MongoDB Succinctly

by Agus Kurniawan

Syncfusion | Technology Resource Portal

MongoDB Succinctly

By
Agus Kurniawan

Foreword by Daniel Jebaraj

Syncfusion
Deliver innovation with ease

2501 Aerial Center Parkway
Suite 200
Morrisville, NC 27560
USA

Important licensing information. Please read.

This book is available for free download from www.syncfusion.com on completion of a registration form.

If you obtained this book from any other source, please register and download a free copy from www.syncfusion.com.

This book is licensed for reading only if obtained from www.syncfusion.com.

This book is licensed strictly for personal or educational use.

Redistribution in any form is prohibited.

The authors and copyright holders provide absolutely no warranty for any information provided.

The authors and copyright holders shall not be liable for any claim, damages, or any other liability arising from, out of, or in connection with the information in this book.

Please do not use this book if the listed terms are unacceptable.

Use shall constitute acceptance of the terms listed.

SYNCFUSION, SUCCINCTLY, DELIVER INNOVATION WITH EASE, ESSENTIAL, and .NET ESSENTIALS are the registered trademarks of Syncfusion, Inc.

Technical Reviewer: Zoran Maksimovic
Copy Editor: Benjamin Ball
Acquisitions Coordinator: Jessica Rightmer, senior marketing strategist, Syncfusion, Inc.
Proofreader: Graham High, content producer, Syncfusion, Inc.

Table of Contents

The Story behind the *Succinctly* Series of Books

Daniel Jebaraj, Vice President
Syncfusion, Inc.

Staying on the cutting edge

As many of you may know, Syncfusion is a provider of software components for the Microsoft platform. This puts us in the exciting but challenging position of always being on the cutting edge.

Whenever platforms or tools are shipping out of Microsoft, which seems to be about every other week these days, we have to educate ourselves, quickly.

Information is plentiful but harder to digest

In reality, this translates into a lot of book orders, blog searches, and Twitter scans.

While more information is becoming available on the Internet and more and more books are being published, even on topics that are relatively new, one aspect that continues to inhibit us is the inability to find concise technology overview books.

We are usually faced with two options: read several 500+ page books or scour the web for relevant blog posts and other articles. Just as everyone else who has a job to do and customers to serve, we find this quite frustrating.

The *Succinctly* series

This frustration translated into a deep desire to produce a series of concise technical books that would be targeted at developers working on the Microsoft platform.

We firmly believe, given the background knowledge such developers have, that most topics can be translated into books that are between 50 and 100 pages.

This is exactly what we resolved to accomplish with the *Succinctly* series. Isn’t everything wonderful born out of a deep desire to change things for the better?

The best authors, the best content

Each author was carefully chosen from a pool of talented experts who shared our vision. The book you now hold in your hands, and the others available in this series, are a result of the authors’ tireless work. You will find original content that is guaranteed to get you up and running in about the time it takes to drink a few cups of coffee.

Free forever

Syncfusion will be working to produce books on several topics. The books will always be free. Any updates we publish will also be free.

Free? What is the catch?

There is no catch here. Syncfusion has a vested interest in this effort.

As a component vendor, our unique claim has always been that we offer deeper and broader frameworks than anyone else on the market. Developer education greatly helps us market and sell against competing vendors who promise to “enable AJAX support with one click,” or “turn the moon to cheese!”

Let us know what you think

If you have any topics of interest, thoughts, or feedback, please feel free to send them to us at succinctly-series@syncfusion.com.

We sincerely hope you enjoy reading this book and that it helps you better understand the topic of study. Thank you for reading.

Please follow us on Twitter and “Like” us on Facebook to help us spread the word about the *Succinctly* series!

About The Author

Agus Kurniawan is one of the founders of PECollege.net. He is a lecturer and author, and has been a Microsoft MVP since 2004. He has more than 10 years of software development experience, especially with Microsoft technology, and some experience related to the Linux platform.

You can reach Agus via email at aguskur@hotmail.com and through his blog, http://blog.aguskurniawan.net.

Introduction to MongoDB

What is MongoDB?

To get a better understanding of MongoDB, the official website for MongoDB is an excellent resource: http://www.mongodb.org/display/DOCS/Introduction.

Installation

If you have a Linux platform, you must first update the repository.

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv 7F0CEB10

sudo apt-get update
```

Then, install the MongoDB database engine.

```
sudo apt-get install mongodb-10gen
```

After that, install the MongoDB driver for Node.js.

```
sudo npm install mongodb
```

For Mac and Windows platforms, you can download MongoDB and install it directly from the project website, http://www.mongodb.org/. I recommend installing the MongoDB server engine as a Windows service.

In order to install MongoDB on Windows:

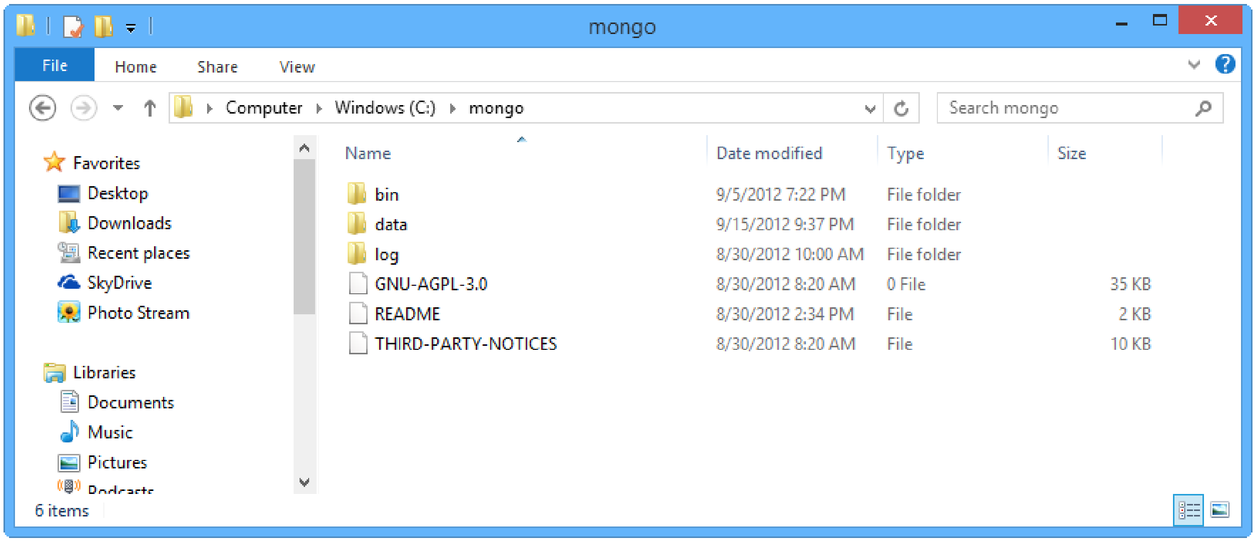
1. Unzip the downloaded files in the **C:\mongo** folder.
2. Open the Command Prompt window with administrator privileges and navigate to the **C:\mongo\bin** folder.
3. Run the following command (you may need to create the c:\mongo\data\db and c:\mongo\log folders before running the command).

```
mongod --logpath "c:\mongo\log\log.log" --logappend --dbpath
"c:\mongo\data\db" --directoryperdb --install
```

After installation, you will get the MongoDB files as shown in Figure 1.

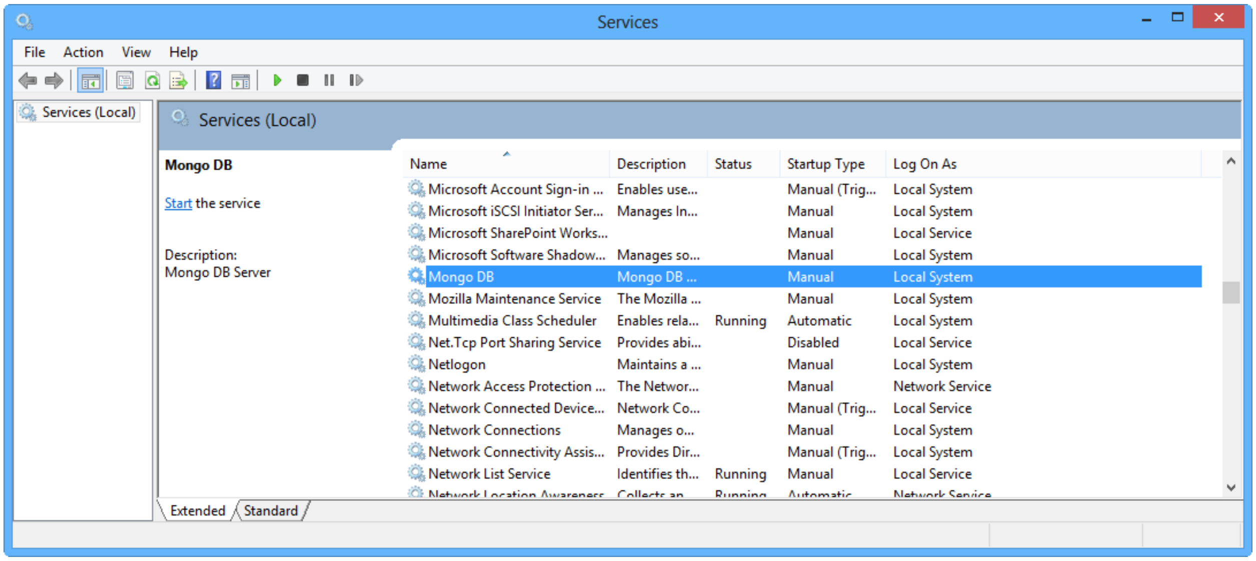
If successful, you will see MongoDB in the Windows **Services** panel, shown in Figure 2.

For more information about installing MongoDB on Windows, visit http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows.



Name	Date modified	Type	Size
bin	9/5/2012 7:22 PM	File folder	
data	9/15/2012 9:37 PM	File folder	
log	8/30/2012 10:00 AM	File folder	
GNU-AGPL-3.0	8/30/2012 8:20 AM	0 File	35 KB
README	8/30/2012 2:34 PM	File	2 KB
THIRD-PARTY-NOTICES	8/30/2012 8:20 AM	File	10 KB

Figure 1: MongoDB files in Windows 8



Mongo DB

Start the service

Description:
Mongo DB Server

Name	Description	Status	Startup Type	Log On As
Microsoft Account Sign-in ...	Enables use...		Manual (Trig...	Local System
Microsoft iSCSI Initiator Ser...	Manages In...		Manual	Local System
Microsoft SharePoint Works...			Manual	Local Service
Microsoft Software Shadow...	Manages so...		Manual	Local System
Mongo DB	Mongo DB ...		Manual	Local System
Mozilla Maintenance Service	The Mozilla ...		Manual	Local System
Multimedia Class Scheduler	Enables rela...	Running	Automatic	Local System
Net.Tcp Port Sharing Service	Provides abi...		Disabled	Local Service
Netlogon	Maintains a ...		Manual	Local System
Network Access Protection ...	The Networ...		Manual	Network Service
Network Connected Device...	Network Co...		Manual (Trig...	Local Service
Network Connections	Manages o...		Manual	Local System
Network Connectivity Assis...	Provides Dir...		Manual (Trig...	Local System
Network List Service	Identifies th...	Running	Manual	Local Service
Network Location Awareness	Collects an...	Running	Automatic	Network Service

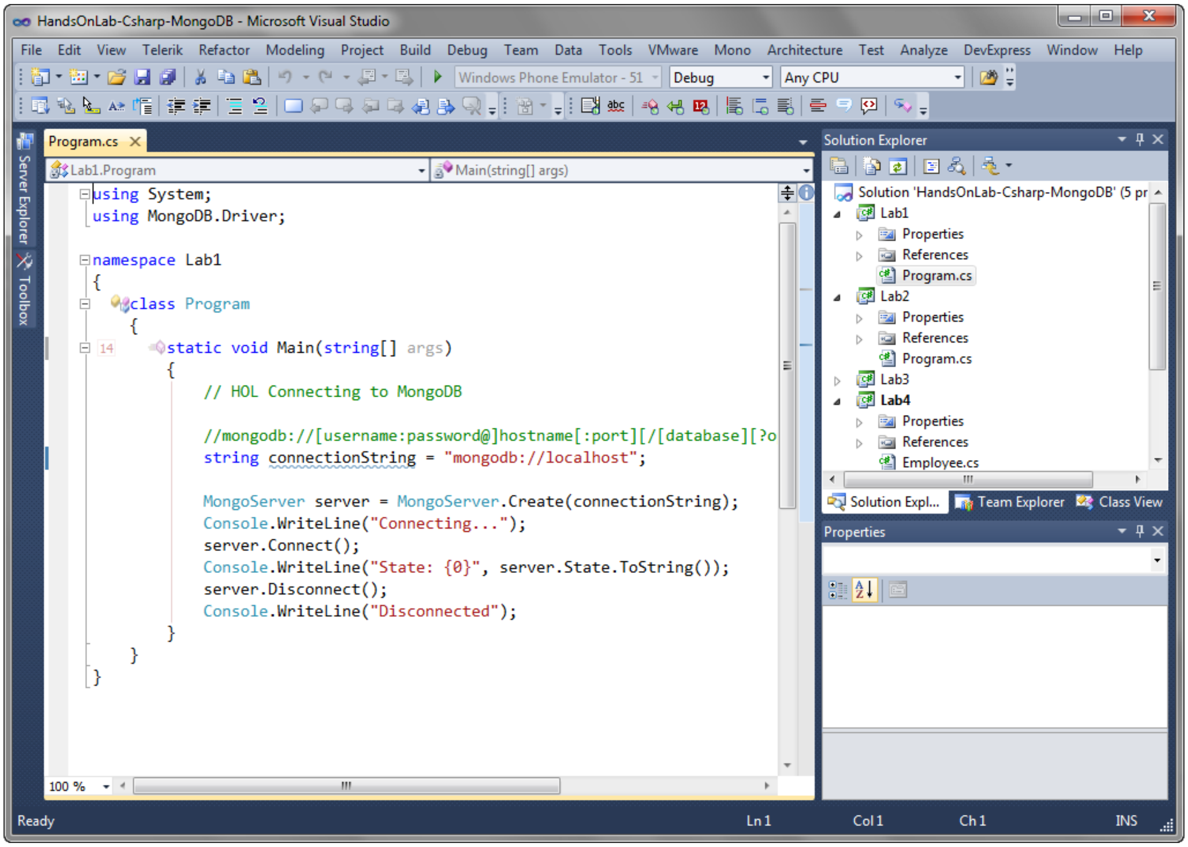
Figure 2: MongoDB engine installed as a Windows service

C# and Development Tools

There are a lot of C# development tools you can use. I recommend using Visual Studio. Microsoft provides a free, Express edition of Visual Studio. You can download it at http://www.microsoft.com/visualstudio/express.

In this book, I use Visual Studio 2012 for testing.

A screenshot of Visual Studio 2010 is shown in Figure 3, and a screenshot of Visual Studio 2012 is shown in Figure 4.



```
using System;
using MongoDB.Driver;

namespace Lab1
{
    class Program
    {
        static void Main(string[] args)
        {
            // HOL Connecting to MongoDB

            //mongodb://[username:password@]hostname[:port][/[database][?o
            string connectionString = "mongodb://localhost";

            MongoServer server = MongoServer.Create(connectionString);
            Console.WriteLine("Connecting...");
            server.Connect();
            Console.WriteLine("State: {0}", server.State.ToString());
            server.Disconnect();
            Console.WriteLine("Disconnected");
        }
    }
}
```

Figure 3: Visual Studio 2010

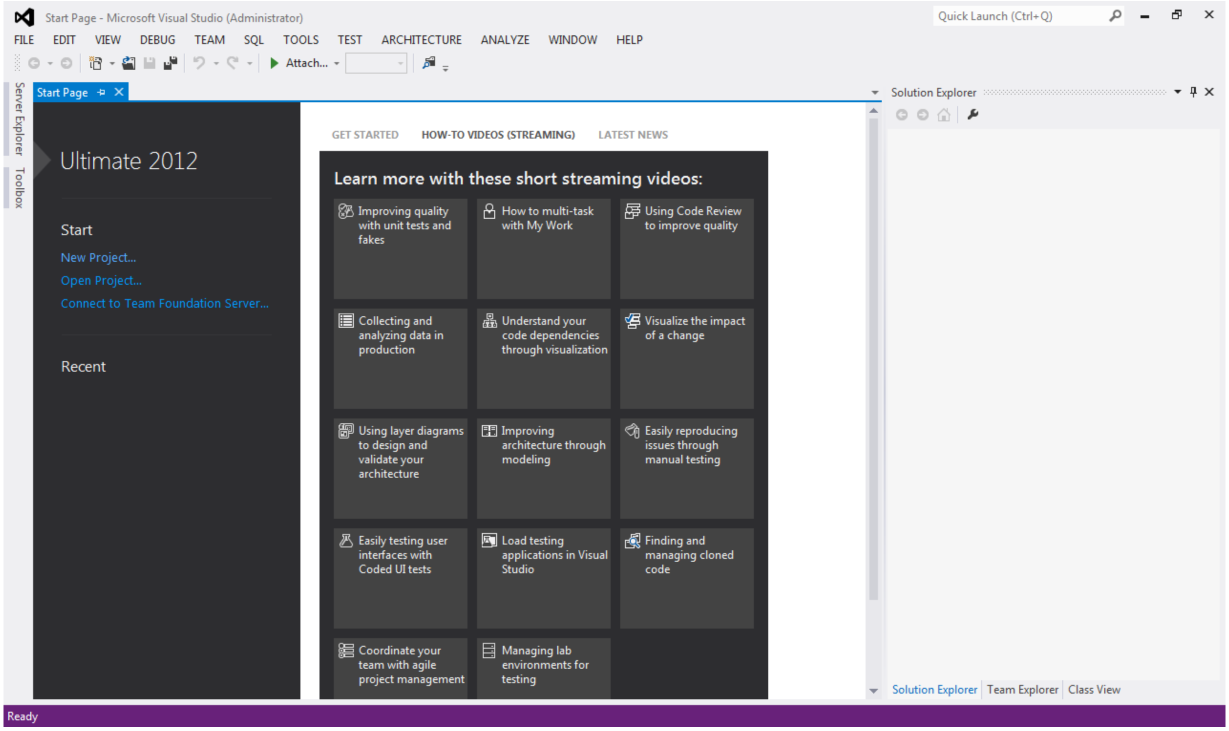

Start Page - Microsoft Visual Studio (Administrator)
Quick Launch (Ctrl+Q)
FILE
EDIT
VIEW
DEBUG
TEAM
SQL
TOOLS
TEST
ARCHITECTURE
ANALYZE
WINDOW
HELP
Attach...
Start Page
Server Explorer
Toolbox
Ultimate 2012
Start
New Project...
Open Project...
Connect to Team Foundation Server...
Recent
GET STARTED
HOW-TO VIDEOS (STREAMING)
LATEST NEWS
Learn more with these short streaming videos:
Improving quality with unit tests and fakes
How to multi-task with My Work
Using Code Review to improve quality
Collecting and analyzing data in production
Understand your code dependencies through visualization
Visualize the impact of a change
Using layer diagrams to design and validate your architecture
Improving architecture through modeling
Easily reproducing issues through manual testing
Easily testing user interfaces with Coded UI tests
Load testing applications in Visual Studio
Finding and managing cloned code
Coordinate your team with agile project management
Managing lab environments for testing
Solution Explorer
Solution Explorer
Team Explorer
Class View
Ready

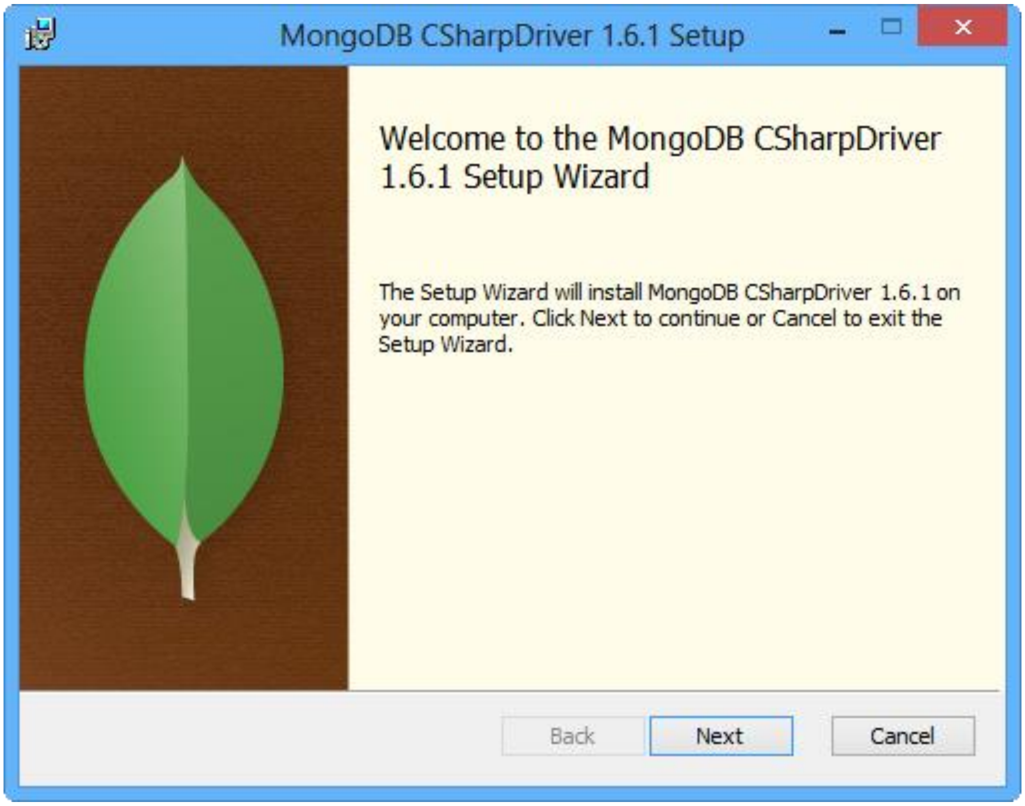
Figure 4: Visual Studio 2012

MongoDB Driver for C#

You can find the MongoDB driver for the .NET platform at http://www.mongodb.org/display/DOCS/Drivers.

I use the official C# driver supported by 10gen. You can download it at https://github.com/mongodb/mongo-csharp-driver/downloads.

If you download the MSI file, you can install MongoDB directly from the setup file. You will see the setup dialog shown in Figure 5.



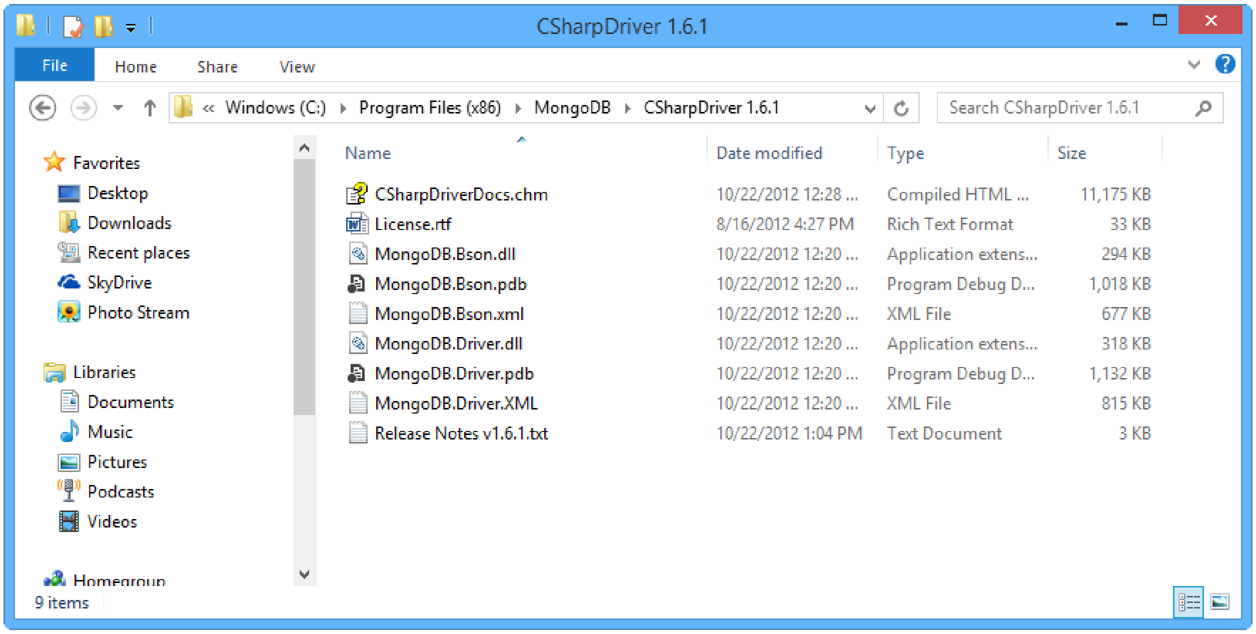
Welcome to the MongoDB CSharpDriver 1.6.1 Setup Wizard

The Setup Wizard will install MongoDB CSharpDriver 1.6.1 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Figure 5: Setup dialog for MongoDB C# driver

Follow all installation instructions.

If successful, you will find the MongoDB driver in the installed folder. For example, C:\Program Files (x86)\MongoDB\CSharpDriver 1.6.1.



Name	Date modified	Type	Size
CSharpDriverDocs.chm	10/22/2012 12:28 ...	Compiled HTML ...	11,175 KB
License.rtf	8/16/2012 4:27 PM	Rich Text Format	33 KB
MongoDB.Bson.dll	10/22/2012 12:20 ...	Application extens...	294 KB
MongoDB.Bson.pdb	10/22/2012 12:20 ...	Program Debug D...	1,018 KB
MongoDB.Bson.xml	10/22/2012 12:20 ...	XML File	677 KB
MongoDB.Driver.dll	10/22/2012 12:20 ...	Application extens...	318 KB
MongoDB.Driver.pdb	10/22/2012 12:20 ...	Program Debug D...	1,132 KB
MongoDB.Driver.XML	10/22/2012 12:20 ...	XML File	815 KB
Release Notes v1.6.1.txt	10/22/2012 1:04 PM	Text Document	3 KB

Figure 6: MongoDB driver files for C#

After creating a C# project in Visual Studio, you can add the MongoDB driver to your project. To do so, right-click on **References** to open the context menu as shown in Figure 7.

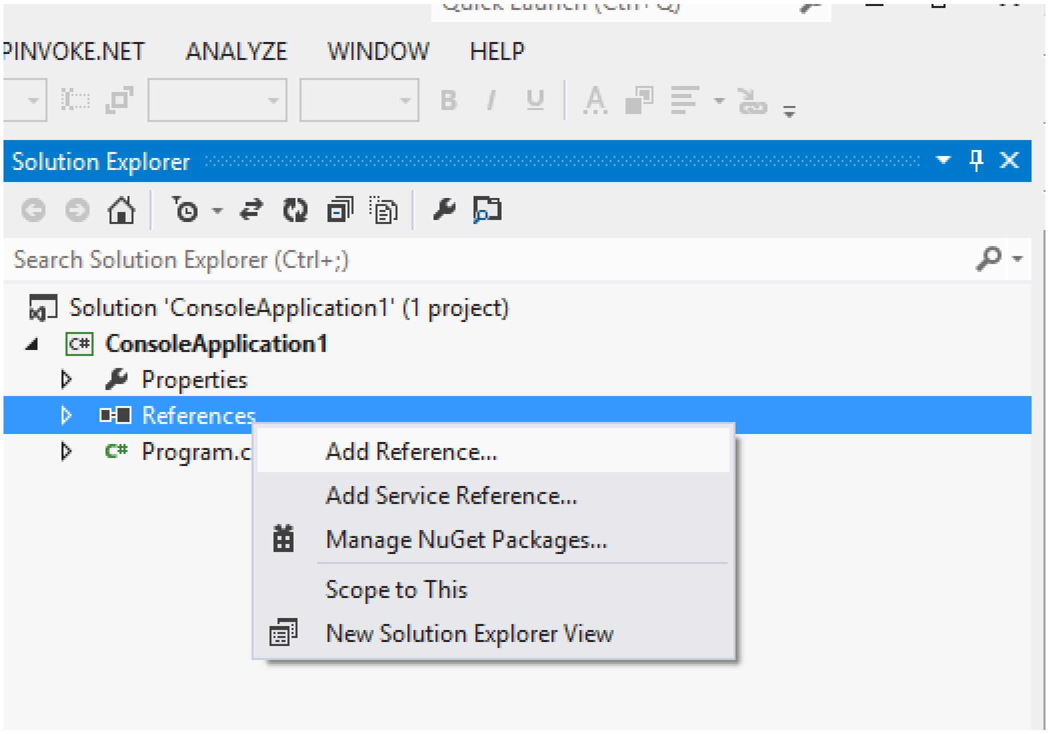
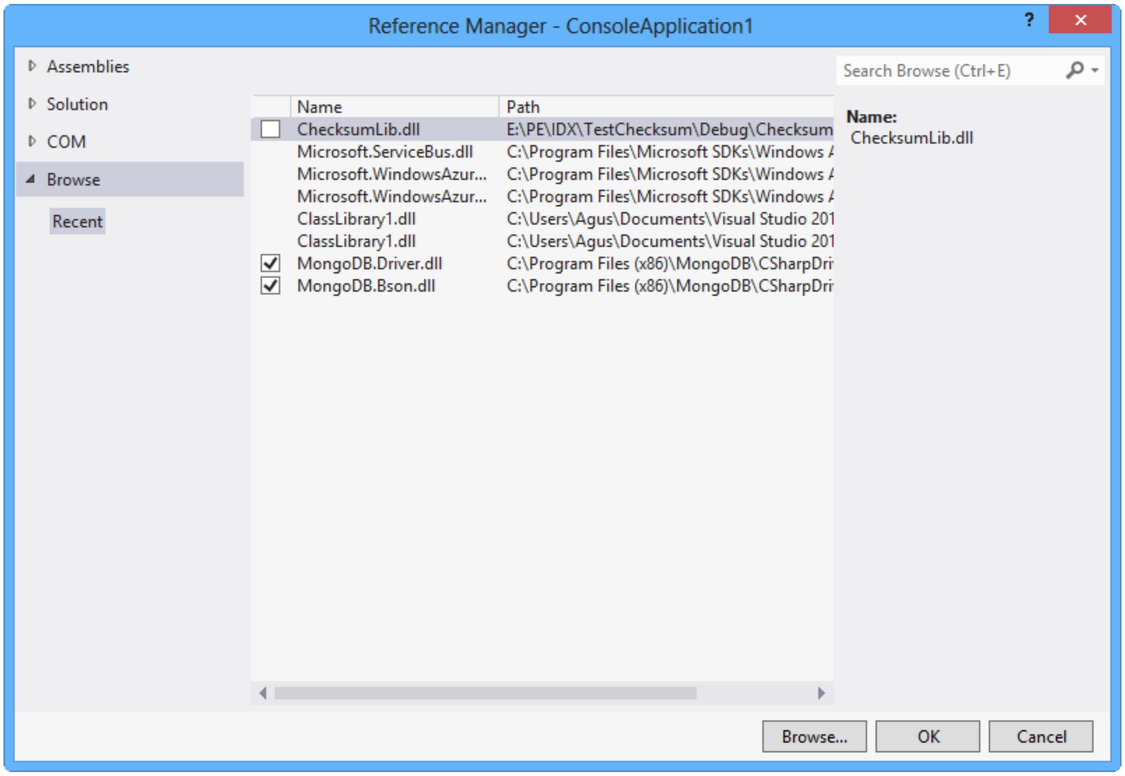


Figure 7: Adding an external library in Visual Studio 2012

Select **Add Reference**. This will open the **Reference Manager** as shown in Figure 8. Add the **MongoDB.Driver.dll** and **MongoDB.Bson.dll** files from the installed MongoDB folder.

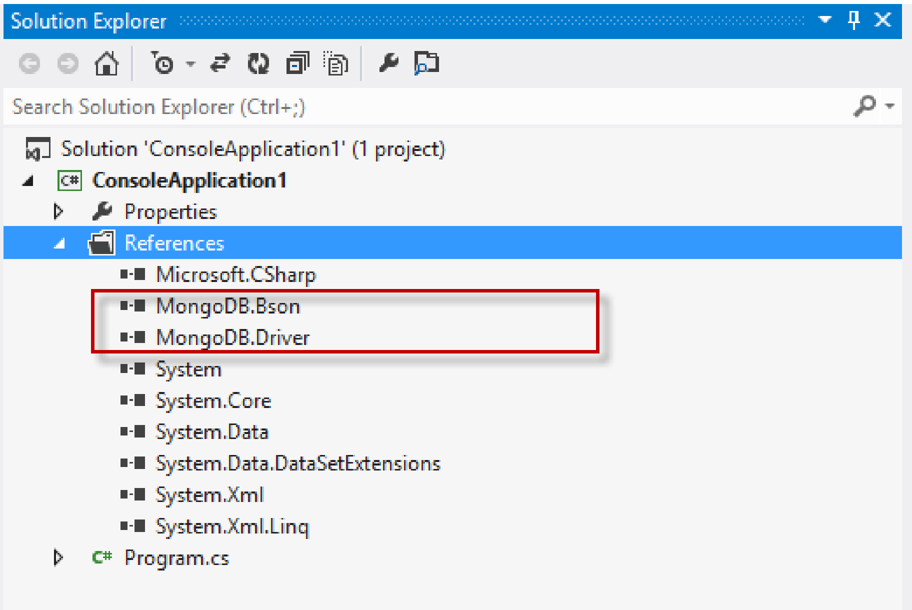


	Name	Path
☐	ChecksumLib.dll	E:\PE\IDX\TestChecksum\Debug\Checksum
	Microsoft.ServiceBus.dll	C:\Program Files\Microsoft SDKs\Windows A
	Microsoft.WindowsAzur...	C:\Program Files\Microsoft SDKs\Windows A
	Microsoft.WindowsAzur...	C:\Program Files\Microsoft SDKs\Windows A
	ClassLibrary1.dll	C:\Users\Agus\Documents\Visual Studio 201
	ClassLibrary1.dll	C:\Users\Agus\Documents\Visual Studio 201
☑	MongoDB.Driver.dll	C:\Program Files (x86)\MongoDB\CSharpDri
☑	MongoDB.Bson.dll	C:\Program Files (x86)\MongoDB\CSharpDri

Name:
ChecksumLib.dll

Figure 8: Adding MongoDB driver files

Click **OK**. You will see the MongoDB driver referenced in your project.



- Solution 'ConsoleApplication1' (1 project)
  - **ConsoleApplication1**
    - Properties
    - References
      - Microsoft.CSharp
      - MongoDB.Bson
      - MongoDB.Driver
      - System
      - System.Core
      - System.Data
      - System.Data.DataSetExtensions
      - System.Xml
      - System.Xml.Linq
    - Program.cs

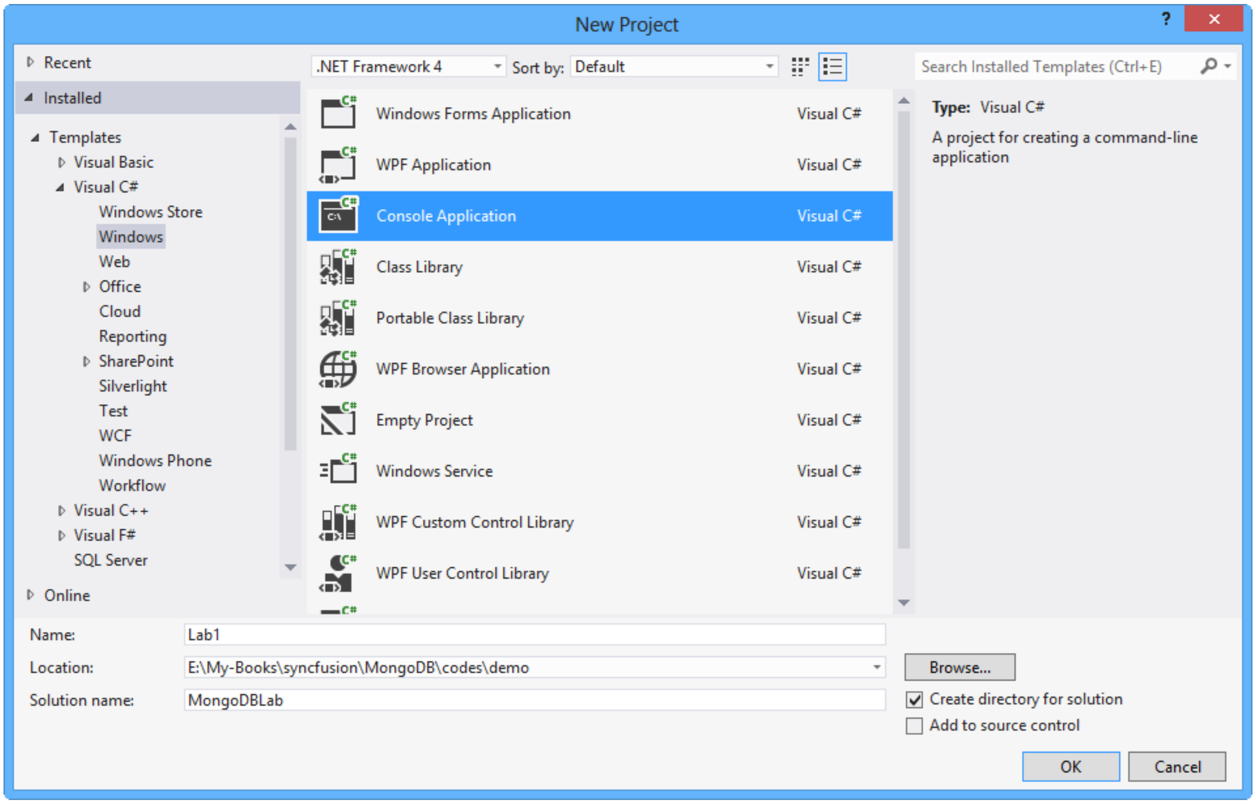
Figure 9: C# project with reference to MongoDB driver

Chapter 1 Connecting to MongoDB

In this chapter, you will learn how to connect to a MongoDB database server using C#.

Creating a Console Application

Create a new **Console Application** project in Visual Studio 2012. Type the project name and folder location. Then, click **OK**.


New Project
Recent
Installed
Templates
Visual Basic
Visual C#
Windows Store
Windows
Web
Office
Cloud
Reporting
SharePoint
Silverlight
Test
WCF
Windows Phone
Workflow
Visual C++
Visual F#
SQL Server
Online
.NET Framework 4
Sort by: Default
Search Installed Templates (Ctrl+E)
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
Class Library Visual C#
Portable Class Library Visual C#
WPF Browser Application Visual C#
Empty Project Visual C#
Windows Service Visual C#
WPF Custom Control Library Visual C#
WPF User Control Library Visual C#
Browse...
OK
Cancel

Type: Visual C#

A project for creating a command-line application

Name: Lab1

Location: E:\My-Books\syncfusion\MongoDB\codes\demo

Solution name: MongoDBLab

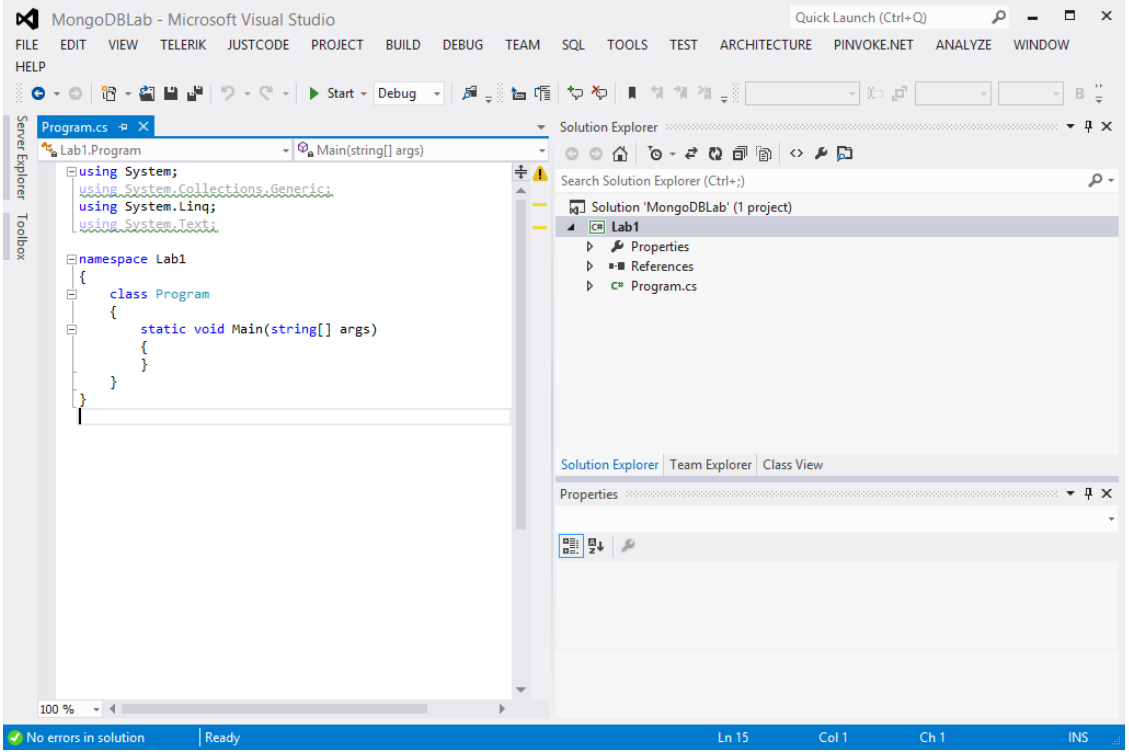
☑ Create directory for solution

□ Add to source control

Figure 10: Choosing Console Application from the project templates

You will get a console application project in Visual Studio 2012, as shown in Figure 11.

Now you’re ready to start writing code.



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

namespace Lab1
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Figure 11: Visual Studio 2012 with console application project

Adding MongoDB Driver Files

After creating the C# project in Visual Studio, you must add the MongoDB driver to your project. Right-click **References** to open a context menu, as shown in Figure 12.

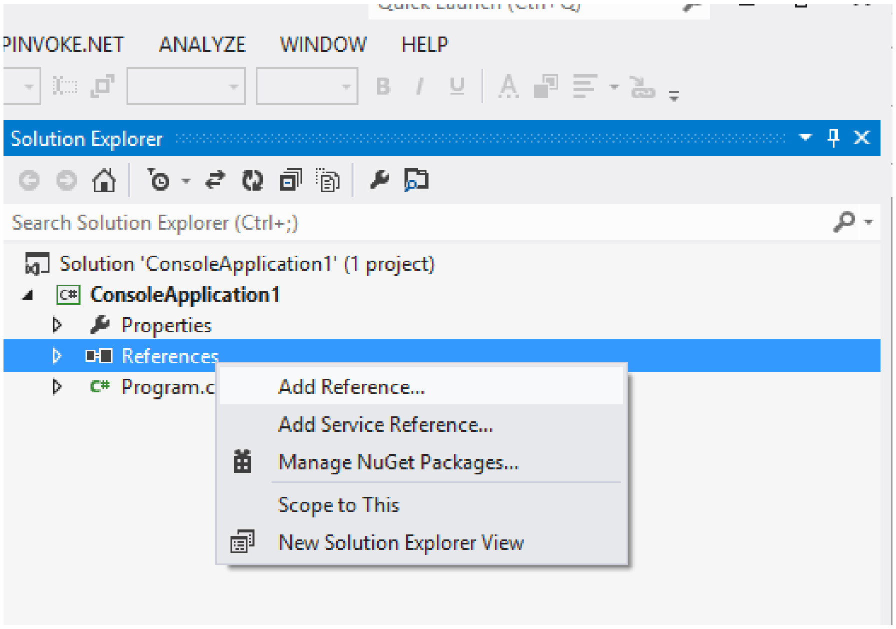
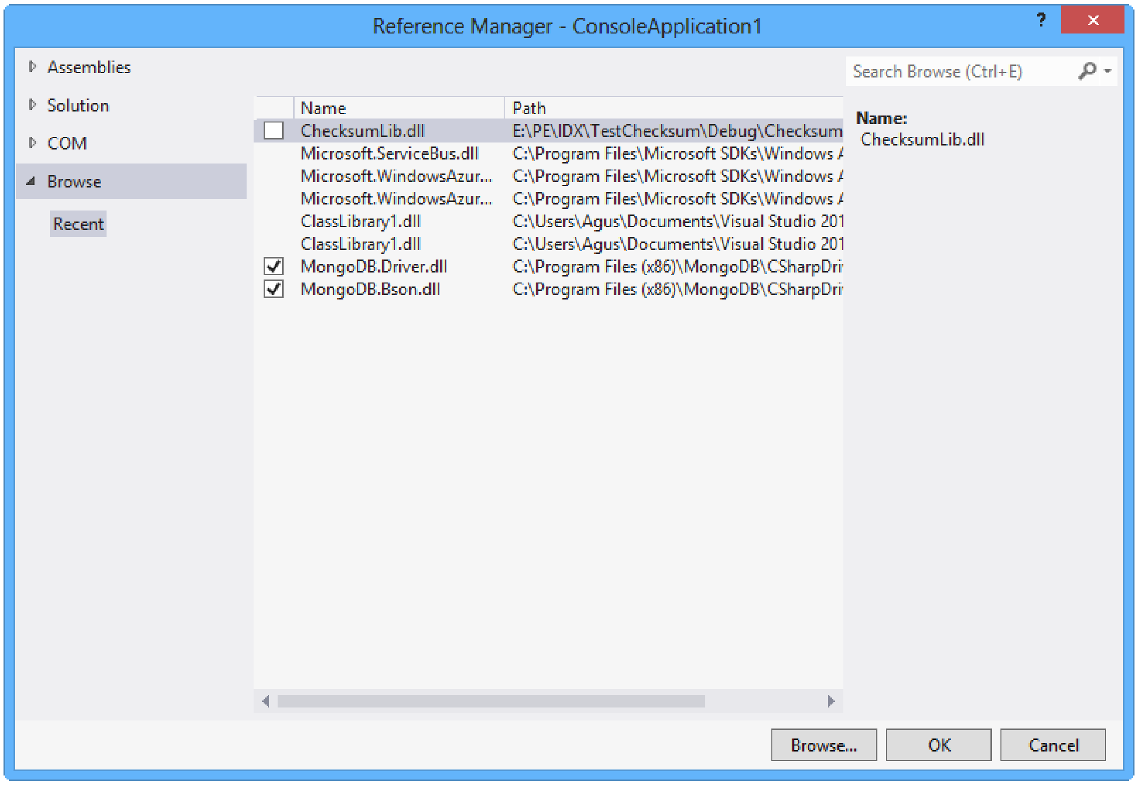


Figure 12: Adding external library in Visual Studio 2012

Select **Add Reference** to open the **Reference Manager** as shown in Figure 13. Add the **MongoDB.Driver.dll** and **MongoDB.Bson.dll** files from the installed MongoDB folder.



Reference Manager - ConsoleApplication1

- Assemblies
- Solution
- COM
- Browse
  - Recent

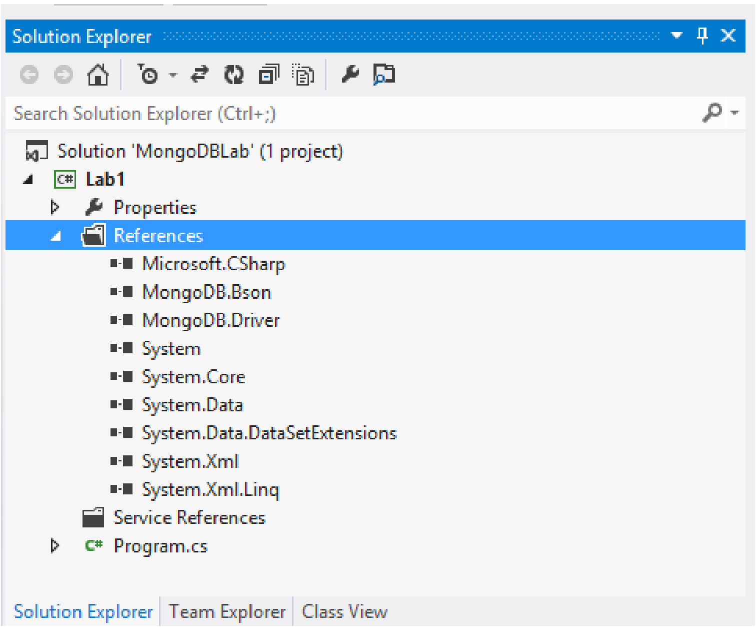
Search Browse (Ctrl+E)

	Name	Path
□	ChecksumLib.dll	E:\PE\IDX\TestChecksum\Debug\Checksum
	Microsoft.ServiceBus.dll	C:\Program Files\Microsoft SDKs\Windows A
	Microsoft.WindowsAzur...	C:\Program Files\Microsoft SDKs\Windows A
	Microsoft.WindowsAzur...	C:\Program Files\Microsoft SDKs\Windows A
	ClassLibrary1.dll	C:\Users\Agus\Documents\Visual Studio 201
	ClassLibrary1.dll	C:\Users\Agus\Documents\Visual Studio 201
☑	MongoDB.Driver.dll	C:\Program Files (x86)\MongoDB\CSharpDri
☑	MongoDB.Bson.dll	C:\Program Files (x86)\MongoDB\CSharpDri

Name:
ChecksumLib.dll

Figure 13: Adding MongoDB driver files

Click **OK**. You will see the MongoDB driver referenced in your project.



Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'MongoDBLab' (1 project)
  - **Lab1**
    - Properties
    - References
      - Microsoft.CSharp
      - MongoDB.Bson
      - MongoDB.Driver
      - System
      - System.Core
      - System.Data
      - System.Data.DataSetExtensions
      - System.Xml
      - System.Xml.Linq
    - Service References
    - Program.cs

Figure 14: Project with MongoDB driver reference

Writing Code

First, you must add namespaces.

