Getting Started with the C# Driver

Introduction

This quick-start provides just enough information to get you started using the C# driver. After you have gotten started you can refer to the rest of the documentation for more information.

Downloading the C# Driver

The simplest way to get started is by using the nuget package. The package page (http://www.nuget.org/packages/mongocsharpdriver/) will provide details on using nuget.

You can also download the C# driver here:

https://github.com/mongodb/mongo-csharp-driver/releases (https://github.com/mongodb/mongo-csharp-driver/releases)

If you downloaded the `.zip` file, simply unzip it and place the contents anywhere you want.

If you downloaded the `.msi` file, double click on the `.msi` file to run the setup program, which will install the C# driver DLLs in the `C:\Program Files (x86)\MongoDB\CSharp Driver 1.x` directory (the exact path may vary on your system).

Add a Reference to the C# Driver DLLs

Right click on the `References` folder in Visual Studio's Solution Explorer and select `Add Reference...`. Navigate to the folder where the C# driver DLLs were installed and add a reference to the following DLLs:

1. MongoDB.Bson.dll
2. MongoDB.Driver.dll

As an alternative you could use the `NuGet` package manager to add the C# driver package to your solution.

Add Required `using` Statements

As a minimum you will need the following using statements:

```
using MongoDB.Bson;
using MongoDB.Driver;
```

Additionally, you will frequently add one or more of these using statements:

```
using MongoDB.Driver.Builders;
using MongoDB.Driver.GridFS;
using MongoDB.Driver.Linq;
```

There are additional namespaces that would only be required in special cases.

Get a Reference to the Client Object

The easiest way to get a reference to a client object is using a connection string:

```
var connectionString = "mongodb://localhost";
var client = new MongoClient(connectionString);
```

If you want to store the client object in a global variable you can. `MongoClient` is thread-safe.

Get a Reference to a Server Object

To get a reference to a server object from the client object, write this:

```
var server = client.GetServer();
```

Get a Reference to a Database Object

To get a reference to a database object from the server object, write this:

```
var database = server.GetDatabase("test"); // "test" is the name of the database
```

If you use more than one database, call `GetDatabase` again for each database you want to use.

`BsonDocument` Object Model vs. Your Own Domain Classes

There are two ways you can work with collections:

1. using the `BsonDocument` object model
2. using your own domain classes

You would use the `BsonDocument` object model when the data you are working with is so free form that it would be difficult or impossible to define domain classes for it.

Because it is so much easier to work with your own domain classes this quick-start will assume that you are going to do that. The C# driver can work with your domain classes provided that they:

1. Have a no-argument constructor
2. Define public read/write fields or properties for the data you want stored in the database

These requirements are essentially the same as those imposed by .NET's XmlSerializer.

In addition, if your domain class is going to be used as the root document it must contain an *Id* field or property (typically named `Id` although you can override that if necessary). Normally the *Id* will be of type `ObjectId`, but there are no constraints on the type of this member.

Consider the following class definition:

```
public class Entity
{
    public ObjectId Id { get; set; }

    public string Name { get; set; }
}
```

Get a Reference to a Collection Object

You would get a reference to a collection containing `Entity` documents like this:

```
// "entities" is the name of the collection
var collection = database.GetCollection<Entity>("entities");
```

Insert a Document

To insert an `Entity`:

```
var entity = new Entity { Name = "Tom" };
collection.Insert(entity);
var id = entity.Id; // Insert will set the Id if necessary (as it was in this example)
```

Find an Existing Document

In this example we will read back an `Entity` assuming we know the *Id* value:

```
var query = Query<Entity>.EQ(e => e.Id, id);
var entity = collection.FindOne(query);
```

`Query<Entity>.EQ` uses the `Query<T>` builder class to build the query. The lambda expression `e => e.Id` is translated to *_id*. This is the name of the field as stored in the database.

NOTE
Normally the name of the field in the database is exactly the same as the name of the field or property in your domain class, but *Id* is an exception and is mapped to `_id` in the database.

Other query operators include: `GT`, `GTE`, `In`, `LT`, `LTE`, `Near`, `NE`, `And`, `Or` (and a few other more specialized ones).

Save a Document

You can save changes to an existing document like this:

```
entity.Name = "Dick";
collection.Save(entity);
```

Update an Existing Document

An alternative to `Save` is `Update`. The difference is that `Save` sends the entire document back to the server, but `Update` sends just the changes. For example:

```
var query = Query<Entity>.EQ(e => e.Id, id);
var update = Update<Entity>.Set(e => e.Name, "Harry"); // update modifiers
collection.Update(query, update);
```

This example uses the `Update<T>` builder to easily build the update modifiers.

Remove an Existing Document

To remove an existing document from a collection you write:

```
var query = Query<Entity>.EQ(e => e.Id, id);
collection.Remove(query);
```

You Do NOT Need to Call Connect or Disconnect

The C# driver has a connection pool to use connections to the server efficiently. There is no need to call `Connect` or `Disconnect`; just let the driver take care of the connections (calling `Connect` is harmless, but calling `Disconnect` is bad because it closes all the connections in the connection pool).

Full Sample Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using MongoDB.Bson;
using MongoDB.Driver;
using MongoDB.Driver.Builders;

namespace ConsoleApplication1
{
    public class Entity
    {
        public ObjectId Id { get; set; }
        public string Name { get; set; }
    }

    class Program
    {
        static void Main(string[] args)
        {
            var connectionString = "mongodb://localhost";
            var client = new MongoClient(connectionString);
            var server = client.GetServer();
            var database = server.GetDatabase("test");
            var collection = database.GetCollection<Entity>("entities");

            var entity = new Entity { Name = "Tom" };
            collection.Insert(entity);
            var id = entity.Id;

            var query = Query<Entity>.EQ(e => e.Id, id);
            entity = collection.FindOne(query);

            entity.Name = "Dick";
            collection.Save(entity);

            var update = Update<Entity>.Set(e => e.Name, "Harry");
            collection.Update(query, update);

            collection.Remove(query);
        }
    }
}
```

OPTIONS