQUEST_TIMEOUT	Long	5000	The default request timeout in milliseconds. In unmanaged proxy mode, the value should match the value used by the proxy service.
REQUST_MAX_CONCURRENT_PER_ITERATOR	Integer	8	The maximum number of concurrent requests per iterator. In unmanaged proxy mode, the value should match the value used by the proxy service.
SCHEMA_RESOURCE	String read-only	N.A.	Name of a schema descriptor file. The file must exists in an application domain.
SOCKET_OPEN_TIMEOUT	Long	3000	Timeout in millisecond to open a socket connection to data store. In unmanaged proxy mode, the value should match the value used by the proxy service.
SOCKET_READ_TIMEOUT	Long	30000	Timeout in millisecond for reading from a socket connection to data store. In unmanaged proxy mode, the value should match the value used by the proxy service.

Option	Type	Default	Description
STATISTICS_INTERVAL	Integer	60	Interval of logging performance statistics in seconds. In unmanaged proxy mode, the value should match the value used by the proxy service.
STORE_HOSTPORT	String	localhost:5000	Host and port of data store server. In managed proxy mode, the proxy service connects to the store of the same location. In unmanaged proxy mode, the value must match the location used by the unmanaged proxy service
STORE_NAME	String	kvstore	The name of the data store. In managed proxy mode, the proxy service connects to a store of the same name. In unmanaged proxy mode, the value must match the store name used by the unmanaged proxy service.
STORE_READ_ZONES	String[]	(null)	List of read zone names separated by comma. In unmanaged proxy mode, the value must match the value used by the proxy service. Otherwise, an exception is raised.
STORE_SECURITY_FILE	String	(null)	The security file used to specify properties for login. Required for connecting to a secure store. In unmanaged proxy mode, the value must match the value used by the proxy service. Otherwise an exception is raised.
STORE_USER_NAME	String	(null)	The name of the user to login to the secured store. Required for connecting to a secure data store. In unmanaged proxy mode, the value must match the value used by the proxy service. Otherwise, an exception is raised.
THREAD_POOL_SIZE	Integer	20	Maximum number of threads in a pool to connect to data store. In unmanaged proxy mode, the value should match the value used by the proxy service.
THROUGHPUT_FLOOR	Integer	0	Threshold for logging lower than expected throughput in request per second. Logged at WARNING level. In unmanaged proxy mode, the value should match the value used by the proxy service.

Usage

All the options above can be set by using the following syntax:

```
Dictionary<Option, object> dict = new Dictionary<Option, object>();
dict.Add(Options.<option>, <value>);
```

Examples

The following is an example for setting the STORE_NAME, PROXY_HOST, and PROXY_PORT:

```
Dictionary<Option, object> dict = new Dictionary<Option, object>();
dict.Add(Options.STORE_NAME, "nosqlstore");
dict.Add(Options.PROXY_HOST, "n1.example.com");
dict.Add(Options.PROXY_PORT, "7001");
```

Third Party Licenses

The Oracle NoSQL Database Client is licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at:.

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

The following applies to this Oracle NoSQL client as well as all other products under the Apache 2.0 license.

Apache License Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions

   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

   "Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
    a. You must give any other recipients of the Work or Derivative Works a copy of this License; and
    b. You must cause any modified files to carry prominent notices stating that You changed the files; and
    c. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
    d. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the

License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions or use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file
7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. Limitation of Liability. In no event and under no legal theory whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description

of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Licensing terms for SLF4J

SLF4J source code and binaries are distributed under the MIT license.

Copyright © 2004, 2013 QOS.ch. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

- The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
- THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.