```
using System;
using System.Linq;
using MongoDB.Driver;
```

Use the following code to connect to the MongoDB server.

```
namespace Lab1
{
    class Program
    {
        static void Main(string[] args)
        {
            string connectionString = "mongodb://localhost";

            MongoServer server = MongoServer.Create(connectionString);
            Console.WriteLine("Connecting...");
            server.Connect();
            Console.WriteLine("State: {0}", server.State.ToString());
            server.Disconnect();
            Console.WriteLine("Disconnected");

            Console.WriteLine("Press any key to continue…");
            Console.Read();
        }
    }
}
```

Explanation

We must define the database connection string.

```
 string connectionString = "mongodb://localhost";
```

If your MongoDB server applies a password policy, you should change the database connection string.

```
string connectionString = "mongodb://[username:password@]hostname[:port]";
```

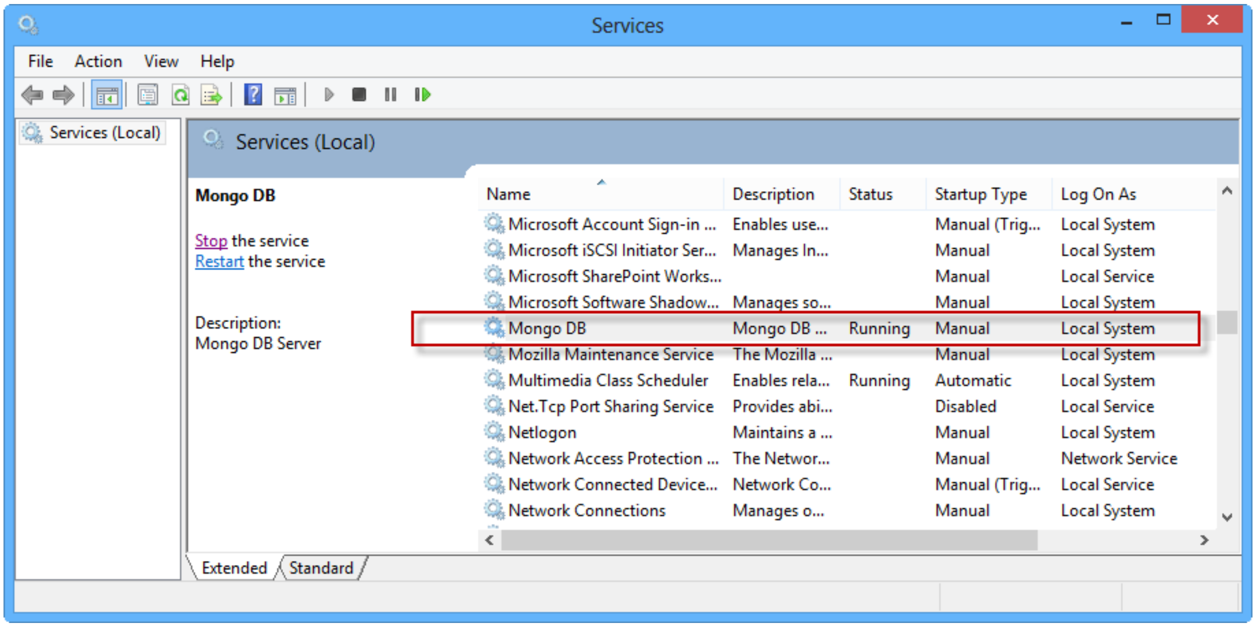
Change the **username**, **password**, **hostname**, and **port**.

The database connection string is passed to the **MongoServer** object, which provides an API to manipulate the MongoDB server.

To connect to the MongoDB server, you can call the **Connect()** method from the **MongoServer** object. Call **Disconnect()** if you want to disconnect from the server.

Testing

Your MongoDB server service should be started and its status should be "Running."



Mongo DB

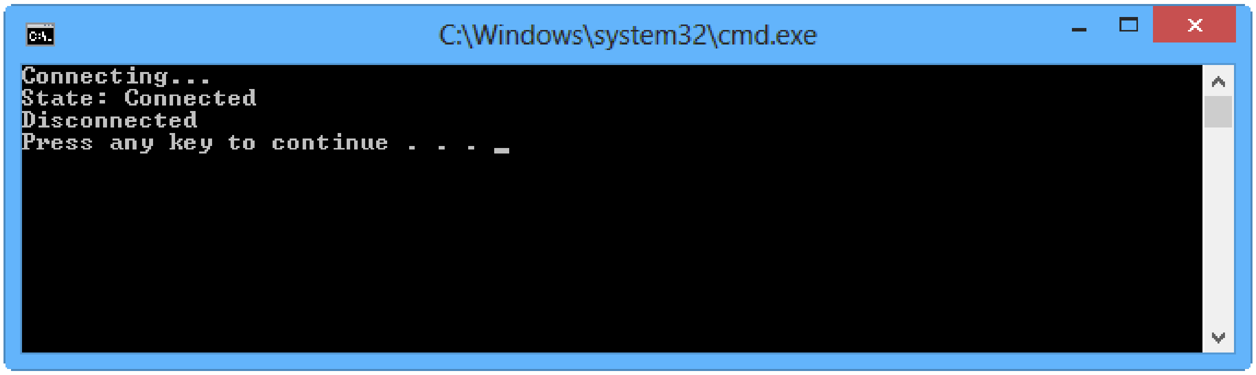
Stop the service
Restart the service

Description:
Mongo DB Server

Name	Description	Status	Startup Type	Log On As
Microsoft Account Sign-in ...	Enables use...		Manual (Trig...	Local System
Microsoft iSCSI Initiator Ser...	Manages In...		Manual	Local System
Microsoft SharePoint Works...			Manual	Local Service
Microsoft Software Shadow...	Manages so...		Manual	Local System
Mongo DB	Mongo DB ...	Running	Manual	Local System
Mozilla Maintenance Service	The Mozilla ...		Manual	Local System
Multimedia Class Scheduler	Enables rela...	Running	Automatic	Local System
Net.Tcp Port Sharing Service	Provides abi...		Disabled	Local Service
Netlogon	Maintains a ...		Manual	Local System
Network Access Protection ...	The Networ...		Manual	Network Service
Network Connected Device...	Network Co...		Manual (Trig...	Local Service
Network Connections	Manages o...		Manual	Local System

Figure 15: MongoDB server service is running

Now you can run the application from Visual Studio 2012. A sample program output is shown in Figure 16.



```
Connecting...
State: Connected
Disconnected
Press any key to continue . . . _
```

Figure 16: Application connected to MongoDB server

Chapter 2 Creating and Deleting the Database

This chapter describes how to create and delete the MongoDB database using C#. I used the console application created in the previous chapter to demonstrate this. Don't forget to add the MongoDB driver files for C# as references.

Include the following namespaces in the **`using`** section.

```
using System;
using System.Linq;
using MongoDB.Driver;
```

Creating a Database

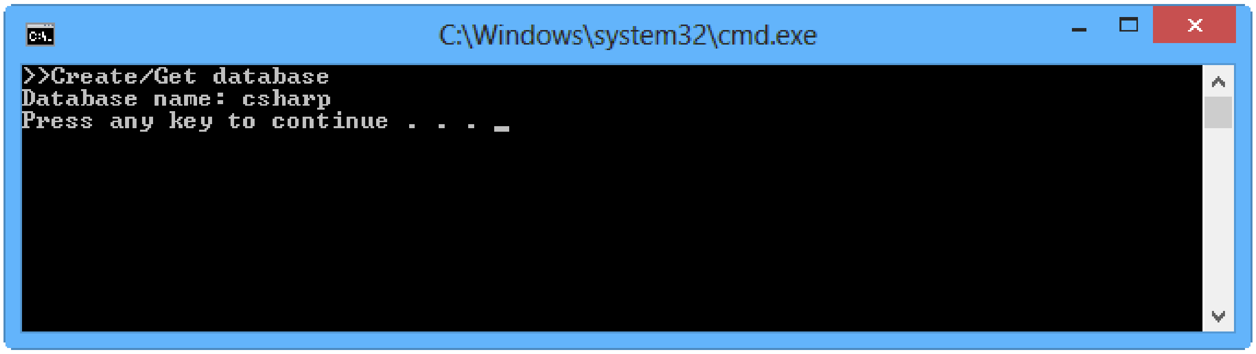
First, declare a connection string and database. Create a new database called **csharp**.

You can create the database by using the **`MongoDatabase`** object. It can be obtained from **`MongoServer`** by calling the **`GetDatabase()`** method.

When you call **`GetDatabase()`**, you should specify the database name as the method's parameter.

```
string connectionString = "mongodb://localhost";
string databaseName = "csharp";

Console.WriteLine(">>Create/Get database");
MongoServer server = MongoServer.Create(connectionString);
MongoDatabase database = server.GetDatabase(databaseName);
Console.WriteLine("Database name: {0}", database.Name);
```

If the database name does not exist, the MongoDB driver will create it automatically. It first creates a database in cache memory. If data comes into this database, MongoDB will flush the database.



```
>>Create/Get database
Database name: csharp
Press any key to continue . . . _
```

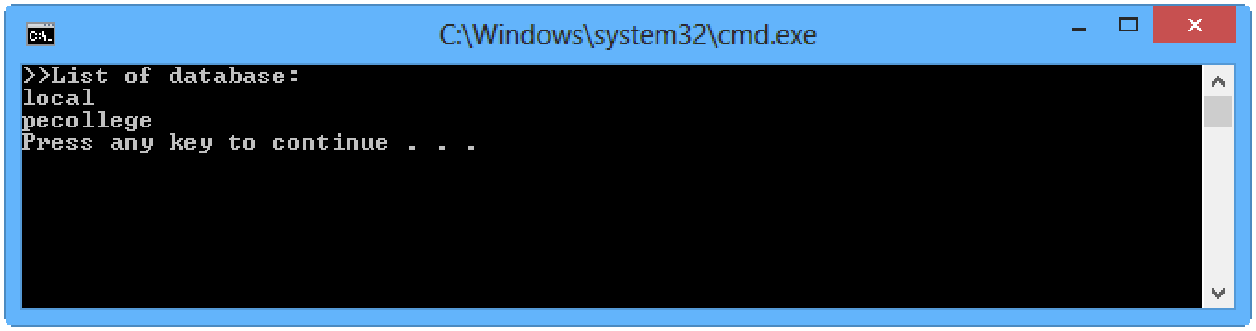
Figure 17: Creating a database in MongoDB

Getting the List of Databases

You may want to know the list of databases in the current MongoDB server. You can do this by calling **`GetDatabaseNames()`** from the **`MongoServer`** object.

The following example code gets a list of database names from the MongoDB server.

```
using System;
using System.Collections.Generic;
using MongoDB.Driver;

namespace Lab2
{
    class Program
    {
        static void Main(string[] args)
        {
            string connectionString = "mongodb://localhost";
            string databaseName = "csharp";

            MongoServer server = MongoServer.Create(connectionString);
            Console.WriteLine(">>List of database:");
            List<string> dbs = new List<string>(server.GetDatabaseNames());
            foreach (var dbName in dbs)
            {
                Console.WriteLine(dbName);
            }
        }

    }
}
```



```
>>List of database:
local
pecollege
Press any key to continue . . .
```

Figure 18: Getting a list of databases

Deleting a Database

To delete a database from the MongoDB server, call the **`Drop()`** method from the **`MongoDatabase`** object.

```
using System;
using System.Collections.Generic;
using MongoDB.Driver;

namespace Lab2
{
    class Program
    {
        static void Main(string[] args)
        {


            string connectionString = "mongodb://localhost";
            string databaseName = "csharp";

            MongoServer server = MongoServer.Create(connectionString);
            MongoDatabase database = server.GetDatabase(databaseName);

            // Drop database.
            Console.WriteLine(">>Drop database collection");
            database.Drop();

        }

    }
}
```

Chapter 3 Database Collection

In the conventional relational database management system (RDBMS) context, database collections are equivalent to tables. This chapter describes how to create, read, and delete MongoDB database collections using C#.

To do this, create a **Console Application** and add the following namespaces.

```
using System;
using System.Linq;
using MongoDB.Driver;
```

Creating a Database Collection

First, get the **`MongoDatabase`** object from the **`MongoServer`** object by passing a database connection string.

```
string connectionString = "mongodb://localhost";
string databaseName = "csharp";

Console.WriteLine(">>Create/Get database");
MongoServer server = MongoServer.Create(connectionString);
MongoDatabase database = server.GetDatabase(databaseName);
Console.WriteLine("Database name: {0}", database.Name);
```

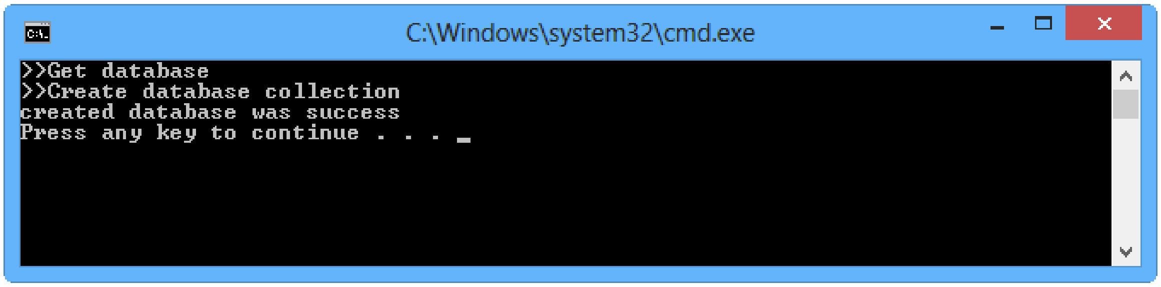
To create a database collection, use **`CreateCollection()`** of the **`MongoDatabase`** object. It returns the **`CommandResult`** object. If successful, you will get the **OK** value.

```
            // Create collection.
            Console.WriteLine(">>Create database collection");
            CommandResult result = database.CreateCollection("employee");
            if (result.Ok)
                Console.WriteLine("created database was success");
            else
                Console.WriteLine(result.ErrorMessage);
```

You can also create another database collection by running the following code.

```
            //Create another collection.
            database.CreateCollection("bank");
            database.CreateCollection("department");
            database.CreateCollection("branch");
```

You can see a sample of the program output in Figure 19.



```
>>Get database
>>Create database collection
created database was success
Press any key to continue . . .
```

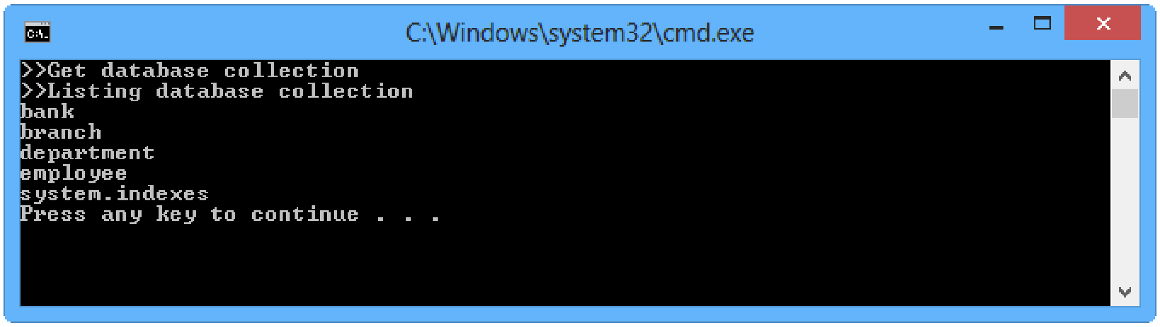
Figure 19: Creating a database collection

Reading a Database Collection

You can get a list of database collections from a database by calling **`GetCollectionNames()`** from the **`MongoDatabase`** object.

```
            // Get collection.
            Console.WriteLine(">>Get database collection");
            // Get collection list.
            Console.WriteLine(">>Listing database collection");
            foreach (string name in database.GetCollectionNames())
            {
                Console.WriteLine(name);
            }
```

The sample output for getting a list of database collection is shown in Figure 20.



```
>>Get database collection
>>Listing database collection
bank
branch
department
employee
system.indexes
Press any key to continue . . .
```

Figure 20: Getting a list of database collections

Deleting a Database Collection

You can delete a database collection by calling **`DropCollection()`** from the **`MongoDatabase`** object. To evaluate this operation, call **`CollectionExists()`** to check whether the collection still exists.

In the following example, the **bank** collection is deleted and then we check whether it still exists.

```
            // Delete collection.
            Console.WriteLine(">>Delete database collection");
            database.DropCollection("bank");
            // Check for collection.
            if (database.CollectionExists("bank"))
                Console.WriteLine("bank collection exists");
            else
                Console.WriteLine("bank collection was not found");
```

Chapter 4 Collection Data

In the conventional RDBMS context, database collection data refers to table row data. This chapter describes how to create, read, update, and delete MongoDB database collection data using C#.

To do this, create a **Console Application** and add the following namespaces.

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

Creating Collection Data

Before creating collection data, get the **`MongoDatabase`** object from the **`MongoServer`**.

```
            string connectionString = "mongodb://localhost";
            string databaseName = "csharp";

            // Get database.
            Console.WriteLine(">>Get database");
            MongoServer server = MongoServer.Create(connectionString);
            MongoDatabase database = server.GetDatabase(databaseName);
```

Create a method called **`CreateDemo()`** with the **`MongoDatabase`** parameter.

```
        private static void CreateDemo(MongoDatabase database)
        {
        }
```

Pass the **`database`** variable to this method.

```
CreateDemo(database);
```

There are two options to create data as a database collection. The first option is to use a BSON document to insert data into a collection, as shown in the following code example.

```
        private static void CreateDemo(MongoDatabase database)
        {
            // // Create data for five employees with BSON.
            Console.WriteLine(">>>>Create collection data by BSON Document");
            MongoCollection<BsonDocument> employees =
database.GetCollection("employee");
            for (int i = 1; i <= 5; i++)
            {
                BsonDocument employee = new BsonDocument {
                    { "name", "Employee " + i },
                    { "email", String.Format("email{0}@email.com", i) },
                    { "createddate", DateTime.Now }
                };

                employees.Insert(employee);
            }

        }
```

Explanation

- To be able to insert data into a collection, we need to have a reference to that collection object. This is achieved by calling the **`GetCollection()`** method with the collection name as the parameter (in our case, “employee”).

- This will return a list of **`BsonDocument`** objects.

A **`BsonDocument`** object is equivalent to a table row in a RDBMS database.

For example, insert an **`employee`** document into a collection with the following fields.

- ID
- name
- email
- createddate

Create a **`BsonDocument`** and pass the document fields in JSON format as shown in the following example.

```
         BsonDocument employee = new BsonDocument {
                    { "name", "Employee " + i },
                    { "email", String.Format("email{0}@email.com", i) },
                    { "createddate", DateTime.Now }
         };
```

This **`BsonDocument`** object can be inserted into a collection by calling **`Insert()`** from the collection object.

```
         employees.Insert(employee);
```

The other option to create data as a database collection is to use the entity object as a document object. For example, declare the **`Employee`** object as follows.

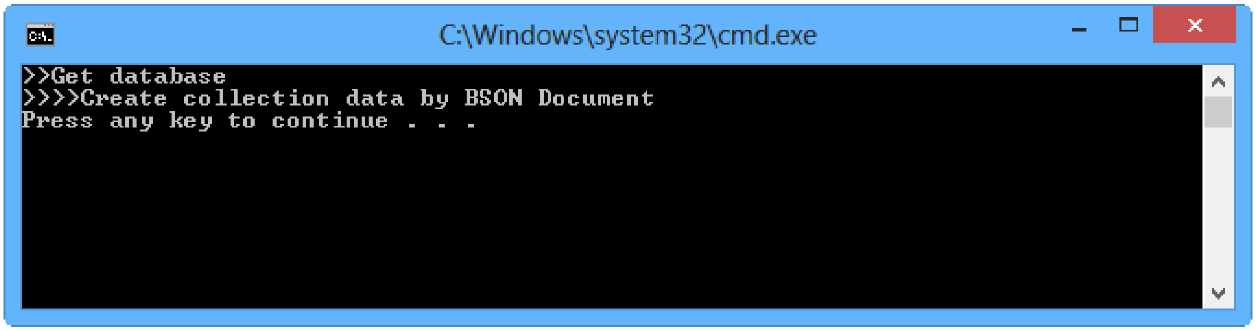
```
using System;
using MongoDB.Bson.Serialization.Attributes;
using MongoDB.Bson;

namespace Lab4
{
    public class Employee
    {
        [BsonElementAttribute("_id")]
        public ObjectId Id { set; get; }
        [BsonElementAttribute("name")]
        public string Name { set; get; }
        [BsonElementAttribute("email")]
        public string Email { set; get; }
        [BsonElementAttribute("createddate")]
        public DateTime CreateDate { set; get; }
    }
}
```

You can expose the class properties using **`BsonElementAttribute`** with the field names.

To insert a document object into a collection, we use the same approach we used for the BSON document. Get the collection list object and then call **`Insert()`** to insert the document object.

```
            Console.WriteLine(">>>>Create collection data by BSON Object");
            MongoCollection<Employee> employeeColl =
database.GetCollection<Employee>("employee");
            for (int i = 1; i <= 5; i++)
            {
                Employee obj = new Employee();
                obj.Id = ObjectId.GenerateNewId();
                obj.Name = "EmployeeObject " + i;
                obj.Email = String.Format("email{0}-object@email.com", i);
                obj.CreateDate = DateTime.Now;

                employeeColl.Insert(obj);
            }
```

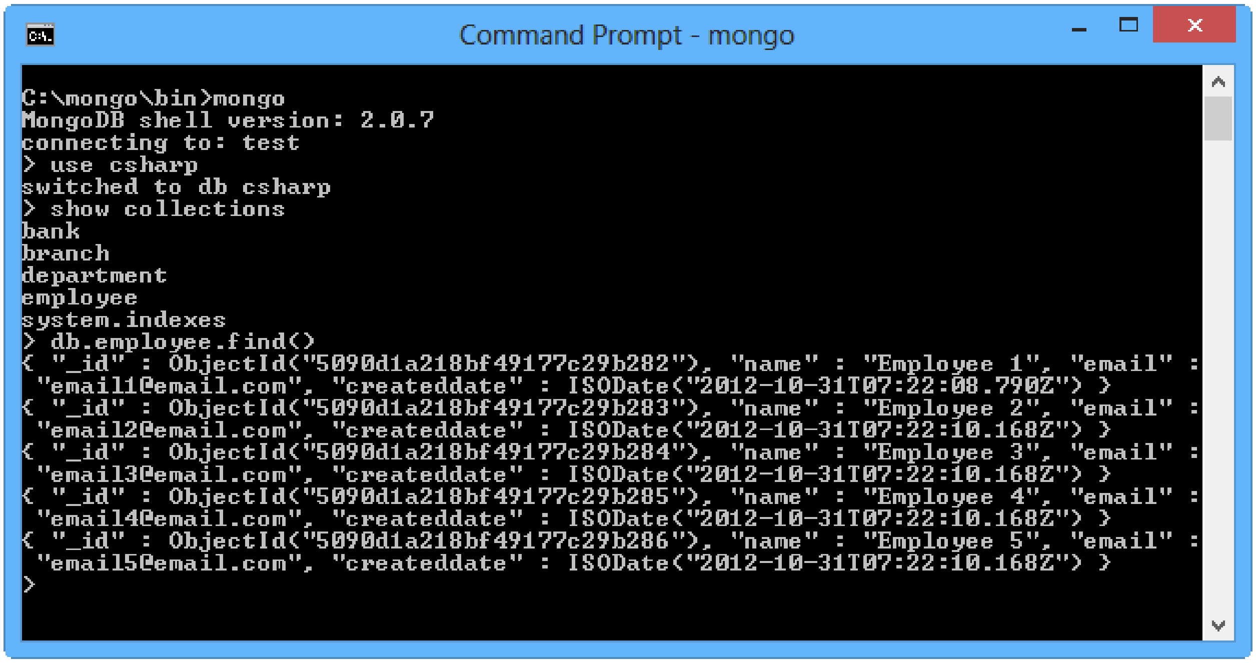
Running the previous code will get a response similar to the output in the following figure.



```
>>Get database
>>>>Create collection data by BSON Document
Press any key to continue . . .
```

Figure 21: Executing app for creating collection data

If you check the MongoDB database using the Mongo shell, you will get the inserted data.



```
C:\mongo\bin>mongo
MongoDB shell version: 2.0.7
connecting to: test
> use csharp
switched to db csharp
> show collections
bank
branch
department
employee
system.indexes
> db.employee.find()
{ "_id" : ObjectId("5090d1a218bf49177c29b282"), "name" : "Employee 1", "email" :
 "email1@email.com", "createddate" : ISODate("2012-10-31T07:22:08.790Z") }
{ "_id" : ObjectId("5090d1a218bf49177c29b283"), "name" : "Employee 2", "email" :
 "email2@email.com", "createddate" : ISODate("2012-10-31T07:22:10.168Z") }
{ "_id" : ObjectId("5090d1a218bf49177c29b284"), "name" : "Employee 3", "email" :
 "email3@email.com", "createddate" : ISODate("2012-10-31T07:22:10.168Z") }
{ "_id" : ObjectId("5090d1a218bf49177c29b285"), "name" : "Employee 4", "email" :
 "email4@email.com", "createddate" : ISODate("2012-10-31T07:22:10.168Z") }
{ "_id" : ObjectId("5090d1a218bf49177c29b286"), "name" : "Employee 5", "email" :
 "email5@email.com", "createddate" : ISODate("2012-10-31T07:22:10.168Z") }
>
```

Figure 22: Showing the inserted data using the Mongo shell

Reading Collection Data

To read all collection data, you can use **`FindAll()`** from the collection list object.

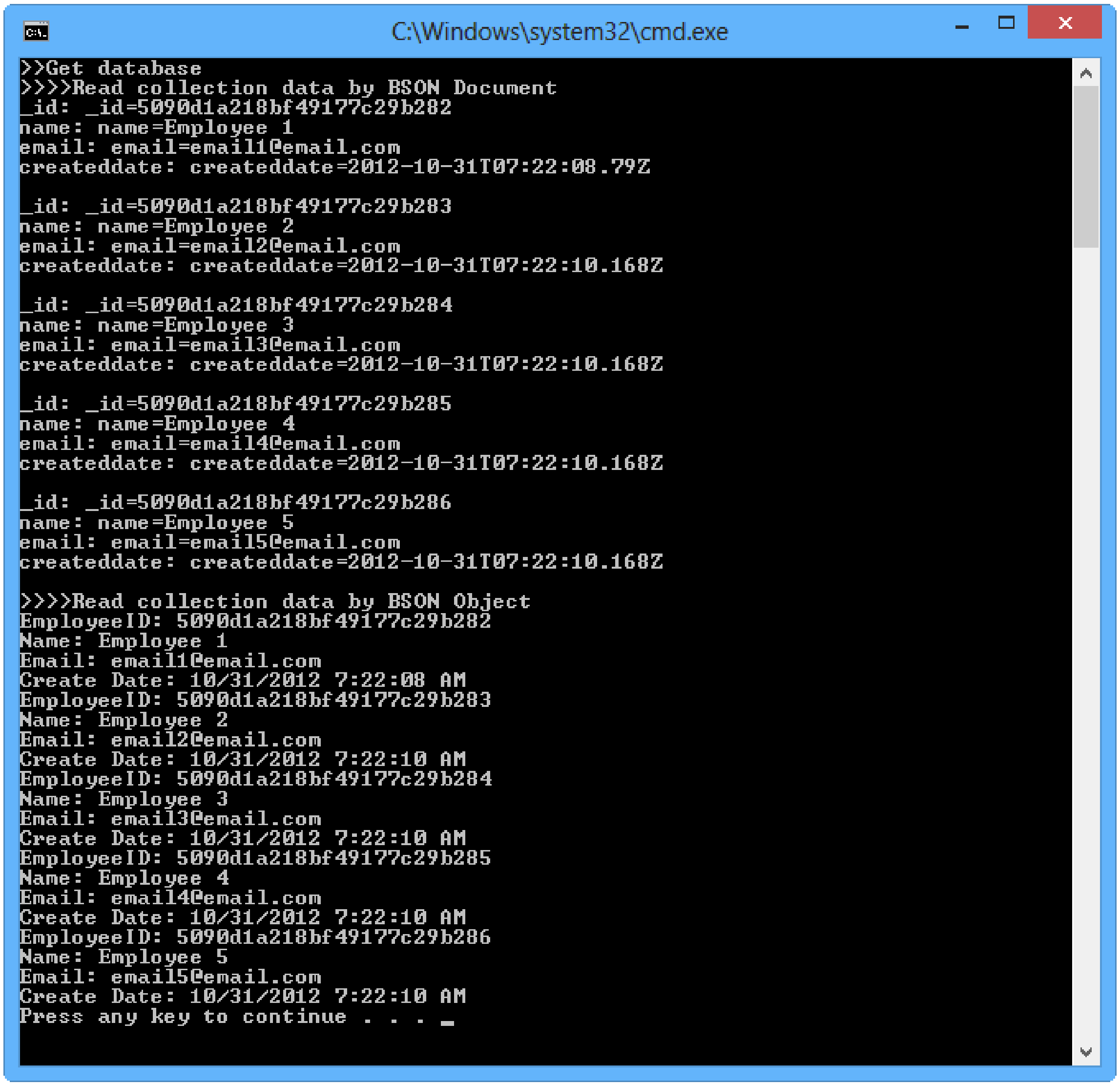
The following example shows all data using the BSON document.

```
            Console.WriteLine(">>>>Read collection data by BSON Document");
            MongoCollection<BsonDocument> employees =
database.GetCollection("employee");
            foreach(BsonDocument doc in employees.FindAll())
            {
                foreach (string name in doc.Names)
                {
                    BsonElement element = doc.GetElement(name);
                    Console.WriteLine("{0}: {1}", name, element);
                }
                Console.WriteLine("");
            }
```

Another solution is to use the entity object, for instance, **`Employee`**.

```
            Console.WriteLine(">>>>Read collection data by BSON Object");
            MongoCollection<Employee> employeeColl =
database.GetCollection<Employee>("employee");
            foreach (Employee doc in employeeColl.FindAll())
            {
                Console.WriteLine("EmployeeID: {0}", doc.Id.ToString());
                Console.WriteLine("Name: {0}", doc.Name);
                Console.WriteLine("Email: {0}", doc.Email);
                Console.WriteLine("Create Date: {0}", doc.CreateDate);
            }
```

If you run the previous code, you will get a list of collection data as shown in Figure 23.



```
>>Get database
>>>>Read collection data by BSON Document
_id: _id=5090d1a218bf49177c29b282
name: name=Employee 1
email: email=email1@email.com
createddate: createddate=2012-10-31T07:22:08.79Z

_id: _id=5090d1a218bf49177c29b283
name: name=Employee 2
email: email=email2@email.com
createddate: createddate=2012-10-31T07:22:10.168Z

_id: _id=5090d1a218bf49177c29b284
name: name=Employee 3
email: email=email3@email.com
createddate: createddate=2012-10-31T07:22:10.168Z

_id: _id=5090d1a218bf49177c29b285
name: name=Employee 4
email: email=email4@email.com
createddate: createddate=2012-10-31T07:22:10.168Z

_id: _id=5090d1a218bf49177c29b286
name: name=Employee 5
email: email=email5@email.com
createddate: createddate=2012-10-31T07:22:10.168Z

>>>>Read collection data by BSON Object
EmployeeID: 5090d1a218bf49177c29b282
Name: Employee 1
Email: email1@email.com
Create Date: 10/31/2012 7:22:08 AM
EmployeeID: 5090d1a218bf49177c29b283
Name: Employee 2
Email: email2@email.com
Create Date: 10/31/2012 7:22:10 AM
EmployeeID: 5090d1a218bf49177c29b284
Name: Employee 3
Email: email3@email.com
Create Date: 10/31/2012 7:22:10 AM
EmployeeID: 5090d1a218bf49177c29b285
Name: Employee 4
Email: email4@email.com
Create Date: 10/31/2012 7:22:10 AM
EmployeeID: 5090d1a218bf49177c29b286
Name: Employee 5
Email: email5@email.com
Create Date: 10/31/2012 7:22:10 AM
Press any key to continue . . . _
```

Figure 23: Running app for retrieving data

Updating Collection Data

In this section, you will update the collection in two ways: using the BSON document and using the entity object.

You can start by getting a collection list object. For example, let’s update a single collection. To do this, you can use **`FindOne()`** to get the **`BsonDocument`** object.

```
            Console.WriteLine(">>>>Update collection data by BSON Document");
            MongoCollection<BsonDocument> employees =
database.GetCollection("employee");
            BsonDocument doc = employees.FindOne();
```

Print this object in the console.

```
            // Show data.
            foreach (string name in doc.Names)
            {
                BsonElement element = doc.GetElement(name);
                Console.WriteLine("{0}: {1}", name, element.Value);
            }
```

Now update this object by calling `Set()`. To save the changes, you can call `Save()`.

```
            doc.Set("name", BsonValue.Create("update-employee"));
            doc.Set("email", BsonValue.Create("update-employee@email.com"));

            employees.Save(doc);
            Console.WriteLine("updated data was success");
```

If you use the entity object, you can start by getting the entity collection list object.

```
            Console.WriteLine(">>>>Update collection data by BSON object");
            MongoCollection<Employee> employeeColl =
database.GetCollection<Employee>("employee");
            Employee emp = employeeColl.FindOne();
```

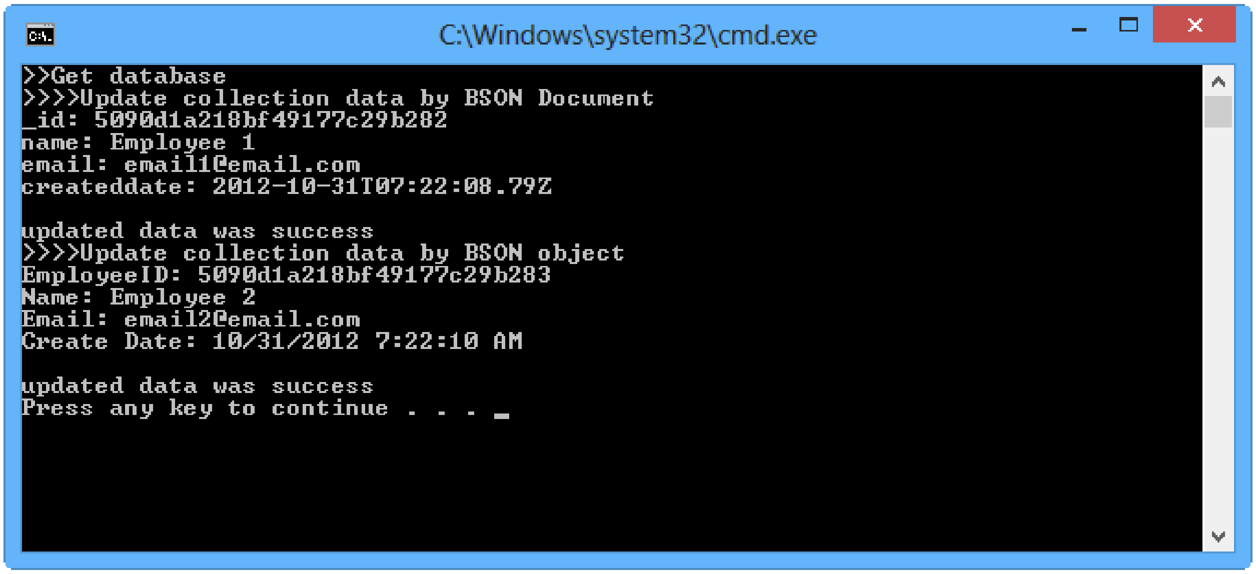
Show this data in the console.

```
            // Show data.
            Console.WriteLine("EmployeeID: {0}", emp.Id.ToString());
            Console.WriteLine("Name: {0}", emp.Name);
            Console.WriteLine("Email: {0}", emp.Email);
            Console.WriteLine("Create Date: {0}", emp.CreateDate);
            Console.WriteLine("");
```

The following example shows how to update the collection data.

```
            emp.Name = "update-employee-object";
            emp.Email = "update-employee-object@email.com";

            employeeColl.Save(emp);
            Console.WriteLine("updated data was success");
```

You can see the program output in the following figure.



```
>>Get database
>>>>Update collection data by BSON Document
_id: 5090d1a218bf49177c29b282
name: Employee 1
email: email1@email.com
createddate: 2012-10-31T07:22:08.79Z

updated data was success
>>>>Update collection data by BSON object
EmployeeID: 5090d1a218bf49177c29b283
Name: Employee 2
Email: email2@email.com
Create Date: 10/31/2012 7:22:10 AM

updated data was success
Press any key to continue . . . _
```

Figure 24: Updating collection data

Deleting Collection Data

It is easy to delete collection data. You can use **`Drop()`** to delete all collection data. In the following example, the **`Drop()`** method returns an **Ok** value if deleting the data is successful.

```
            Console.WriteLine(">>>>Delete all data in employee collection");
            MongoCollection<BsonDocument> employees =
database.GetCollection("employee");
            CommandResult result = employees.Drop();
            if (result.Ok)
                Console.WriteLine("deleted all data in employee collection
was success");
            else
                Console.WriteLine(result.ErrorMessage);
```

If you want to delete specific data, you can use the **`Query`** object which is covered in the next chapter.

Chapter 5 Finding and Querying Data

This chapter describes how to find data in a database collection using C#. To do this, we create a **Console Application**.

Once created, include the following namespaces.

```
using System;
using MongoDB.Driver;
using MongoDB.Bson;
using MongoDB.Driver.Builders;
```

Finding Data

MongoDB is a NoSQL database, so if you want to find data you can use the **`FindOne()`** and **`Find()`** methods.

First, prepare the database object for MongoDB.

```
            string connectionString = "mongodb://localhost";
            string databaseName = "csharp";

            // Get database.
            Console.WriteLine(">>Get database");
            MongoServer server = MongoServer.Create(connectionString);
            MongoDatabase database = server.GetDatabase(databaseName);
```

Next, generate data for testing.

```
            GenerateData(database);
```

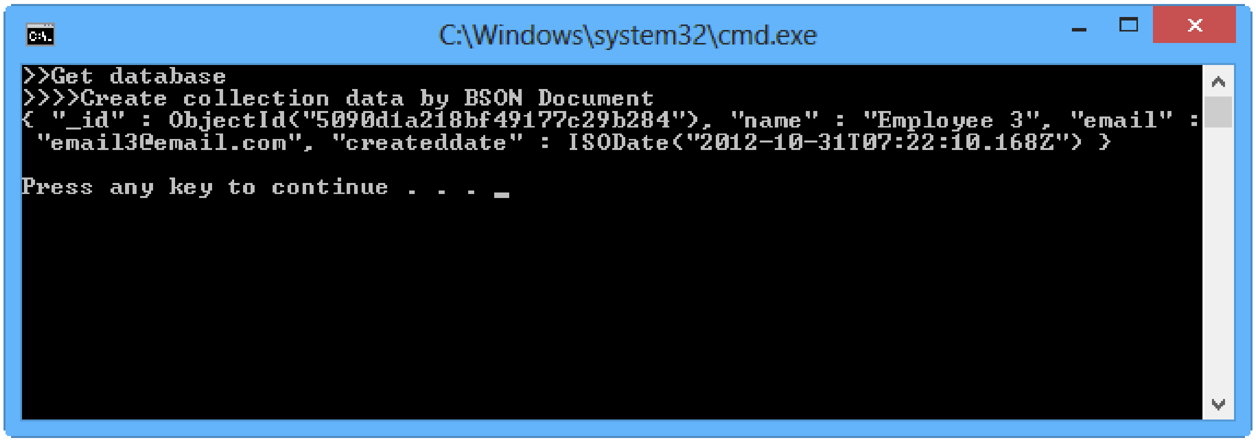
The following example is an implementation of the **`GenerateData()`** method used to build data that includes ten employees, each with a **`level`** field that can be **`Architect`** or **`Programmer`**.

```
       private static void GenerateData(MongoDatabase database)
        {
            // // Create data for ten employees with BSON.
            Console.WriteLine(">>>>Create collection data by BSON Document");
            MongoCollection<BsonDocument> employees =
database.GetCollection("employee");
            for (int i = 1; i <= 10; i++)
            {
                if (i < 5)
                {
                    BsonDocument employee = new BsonDocument {
                        { "name", "Employee " + i },
                        { "email", String.Format("email{0}@email.com", i) },
                        { "level", "Programmer" },
                        { "createddate", DateTime.Now }
                    };
                    employees.Insert(employee);
                }
                else
                {
                    BsonDocument employee = new BsonDocument {
                        { "name", "Employee " + i },
                        { "email", String.Format("email{0}@email.com", i) },
                        { "level", "Architect" },
                        { "createddate", DateTime.Now }
                    };
                    employees.Insert(employee);
                }
            }


        }
```

To find a single piece of data, use **`FindOne()`** from the collection object. **`FindOne()`** returns one document that satisfies the specified query criteria. If more than one document satisfies the query, this method returns the first document.

The following example demonstrates finding a single document. The output is shown in Figure 25.

```
            MongoCollection<BsonDocument> employees =
database.GetCollection("employee");

            // Find one.
            BsonDocument employee = employees.FindOne();
            Console.WriteLine(employee.ToString());
            Console.WriteLine("");
```



```
>>Get database
>>>>Create collection data by BSON Document
{ "_id" : ObjectId("5090d1a218bf49177c29b284"), "name" : "Employee 3", "email" :
 "email3@email.com", "createddate" : ISODate("2012-10-31T07:22:10.168Z") }

Press any key to continue . . . _
```

Figure 25: Finding a single document

Query

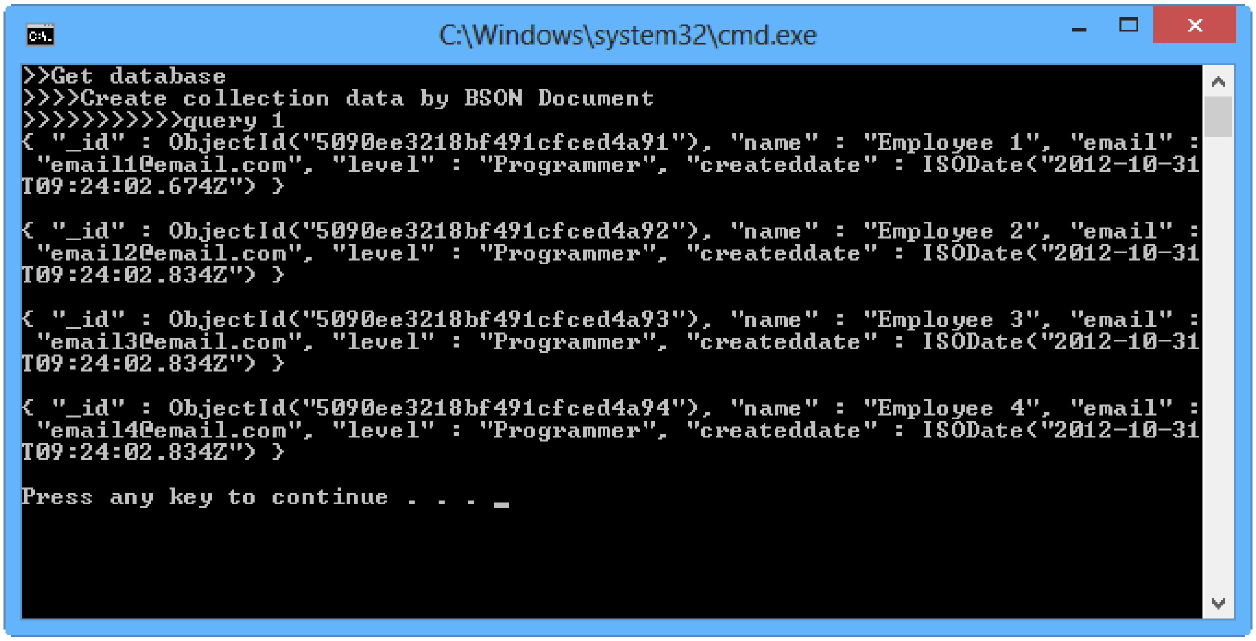
In the previous section, we searched for individual data without criteria. Now we will find data based on criteria.

To query data in MongoDB, you can use the **`QueryDocument`** object and pass it key and value parameters.

Create a **`QueryDocument`** object for data with the level "**`Programmer`**". After that, pass the **`QueryDocument`** object to the **`Find()`** method and return a list of documents. Display this list in the console.

```
            Console.WriteLine(">>>>>>>>>>>query 1");
            var query1 = new QueryDocument("level", "Programmer");
            foreach (BsonDocument emp in employees.Find(query1))
            {
                Console.WriteLine(emp.ToString());
                Console.WriteLine("");
            }
```

If you run the previous code, you will get data filtered to the programmer level, as shown in Figure 26.



```
>>Get database
>>>>Create collection data by BSON Document
>>>>>>>>>>>query 1
{ "_id" : ObjectId("5090ee3218bf491cfced4a91"), "name" : "Employee 1", "email" :
 "email1@email.com", "level" : "Programmer", "createddate" : ISODate("2012-10-31
T09:24:02.674Z") }

{ "_id" : ObjectId("5090ee3218bf491cfced4a92"), "name" : "Employee 2", "email" :
 "email2@email.com", "level" : "Programmer", "createddate" : ISODate("2012-10-31
T09:24:02.834Z") }

{ "_id" : ObjectId("5090ee3218bf491cfced4a93"), "name" : "Employee 3", "email" :
 "email3@email.com", "level" : "Programmer", "createddate" : ISODate("2012-10-31
T09:24:02.834Z") }

{ "_id" : ObjectId("5090ee3218bf491cfced4a94"), "name" : "Employee 4", "email" :
 "email4@email.com", "level" : "Programmer", "createddate" : ISODate("2012-10-31
T09:24:02.834Z") }

Press any key to continue . . . _
```

Figure 26: Finding with criteria

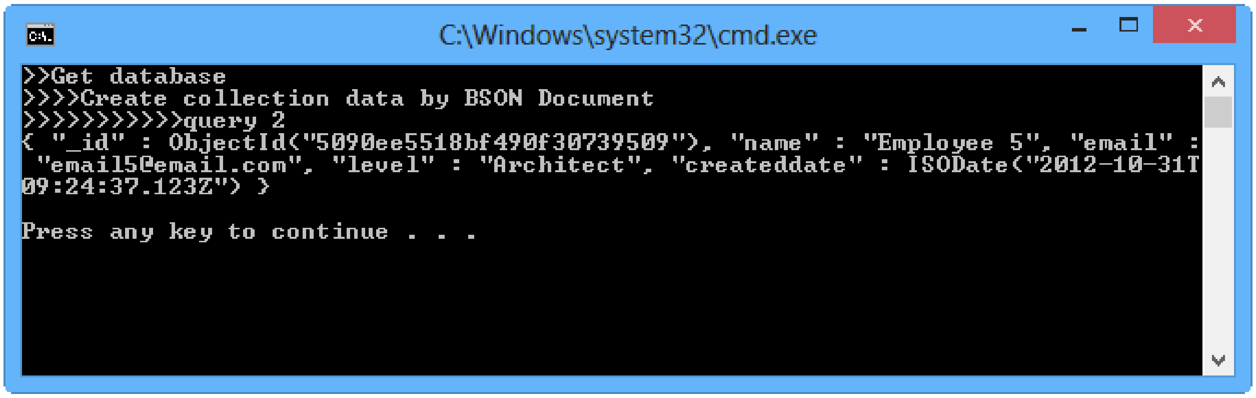
You can also use the **Query** object to query data. It provides many methods such as **EQ** (equal), **And**, **Or**, etc. These methods are useful for building query criteria.

For example, query data with the name "**Employee 5**" and call the **EQ()** method. Running the following code will return a response as shown in Figure 27.

```
Console.WriteLine(">>>>>>>>>>>query 2");
var query2 = Query.EQ("name", BsonValue.Create("Employee 5"));
BsonDocument emp2 = employees.FindOne(query2);
Console.WriteLine(emp2.ToString());
Console.WriteLine("");
```

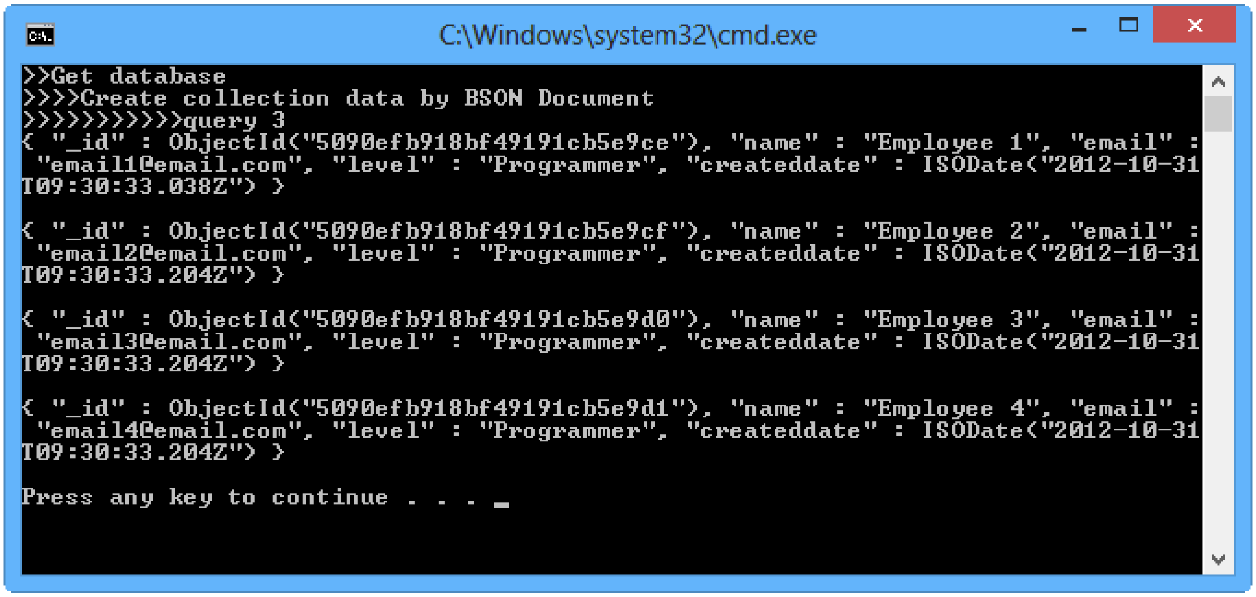
Try a query with **and** criteria. You can use the **And()** method of the query object. Running the following code will return the program output shown in Figure 28.

```
Console.WriteLine(">>>>>>>>>>>query 3");
var query3 = Query.And(
        Query.NE("name", BsonValue.Create("Employee")),
        Query.EQ("level", BsonValue.Create("Programmer"))
    );
foreach (BsonDocument emp in employees.Find(query3))
{
    Console.WriteLine(emp.ToString());
    Console.WriteLine("");
}
```



```
>>Get database
>>>>Create collection data by BSON Document
>>>>>>>>>>>>query 2
{ "_id" : ObjectId("5090ee5518bf490f30739509"), "name" : "Employee 5", "email" :
 "email5@email.com", "level" : "Architect", "createddate" : ISODate("2012-10-31T
09:24:37.123Z") }

Press any key to continue . . .
```

Figure 27: Querying data with EQ()



```
>>Get database
>>>>Create collection data by BSON Document
>>>>>>>>>>>>query 3
{ "_id" : ObjectId("5090efb918bf49191cb5e9ce"), "name" : "Employee 1", "email" :
 "email1@email.com", "level" : "Programmer", "createddate" : ISODate("2012-10-31
T09:30:33.038Z") }

{ "_id" : ObjectId("5090efb918bf49191cb5e9cf"), "name" : "Employee 2", "email" :
 "email2@email.com", "level" : "Programmer", "createddate" : ISODate("2012-10-31
T09:30:33.204Z") }

{ "_id" : ObjectId("5090efb918bf49191cb5e9d0"), "name" : "Employee 3", "email" :
 "email3@email.com", "level" : "Programmer", "createddate" : ISODate("2012-10-31
T09:30:33.204Z") }

{ "_id" : ObjectId("5090efb918bf49191cb5e9d1"), "name" : "Employee 4", "email" :
 "email4@email.com", "level" : "Programmer", "createddate" : ISODate("2012-10-31
T09:30:33.204Z") }

Press any key to continue . . . _
```

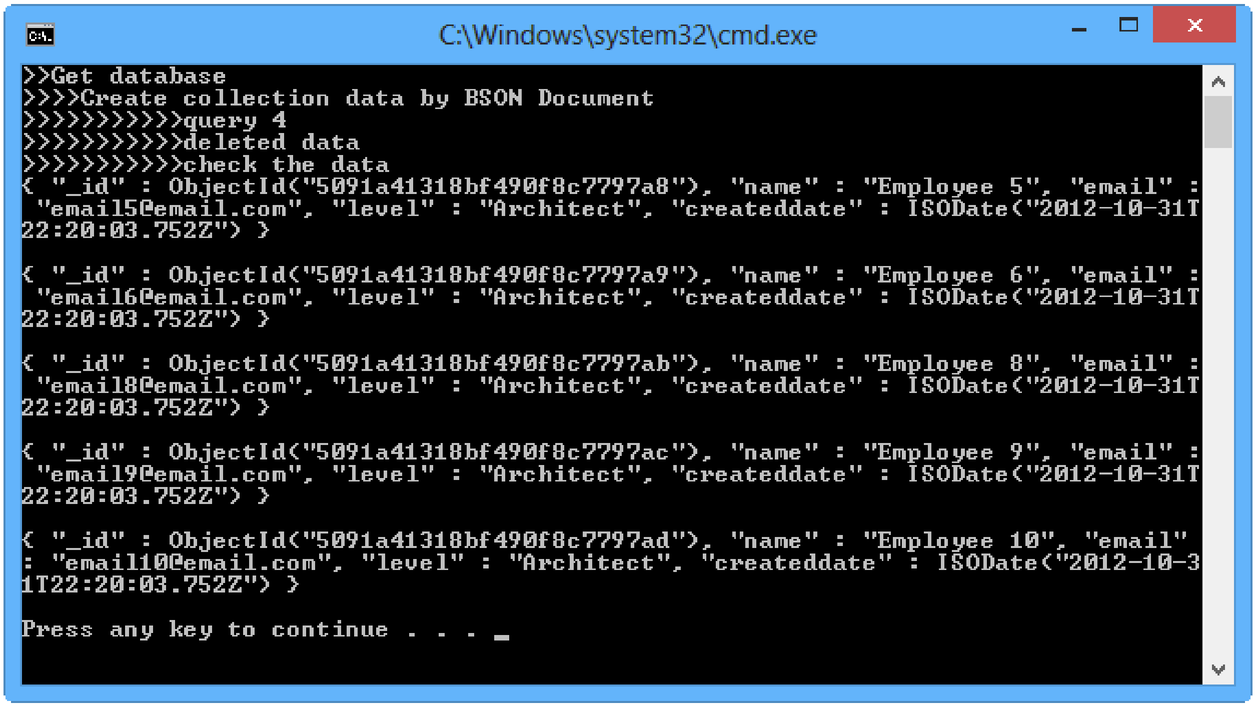
Figure 28: Querying data with EQ() and NE()

Query and Remove

Now we can remove data based on specific criteria.

For example, if you want to delete data that has the name "**`Employee 7`**" or the level "**`Programmer`**", use the following code example. You will get the output shown in Figure 29.

```
            Console.WriteLine(">>>>>>>>>>>>query 4");
            var query4 = Query.Or(
                    Query.EQ("name", BsonValue.Create("Employee 7")),
                    Query.EQ("level", BsonValue.Create("Programmer"))
                );
            employees.Remove(query4);
            foreach (BsonDocument emp in employees.FindAll())
            {
                Console.WriteLine(emp.ToString());
                Console.WriteLine("");
            }
```



```
>>Get database
>>>>Create collection data by BSON Document
>>>>>>>>>>>query 4
>>>>>>>>>>>>deleted data
>>>>>>>>>>>check the data
{ "_id" : ObjectId("5091a41318bf490f8c7797a8"), "name" : "Employee 5", "email" :
 "email5@email.com", "level" : "Architect", "createddate" : ISODate("2012-10-31T
22:20:03.752Z") }

{ "_id" : ObjectId("5091a41318bf490f8c7797a9"), "name" : "Employee 6", "email" :
 "email6@email.com", "level" : "Architect", "createddate" : ISODate("2012-10-31T
22:20:03.752Z") }

{ "_id" : ObjectId("5091a41318bf490f8c7797ab"), "name" : "Employee 8", "email" :
 "email8@email.com", "level" : "Architect", "createddate" : ISODate("2012-10-31T
22:20:03.752Z") }

{ "_id" : ObjectId("5091a41318bf490f8c7797ac"), "name" : "Employee 9", "email" :
 "email9@email.com", "level" : "Architect", "createddate" : ISODate("2012-10-31T
22:20:03.752Z") }

{ "_id" : ObjectId("5091a41318bf490f8c7797ad"), "name" : "Employee 10", "email"
: "email10@email.com", "level" : "Architect", "createddate" : ISODate("2012-10-3
1T22:20:03.752Z") }

Press any key to continue . . . _
```

Figure 29: Removing data based on criteria

Chapter 6 Binary and Image Collection Data

In this chapter, we're going to learn how to work with binary and image data in MongoDB.

First, create a **Console Application** and include the following namespaces.

```
using System;
using MongoDB.Driver;
using MongoDB.Bson;
using MongoDB.Driver.Builders;
using MongoDB.Driver.GridFS;
using System.IO;
```

Inserting an Image or Binary File

In the first scenario, you will learn to insert an image file into the MongoDB database.

Use the collection **sign** for image manipulation.

```
            string connectionString = "mongodb://localhost";
            string databaseName = "csharp";

            // Get database.
            Console.WriteLine(">>Get database");
            MongoServer server = MongoServer.Create(connectionString);
            MongoDatabase database = server.GetDatabase(databaseName);

            MongoCollection<BsonDocument> signs =
database.GetCollection("sign");
```

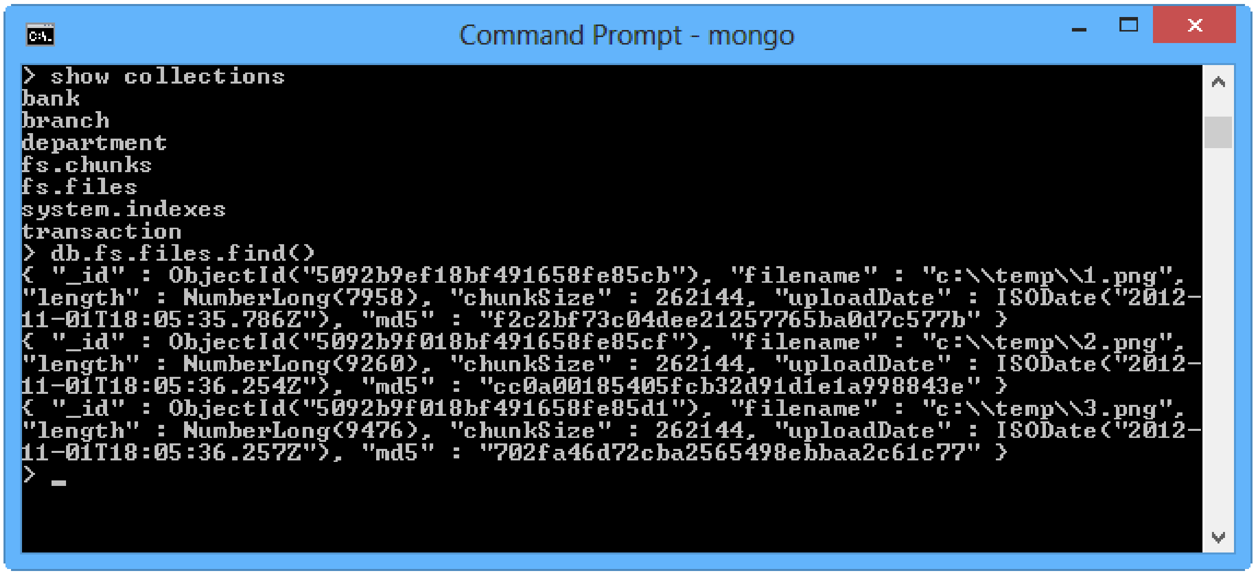
To work with a binary file such as an image, document file, or some other binary file, you can use the **MongoGridFS** object. **GridFS** is a specification for storing and retrieving files bigger than 16 MB, which is the BSON document size limit. **GridFS** by default will store the files in two separate collections:

- **fs.chunks**: Stores the file as binary chunks.
- **fs.files**: Stores the file's metadata.

We can insert a binary file by calling **Upload()**. In the following example, we insert three image files through **GridFS**.

```
            // Insert image file in GridFS.
            Console.WriteLine(">>Insert file into GridFS");
            MongoGridFS gfs = new MongoGridFS(database);
            MongoGridFSFileInfo gfsi1 = gfs.Upload(@"c:\temp\1.png");
            MongoGridFSFileInfo gfsi2 = gfs.Upload(@"c:\temp\2.png");
            MongoGridFSFileInfo gfsi3 = gfs.Upload(@"c:\temp\3.png");
```

If successful, you can see these image files inside MongoDB. Figure 30 shows the Mongo shell displaying the image files, names, sizes, and checksums.



```
> show collections
bank
branch
department
fs.chunks
fs.files
system.indexes
transaction
> db.fs.files.find()
{ "_id" : ObjectId("5092b9ef18bf491658fe85cb"), "filename" : "c:\\temp\\1.png",
"length" : NumberLong(7958), "chunkSize" : 262144, "uploadDate" : ISODate("2012-
11-01T18:05:35.786Z"), "md5" : "f2c2bf73c04dee21257765ba0d7c577b" }
{ "_id" : ObjectId("5092b9f018bf491658fe85cf"), "filename" : "c:\\temp\\2.png",
"length" : NumberLong(9260), "chunkSize" : 262144, "uploadDate" : ISODate("2012-
11-01T18:05:36.254Z"), "md5" : "cc0a00185405fcb32d91d1e1a998843e" }
{ "_id" : ObjectId("5092b9f018bf491658fe85d1"), "filename" : "c:\\temp\\3.png",
"length" : NumberLong(9476), "chunkSize" : 262144, "uploadDate" : ISODate("2012-
11-01T18:05:36.257Z"), "md5" : "702fa46d72cba2565498ebbaa2c61c77" }
> _
```

Figure 30: Showing image files in MongoDB using the Mongo shell

Mapping GridFS and Collection Data

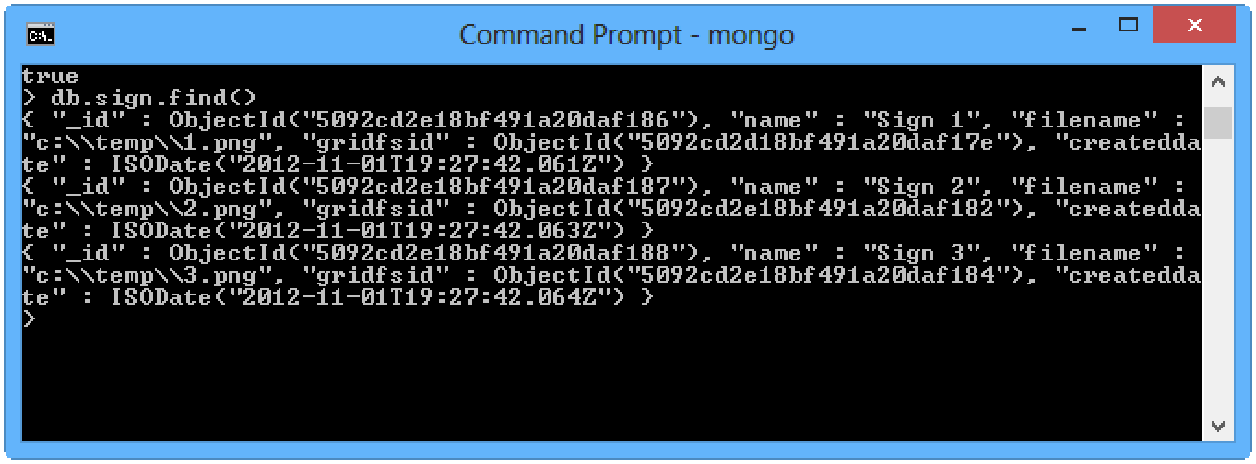
After inserting a binary file in GridFS, we have to map the GridFS ID to our collection data. This is important because GridFS doesn't provide the data information as a document in a collection. GridFS only provides information related to binary data information, such as the size and checksum.

An easier way to map between GridFS and a document is to attach the GridFS ID in one of document fields. For example, examine the following code.

```
            // Map image file into document.
            Console.WriteLine(">>Map GridFS file and collection document");
            BsonDocument sign1 = new BsonDocument {
                    { "name", "Sign 1" },
                    { "filename", gfsi1.Name },
                    { "gridfsid", gfsi1.Id.AsObjectId },
                    { "createddate", DateTime.Now }
                };
            signs.Insert(sign1);
            BsonDocument sign2 = new BsonDocument {
                    { "name", "Sign 2" },
                    { "filename", gfsi2.Name },
                    { "gridfsid", gfsi2.Id.AsObjectId },
                    { "createddate", DateTime.Now }
                };
            signs.Insert(sign2);
            BsonDocument sign3 = new BsonDocument {
                    { "name", "Sign 3" },
                    { "filename", gfsi3.Name },
                    { "gridfsid", gfsi3.Id.AsObjectId },
                    { "createddate", DateTime.Now }
                };
            signs.Insert(sign3);
```

The **gridfsid** field consists of the GridFS ID with the **ObjectId** data type.

Run the previous code example to see the sign collection in MongoDB, as shown in Figure 31.



```
true
> db.sign.find()
{ "_id" : ObjectId("5092cd2e18bf491a20daf186"), "name" : "Sign 1", "filename" :
"c:\\temp\\1.png", "gridfsid" : ObjectId("5092cd2d18bf491a20daf17e"), "createdda
te" : ISODate("2012-11-01T19:27:42.061Z") }
{ "_id" : ObjectId("5092cd2e18bf491a20daf187"), "name" : "Sign 2", "filename" :
"c:\\temp\\2.png", "gridfsid" : ObjectId("5092cd2e18bf491a20daf182"), "createdda
te" : ISODate("2012-11-01T19:27:42.063Z") }
{ "_id" : ObjectId("5092cd2e18bf491a20daf188"), "name" : "Sign 3", "filename" :
"c:\\temp\\3.png", "gridfsid" : ObjectId("5092cd2e18bf491a20daf184"), "createdda
te" : ISODate("2012-11-01T19:27:42.064Z") }
>
```

Figure 31: Mapping GridFS IDs to document field

Reading GridFS Data

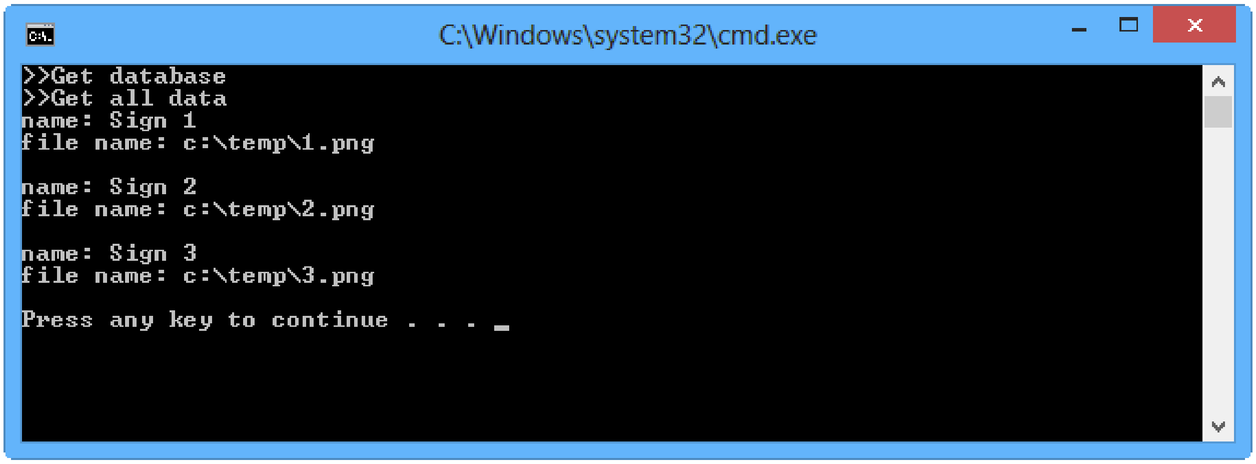
To get binary data from GridFS, you can use the **Download()** method with specific criteria.

You have already mapped the GridFS IDs into one of the document fields, so you can use the GridFS ID to retrieve the GridFS object.

Use the following code.

```
            // Get all data.
            Console.WriteLine(“>>Get all data”);
            foreach (BsonDocument sign in signs.FindAll())
            {
                string 45ilename =
sign.GetElement(“filename”).Value.AsString;
                string signName = sign.GetElement(“name”).Value.AsString;
                ObjectId gridfsId =
sign.GetElement(“gridfsid”).Value.AsObjectId;

                Console.WriteLine(“name: {0}”,signName);
                Console.WriteLine(“file name: {0}”, 45ilename);
                Console.WriteLine(“”);

                gfs.Download(“c:/temp/download-“ +
Path.GetFileName(45ilename), Query.EQ(“_id”, BsonValue.Create(gridfsId)));
            }
```

This example retrieves GridFS data by calling the **`Download()`** method. The data can then be stored in a folder.



```
>>Get database
>>Get all data
name: Sign 1
file name: c:\temp\1.png

name: Sign 2
file name: c:\temp\2.png

name: Sign 3
file name: c:\temp\3.png

Press any key to continue . . .
```

Figure 32: Showing GridFS and collection

Deleting GridFS Data

You can delete GridFS data by calling **`DeleteById()`** and passing the GridFS IDs to this method.

The following code example deletes data from GridFS.

```
            // Remove file by Id.
            Console.WriteLine(">>Delete all");
            gfs.DeleteById(gfsi1.Id.AsObjectId);
            gfs.DeleteById(gfsi2.Id.AsObjectId);
            gfs.DeleteById(gfsi3.Id.AsObjectId);
```

Chapter 7 Embedded Document

In this chapter, you will learn how to work with an embedded document in MongoDB.

Embedded documents are single document structures that embed document structures as sub-documents in a field or array within a document.

Rather than linking various sub-documents by their respective identifiers which usually happens in relational databases, embedded documents contain the full data structure. The advantage of embedded documents is that this model allows applications to retrieve data in a single database request.

Use the console application for testing. Write the following namespace and don't forget to add a reference to the MongoDB driver for C#.

```
using System;
using MongoDB.Driver;
using MongoDB.Bson;
using MongoDB.Driver.Builders;
```

Preparation

Prepare your database and collection with the following code.

```
            string connectionString = "mongodb://localhost";
            string databaseName = "csharp";

            // Get database.
            Console.WriteLine(">>Get database");
            MongoServer server = MongoServer.Create(connectionString);
            MongoDatabase database = server.GetDatabase(databaseName);
```

You can change the database connection string.

Simple Embedded Document

In this scenario, we’re building an application to:

- Create an embedded document.
- Read all embedded documents.
- Find and query.

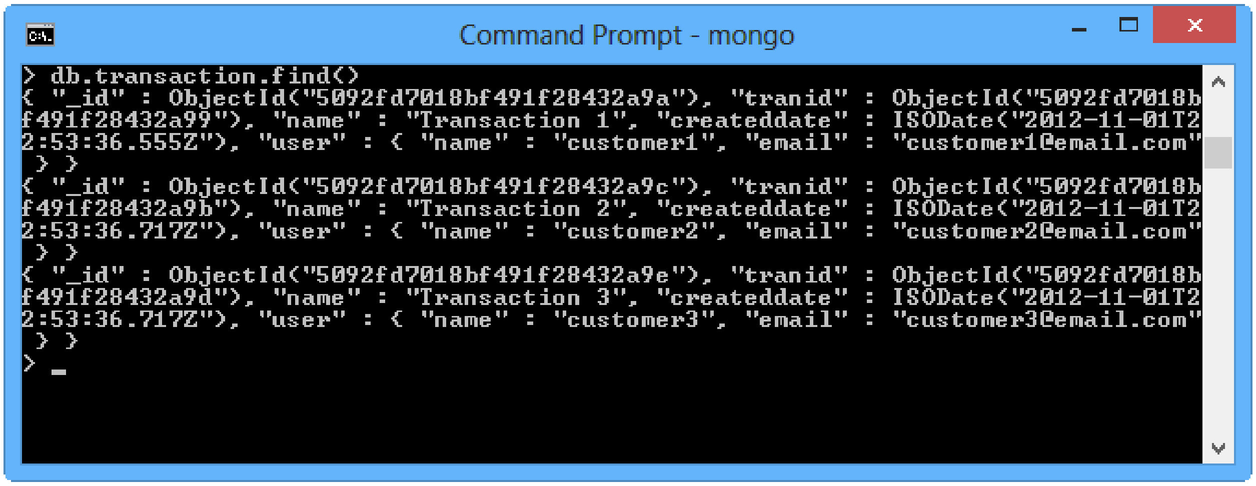
What does an embedded document look like? The following code example is a simple embedded document in JSON format where the user field contains an embedded sub-document.

```
    {
        'tranid': 1,
        'name': 'abcdedf',
        'createddate': '2/10/2012',
        'user': {
            'name': 'mr.abc',
            'email': 'abc@email.com'
        }
    }
```

Now you can insert a simple embedded document. The following code example does this.

```
        private static void CreateDemo(MongoDatabase database)
        {
            // // Create data for 3 transaction with BSON.
            Console.WriteLine(">>>>Create collection data by BSON Document");
            MongoCollection<BsonDocument> transactions = 
database.GetCollection("transaction");
            for (int i = 1; i <= 3; i++)
            {
                BsonDocument transaction = new BsonDocument {
                    { "tranid", ObjectId.GenerateNewId() },
                    { "name", String.Format("Transaction {0}", i) },
                    { "createddate", DateTime.Now },
                    { "user", new BsonDocument{
                            {"name", "customer" + i},
                            {"email", 
string.Format("customer{0}@email.com",i)}
                        }
                    }
                };

                transactions.Insert(transaction);
            }

        }
```

Run this code, and then check it on the MongoDB database. You should get the inserted data. You can see the sample data in Figure 33.



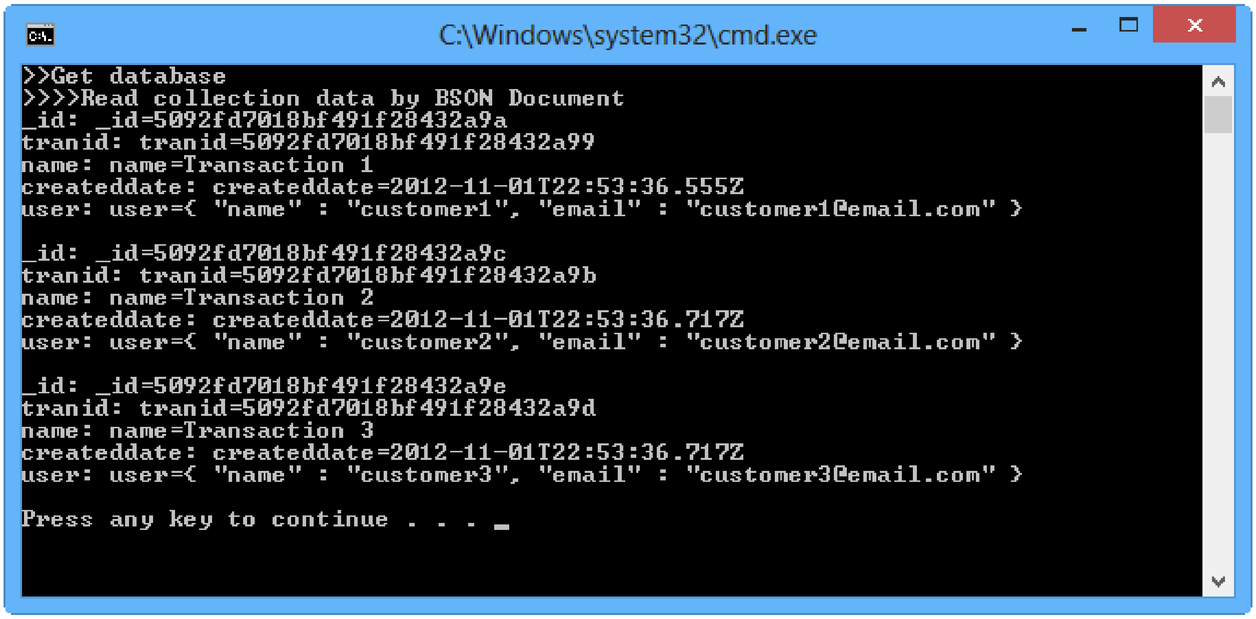
```
> db.transaction.find()
{ "_id" : ObjectId("5092fd7018bf491f28432a9a"), "tranid" : ObjectId("5092fd7018b
f491f28432a99"), "name" : "Transaction 1", "createddate" : ISODate("2012-11-01T2
2:53:36.555Z"), "user" : { "name" : "customer1", "email" : "customer1@email.com"
 } }
{ "_id" : ObjectId("5092fd7018bf491f28432a9c"), "tranid" : ObjectId("5092fd7018b
f491f28432a9b"), "name" : "Transaction 2", "createddate" : ISODate("2012-11-01T2
2:53:36.717Z"), "user" : { "name" : "customer2", "email" : "customer2@email.com"
 } }
{ "_id" : ObjectId("5092fd7018bf491f28432a9e"), "tranid" : ObjectId("5092fd7018b
f491f28432a9d"), "name" : "Transaction 3", "createddate" : ISODate("2012-11-01T2
2:53:36.717Z"), "user" : { "name" : "customer3", "email" : "customer3@email.com"
 } }
> _
```

Figure 33: Inserting an embedded document

To read your embedded document, get the normal document and then go deep into the embedded document.

```
        private static void ReadDemo(MongoDatabase database)
        {
            Console.WriteLine(">>>>Read collection data by BSON Document");
            MongoCollection<BsonDocument> transactions =
database.GetCollection("transaction");
            foreach (BsonDocument doc in transactions.FindAll())
            {
                foreach (string name in doc.Names)
                {
                    BsonElement element = doc.GetElement(name);
                    Console.WriteLine("{0}: {1}", name, element);
                }
                Console.WriteLine("");
            }
        }
```

Running the previous example will get a list of documents. The output is shown in Figure 34.



```
>>Get database
>>>>Read collection data by BSON Document
_id: _id=5092fd7018bf491f28432a9a
tranid: tranid=5092fd7018bf491f28432a99
name: name=Transaction 1
createddate: createddate=2012-11-01T22:53:36.555Z
user: user={ "name" : "customer1", "email" : "customer1@email.com" }

_id: _id=5092fd7018bf491f28432a9c
tranid: tranid=5092fd7018bf491f28432a9b
name: name=Transaction 2
createddate: createddate=2012-11-01T22:53:36.717Z
user: user={ "name" : "customer2", "email" : "customer2@email.com" }

_id: _id=5092fd7018bf491f28432a9e
tranid: tranid=5092fd7018bf491f28432a9d
name: name=Transaction 3
createddate: createddate=2012-11-01T22:53:36.717Z
user: user={ "name" : "customer3", "email" : "customer3@email.com" }

Press any key to continue . . . _
```

Figure 34: Showing embedded documents

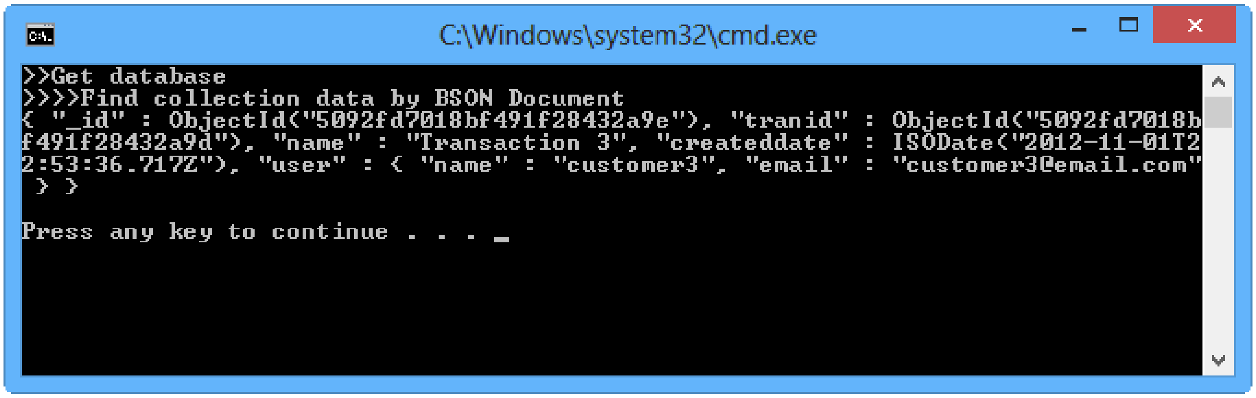
Next, we're going to find and query data.

To find data in an embedded document, you can use the **`Query`** object with query criteria.

For instance, to find a single document with the name "customer3", put the filter criteria into the **`Query`** object, and then call the **`FindOne()`** method to get the data.

```
        private static void FindDemo(MongoDatabase database)
        {
            Console.WriteLine(">>>>Find collection data by BSON Document");
            MongoCollection<BsonDocument> transactions =
database.GetCollection("transaction");

            var query = Query.EQ("user.name", BsonValue.Create("customer3"));
            BsonDocument doc = transactions.FindOne(query);
            Console.WriteLine(doc.ToString());
            Console.WriteLine("");
        }
```

The program output is shown in Figure 35.



```
>>Get database
>>>>Find collection data by BSON Document
{ "_id" : ObjectId("5092fd7018bf491f28432a9e"), "tranid" : ObjectId("5092fd7018b
f491f28432a9d"), "name" : "Transaction 3", "createddate" : ISODate("2012-11-01T2
2:53:36.717Z"), "user" : { "name" : "customer3", "email" : "customer3@email.com"
 } }

Press any key to continue . . . _
```

Figure 35: Finding with filter criteria

To update the existing embedded document, use the following code example.

```
        private static void UpdateDemo(MongoDatabase database)
        {
            Console.WriteLine(">>>>Update collection data by BSON Document");
            MongoCollection<BsonDocument> transactions =
database.GetCollection("transaction");
            BsonDocument doc = transactions.FindOne();
            // Show data.
            foreach (string name in doc.Names)
            {
                BsonElement element = doc.GetElement(name);
                Console.WriteLine("{0}: {1}", name, element.Value);
            }
            Console.WriteLine("");

            BsonDocument user = doc["user"].AsBsonDocument;
            user.Set("name", BsonValue.Create("update-customer"));
            user.Set("email", BsonValue.Create("update-customer@email.com"));

            doc.SetElement(new BsonElement("user",user));
            transactions.Save(doc);
            Console.WriteLine("updated data was success");

            // Show data.
            ReadDemo(database);
        }
```

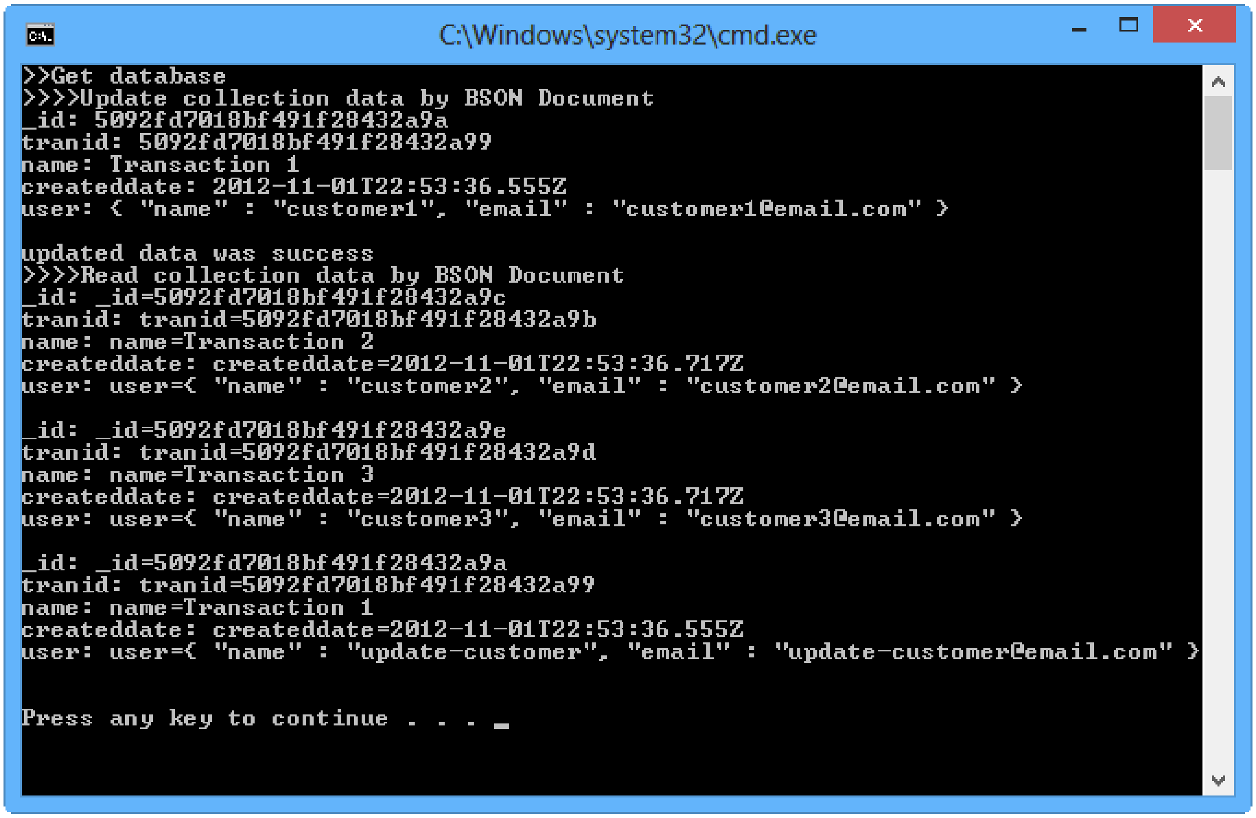
After you get a document object, you can get the embedded document object from its field.

```
            BsonDocument user = doc["user"].AsBsonDocument;
```

Call **Set()** to update the embedded document field value. Then replace the existing data by calling **SetElement()**. Don't forget to call **Save()**.

```
            BsonDocument user = doc["user"].AsBsonDocument;
            user.Set("name", BsonValue.Create("update-customer"));
            user.Set("email", BsonValue.Create("update-customer@email.com"));

            doc.SetElement(new BsonElement("user",user));
            transactions.Save(doc);
```

Run this code. You can see the program output in Figure 36.



```
>>Get database
>>>>Update collection data by BSON Document
_id: 5092fd7018bf491f28432a9a
tranid: 5092fd7018bf491f28432a99
name: Transaction 1
createddate: 2012-11-01T22:53:36.555Z
user: { "name" : "customer1", "email" : "customer1@email.com" }

updated data was success
>>>>Read collection data by BSON Document
_id: _id=5092fd7018bf491f28432a9c
tranid: tranid=5092fd7018bf491f28432a9b
name: name=Transaction 2
createddate: createddate=2012-11-01T22:53:36.717Z
user: user={ "name" : "customer2", "email" : "customer2@email.com" }

_id: _id=5092fd7018bf491f28432a9e
tranid: tranid=5092fd7018bf491f28432a9d
name: name=Transaction 3
createddate: createddate=2012-11-01T22:53:36.717Z
user: user={ "name" : "customer3", "email" : "customer3@email.com" }

_id: _id=5092fd7018bf491f28432a9a
tranid: tranid=5092fd7018bf491f28432a99
name: name=Transaction 1
createddate: createddate=2012-11-01T22:53:36.555Z
user: user={ "name" : "update-customer", "email" : "update-customer@email.com" }


Press any key to continue . . . _
```

Figure 36: Updating data for an embedded document

Embedded Document Collection

Sometimes you need an array of embedded documents such as **order** and **orderdetail**. One **order** has many **orderdetail**. For instance, the following is a sample of an embedded document collection

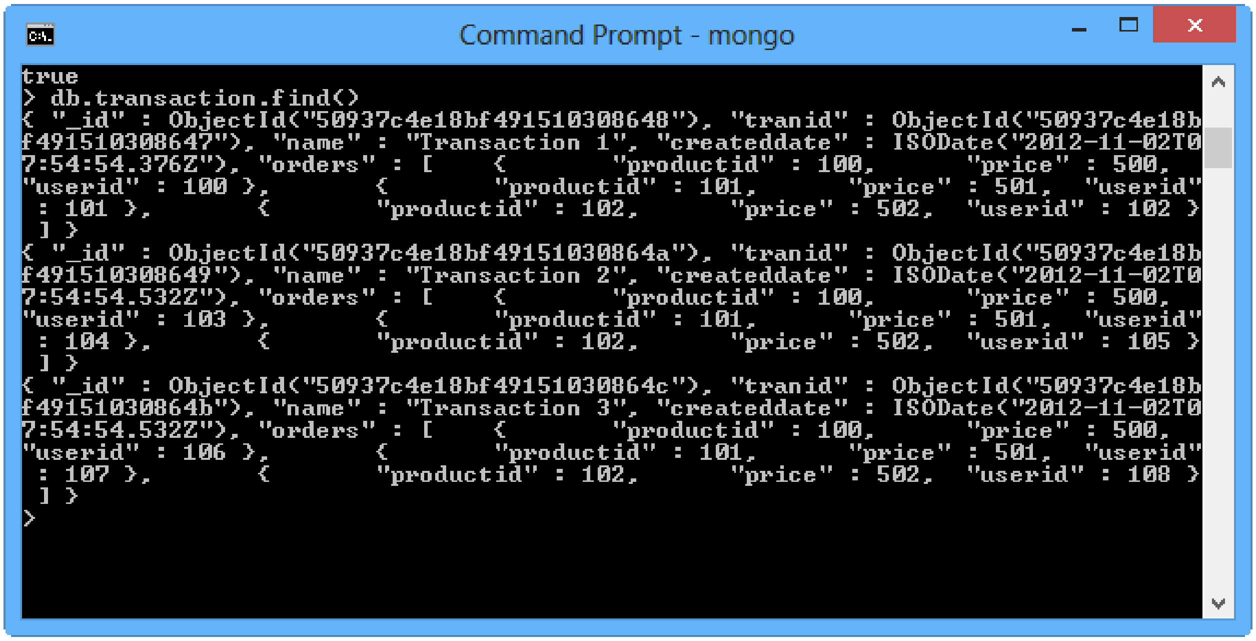
```
    {
        'tranid': 1,
        'name': 'trans 1',
        'createddate': '2/10/2012',
        'orders': [
            {
                'productid': 'product 1',
                'price': '100',
                'userid': 'user1'
            },
            {
                'productid': 'product 2',
                'price': '110',
                'userid': 'user1'
            },
            {
                'productid': 'product 3',
                'price': '90',
                'userid': 'user1'
            },
            {
                'productid': 'product 4',
                'price': '210',
                'userid': 'user1'
            }
        ]
    }
```

You can see that the **orders** field stores a collection of embedded documents.

First, we will create an embedded document collection. Use the **BsonArray** object to store embedded document collections. The following code example creates an array of embedded documents.

```
        private static void CreateDemo(MongoDatabase database)
        {
            // // create data for 5 transaction with BSON
            Console.WriteLine(">>>>Create collection data by BSON Document");
            MongoCollection<BsonDocument> transactions =
database.GetCollection("transaction");
            int index = 0;
            for (int i = 1; i <= 3; i++)
            {
                BsonDocument transaction = new BsonDocument {
                    { "tranid", ObjectId.GenerateNewId() },
                    { "name", String.Format("Transaction {0}", i) },
                    { "createddate", DateTime.Now }
                };
                BsonArray orders = new BsonArray();

                for (int j = 0; j < 3; j++)
                {
                    orders.Add(new BsonDocument{
                        {"productid", 100+j},
                        {"price", 500+j},
                        {"userid", 100+index}
                    });
                    index++;
                }
                BsonElement element = new BsonElement("orders", orders);
                transaction.Add(element);

                transactions.Insert(transaction);
            }

        }
```

Run this code. You can check the result by using the Mongo shell to open your collection. You will get a list of embedded documents with embedded collections. A sample of the program output is shown in Figure 37.



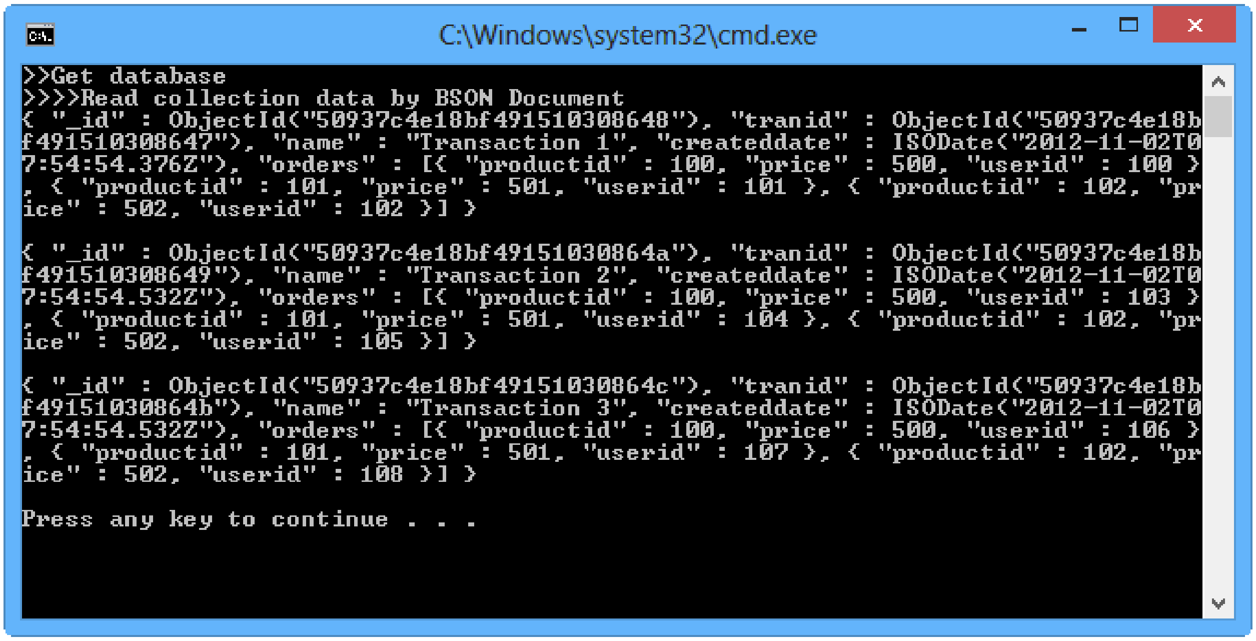
```
true
> db.transaction.find()
{ "_id" : ObjectId("50937c4e18bf491510308648"), "tranid" : ObjectId("50937c4e18b
f491510308647"), "name" : "Transaction 1", "createddate" : ISODate("2012-11-02T0
7:54:54.376Z"), "orders" : [    {       "productid" : 100,      "price" : 500,
"userid" : 100 },       {       "productid" : 101,      "price" : 501,  "userid"
 : 101 },       {       "productid" : 102,      "price" : 502,  "userid" : 102 }
 ] }
{ "_id" : ObjectId("50937c4e18bf49151030864a"), "tranid" : ObjectId("50937c4e18b
f491510308649"), "name" : "Transaction 2", "createddate" : ISODate("2012-11-02T0
7:54:54.532Z"), "orders" : [    {       "productid" : 100,      "price" : 500,
"userid" : 103 },       {       "productid" : 101,      "price" : 501,  "userid"
 : 104 },       {       "productid" : 102,      "price" : 502,  "userid" : 105 }
 ] }
{ "_id" : ObjectId("50937c4e18bf49151030864c"), "tranid" : ObjectId("50937c4e18b
f49151030864b"), "name" : "Transaction 3", "createddate" : ISODate("2012-11-02T0
7:54:54.532Z"), "orders" : [    {       "productid" : 100,      "price" : 500,
"userid" : 106 },       {       "productid" : 101,      "price" : 501,  "userid"
 : 107 },       {       "productid" : 102,      "price" : 502,  "userid" : 108 }
 ] }
>
```

Figure 37: Inserting embedded document collection

After creating the embedded document collection, you can read the data.

You can use **`FindAll()`** to retrieve all data. Running the following code will get a list of the embedded documents collection in the console, as shown in Figure 38.

```
        private static void ReadDemo(MongoDatabase database)
        {
            Console.WriteLine(">>>>Read collection data by BSON Document");
            MongoCollection<BsonDocument> transactions =
database.GetCollection("transaction");
            foreach (BsonDocument doc in transactions.FindAll())
            {
                Console.WriteLine(doc.ToString());
                Console.WriteLine("");
            }

        }
```


C:\Windows\system32\cmd.exe

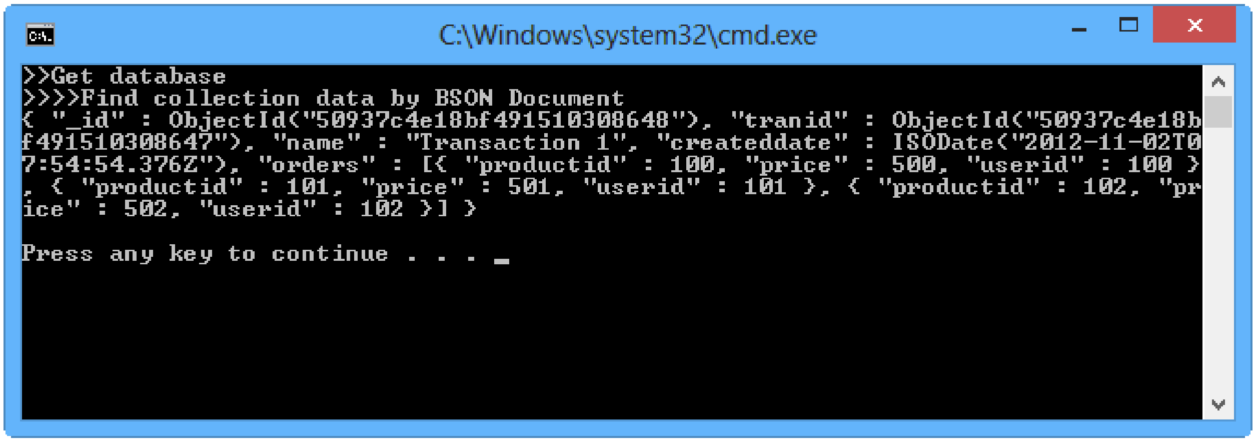
```
>>Get database
>>>>Read collection data by BSON Document
{ "_id" : ObjectId("50937c4e18bf4915103086 48"), "tranid" : ObjectId("50937c4e18b
f49151030864 7"), "name" : "Transaction 1", "createddate" : ISODate("2012-11-02T0
7:54:54.376Z"), "orders" : [{ "productid" : 100, "price" : 500, "userid" : 100 }
, { "productid" : 101, "price" : 501, "userid" : 101 }, { "productid" : 102, "pr
ice" : 502, "userid" : 102 }] }

{ "_id" : ObjectId("50937c4e18bf49151030864a"), "tranid" : ObjectId("50937c4e18b
f491510308649"), "name" : "Transaction 2", "createddate" : ISODate("2012-11-02T0
7:54:54.532Z"), "orders" : [{ "productid" : 100, "price" : 500, "userid" : 103 }
, { "productid" : 101, "price" : 501, "userid" : 104 }, { "productid" : 102, "pr
ice" : 502, "userid" : 105 }] }

{ "_id" : ObjectId("50937c4e18bf49151030864c"), "tranid" : ObjectId("50937c4e18b
f49151030864b"), "name" : "Transaction 3", "createddate" : ISODate("2012-11-02T0
7:54:54.532Z"), "orders" : [{ "productid" : 100, "price" : 500, "userid" : 106 }
, { "productid" : 101, "price" : 501, "userid" : 107 }, { "productid" : 102, "pr
ice" : 502, "userid" : 108 }] }

Press any key to continue . . .
```

Figure 38: Reading all embedded documents collection

You also can find documents based on the embedded document field. Use **`Find()`** with the **`Query`** object for filtering criteria. The following code example implements this scenario.

```
        private static void FindDemo(MongoDatabase database)
        {
            Console.WriteLine(">>>>Find collection data by BSON Document");
            MongoCollection<BsonDocument> transactions =
database.GetCollection("transaction");

            var query = Query.EQ("orders.userid", BsonValue.Create(100));
            foreach (BsonDocument doc in transactions.Find(query))
            {
                Console.WriteLine(doc.ToString());
                Console.WriteLine("");
            }
        }
```

Running this code will produce the program output in Figure 39.



```
>>Get database
>>>>Find collection data by BSON Document
{ "_id" : ObjectId("50937c4e18bf491510308648"), "tranid" : ObjectId("50937c4e18b
f491510308647"), "name" : "Transaction 1", "createddate" : ISODate("2012-11-02T0
7:54:54.376Z"), "orders" : [{ "productid" : 100, "price" : 500, "userid" : 100 }
, { "productid" : 101, "price" : 501, "userid" : 101 }, { "productid" : 102, "pr
ice" : 502, "userid" : 102 }] }

Press any key to continue . . .
```

Figure 39: Reading the data based on the filter criteria

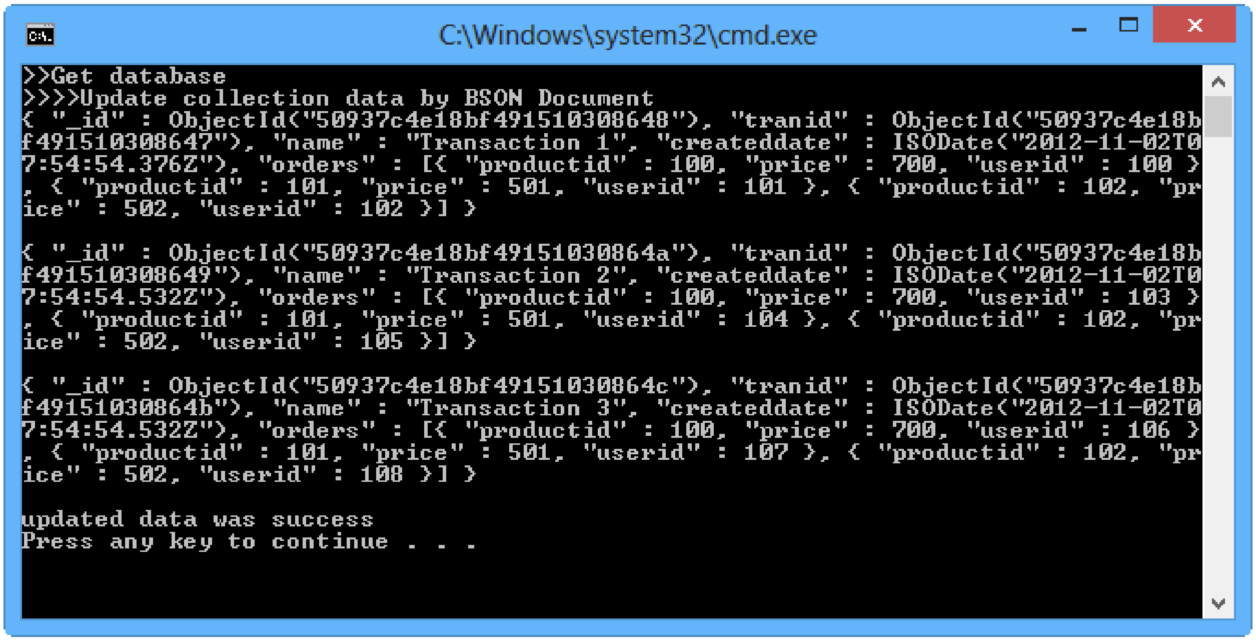
You can use **`Update.Set()`** with a parameter and value to update the data inside an embedded document. After that, call **`Update()`** from the collection object.

The following example updates an embedded document.

```
        private static void UpdateDemo(MongoDatabase database)
        {
            Console.WriteLine(">>>>Update collection data by BSON Document");
            MongoCollection<BsonDocument> transactions =
database.GetCollection("transaction");

            var query = Query.EQ("orders.productid", BsonValue.Create(100));
            var update = Update.Set("orders.$.price", BsonValue.Create(700));
            var option = new MongoUpdateOptions();
            option.Flags = UpdateFlags.Multi;
            transactions.Update(query, update, option);
            foreach (BsonDocument doc in transactions.Find(query))
            {
                Console.WriteLine(doc.ToString());
                Console.WriteLine("");
            }
            Console.WriteLine("updated data was success");

        }
```

The dollar sign in **`orders.$.price`** acts as a placeholder to update the first element that matches the query condition in an update.

Running this example will produce the program output in Figure 40. All prices become 700.



```
>>Get database
>>>>Update collection data by BSON Document
{ "_id" : ObjectId("50937c4e18bf491510308648"), "tranid" : ObjectId("50937c4e18b
f491510308647"), "name" : "Transaction 1", "createddate" : ISODate("2012-11-02T0
7:54:54.376Z"), "orders" : [{ "productid" : 100, "price" : 700, "userid" : 100 }
, { "productid" : 101, "price" : 501, "userid" : 101 }, { "productid" : 102, "pr
ice" : 502, "userid" : 102 }] }

{ "_id" : ObjectId("50937c4e18bf49151030864a"), "tranid" : ObjectId("50937c4e18b
f491510308649"), "name" : "Transaction 2", "createddate" : ISODate("2012-11-02T0
7:54:54.532Z"), "orders" : [{ "productid" : 100, "price" : 700, "userid" : 103 }
, { "productid" : 101, "price" : 501, "userid" : 104 }, { "productid" : 102, "pr
ice" : 502, "userid" : 105 }] }

{ "_id" : ObjectId("50937c4e18bf49151030864c"), "tranid" : ObjectId("50937c4e18b
f49151030864b"), "name" : "Transaction 3", "createddate" : ISODate("2012-11-02T0
7:54:54.532Z"), "orders" : [{ "productid" : 100, "price" : 700, "userid" : 106 }
, { "productid" : 101, "price" : 501, "userid" : 107 }, { "productid" : 102, "pr
ice" : 502, "userid" : 108 }] }

updated data was success
Press any key to continue . . .
```

Figure 40: Updating data in an embedded document collection

To delete the data, you can use the **`Drop()`** method of the collection object. The following code example deletes the collection.

```
        private static void DeleteDemo(MongoDatabase database)
        {
            Console.WriteLine(">>>>Delete all data in transaction 
collection");
            MongoCollection<BsonDocument> transactions = 
database.GetCollection("transaction");
            CommandResult result = transactions.Drop();
            if (result.Ok)
                Console.WriteLine("deleted all data in transaction collection 
was success");
            else
                Console.WriteLine(result.ErrorMessage);
        }
```

Chapter 8 LINQ

In this chapter, you will learn how to use LINQ in MongoDB.

In general, only LINQ queries that can be translated to an equivalent MongoDB query are supported. A runtime exception is thrown if the LINQ query contains a predicate that can't be translated. The error message will give information about the unsupported query.

Preparation

After creating the console application, add the MongoDB driver files into your project.

Write the following code for the namespace.

```
using System;
using System.Linq;
using MongoDB.Driver;
using MongoDB.Bson;
using MongoDB.Driver.Linq;
```

Next, we prepare a database and generate data for testing.

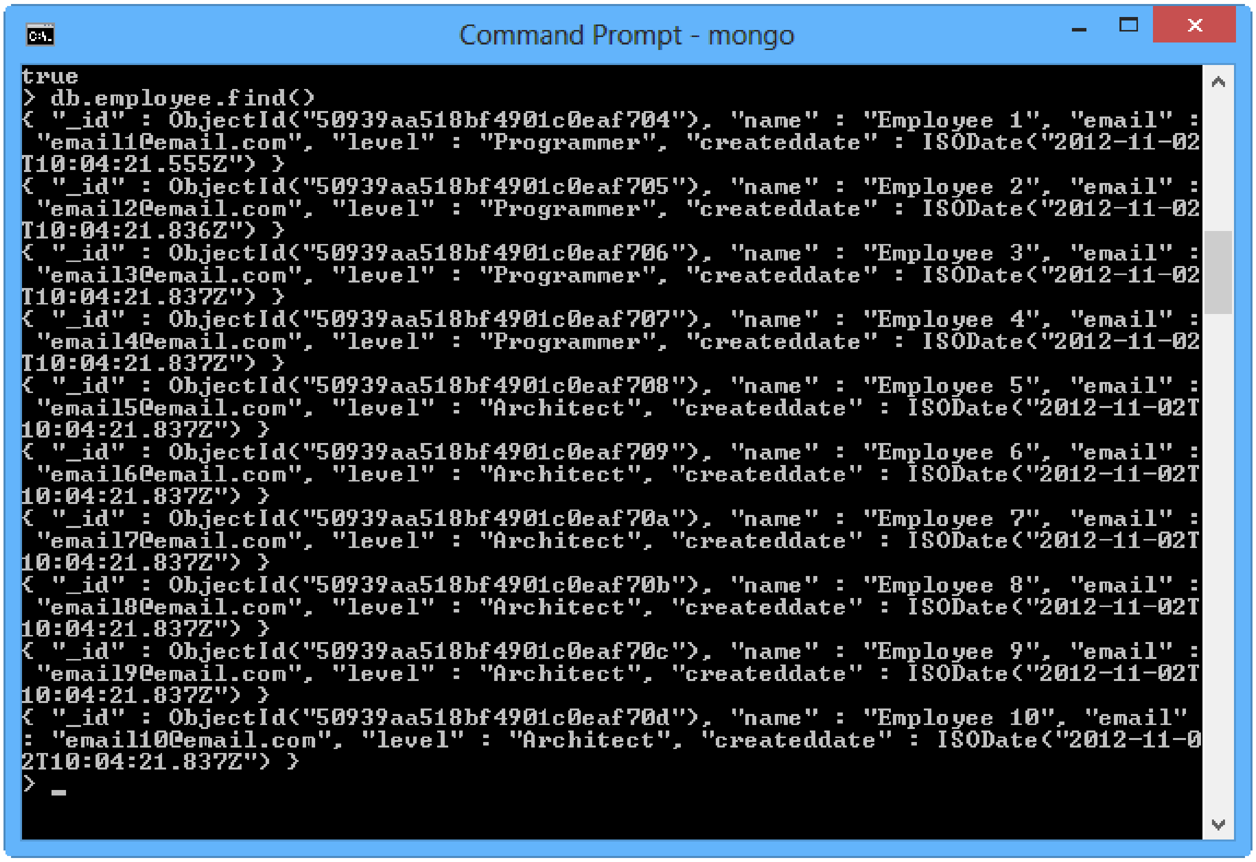
```
            string connectionString = "mongodb://localhost";
            string databaseName = "csharp";

            // Get database.
            Console.WriteLine(">>Get database");
            MongoServer server = MongoServer.Create(connectionString);
            MongoDatabase database = server.GetDatabase(databaseName);

            GenerateData(database);
```

The **`GenerateData()`** method generates 10 documents for testing purposes.

The following code example shows an implementation of the **`GenerateData()`** method.

```
        private static void GenerateData(MongoDatabase database)
        {
            // // Create data for 10 employees with BSON.
            Console.WriteLine(">>>>Create collection data by BSON Document");
            MongoCollection<BsonDocument> employees = 
database.GetCollection("employee");
            for (int i = 1; i <= 10; i++)
            {
                if (i < 5)
                {
                    BsonDocument employee = new BsonDocument {
                        { "name", "Employee " + i },
                        { "email", String.Format("email{0}@email.com", i) },
                        { "level", "Programmer" },
                        { "createddate", DateTime.Now }
                    };
                    employees.Insert(employee);
                }
                else
                {
                    BsonDocument employee = new BsonDocument {
                        { "name", "Employee " + i },
                        { "email", String.Format("email{0}@email.com", i) },
                        { "level", "Architect" },
                        { "createddate", DateTime.Now }
                    };
                    employees.Insert(employee);
                }
            }

        }
```

If you run this code, you can see the data in the MongoDB database as shown in Figure 41.



```
true
> db.employee.find()
{ "_id" : ObjectId("50939aa518bf4901c0eaf704"), "name" : "Employee 1", "email" :
 "email1@email.com", "level" : "Programmer", "createddate" : ISODate("2012-11-02
T10:04:21.555Z") }
{ "_id" : ObjectId("50939aa518bf4901c0eaf705"), "name" : "Employee 2", "email" :
 "email2@email.com", "level" : "Programmer", "createddate" : ISODate("2012-11-02
T10:04:21.836Z") }
{ "_id" : ObjectId("50939aa518bf4901c0eaf706"), "name" : "Employee 3", "email" :
 "email3@email.com", "level" : "Programmer", "createddate" : ISODate("2012-11-02
T10:04:21.837Z") }
{ "_id" : ObjectId("50939aa518bf4901c0eaf707"), "name" : "Employee 4", "email" :
 "email4@email.com", "level" : "Programmer", "createddate" : ISODate("2012-11-02
T10:04:21.837Z") }
{ "_id" : ObjectId("50939aa518bf4901c0eaf708"), "name" : "Employee 5", "email" :
 "email5@email.com", "level" : "Architect", "createddate" : ISODate("2012-11-02T
10:04:21.837Z") }
{ "_id" : ObjectId("50939aa518bf4901c0eaf709"), "name" : "Employee 6", "email" :
 "email6@email.com", "level" : "Architect", "createddate" : ISODate("2012-11-02T
10:04:21.837Z") }
{ "_id" : ObjectId("50939aa518bf4901c0eaf70a"), "name" : "Employee 7", "email" :
 "email7@email.com", "level" : "Architect", "createddate" : ISODate("2012-11-02T
10:04:21.837Z") }
{ "_id" : ObjectId("50939aa518bf4901c0eaf70b"), "name" : "Employee 8", "email" :
 "email8@email.com", "level" : "Architect", "createddate" : ISODate("2012-11-02T
10:04:21.837Z") }
{ "_id" : ObjectId("50939aa518bf4901c0eaf70c"), "name" : "Employee 9", "email" :
 "email9@email.com", "level" : "Architect", "createddate" : ISODate("2012-11-02T
10:04:21.837Z") }
{ "_id" : ObjectId("50939aa518bf4901c0eaf70d"), "name" : "Employee 10", "email"
: "email10@email.com", "level" : "Architect", "createddate" : ISODate("2012-11-0
2T10:04:21.837Z") }
> _
```

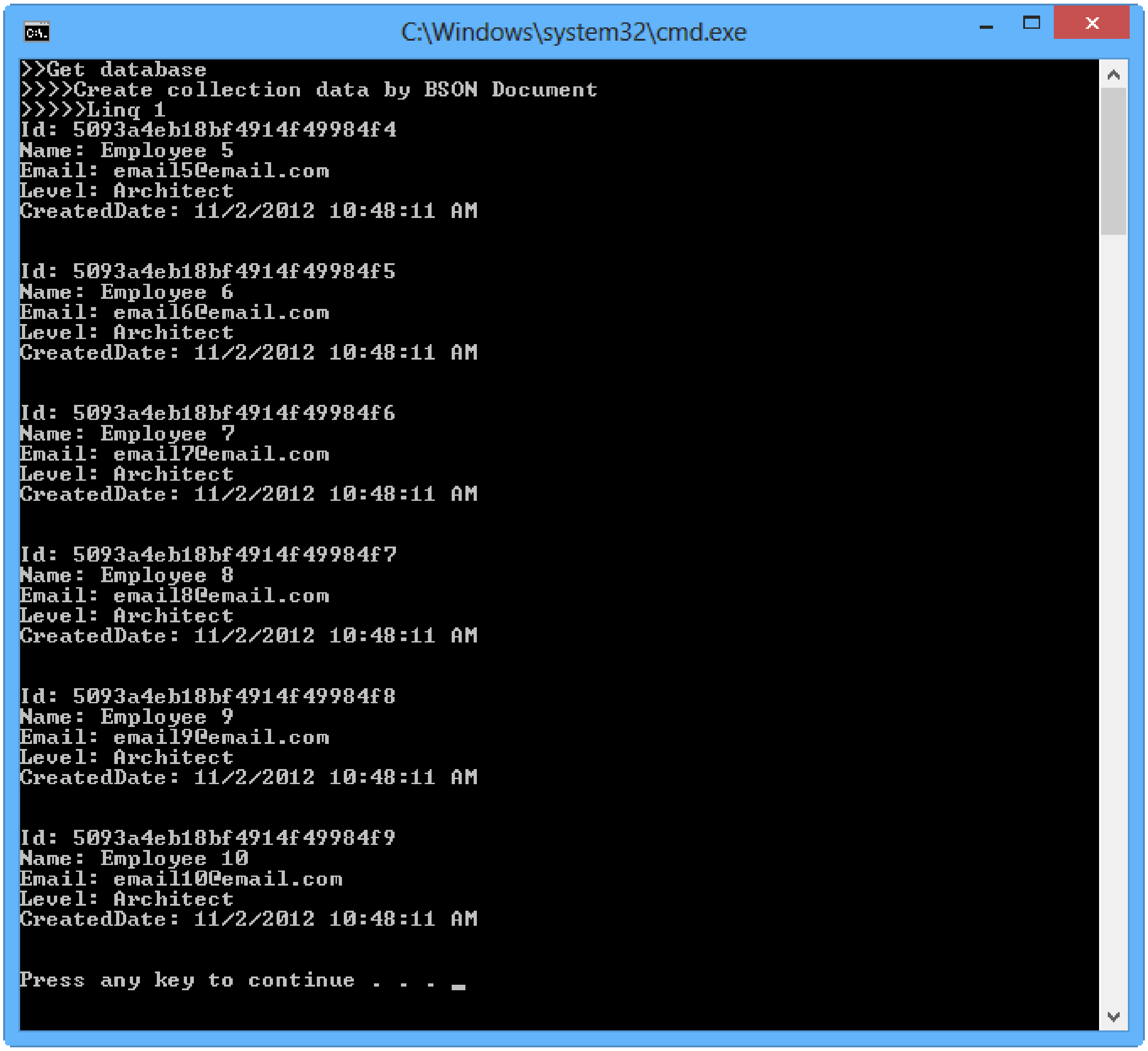
Figure 41: Generating data for testing purposes

Querying with LINQ

To query using LINQ, get the collection object based on the entity object. For example, get the **`employee`** collection and convert it into the **`Employee`** entity object.

The following example shows LINQ code for MongoDB. Running it will produce the output shown in Figure 42.

```
var employees = database.GetCollection<Employee>("employee");

// LINQ #1
Console.WriteLine(">>>>>Linq 1");
var query = from a in employees.AsQueryable<Employee>()
            where a.Level=="Architect"
            select a;
foreach (var employee in query)
{
    Console.WriteLine(employee);
}
```



```
>>Get database
>>>>Create collection data by BSON Document
>>>>>Linq 1
Id: 5093a4eb18bf4914f49984f4
Name: Employee 5
Email: email5@email.com
Level: Architect
CreatedDate: 11/2/2012 10:48:11 AM


Id: 5093a4eb18bf4914f49984f5
Name: Employee 6
Email: email6@email.com
Level: Architect
CreatedDate: 11/2/2012 10:48:11 AM


Id: 5093a4eb18bf4914f49984f6
Name: Employee 7
Email: email7@email.com
Level: Architect
CreatedDate: 11/2/2012 10:48:11 AM


Id: 5093a4eb18bf4914f49984f7
Name: Employee 8
Email: email8@email.com
Level: Architect
CreatedDate: 11/2/2012 10:48:11 AM


Id: 5093a4eb18bf4914f49984f8
Name: Employee 9
Email: email9@email.com
Level: Architect
CreatedDate: 11/2/2012 10:48:11 AM


Id: 5093a4eb18bf4914f49984f9
Name: Employee 10
Email: email10@email.com
Level: Architect
CreatedDate: 11/2/2012 10:48:11 AM


Press any key to continue . . . _
```

Figure 42: Showing data using LINQ

You can use the entity object and LINQ to retrieve MongoDB data.

The **`Employee`** entity object is defined as follows.

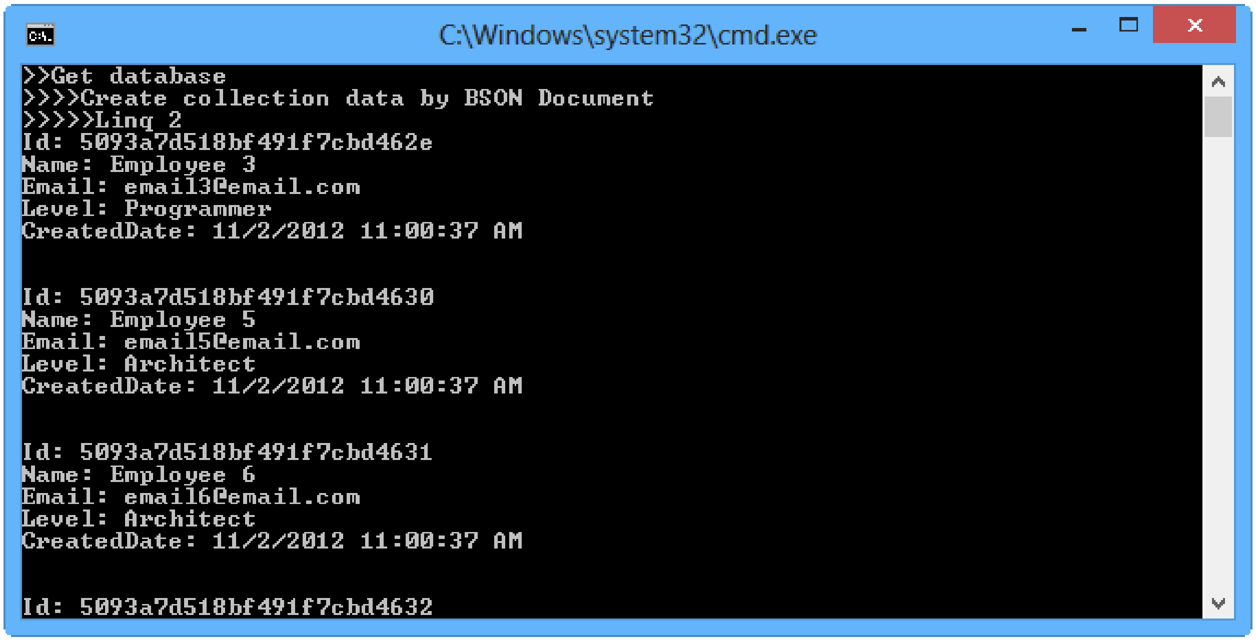
```
    public class Employee
    {
        [BsonElementAttribute("_id")]
        public ObjectId Id { set; get; }
        [BsonElementAttribute("name")]
        public string Name { set; get; }
        [BsonElementAttribute("email")]
        public string Email { set; get; }
        [BsonElementAttribute("level")]
        public string Level { set; get; }
        [BsonElementAttribute("createddate")]
        public DateTime CreateDate { set; get; }

        public override string ToString()
        {
            string str = string.Format("Id: {0}\r\nName: {1}\r\nEmail:
{2}\r\nLevel: {3}\r\nCreatedDate: {4}\r\n\r\n",
                         Id,Name,Email,Level,CreateDate);
            return str;
        }
    }
```

A namespace must be added for the entity object, as shown in the following example.

```

using System;
using MongoDB.Bson.Serialization.Attributes;
using MongoDB.Bson;

```

Now you can query the data as follows. Running the code will produce the output shown in Figure 43.

```
            // LINQ #2
            Console.WriteLine(">>>>>Linq 2");
            var query2 = from a in employees.AsQueryable<Employee>()
                        where a.Level == "Architect" || a.Email ==
"email3@email.com"
                        select a;
            foreach (var employee in query2)
            {
                Console.WriteLine(employee);
            }
```



```
>>Get database
>>>>Create collection data by BSON Document
>>>>>Linq 2
Id: 5093a7d518bf491f7cbd462e
Name: Employee 3
Email: email3@email.com
Level: Programmer
CreatedDate: 11/2/2012 11:00:37 AM


Id: 5093a7d518bf491f7cbd4630
Name: Employee 5
Email: email5@email.com
Level: Architect
CreatedDate: 11/2/2012 11:00:37 AM


Id: 5093a7d518bf491f7cbd4631
Name: Employee 6
Email: email6@email.com
Level: Architect
CreatedDate: 11/2/2012 11:00:37 AM


Id: 5093a7d518bf491f7cbd4632
```

Figure 43: Querying data using LINQ and entity object

Chapter 9 Working with MongoDB Shell

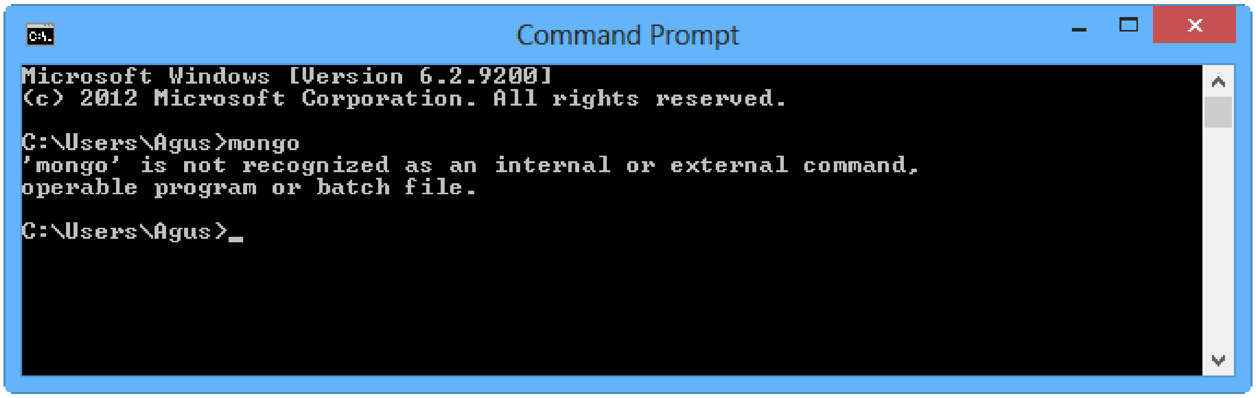
In this chapter, you will learn how to use the MongoDB shell.

What is MongoDB Shell?

MongoDB provides a shell to manage a MongoDB server. To start the MongoDB shell, you must call the Mongo shell. Open the Command Prompt window, type the following, and press Enter.

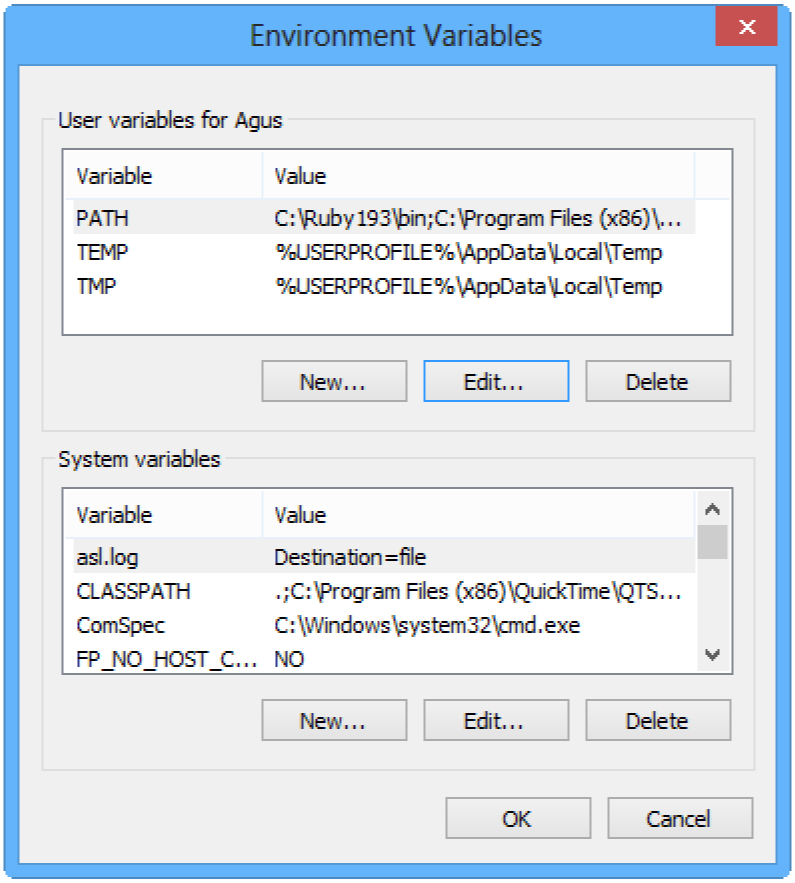
```
mongo
```

You will get an error message as shown in Figure 44. You can configure the MongoDB path using environment variables, as shown in Figure 45. If you want to configure the path for the user, select **PATH** in the **User Variable** menu. You will get a dialog as shown in Figure 46. Add the MongoDB folder where **mongo.exe** is located, for instance **c:\mongo\bin**, as the **Variable value**. After that, click **OK**.



```
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\Users\Agus>mongo
'mongo' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\Agus>_
```

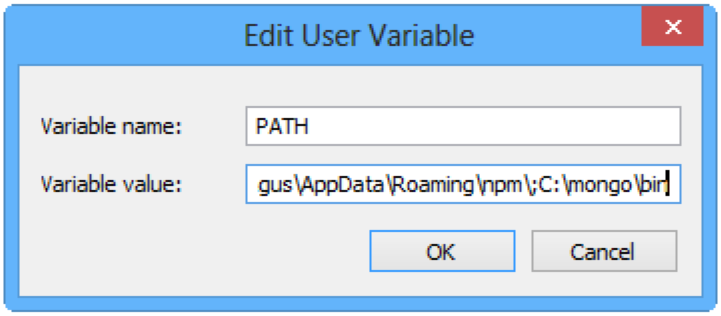
Figure 44: Mongo command is unkown in the Command Prompt



Variable	Value
PATH	C:\Ruby193\bin;C:\Program Files (x86)\...
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

Variable	Value
asl.log	Destination=file
CLASSPATH	.;C:\Program Files (x86)\QuickTime\QTS...
ComSpec	C:\Windows\system32\cmd.exe
FP_NO_HOST_C...	NO

Figure 45: Configuring the path in Environment Variables



Variable name: PATH

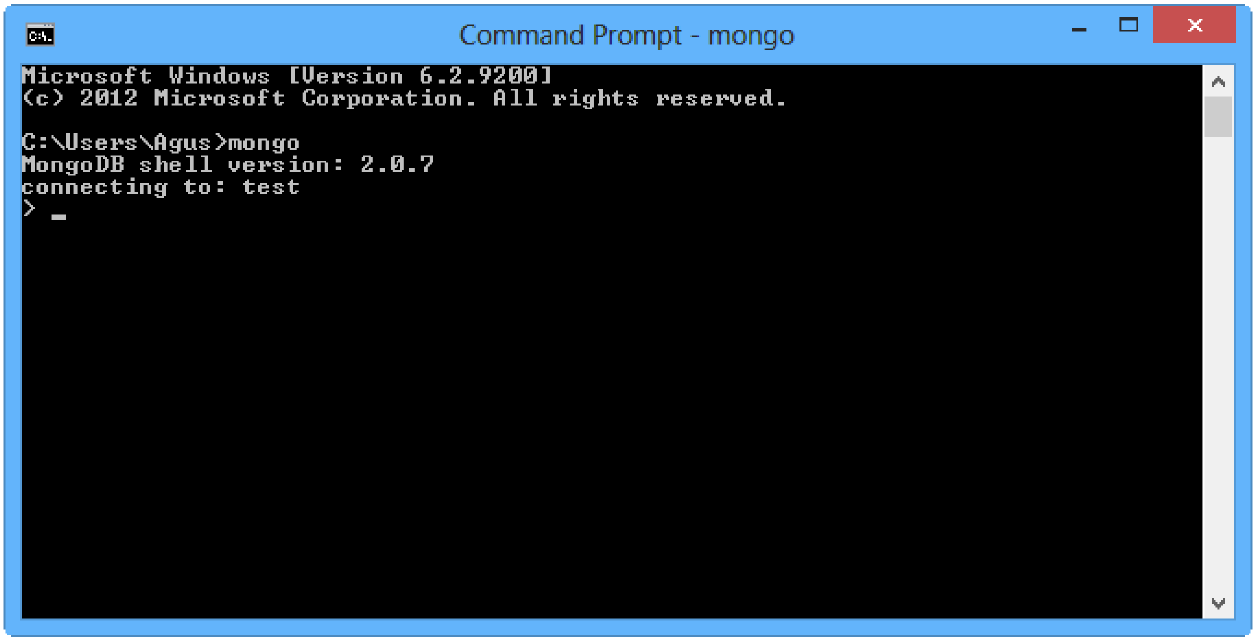
Variable value: gus\AppData\Roaming\npm\;C:\mongo\bin

Figure 46: Adding the Mongo path

Now type the following in the Command Prompt and press Enter.

```
mongo
```

If successful, you will start the Mongo shell as shown in Figure 47.

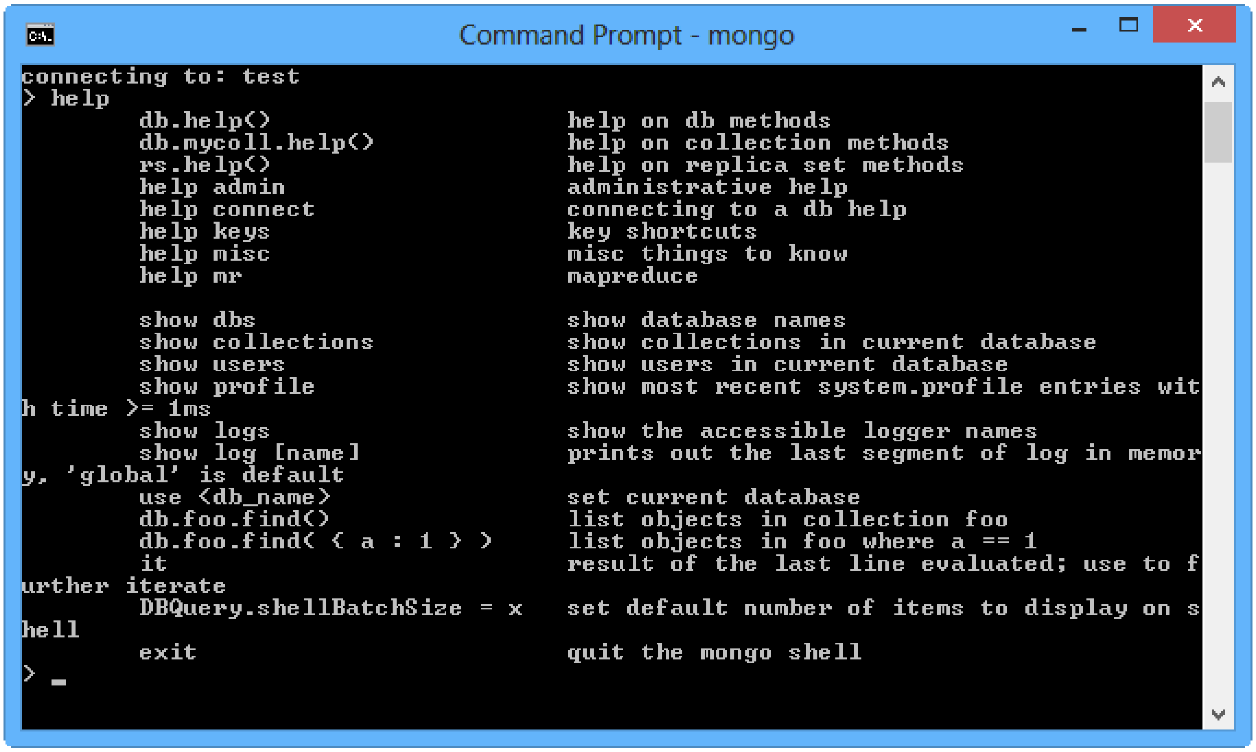


```
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\Users\Agus>mongo
MongoDB shell version: 2.0.7
connecting to: test
> _
```

Figure 47: The Mongo shell

Type the following command to get information about the Mongo shell.

```
help
```



```
connecting to: test
> help
        db.help()                    help on db methods
        db.mycoll.help()             help on collection methods
        rs.help()                    help on replica set methods
        help admin                   administrative help
        help connect                 connecting to a db help
        help keys                    key shortcuts
        help misc                    misc things to know
        help mr                      mapreduce

        show dbs                     show database names
        show collections             show collections in current database
        show users                   show users in current database
        show profile                 show most recent system.profile entries wit
h time >= 1ms
        show logs                    show the accessible logger names
        show log [name]              prints out the last segment of log in memor
y, 'global' is default
        use <db_name>                set current database
        db.foo.find()                list objects in collection foo
        db.foo.find( { a : 1 } )     list objects in foo where a == 1
        it                           result of the last line evaluated; use to f
urther iterate
        DBQuery.shellBatchSize = x   set default number of items to display on s
hell
        exit                         quit the mongo shell
> _
```

Figure 48: Help in the Mongo shell

The following is another command available for getting information about the Mongo shell.

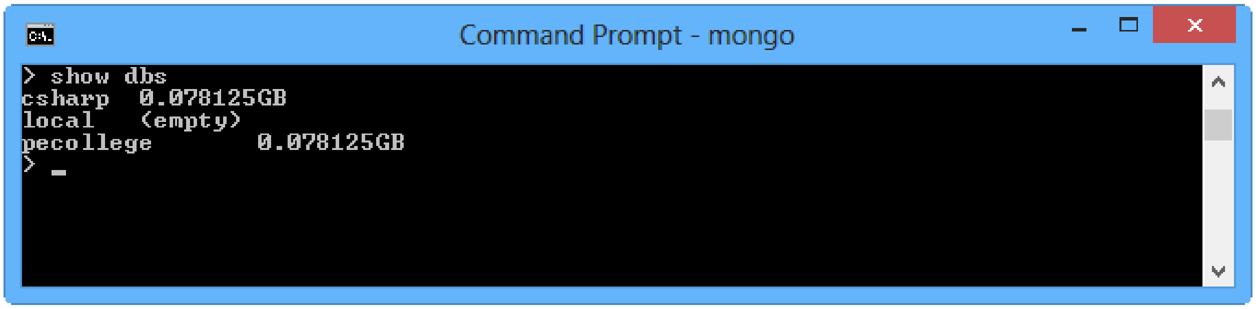
```
db.help()
```

Databases

The Mongo shell doesn't explicitly provide commands for creating databases. When you insert at least one document into a collection, MongoDB will create a database if one doesn't exist.

First, get the list of databases in MongoDB.

```
show dbs
```



```
> show dbs
csharp  0.078125GB
local   (empty)
pecollege       0.078125GB
> _
```

Figure 49: A list of databases

In this scenario, we will create a database called mydb by trying to insert one document. This action will make MongoDB create a database.

To navigate to our new database, use the following command. Don't worry if the database doesn't exist.

```
use mydb
```

The sample shell output is shown in Figure 50.

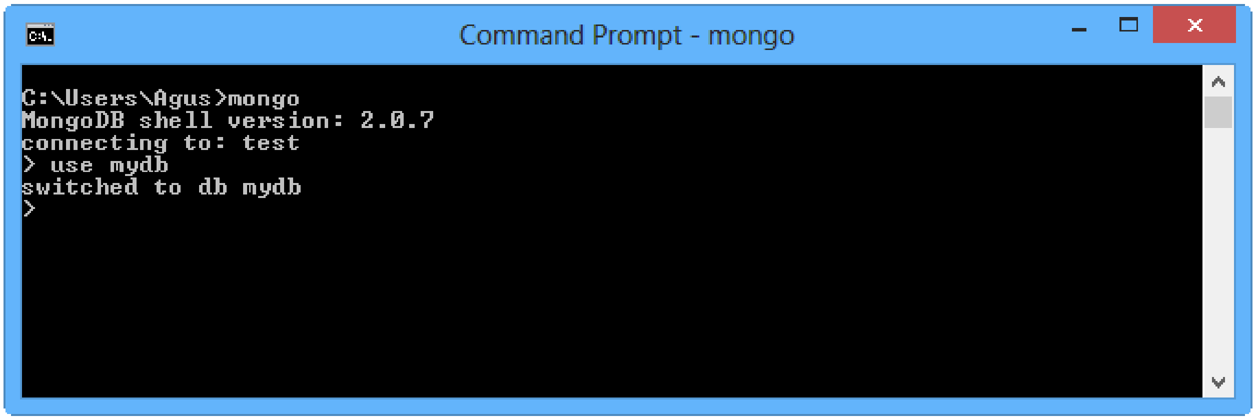
Next, we insert data into a collection called customers.

```
Db.customers.insert({firstname:'agus',lastname:'kurniawan'})
```

Use the following command.

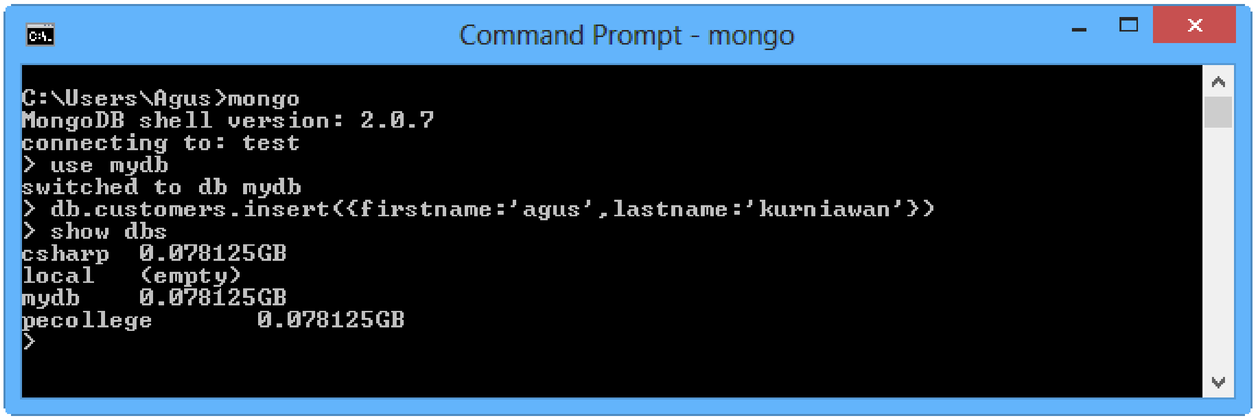
```
show dbs
```

You see a new database, as shown in Figure 51.



```
C:\Users\Agus>mongo
MongoDB shell version: 2.0.7
connecting to: test
> use mydb
switched to db mydb
>
```

Figure 50: Switching to a database



```
C:\Users\Agus>mongo
MongoDB shell version: 2.0.7
connecting to: test
> use mydb
switched to db mydb
> db.customers.insert({firstname:'agus',lastname:'kurniawan'})
> show dbs
csharp  0.078125GB
local   (empty)
mydb    0.078125GB
pecollege       0.078125GB
>
```

Figure 51: Showing a new database

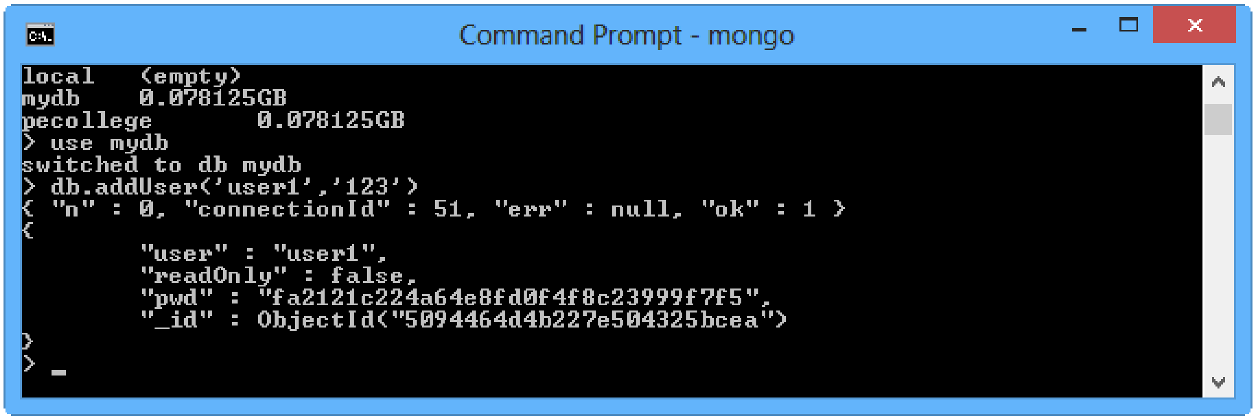
Database User

You may want to add database users for security purposes. You can use **`addUser()`** with username and password parameters.

The following example is a simple script to add a database user with the username “**`user1`**” and the password “**`123`**”.

```
db.addUser('user1','password');
```

If it is successful, you will see the response in the Mongo shell as shown in Figure 52.



```
local   (empty)
mydb    0.078125GB
pecollege        0.078125GB
> use mydb
switched to db mydb
> db.addUser('user1','123')
{ "n" : 0, "connectionId" : 51, "err" : null, "ok" : 1 }
{
        "user" : "user1",
        "readOnly" : false,
        "pwd" : "fa2121c224a64e8fd0f4f8c23999f7f5",
        "_id" : ObjectId("5094464d4b227e504325bcea")
}
> _
```

Figure 52: Adding a database user

Now your database has a database user. This means you can access the database if you pass the username and password.

Let's test our new user in C#. First, create a console application and add the MongoDB driver files.

Write the following namespace.

```
using System;
using MongoDB.Driver;
```

To pass a database user, you can use the **`MongoCredentials`** object. Just put the username and password into this object.

```
            string connectionString = "mongodb://localhost";
            string databaseName = "mydb";

            MongoServer server = MongoServer.Create(connectionString);

            MongoCredentials credential = new MongoCredentials("user1",
"123");
            MongoDatabase database = server.GetDatabase(databaseName,
credential);
```

You can test the credentials by attempting to retrieve a list of database collections. Run the following code.

```
            Console.WriteLine(">>Listing database collection");
            foreach (string name in database.GetCollectionNames())
            {
                Console.WriteLine(name);
            }
```

You will get a list of database collections. If you don't pass the database user, you will get an error.

Document

We have already created documents in a collection using C#, so let's look at how to do it using the Mongo shell.

`insert()` can be used to insert a document into a collection. To demonstrate this, insert two new customers into a collection called **`customer`** in the **`mydb`** database we made earlier. First, activate your database with the following command.

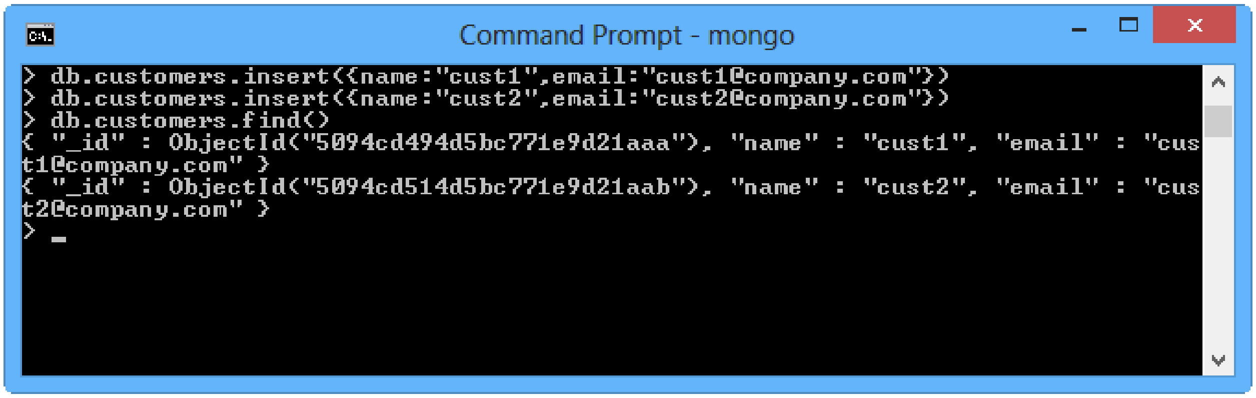
```
use mydb
```

Next, insert the data.

```
db.customers.insert({name:"cust1",email:"cust1@company.com"})
db.customers.insert({name:"cust2",email:"cust2@company.com"})
```

After creating the documents, we can check the data using **`find()`**. If you run the following code, you will see the program output shown in Figure 53.

```
db.customers.find()
```

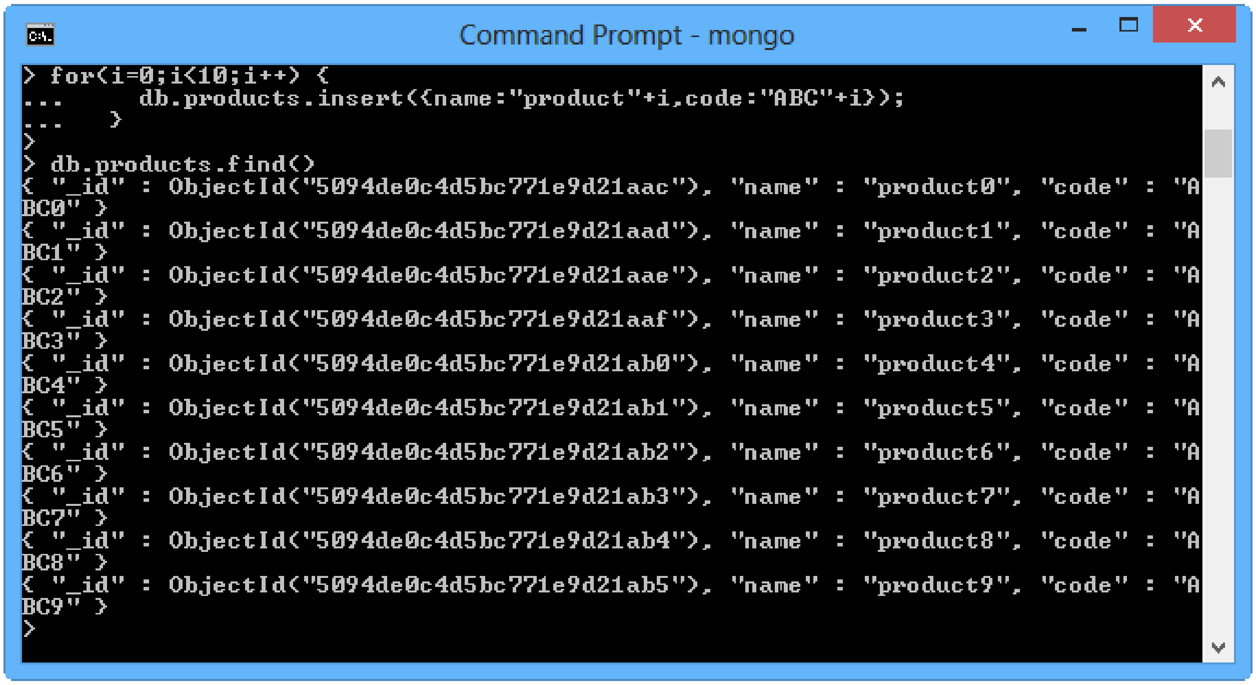


```
> db.customers.insert({name:"cust1",email:"cust1@company.com"})
> db.customers.insert({name:"cust2",email:"cust2@company.com"})
> db.customers.find()
{ "_id" : ObjectId("5094cd494d5bc771e9d21aaa"), "name" : "cust1", "email" : "cus
t1@company.com" }
{ "_id" : ObjectId("5094cd514d5bc771e9d21aab"), "name" : "cust2", "email" : "cus
t2@company.com" }
> _
```

Figure 53: Showing data using find()

You can add many documents to a collection using the Mongo shell. For instance, you can use a looping **`for`** in the Mongo shell. The output of the following program is shown in Figure 54.

```
for(i=0;i<10;i++) {
    db.products.insert({name:"product"+i,code:"ABC"+i});
  }
```

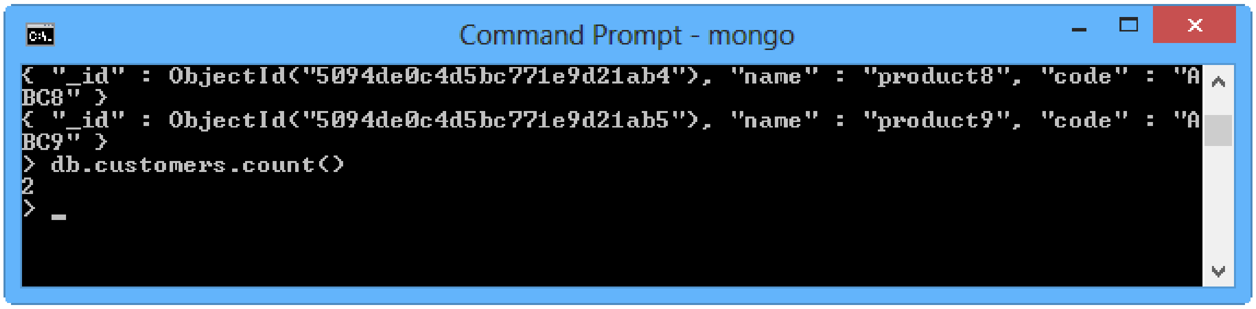


```
> for(i=0;i<10;i++) {
...     db.products.insert({name:"product"+i,code:"ABC"+i});
...   }
>
> db.products.find()
{ "_id" : ObjectId("5094de0c4d5bc771e9d21aac"), "name" : "product0", "code" : "A
BC0" }
{ "_id" : ObjectId("5094de0c4d5bc771e9d21aad"), "name" : "product1", "code" : "A
BC1" }
{ "_id" : ObjectId("5094de0c4d5bc771e9d21aae"), "name" : "product2", "code" : "A
BC2" }
{ "_id" : ObjectId("5094de0c4d5bc771e9d21aaf"), "name" : "product3", "code" : "A
BC3" }
{ "_id" : ObjectId("5094de0c4d5bc771e9d21ab0"), "name" : "product4", "code" : "A
BC4" }
{ "_id" : ObjectId("5094de0c4d5bc771e9d21ab1"), "name" : "product5", "code" : "A
BC5" }
{ "_id" : ObjectId("5094de0c4d5bc771e9d21ab2"), "name" : "product6", "code" : "A
BC6" }
{ "_id" : ObjectId("5094de0c4d5bc771e9d21ab3"), "name" : "product7", "code" : "A
BC7" }
{ "_id" : ObjectId("5094de0c4d5bc771e9d21ab4"), "name" : "product8", "code" : "A
BC8" }
{ "_id" : ObjectId("5094de0c4d5bc771e9d21ab5"), "name" : "product9", "code" : "A
BC9" }
>
```

Figure 54: Adding many documents in a collection

If you want to know how many document items there are inside a collection, use **`count()`** from the collection object. For instance, if you want to know the number of customer items, use the following code example. You will get the result shown in Figure 55.

```
db.customers.count()
```



```
{ "_id" : ObjectId("5094de0c4d5bc771e9d21ab4"), "name" : "product8", "code" : "A
BC8" }
{ "_id" : ObjectId("5094de0c4d5bc771e9d21ab5"), "name" : "product9", "code" : "A
BC9" }
> db.customers.count()
2
> _
```

Figure 55: Getting the total number of document items

Editing a Document

If you want to modify an existing document item, you can use the **`update()`** method. You must specify which document item you want to modify. For instance, if you want to modify the name of a product item that has the code **`ABC7`**, use **`$set`** to set the new value to the **`name`** field. The following example does this.

```
use mydb
db.products.update({code:"ABC7"},{$set: {name:"product baru"}})
```

Deleting a Document

You can delete document items from a collection using **`remove()`**. For instance, if you want to delete all document items for the products collection, use the following.

```
use mydb
db.products.remove()
```

If you want to delete specific document items, pass a criteria in the **`remove()`** method. For instance, if you want to delete all document items with the name **`product2`**, use the following.

```
use mydb
db.products.remove({name:"product2"})
```

Comparison Operators

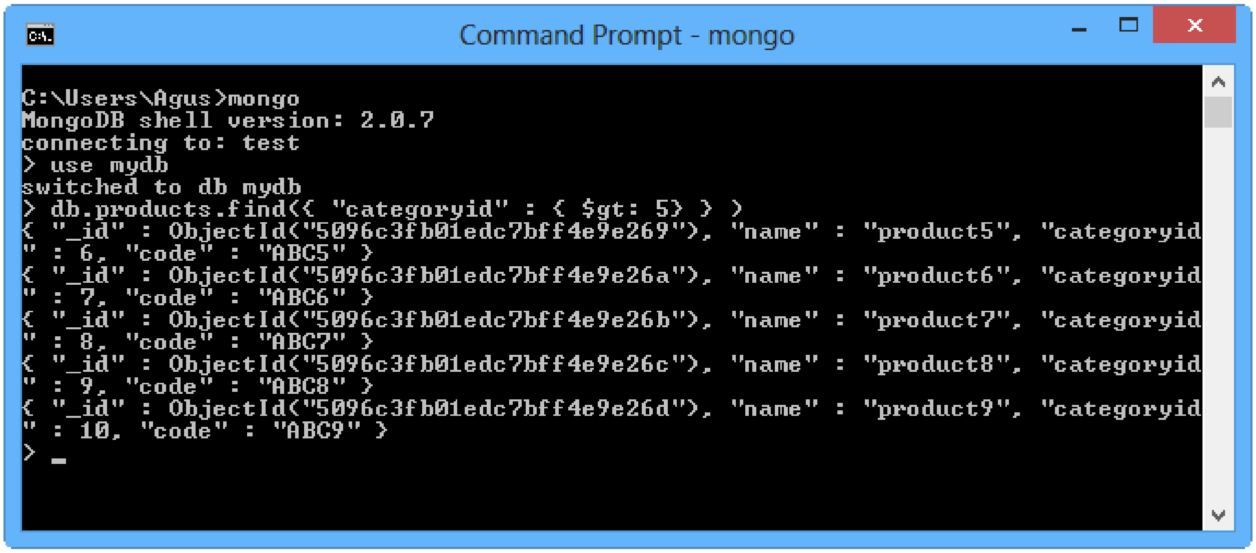
You normally use comparison operators such as >, <, <=, and >= in a SQL query. In the Mongo shell, the same comparison operators are available, but they use a different notation. The following table shows the Mongo shell equivalent of SQL comparison operators.

Mongo Shell Comparison Operator	SQL Equivalent
gt	>
lt	<
gte	>=
lte	<=

For example, if you want to filter product data by **`categoryid`** values greater than 5, you can do so in the Mongo shell by using the following code.

```
use mydb
db.products.find({ "categoryid" : { $gt: 5} } )
```

The sample query output is shown in Figure 56.



```
C:\Users\Agus>mongo
MongoDB shell version: 2.0.7
connecting to: test
> use mydb
switched to db mydb
> db.products.find({ "categoryid" : { $gt: 5} } )
{ "_id" : ObjectId("5096c3fb01edc7bff4e9e269"), "name" : "product5", "categoryid
" : 6, "code" : "ABC5" }
{ "_id" : ObjectId("5096c3fb01edc7bff4e9e26a"), "name" : "product6", "categoryid
" : 7, "code" : "ABC6" }
{ "_id" : ObjectId("5096c3fb01edc7bff4e9e26b"), "name" : "product7", "categoryid
" : 8, "code" : "ABC7" }
{ "_id" : ObjectId("5096c3fb01edc7bff4e9e26c"), "name" : "product8", "categoryid
" : 9, "code" : "ABC8" }
{ "_id" : ObjectId("5096c3fb01edc7bff4e9e26d"), "name" : "product9", "categoryid
" : 10, "code" : "ABC9" }
> _
```

Figure 56: Conditional operator usage in the Mongo shell

Limiting and Sorting

Imagine that you perform a query and get an enormous amount of data. It will have an impact on performance, so you want to set a limit for the amount of data that can be retrieved at once. You can set one by calling the **`limit()`** method. For instance, if you want to limit your query to 10 documents, use the following code.

```
use mydb
db.customers.find({"country": "germany"}).limit(10)
```

If you want to sort the result data, you can use the **`sort()`** method. It is similar to the ORDER BY command in a SQL query. For instance, if you want to sort data by the **`country`** field, use the following.

```
use mydb
db.customers.find().sort({country:-1})
```

The value (-1) means the sort is organized by descending order. The values 0 or 1 can be used for ascending order.

AND and OR Operators

Sometimes you want to query with filtering criteria that uses OR and AND operators. You can use **`$or`** for an OR operator and **`$and`** for an AND operator.

For example, if you want to find products with the name “abc” that have a category field value of “1” or a deleted field value of “1”, you can use the following code.

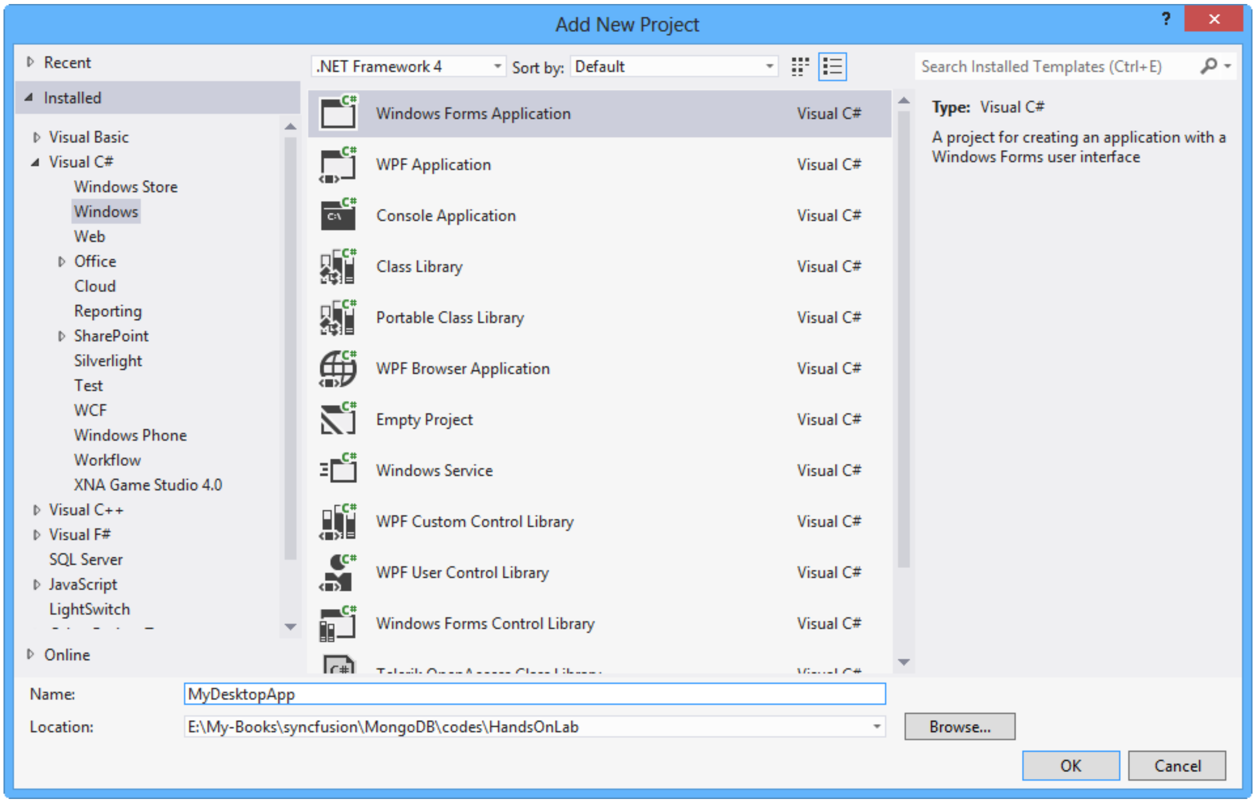
```
use mydb
db.products.find( { name : "abc" , $or : [ { category : 1 } , { deleted : -1
} ] } )
```

Chapter 10 MongoDB and Windows Forms

In this chapter, you will learn how to use the MongoDB database in a Windows Forms application.

Creating a Project

Create a new project in Visual Studio 2012. Choose **Windows Forms Application** as shown in Figure 57.



Add New Project

.NET Framework 4 Sort by: Default

Type: Visual C#

A project for creating an application with a Windows Forms user interface

Name: MyDesktopApp

Location: E:\My-Books\syncfusion\MongoDB\codes\HandsOnLab

Recent
Installed
Visual Basic
Visual C#
Windows Store
Windows
Web
Office
Cloud
Reporting
SharePoint
Silverlight
Test
WCF
Windows Phone
Workflow
XNA Game Studio 4.0
Visual C++
Visual F#
SQL Server
JavaScript
LightSwitch
Online
Search Installed Templates (Ctrl+E)
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
Class Library Visual C#
Portable Class Library Visual C#
WPF Browser Application Visual C#
Empty Project Visual C#
Windows Service Visual C#
WPF Custom Control Library Visual C#
WPF User Control Library Visual C#
Windows Forms Control Library Visual C#
Browse...
OK
Cancel

Figure 57: Creating a Windows Forms Application project in Visual Studio 2012

Provide a project name and location. Click **OK**. You will get a form as shown in Figure 58.

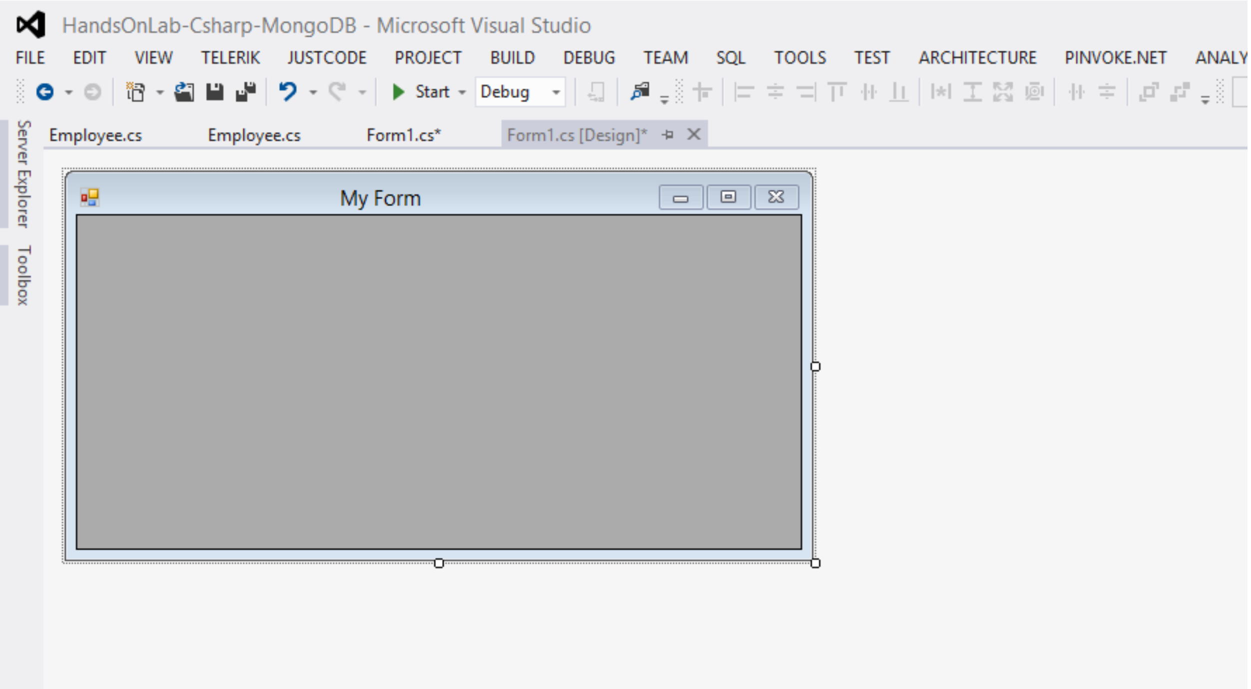
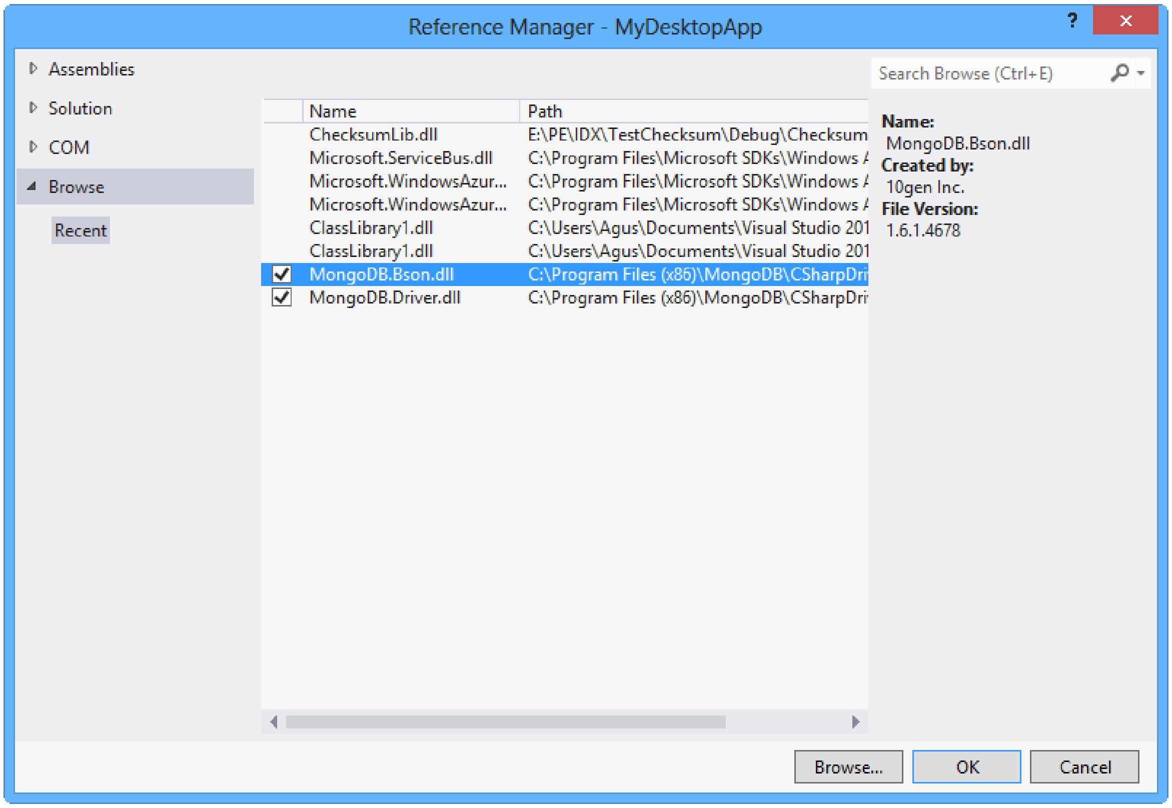


Figure 58: A form in Visual Studio 2012

Adding the MongoDB Driver

The first thing to do is to add the MongoDB driver files. Right-click on the **References** folder in your project. Select **Add reference** from the context menu to open the **Reference Manager** as shown in Figure 59.

Add MongoDB drive files by clicking **Browse**. There are two files you should add to your project: **MongoDB.Bson.dll** and **MongoDB.Driver.dll**. Click **OK**.



	Name	Path
	ChecksumLib.dll	E:\PE\IDX\TestChecksum\Debug\Checksum
	Microsoft.ServiceBus.dll	C:\Program Files\Microsoft SDKs\Windows A
	Microsoft.WindowsAzur...	C:\Program Files\Microsoft SDKs\Windows A
	Microsoft.WindowsAzur...	C:\Program Files\Microsoft SDKs\Windows A
	ClassLibrary1.dll	C:\Users\Agus\Documents\Visual Studio 201
	ClassLibrary1.dll	C:\Users\Agus\Documents\Visual Studio 201
☑	MongoDB.Bson.dll	C:\Program Files (x86)\MongoDB\CSharpDri
☑	MongoDB.Driver.dll	C:\Program Files (x86)\MongoDB\CSharpDri

Name:
MongoDB.Bson.dll
Created by:
10gen Inc.
File Version:
1.6.1.4678

Figure 59: Adding MongoDB driver files

Design Form

To modify your form, select a **DataGridView** from the **Toolbox** and put it on the form. Set its dock value to **Fill** to produce the form shown in Figure 60.

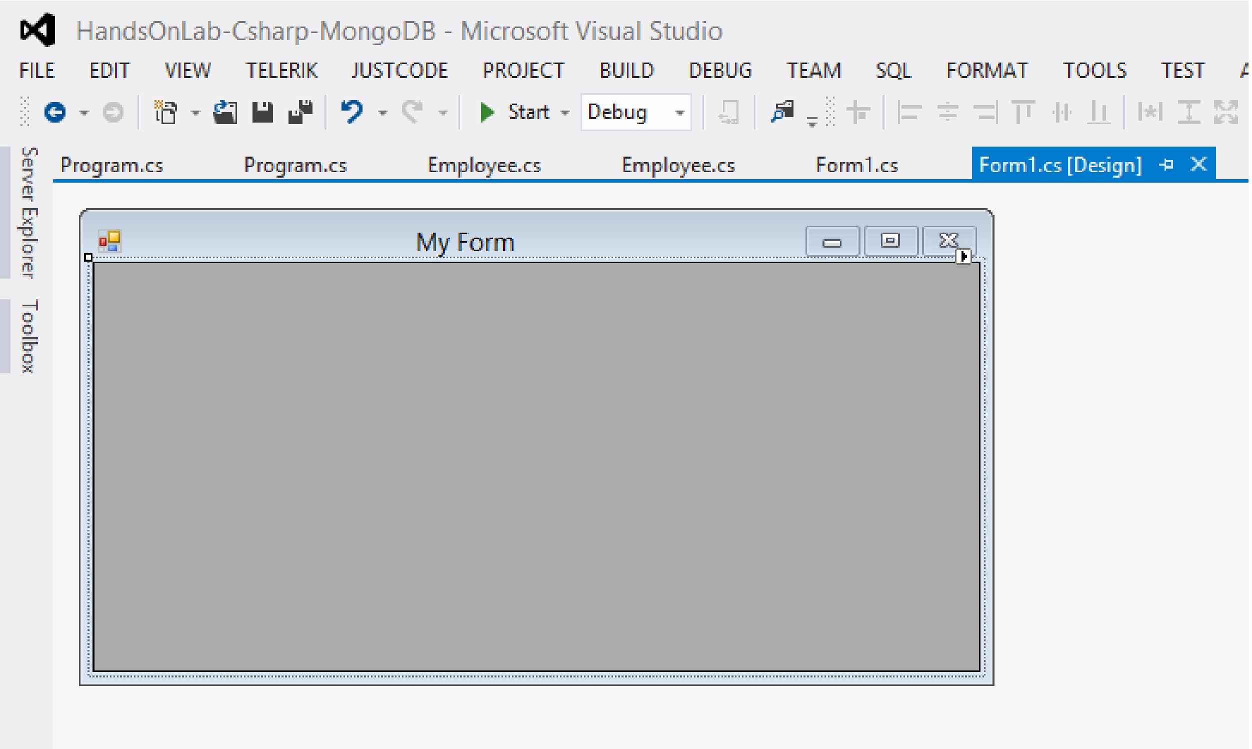


Figure 60: Adding DataGridView to a form

Writing Code

In Visual Studio 2012, open the **Form1.cs** file. You will see the following code.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;


namespace MyDesktopApp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

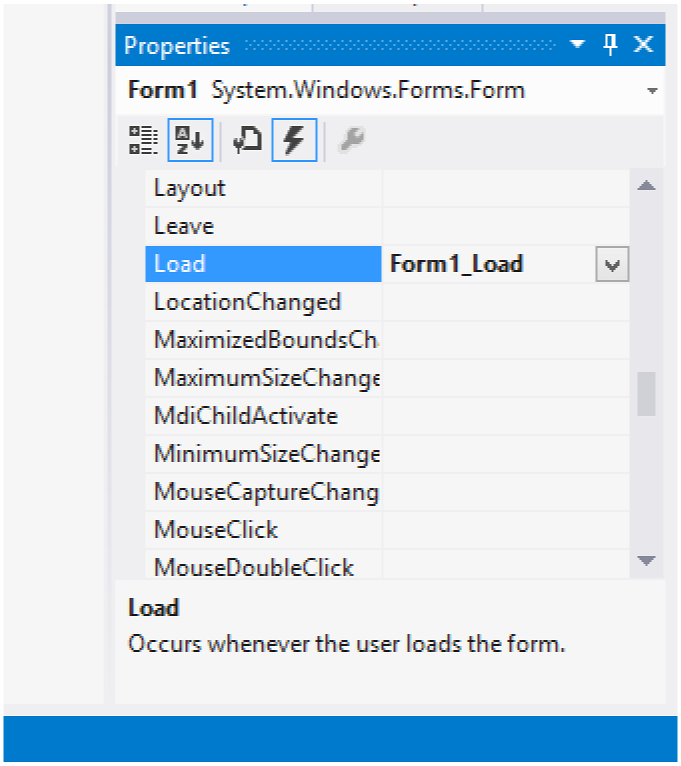
Write the following namespaces for the MongoDB driver.

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

You need an entity object called **`Employee`**. Add a class and save it as **`Employee.cs`**. This object is defined as follows.

```
using System;
using MongoDB.Bson.Serialization.Attributes;
using MongoDB.Bson;

namespace MyDesktopApp
{
    class Employee
    {
        [BsonElementAttribute("_id")]
        public ObjectId Id { set; get; }
        [BsonElementAttribute("name")]
        public string Name { set; get; }
        [BsonElementAttribute("email")]
        public string Email { set; get; }
        [BsonElementAttribute("level")]
        public string Level { set; get; }
        [BsonElementAttribute("createddate")]
        public DateTime CreateDate { set; get; }

        public override string ToString()
        {
            string str = string.Format("Id: {0}\r\nName: {1}\r\nEmail:
{2}\r\nLevel: {3}\r\nCreatedDate: {4}\r\n\r\n",
                         Id, Name, Email, Level, CreateDate);
            return str;
        }
    }
}
```

Next, you need to modify the form. Click inside the form and open the form events under **Properties**. Use the arrow keys to highlight **Load** as shown in Figure 62, and press Enter.



Form1 System.Windows.Forms.Form

Layout
Leave
Load Form1_Load
LocationChanged
MaximizedBoundsCh
MaximumSizeChange
MdiChildActivate
MinimumSizeChange
MouseCaptureChang
MouseClick
MouseDoubleClick

Load
Occurs whenever the user loads the form.

Figure 61: Adding the Load event

A code editor will open containing the following code.

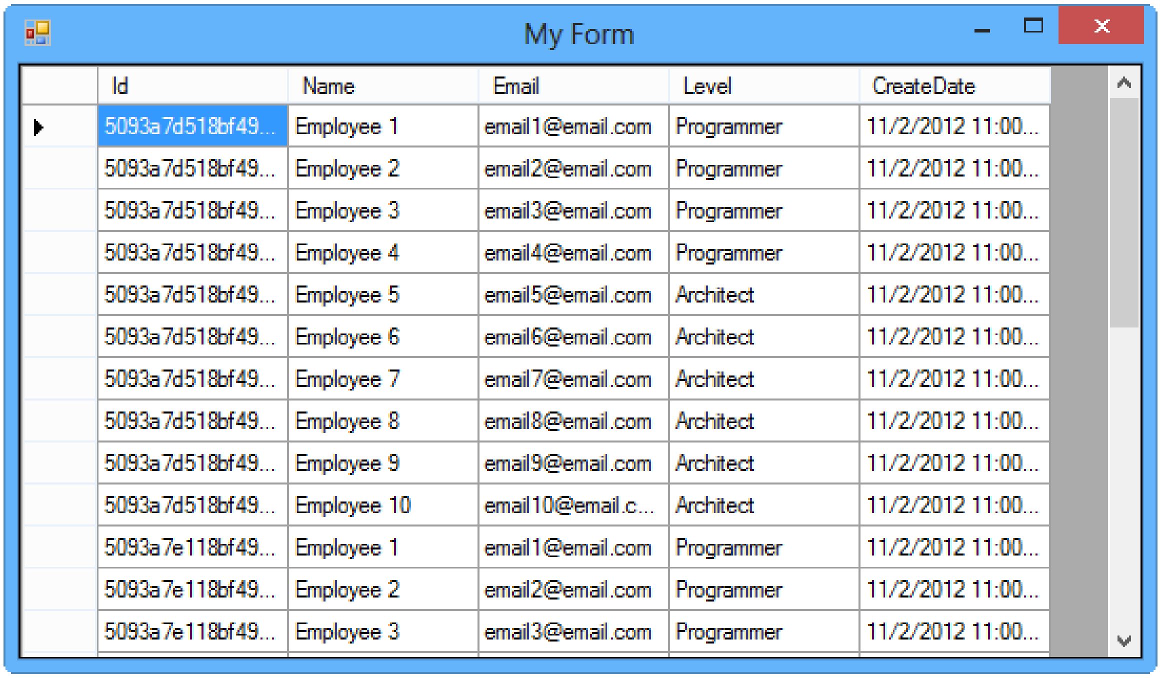
```
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }
    }
```

Next, we have to add code inside the **`Form1_Load()`** method. Use the following code example to retrieve data from MongoDB.

```
        private void Form1_Load(object sender, EventArgs e)
        {
            string connectionString = "mongodb://localhost";
            string databaseName = "csharp";

            // Get database.
            Console.WriteLine(">>Get database");
            MongoServer server = MongoServer.Create(connectionString);
            MongoDatabase database = server.GetDatabase(databaseName);

            var employees = database.GetCollection<Employee>("employee");
            var data = from a in employees.AsQueryable<Employee>() select a;
            dataGridView1.DataSource = data.ToArray<Employee>();
        }
```

`dataGridView1` is the **`DataGridView`** component that will show the data in tabular form.

Compile and run this project. If successful, you will see the following.



	Id	Name	Email	Level	CreateDate
▶	5093a7d518bf49...	Employee 1	email1@email.com	Programmer	11/2/2012 11:00...
	5093a7d518bf49...	Employee 2	email2@email.com	Programmer	11/2/2012 11:00...
	5093a7d518bf49...	Employee 3	email3@email.com	Programmer	11/2/2012 11:00...
	5093a7d518bf49...	Employee 4	email4@email.com	Programmer	11/2/2012 11:00...
	5093a7d518bf49...	Employee 5	email5@email.com	Architect	11/2/2012 11:00...
	5093a7d518bf49...	Employee 6	email6@email.com	Architect	11/2/2012 11:00...
	5093a7d518bf49...	Employee 7	email7@email.com	Architect	11/2/2012 11:00...
	5093a7d518bf49...	Employee 8	email8@email.com	Architect	11/2/2012 11:00...
	5093a7d518bf49...	Employee 9	email9@email.com	Architect	11/2/2012 11:00...
	5093a7d518bf49...	Employee 10	email10@email.c...	Architect	11/2/2012 11:00...
	5093a7e118bf49...	Employee 1	email1@email.com	Programmer	11/2/2012 11:00...
	5093a7e118bf49...	Employee 2	email2@email.com	Programmer	11/2/2012 11:00...
	5093a7e118bf49...	Employee 3	email3@email.com	Programmer	11/2/2012 11:00...

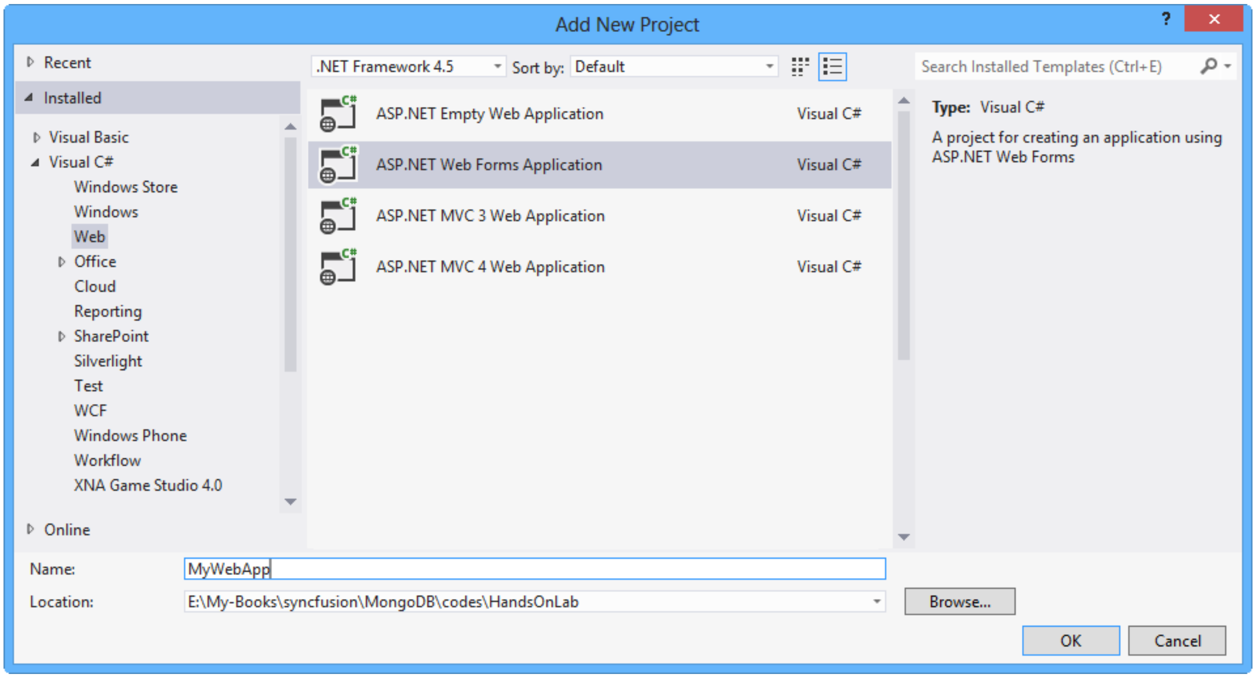
Figure 62: Showing data on grid

Chapter 11 MongoDB and ASP.NET

In this chapter, you will learn how to use MongoDB in an ASP.NET Web Forms application.

Creating a Project

Create a new project in Visual Studio 2012. Select **ASP.NET Web Forms Application** as shown in Figure 63.



Type: Visual C#

A project for creating an application using ASP.NET Web Forms

Name: MyWebApp

Location: E:\My-Books\syncfusion\MongoDB\codes\HandsOnLab

Figure 63: Creating a new ASP.NET Web Form Application

Provide a project name and location, and then click **OK**. An ASP.NET application project will be created.

Adding the MongoDB Driver

To access MongoDB, you must add the MongoDB driver files to your project. You can do this by right-clicking **References** in the Solution Explorer and selecting **Add References** as shown in Figure 64. Add the **MongoDB.Bson.dll** and **MongoDB.Driver.dll** files.

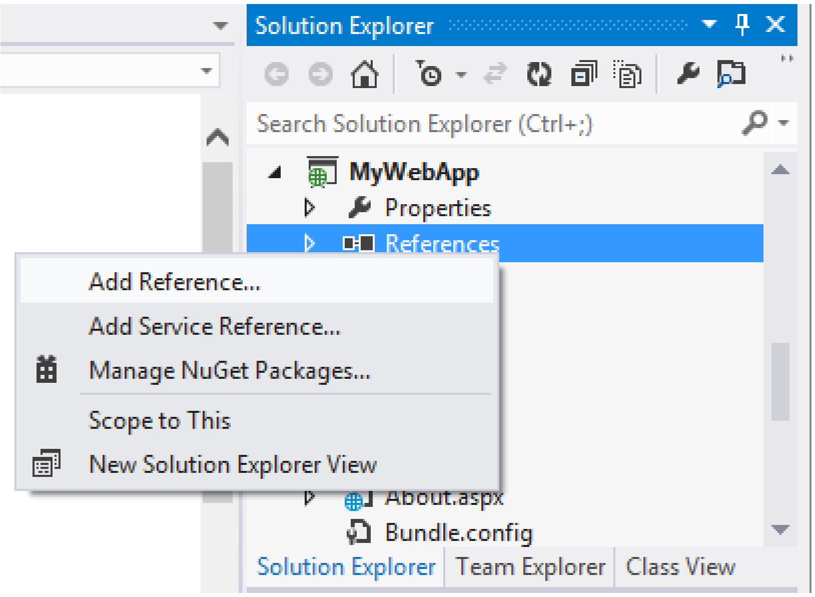
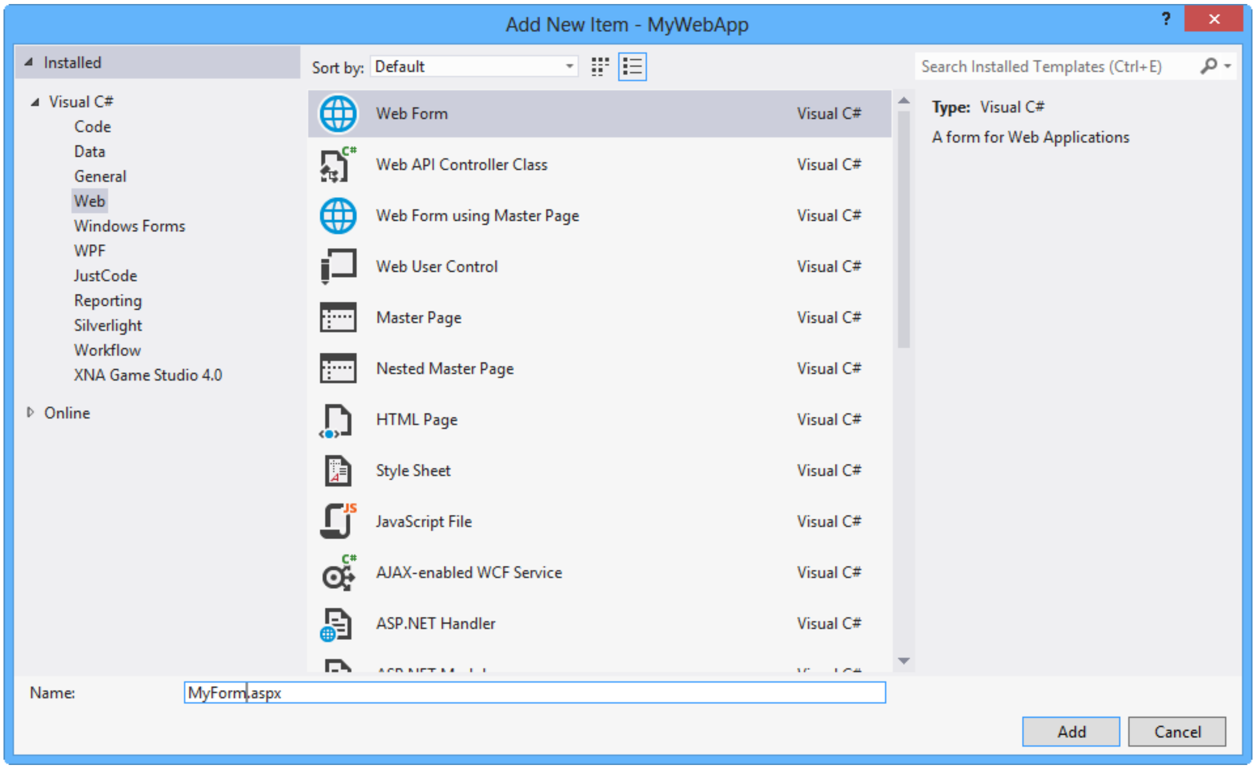


Figure 64: Adding references

Design Form

Add a new web form for displaying data from MongoDB by right-clicking on the project, selecting **Add**, and then choosing **New Item**. You will see the **Add New Item** window shown in Figure 65.


Add New Item - MyWebApp
Installed
Visual C#
Code
Data
General
Web
Windows Forms
WPF
JustCode
Reporting
Silverlight
Workflow
XNA Game Studio 4.0
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
Web Form Visual C#
Web API Controller Class Visual C#
Web Form using Master Page Visual C#
Web User Control Visual C#
Master Page Visual C#
Nested Master Page Visual C#
HTML Page Visual C#
Style Sheet Visual C#
JavaScript File Visual C#
AJAX-enabled WCF Service Visual C#
ASP.NET Handler Visual C#
Type: Visual C#
A form for Web Applications
Add
Cancel

Name: MyForm.aspx

Figure 65: Adding a new web form

Select **Web Form** and give it the name **MyForm.aspx**. When finished, click **Add**.

After creating the web form, open the file. Drag a **GridView** from the **Toolbox** and put it inside the **`<form>`** tag. The following code example shows a GridView added to the form.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="MyForm.aspx.cs"
Inherits="MyWebApp.MyForm" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:gridview runat="server" ID="gridview"></asp:gridview>
    </div>
    </form>
</body>
</html>
```

Writing Code

Right-click on the web form and select **View Code** from the context menu. This will open the code editor where you will see the following code example.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace MyWebApp
{
    public partial class MyForm : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }
    }
}
```

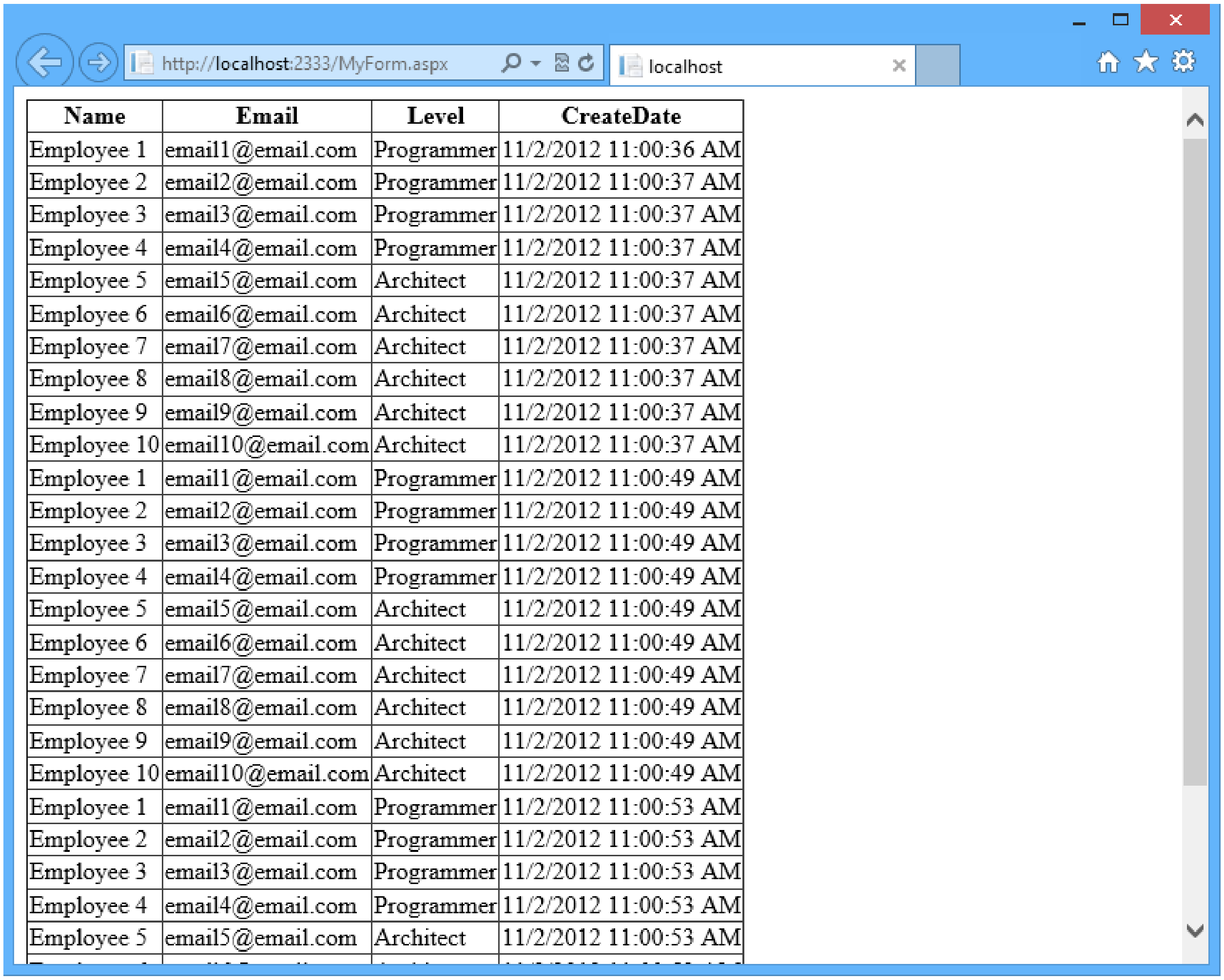
Add the following namespaces for the MongoDB driver.

```
Using MongoDB.Driver;
using MongoDB.Bson;
using MongoDB.Driver.Linq;
```

Next, write the following code inside the **Page_Load()** method.

```
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
              string connectionString = "mongodb://localhost";
              string databaseName = "csharp";

              // Get database.
              Console.WriteLine(">>Get database");
              MongoServer server = MongoServer.Create(connectionString);
              MongoDatabase database = server.GetDatabase(databaseName);

              var employees = database.GetCollection<Employee>("employee");
              var data = from a in employees.AsQueryable<Employee>() select a;
                gridview.DataSource = data.ToArray<Employee>();
                gridview.DataBind();
            }
        }
```

Now you can compile and run the page. The sample output is shown in Figure 66.



Name	Email	Level	CreateDate
Employee 1	email1@email.com	Programmer	11/2/2012 11:00:36 AM
Employee 2	email2@email.com	Programmer	11/2/2012 11:00:37 AM
Employee 3	email3@email.com	Programmer	11/2/2012 11:00:37 AM
Employee 4	email4@email.com	Programmer	11/2/2012 11:00:37 AM
Employee 5	email5@email.com	Architect	11/2/2012 11:00:37 AM
Employee 6	email6@email.com	Architect	11/2/2012 11:00:37 AM
Employee 7	email7@email.com	Architect	11/2/2012 11:00:37 AM
Employee 8	email8@email.com	Architect	11/2/2012 11:00:37 AM
Employee 9	email9@email.com	Architect	11/2/2012 11:00:37 AM
Employee 10	email10@email.com	Architect	11/2/2012 11:00:37 AM
Employee 1	email1@email.com	Programmer	11/2/2012 11:00:49 AM
Employee 2	email2@email.com	Programmer	11/2/2012 11:00:49 AM
Employee 3	email3@email.com	Programmer	11/2/2012 11:00:49 AM
Employee 4	email4@email.com	Programmer	11/2/2012 11:00:49 AM
Employee 5	email5@email.com	Architect	11/2/2012 11:00:49 AM
Employee 6	email6@email.com	Architect	11/2/2012 11:00:49 AM
Employee 7	email7@email.com	Architect	11/2/2012 11:00:49 AM
Employee 8	email8@email.com	Architect	11/2/2012 11:00:49 AM
Employee 9	email9@email.com	Architect	11/2/2012 11:00:49 AM
Employee 10	email10@email.com	Architect	11/2/2012 11:00:49 AM
Employee 1	email1@email.com	Programmer	11/2/2012 11:00:53 AM
Employee 2	email2@email.com	Programmer	11/2/2012 11:00:53 AM
Employee 3	email3@email.com	Programmer	11/2/2012 11:00:53 AM
Employee 4	email4@email.com	Programmer	11/2/2012 11:00:53 AM
Employee 5	email5@email.com	Architect	11/2/2012 11:00:53 AM

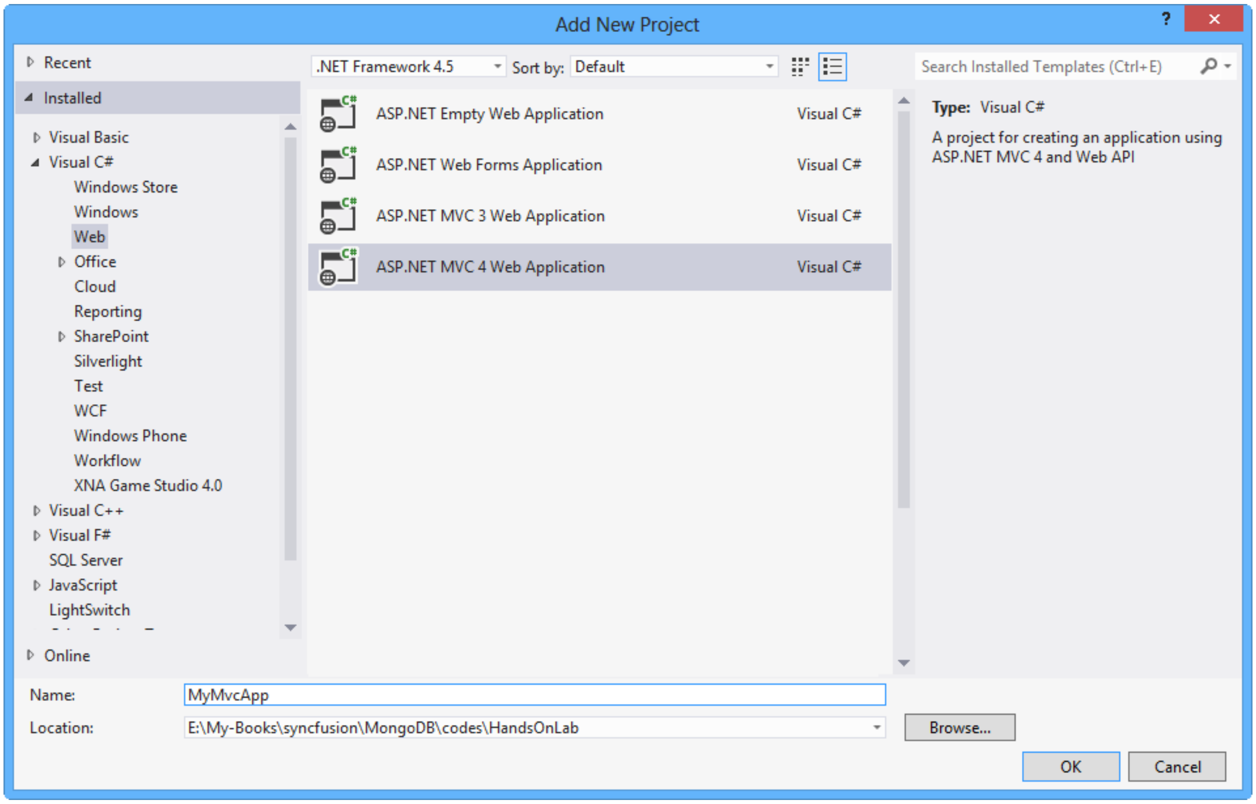
Figure 66: Displaying MongoDB data in a web form application

Chapter 12 MongoDB and ASP.NET MVC

In this chapter, you will learn how to use MongoDB in an ASP.NET MVC application.

Creating a Project

First, create a new project in Visual Studio 2012 and choose **ASP.NET MVC 4 Web Application** as shown in Figure 67.



.NET Framework 4.5

Sort by: Default

Type: Visual C#

A project for creating an application using ASP.NET MVC 4 and Web API

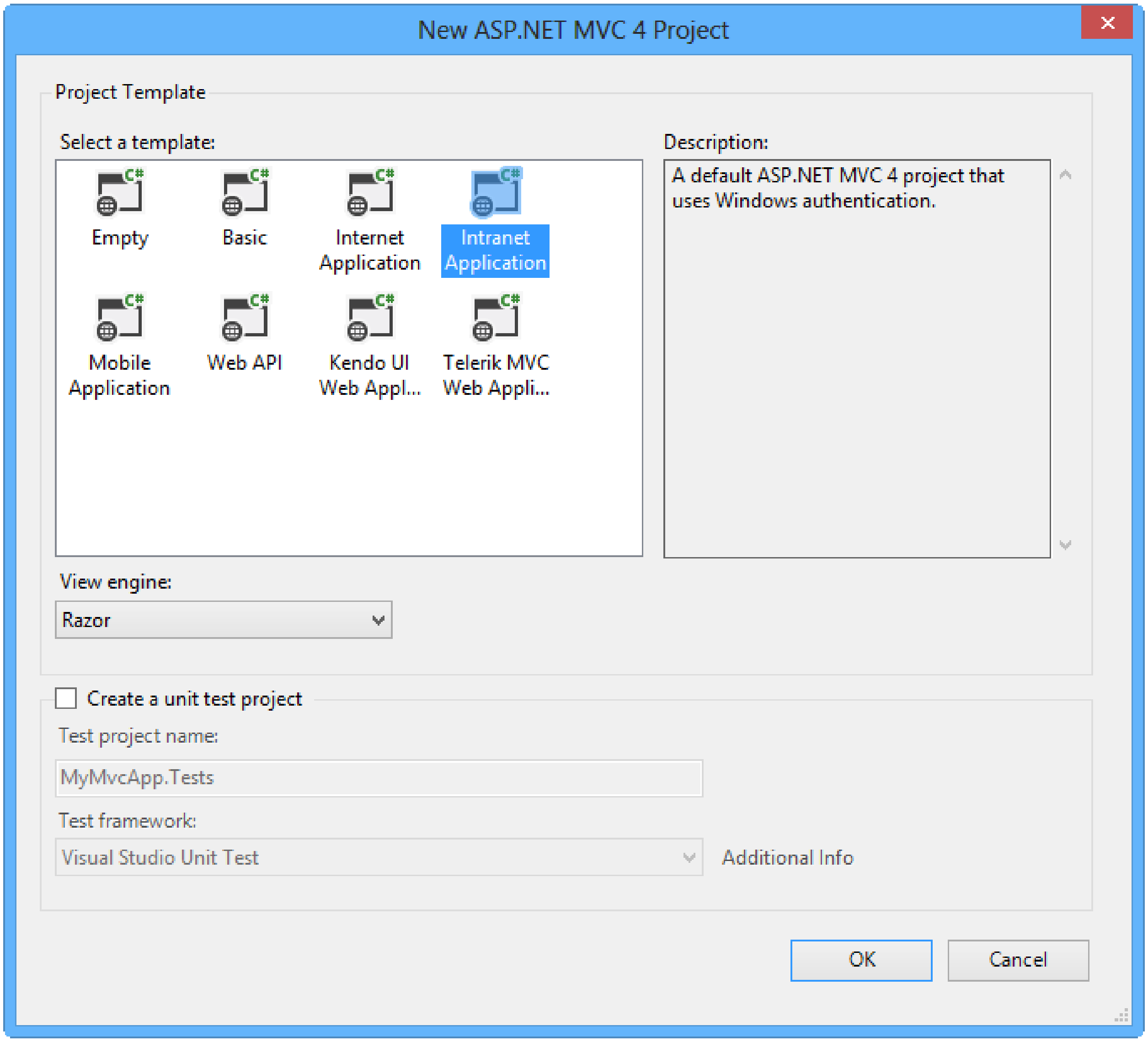
Name: MyMvcApp

Location: E:\My-Books\syncfusion\MongoDB\codes\HandsOnLab

Figure 67: Creating an ASP.NET MVC project

Provide a project name and location, and then click **OK**.

You will then see the New ASP.NET MVC 4 Project window as shown in Figure 68. Select **Internet Application**, and select **Razor** as the **View engine**. Click **OK** when you are done.



New ASP.NET MVC 4 Project

Project Template

Select a template: Intranet Application

Description: A default ASP.NET MVC 4 project that uses Windows authentication.

View engine: Razor

□ Create a unit test project

Test project name: MyMvcApp.Tests

Test framework: Visual Studio Unit Test

Figure 68: Choosing an ASP.NET MVC project template

Adding the MongoDB Driver

Add the MongoDB driver files to your project to access MongoDB by right-clicking **References** and selecting **Add Reference** as shown in Figure 69. The Reference Manager will open. Add the **MongoDB.Bson.dll** and **MongoDB.Driver.dll** files for the MongoDB driver.

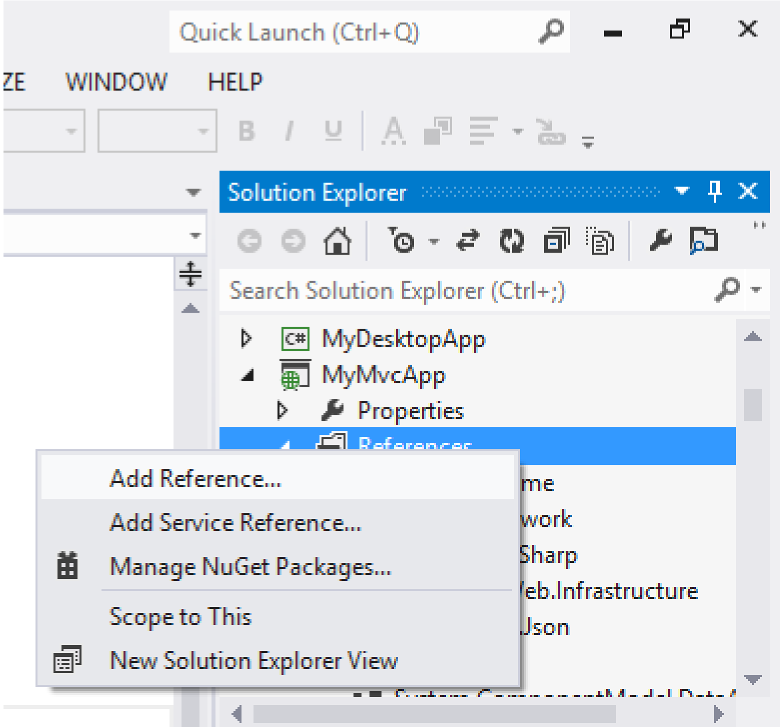


Figure 69: Adding References

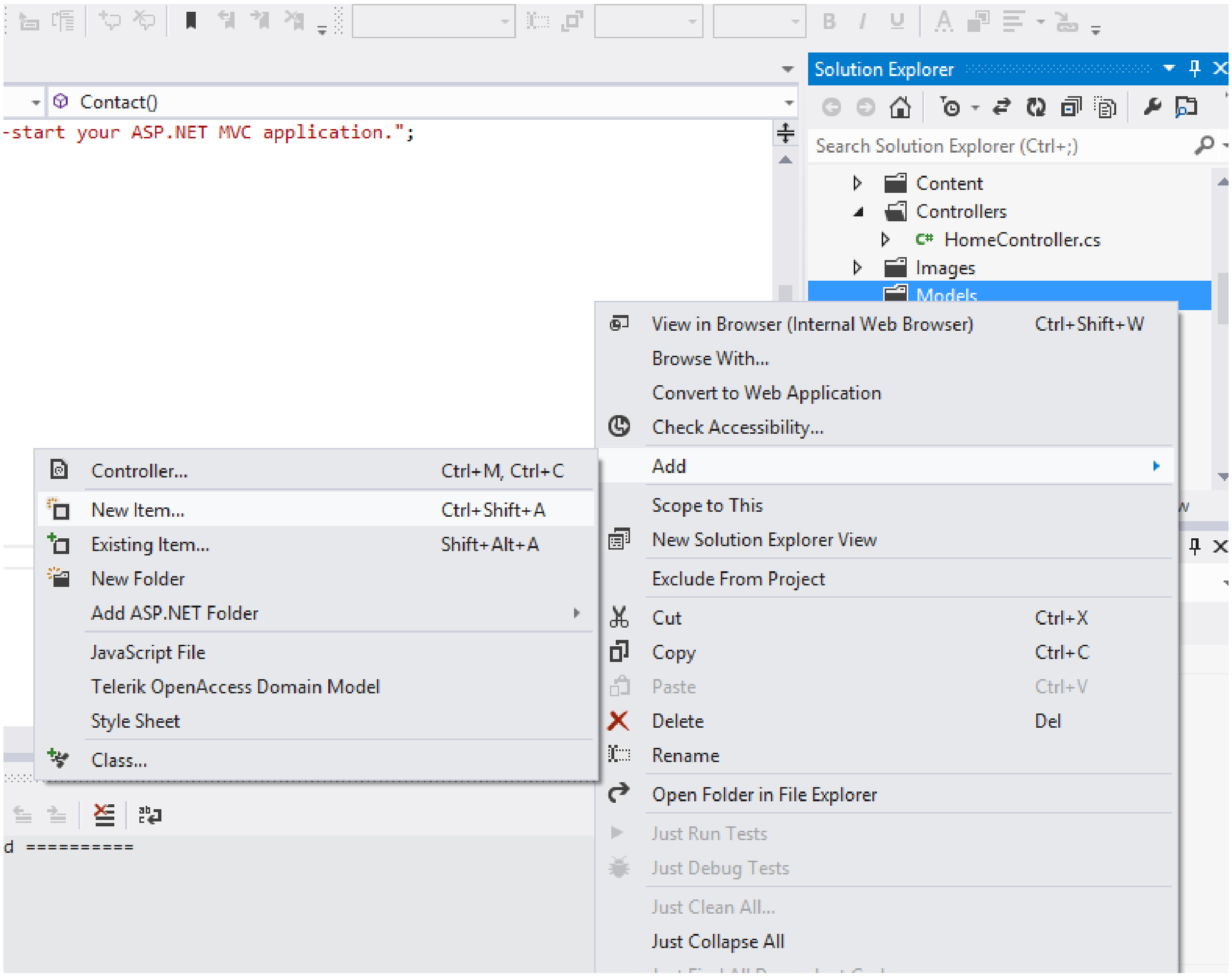
Adding a Model

Create a model called **Employee**. This model will be the entity object of the MongoDB document.

Right-click the **Models** folder, select **Add**, and then choose **New Item** as shown in Figure 70.

A dialog will appear. Select **Class** and enter the file name **Employee.cs**. The following code is an example of the Employee.cs file.

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

namespace MyMvcApp.Models
{
    public class Employee
    {
    }
}
```



```
-start your ASP.NET MVC application.";
```

Figure 70: Adding a new item for a new model

First, add the following namespaces for the MongoDB driver.

```
using MongoDB.Bson.Serialization.Attributes;
using MongoDB.Bson;
```

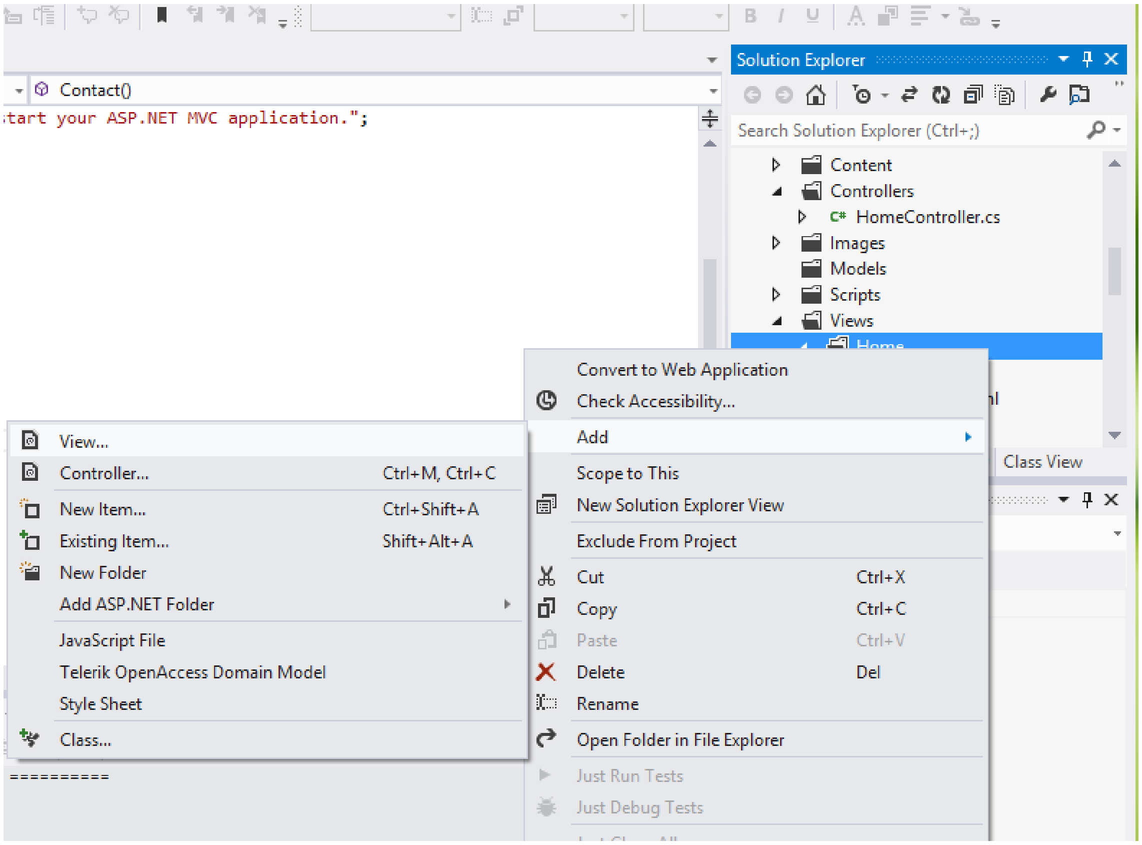
To implement the **`Employee`** model, write the following code.

```
    class Employee
    {
        [BsonElementAttribute("_id")]
        public ObjectId Id { set; get; }
        [BsonElementAttribute("name")]
        public string Name { set; get; }
        [BsonElementAttribute("email")]
        public string Email { set; get; }
        [BsonElementAttribute("level")]
        public string Level { set; get; }
        [BsonElementAttribute("createddate")]
        public DateTime CreateDate { set; get; }

        public override string ToString()
        {
            string str = string.Format("Id: {0}\r\nName: {1}\r\nEmail:
{2}\r\nLevel: {3}\r\nCreatedDate: {4}\r\n\r\n",
                         Id, Name, Email, Level, CreateDate);
            return str;
        }
    }
```

The **`Employee`** model will be mapped to the document object of MongoDB.

Adding a View

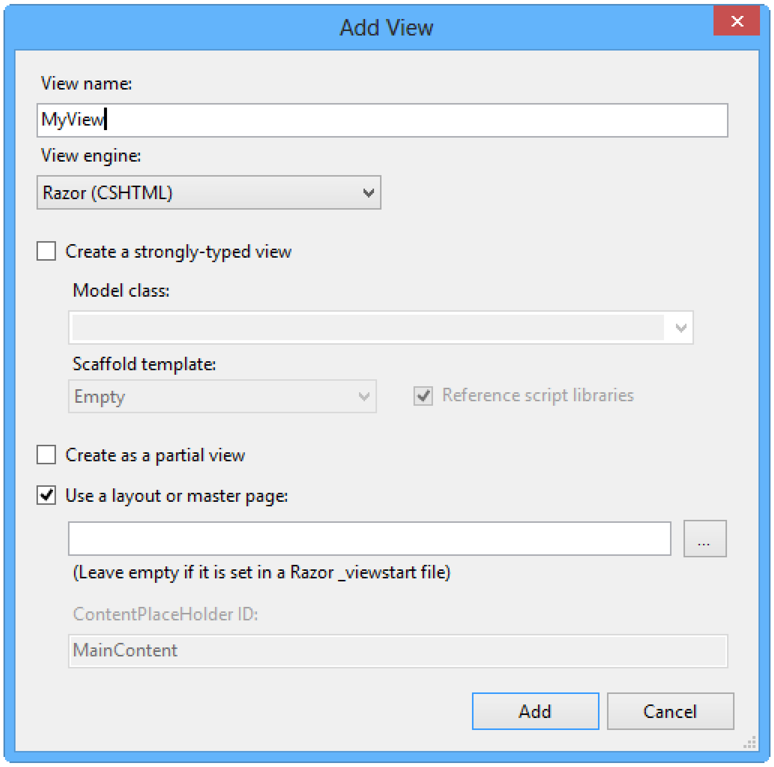
Your ASP.NET MVC project has a Views folder that consists of view objects. Open it, and then right-click the **Home** folder inside. You will get a context menu as shown in Figure 71. Choose **Add** from the menu, and then select **View**.



```
tart your ASP.NET MVC application.";
```

Figure 71: Adding new View

The Add View window will open, as shown in Figure 72. Name the view **MyView**. For the **View engine**, choose **Razor**. Click **Add** when you are finished.



View name: MyView

View engine: Razor (CSHTML)

☐ Create a strongly-typed view

Model class:

Scaffold template: Empty

☑ Reference script libraries

☐ Create as a partial view

☑ Use a layout or master page:

(Leave empty if it is set in a Razor _viewstart file)

ContentPlaceHolder ID: MainContent

Figure 72: Add View Window

Next, we will modify the **MyView.cshtml** file. Add the model to the View form.

```
@model MyMvcApp.Models.Employee[]
```

The following code example adds a table to the View HTML.

```
@model MyMvcApp.Models.Employee[]
@{
    ViewBag.Title = "MyView";
}

<h2>List of Employee</h2>
<br />
<table>
   <tr>
       <td>Name</td>
   </tr>
    @for (int i = 0; i < Model.Count(); i++)
    {
        <tr>
            <td>@Model[i].Id</td>
            <td>@Model[i].Name</td>
            <td>@Model[i].Email</td>
            <td>@Model[i].Level</td>
            <td>@Model[i].CreateDate</td>
        </tr>
    }
</table>
```

Modifying the Controller

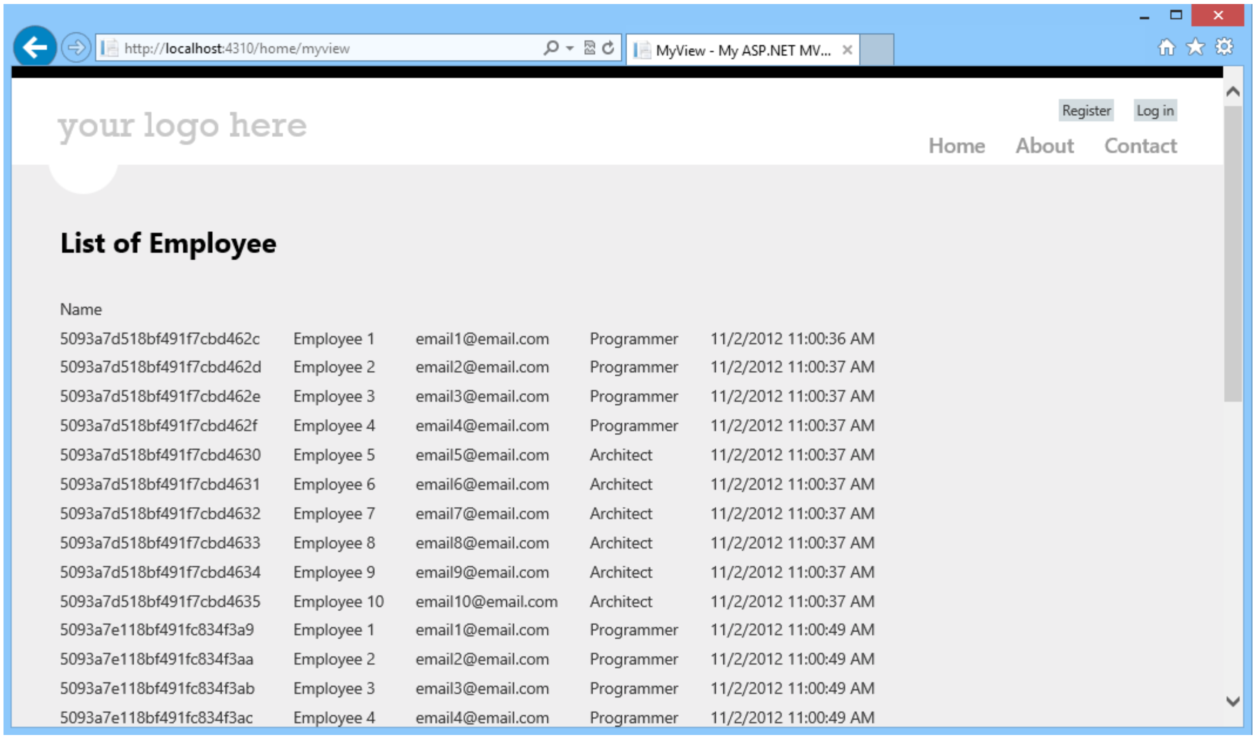
Now that a model and a view has been added to the application, we need to map the view and controller, and push the model to the view.

Because the view form has been added under the Views and Home folders, the view is mapped to a Home controller.

Open the Home controller file, **`HomeController.cs`**. Add a new method for handling view requests as illustrated in the following example.

```
        public ActionResult MyView()
        {
            string connectionString = "mongodb://localhost";
            string databaseName = "csharp";

            // Get database.
            Console.WriteLine(">>Get database");
            MongoServer server = MongoServer.Create(connectionString);
            MongoDatabase database = server.GetDatabase(databaseName);

            var employees = database.GetCollection<Employee>("employee");
            var data = from a in employees.AsQueryable<Employee>() select a;

            return View(data.ToArray<Employee>());
        }
```

Save all the files, and then compile and run the application. Navigate to the URL that your view created, for instance, http://localhost:4310/home/myview.



List of Employee

Name				
5093a7d518bf491f7cbd462c	Employee 1	email1@email.com	Programmer	11/2/2012 11:00:36 AM
5093a7d518bf491f7cbd462d	Employee 2	email2@email.com	Programmer	11/2/2012 11:00:37 AM
5093a7d518bf491f7cbd462e	Employee 3	email3@email.com	Programmer	11/2/2012 11:00:37 AM
5093a7d518bf491f7cbd462f	Employee 4	email4@email.com	Programmer	11/2/2012 11:00:37 AM
5093a7d518bf491f7cbd4630	Employee 5	email5@email.com	Architect	11/2/2012 11:00:37 AM
5093a7d518bf491f7cbd4631	Employee 6	email6@email.com	Architect	11/2/2012 11:00:37 AM
5093a7d518bf491f7cbd4632	Employee 7	email7@email.com	Architect	11/2/2012 11:00:37 AM
5093a7d518bf491f7cbd4633	Employee 8	email8@email.com	Architect	11/2/2012 11:00:37 AM
5093a7d518bf491f7cbd4634	Employee 9	email9@email.com	Architect	11/2/2012 11:00:37 AM
5093a7d518bf491f7cbd4635	Employee 10	email10@email.com	Architect	11/2/2012 11:00:37 AM
5093a7e118bf491fc834f3a9	Employee 1	email1@email.com	Programmer	11/2/2012 11:00:49 AM
5093a7e118bf491fc834f3aa	Employee 2	email2@email.com	Programmer	11/2/2012 11:00:49 AM
5093a7e118bf491fc834f3ab	Employee 3	email3@email.com	Programmer	11/2/2012 11:00:49 AM
5093a7e118bf491fc834f3ac	Employee 4	email4@email.com	Programmer	11/2/2012 11:00:49 AM

Figure 73: Executing an ASP.NET MVC application with MongoDB data

Chapter 13 Export and Import Database

In this chapter, you will learn how to export and import a MongoDB database.

Exporting Data

You may want to export the MongoDB database for external use. MongoDB provides an export mechanism called **`mongoexport`** in the Mongo shell. The output can be defined as JSON, TSV, or CSV.

The following is an example of the **`mongoexport`** parameters.

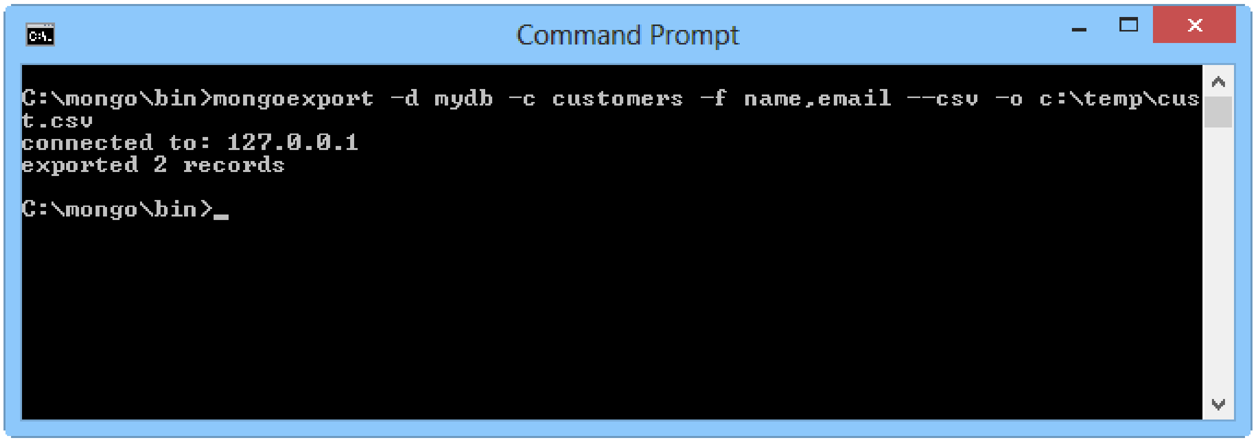
```
options:
  --help                  produce help message
  -v [ --verbose ]        be more verbose (include multiple times for more
                          verbosity e.g. -vvvvv)
  -h [ --host ] arg       mongo host to connect to ("left,right" for pairs)
  -d [ --db ] arg         database to use
  -c [ --collection ] arg where 'arg' is the collection to use
  -u [ --username ] arg   username
  -p [ --password ] arg   password
  --dbpath arg            directly access mongod data files in the given path,
                          instead of connecting to a mongod instance - needs to
                          lock the data directory, so cannot be used if a
                          mongod is currently accessing the same path
  --directoryperdb        if dbpath specified, each db is in a separate
                          directory
  -q [ --query ] arg      query filter, as a JSON string
  -f [ --fields ] arg     comma separated list of field names e.g. -f name,age
  --csv                   export to CSV instead of JSON, requires -f
  -o [ --out ] arg        output file; if not specified, stdout is used
```

To execute **`mongoexport`**, navigate to the folder where MongoDB is installed (e.g., c:\mongo\bin) in the Command Prompt window.

Let’s export the collection **`customers`** from the database **`mydb`** as a CSV file with the **`name`** and **`email`** fields. The following code example performs this export.

```
mongoexport -d mydb -c customers -f name,email --csv -o c:\temp\cust.csv
```

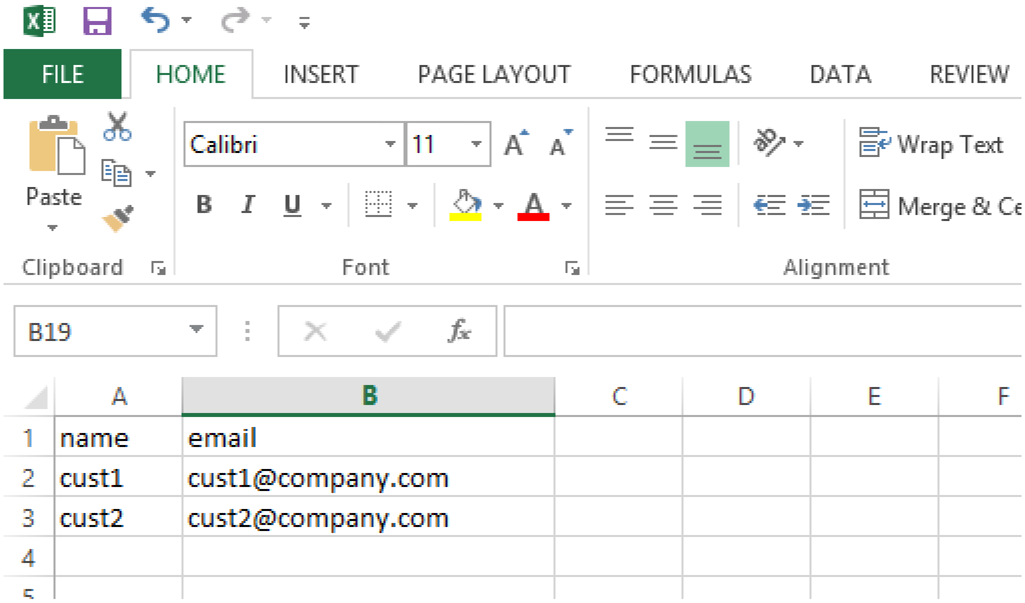
You can see the sample output in Figure 74.



```
C:\mongo\bin>mongoexport -d mydb -c customers -f name,email --csv -o c:\temp\cus
t.csv
connected to: 127.0.0.1
exported 2 records

C:\mongo\bin>_
```

Figure 74: Exporting data

Figure 75 shows the resulting CSV file opened in Excel.



	A	B	C	D	E	F
1	name	email				
2	cust1	cust1@company.com				
3	cust2	cust2@company.com				
4						

Figure 75: Sample export result

If your database applies access restrictions, you can pass a username and password. You can add **-u** for username and **-p** for password, as shown in the following code example.

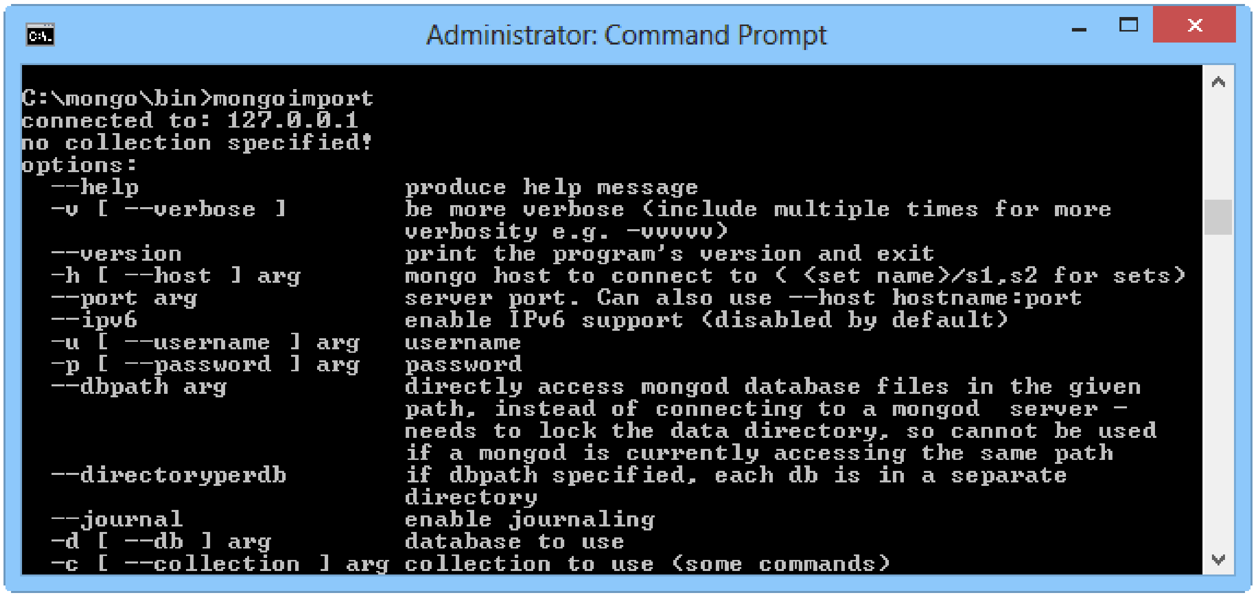
```
mongoexport -d mydb -u user -p password -c customers -f name,email --csv -o
c:\temp\cust.csv
```

If you want to export data from a remote MongoDB server, you can define the server name explicitly. Use **-h** to define a server name.

```
mongoexport -h dbserver01 -d mydb -u user -p password -c customers -f
name,email --csv -o c:\temp\cust.csv
```

Importing Data

You can also import data to a MongoDB database. MongoDB provides mongoimport.exe for importing data. If you type **`mongoimport`**, you can view information about its parameters, as shown in Figure 76.



```
C:\mongo\bin>mongoimport
connected to: 127.0.0.1
no collection specified!
options:
  --help                  produce help message
  -v [ --verbose ]        be more verbose (include multiple times for more
                          verbosity e.g. -vvvvv)
  --version               print the program's version and exit
  -h [ --host ] arg       mongo host to connect to ( <set name>/s1,s2 for sets)
  --port arg              server port. Can also use --host hostname:port
  --ipv6                  enable IPv6 support (disabled by default)
  -u [ --username ] arg   username
  -p [ --password ] arg   password
  --dbpath arg            directly access mongod database files in the given
                          path, instead of connecting to a mongod  server -
                          needs to lock the data directory, so cannot be used
                          if a mongod is currently accessing the same path
  --directoryperdb        if dbpath specified, each db is in a separate
                          directory
  --journal               enable journaling
  -d [ --db ] arg         database to use
  -c [ --collection ] arg collection to use (some commands)
```

Figure 76: Information about mongoimport

The following is a summary of **`mongoimport`** parameters.

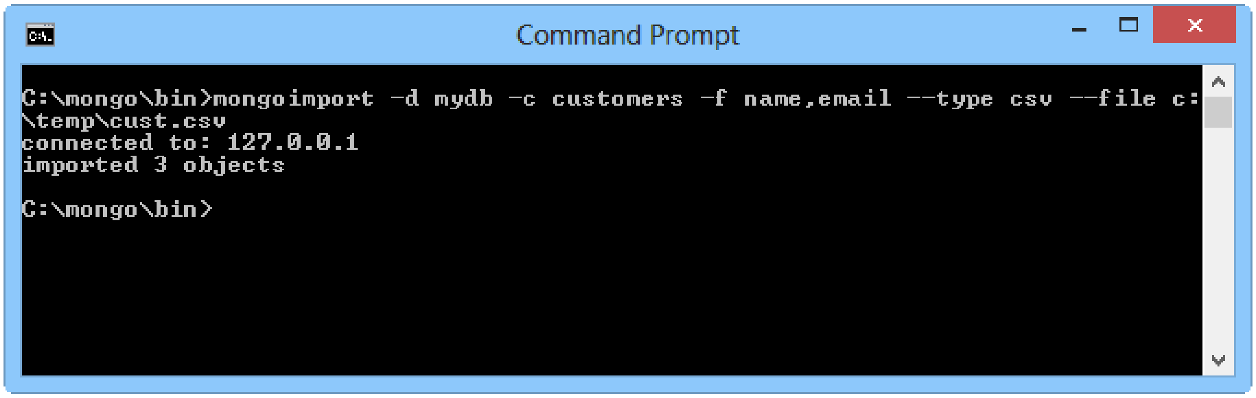
```
options:
  --help                  produce help message
  -v [ --verbose ]        be more verbose (include multiple times for more
                          verbosity e.g. -vvvvv)
  -h [ --host ] arg       mongo host to connect to ("left,right" for pairs)
  --port arg              server port (can also use --host hostname:port)
  --ipv6                  enable IPv6 support (disabled by default)
  -d [ --db ] arg         database to use
  -c [ --collection ] arg collection to use (some commands)
  -u [ --username ] arg   username
  -p [ --password ] arg   password
  --dbpath arg            directly access mongod data files in the given path,
                          instead of connecting to a mongod instance - needs to
                          lock the data directory, so this cannot be used if a
                          mongod is currently accessing the same path
  --directoryperdb        if dbpath specified, each db is in a separate
```

```
                            directory
  -f [ --fields ] arg       comma separated list of field names
                            e.g. -f name,age
  --fieldFile arg           file with fields names - 1 per line
  --ignoreBlanks            if given, empty fields in csv and tsv
                            will be ignored
  --type arg                type of file to import (json, csv, tsv).
                            default: json
  --file arg                file to import from; if not specified stdin is used
  --drop                    drop collection first
  --headerline              CSV,TSV only - use first line as headers
  --upsert                  insert or update objects that already exist
  --upsertFields arg        comma-separated fields for the query part of the
                            upsert. You should make sure this is indexed.
  --stopOnError             stop importing at the first error rather
                            than continuing
  --jsonArray               load a json array, not one item per line.
                            Currently limited to 4MB.
```

For example, import the data file **c:\temp\cust.csv** into the MongoDB database **mydb**. The data can be inserted into the **`customers`** collection with the following code example.

```
mongoimport -d mydb -c customers -f name,email --csv -o c:\temp\cust.csv
```

Running the sample will produce the output shown in Figure 77.



```
C:\mongo\bin>mongoimport -d mydb -c customers -f name,email --type csv --file c:
\temp\cust.csv
connected to: 127.0.0.1
imported 3 objects

C:\mongo\bin>
```

Figure 77: Importing data

Chapter 14 Back up and Restore

In this chapter, you will learn how to back up and restore a database in MongoDB.

Back up

MongoDB provides a tool called **mongodump.exe** to back up your MongoDB database. If you run **mongodump** for help, you get a list of **mongodump** parameters.

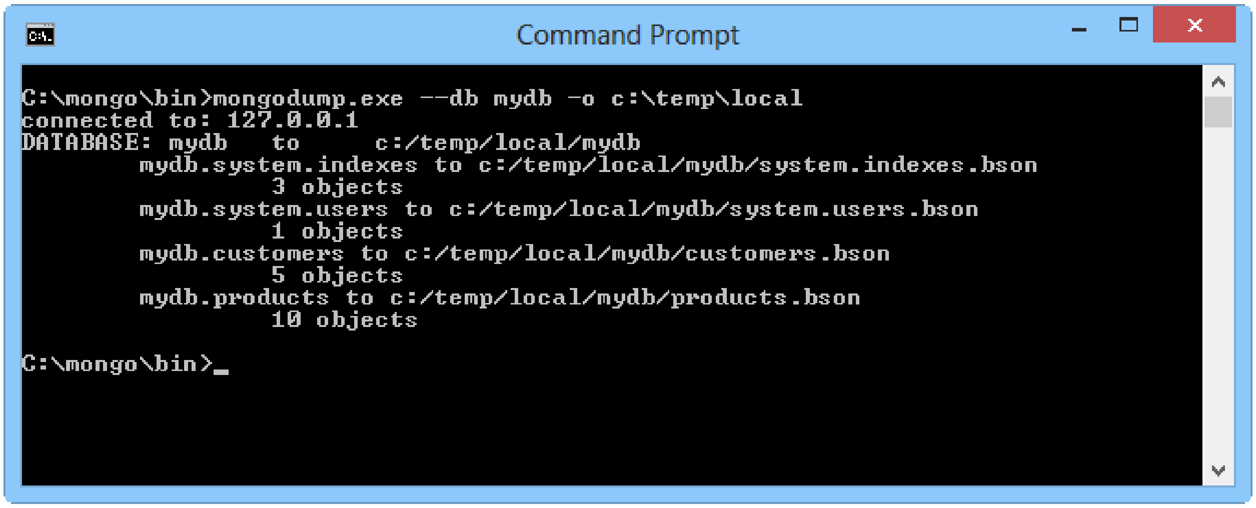
The following is a list of **mongodump** parameters you can use.

```
options:
  --help                    produce help message
  -v [ --verbose ]          be more verbose (include multiple times for more
                            verbosity e.g. -vvvvv)
  -h [ --host ] arg         mongo host to connect to ("left,right" for pairs)
  -d [ --db ] arg           database to use
  -c [ --collection ] arg   collection to use (some commands)
  -u [ --username ] arg     username
  -p [ --password ] arg     password
  --dbpath arg              directly access mongod data files in the
                            given path, instead of connecting to a mongod
                            instance – needs to lock the data directory, so
                            cannot be used if a mongod is currently accessing
                            the same path
  --directoryperdb          if dbpath specified, each db is in a separate
                            directory
  -o [ --out ] arg (=dump)  output directory
  -q [ --query ] arg        json query
  --oplog                   point in time backup (requires an oplog)
  --repair                  repairs documents as it dumps from a corrupt db
                            (requires --dbpath and -d/--db)
  --forceTableScan          force a table scan (do not use $snapshot)
```

For example, if you wanted to back up the database **mydb** into the folder c:\temp\local, you would use the following code example.

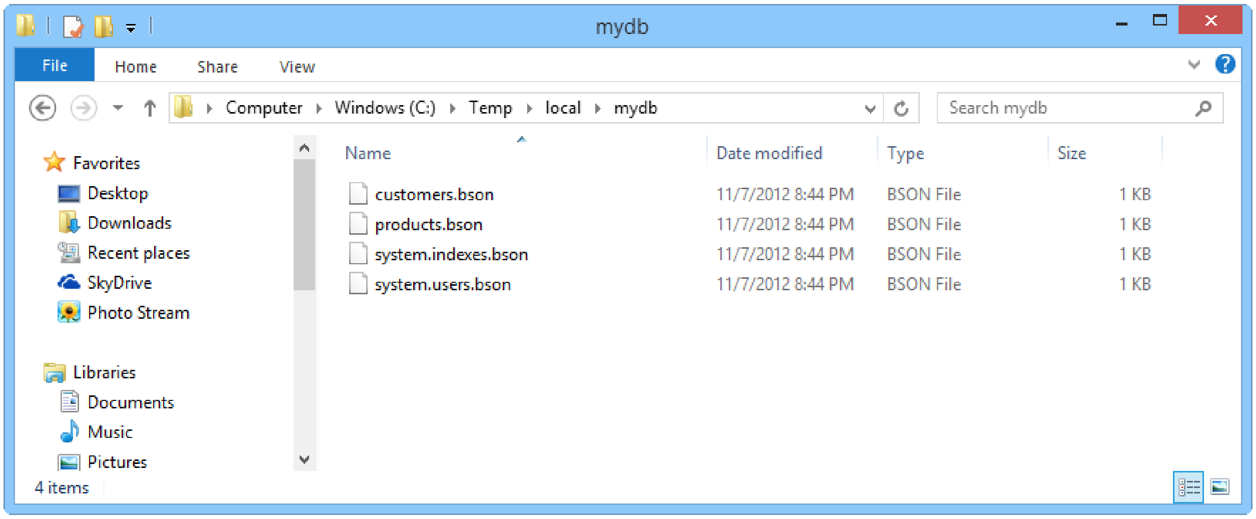
```
mongodump.exe –-db mydb –o c:\Temp\local
```

The sample output is shown in Figure 78. If you open the folder where you placed the backup, you will see a list of BSON files as shown in Figure 79.



```
C:\mongo\bin>mongodump.exe --db mydb -o c:\temp\local
connected to: 127.0.0.1
DATABASE: mydb   to     c:/temp/local/mydb
        mydb.system.indexes to c:/temp/local/mydb/system.indexes.bson
                 3 objects
        mydb.system.users to c:/temp/local/mydb/system.users.bson
                 1 objects
        mydb.customers to c:/temp/local/mydb/customers.bson
                 5 objects
        mydb.products to c:/temp/local/mydb/products.bson
                 10 objects

C:\mongo\bin>_
```

Figure 78: Backing up the mydb database



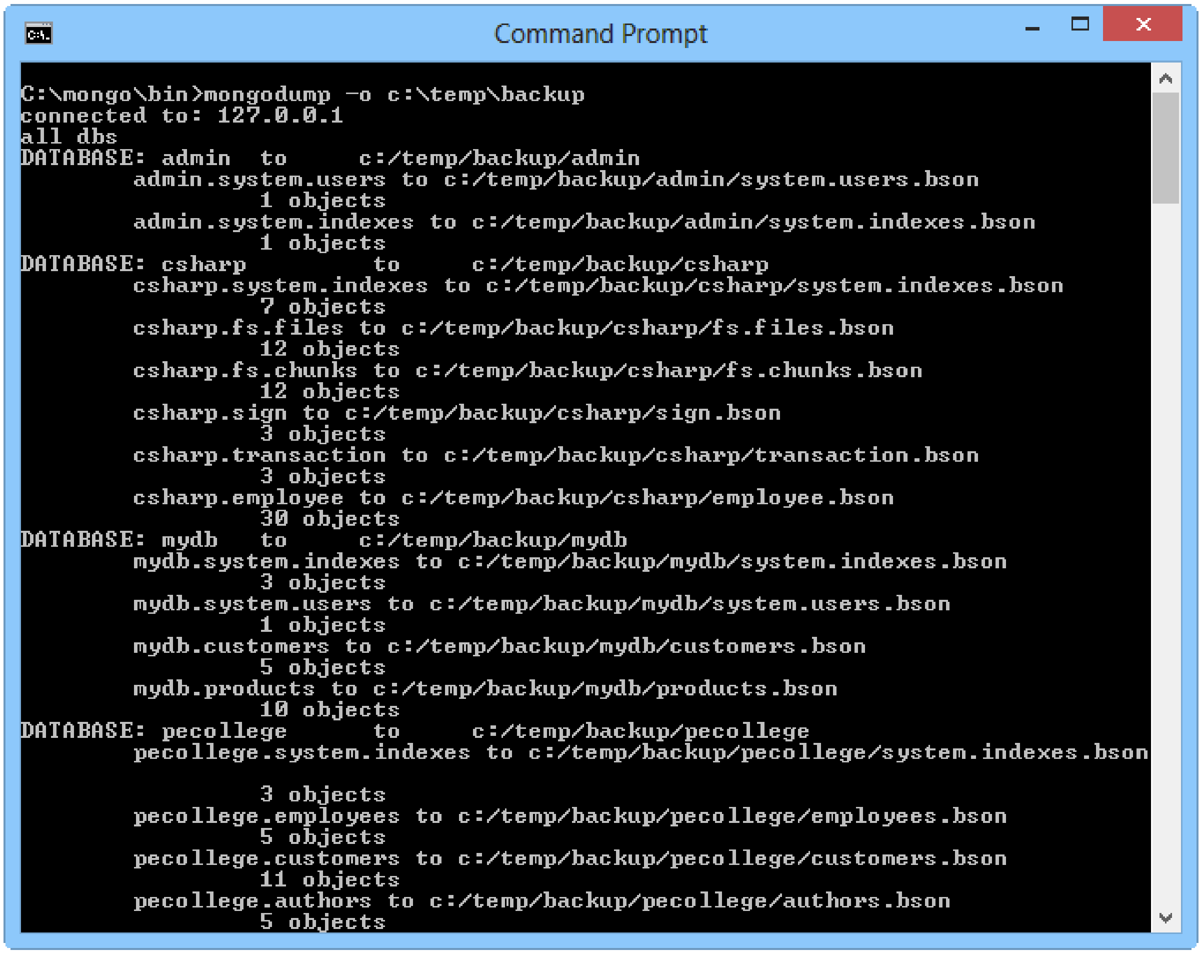
Name	Date modified	Type	Size
customers.bson	11/7/2012 8:44 PM	BSON File	1 KB
products.bson	11/7/2012 8:44 PM	BSON File	1 KB
system.indexes.bson	11/7/2012 8:44 PM	BSON File	1 KB
system.users.bson	11/7/2012 8:44 PM	BSON File	1 KB

Figure 79: Backup files

You can also back up all databases. For instance, you can back up data in the folder c:\temp\backup using the following code example.

```
mongodump.exe -o c:\Temp\backup
```

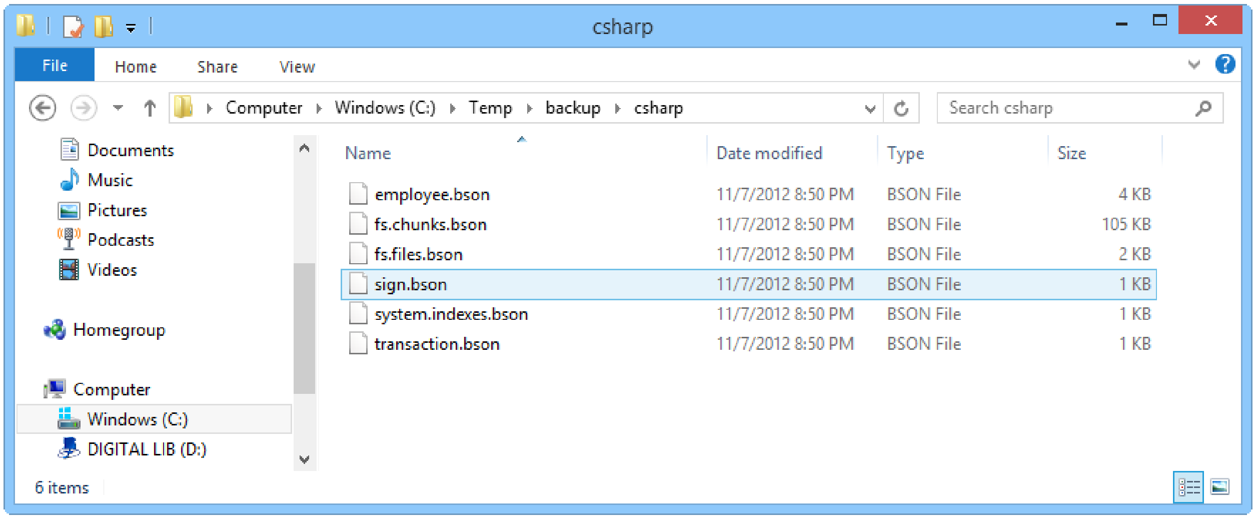
The console output is shown in Figure 80.



```
C:\mongo\bin>mongodump -o c:\temp\backup
connected to: 127.0.0.1
all dbs
DATABASE: admin  to     c:/temp/backup/admin
        admin.system.users to c:/temp/backup/admin/system.users.bson
                 1 objects
        admin.system.indexes to c:/temp/backup/admin/system.indexes.bson
                 1 objects
DATABASE: csharp         to     c:/temp/backup/csharp
        csharp.system.indexes to c:/temp/backup/csharp/system.indexes.bson
                 7 objects
        csharp.fs.files to c:/temp/backup/csharp/fs.files.bson
                 12 objects
        csharp.fs.chunks to c:/temp/backup/csharp/fs.chunks.bson
                 12 objects
        csharp.sign to c:/temp/backup/csharp/sign.bson
                 3 objects
        csharp.transaction to c:/temp/backup/csharp/transaction.bson
                 3 objects
        csharp.employee to c:/temp/backup/csharp/employee.bson
                 30 objects
DATABASE: mydb   to     c:/temp/backup/mydb
        mydb.system.indexes to c:/temp/backup/mydb/system.indexes.bson
                 3 objects
        mydb.system.users to c:/temp/backup/mydb/system.users.bson
                 1 objects
        mydb.customers to c:/temp/backup/mydb/customers.bson
                 5 objects
        mydb.products to c:/temp/backup/mydb/products.bson
                 10 objects
DATABASE: pecollege      to     c:/temp/backup/pecollege
        pecollege.system.indexes to c:/temp/backup/pecollege/system.indexes.bson

                 3 objects
        pecollege.employees to c:/temp/backup/pecollege/employees.bson
                 5 objects
        pecollege.customers to c:/temp/backup/pecollege/customers.bson
                 11 objects
        pecollege.authors to c:/temp/backup/pecollege/authors.bson
                 5 objects
```

Figure 80: Backing up all databases

If you open the c:\temp\backup folder, you will see all databases in the BSON file format. For instance, if you open the **csharp** database you will see several BSON files as shown in Figure 81.

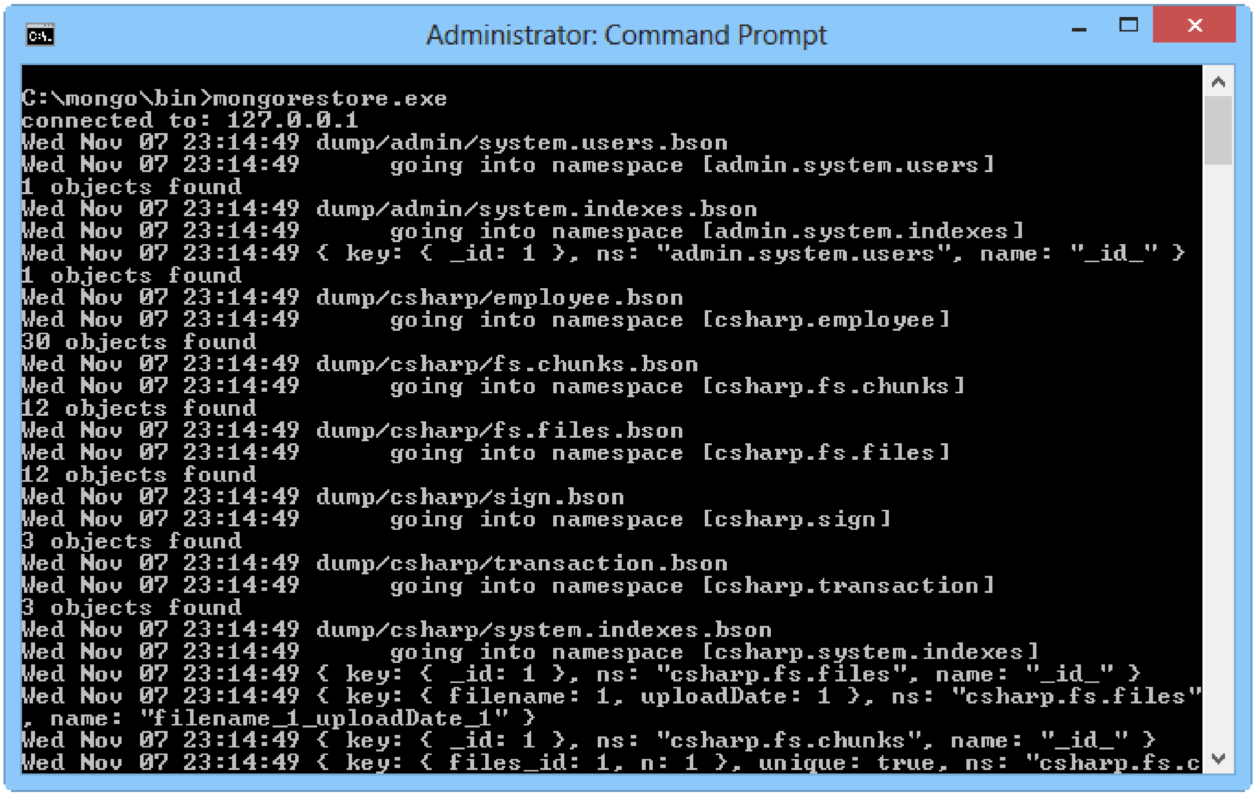


Name	Date modified	Type	Size
employee.bson	11/7/2012 8:50 PM	BSON File	4 KB
fs.chunks.bson	11/7/2012 8:50 PM	BSON File	105 KB
fs.files.bson	11/7/2012 8:50 PM	BSON File	2 KB
sign.bson	11/7/2012 8:50 PM	BSON File	1 KB
system.indexes.bson	11/7/2012 8:50 PM	BSON File	1 KB
transaction.bson	11/7/2012 8:50 PM	BSON File	1 KB

Figure 81: Showing database csharp

Restore

After backing up the data, you can also restore the data to MongoDB. To restore the data, you use **`mongorestore.exe`**. If you type it in the Command Prompt window and press Enter, you will get a response as shown in Figure 82.



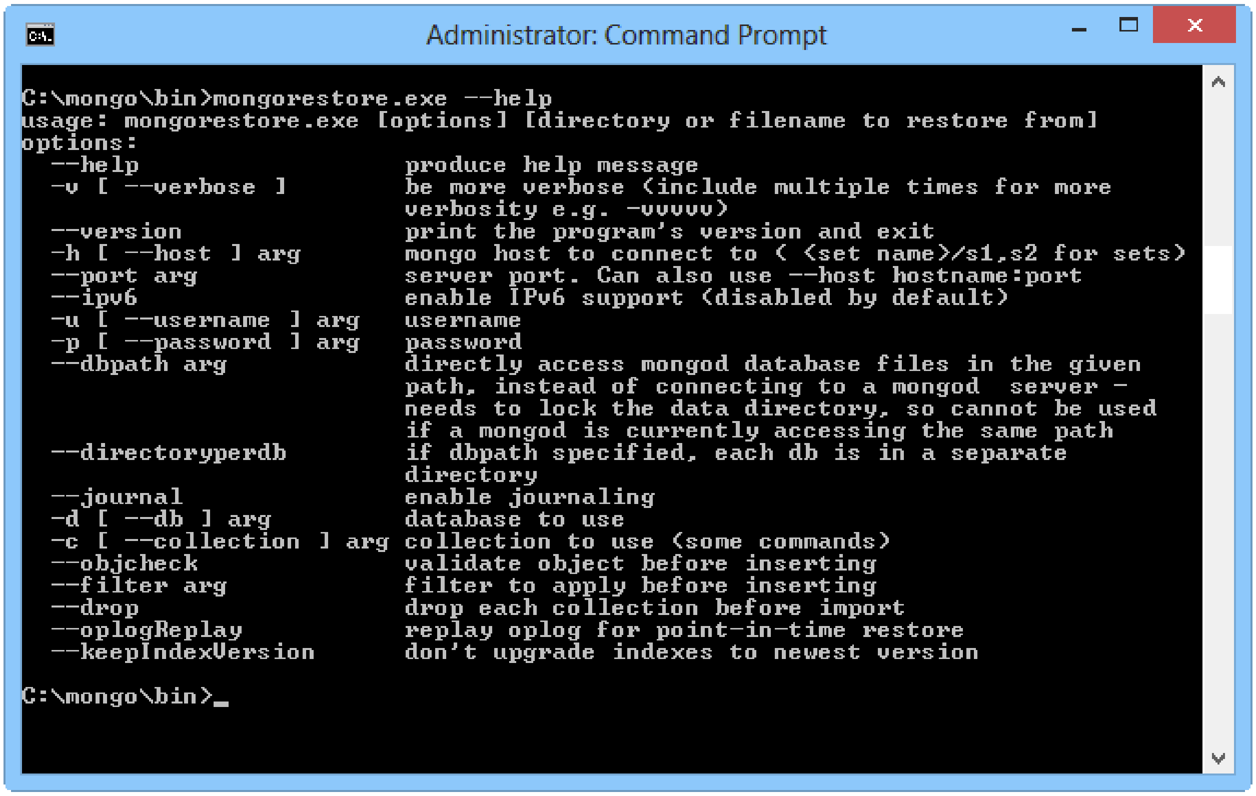
```
C:\mongo\bin>mongorestore.exe
connected to: 127.0.0.1
Wed Nov 07 23:14:49 dump/admin/system.users.bson
Wed Nov 07 23:14:49      going into namespace [admin.system.users]
1 objects found
Wed Nov 07 23:14:49 dump/admin/system.indexes.bson
Wed Nov 07 23:14:49      going into namespace [admin.system.indexes]
Wed Nov 07 23:14:49 { key: { _id: 1 }, ns: "admin.system.users", name: "_id_" }
1 objects found
Wed Nov 07 23:14:49 dump/csharp/employee.bson
Wed Nov 07 23:14:49      going into namespace [csharp.employee]
30 objects found
Wed Nov 07 23:14:49 dump/csharp/fs.chunks.bson
Wed Nov 07 23:14:49      going into namespace [csharp.fs.chunks]
12 objects found
Wed Nov 07 23:14:49 dump/csharp/fs.files.bson
Wed Nov 07 23:14:49      going into namespace [csharp.fs.files]
12 objects found
Wed Nov 07 23:14:49 dump/csharp/sign.bson
Wed Nov 07 23:14:49      going into namespace [csharp.sign]
3 objects found
Wed Nov 07 23:14:49 dump/csharp/transaction.bson
Wed Nov 07 23:14:49      going into namespace [csharp.transaction]
3 objects found
Wed Nov 07 23:14:49 dump/csharp/system.indexes.bson
Wed Nov 07 23:14:49      going into namespace [csharp.system.indexes]
Wed Nov 07 23:14:49 { key: { _id: 1 }, ns: "csharp.fs.files", name: "_id_" }
Wed Nov 07 23:14:49 { key: { filename: 1, uploadDate: 1 }, ns: "csharp.fs.files"
, name: "filename_1_uploadDate_1" }
Wed Nov 07 23:14:49 { key: { _id: 1 }, ns: "csharp.fs.chunks", name: "_id_" }
Wed Nov 07 23:14:49 { key: { files_id: 1, n: 1 }, unique: true, ns: "csharp.fs.c
```

Figure 82: Executing mongorestore.exe

As you can see, mongorestore.exe writes the dumped data to restore the database. It will run automatically. You can use the following script for the restore process.

```
mongorestore.exe --help
```

You will get a response as shown in Figure 83.



```
C:\mongo\bin>mongorestore.exe --help
usage: mongorestore.exe [options] [directory or filename to restore from]
options:
  --help                    produce help message
  -v [ --verbose ]          be more verbose (include multiple times for more
                            verbosity e.g. -vvvvv)
  --version                 print the program's version and exit
  -h [ --host ] arg         mongo host to connect to ( <set name>/s1,s2 for sets)
  --port arg                server port. Can also use --host hostname:port
  --ipv6                    enable IPv6 support (disabled by default)
  -u [ --username ] arg     username
  -p [ --password ] arg     password
  --dbpath arg              directly access mongod database files in the given
                            path, instead of connecting to a mongod  server -
                            needs to lock the data directory, so cannot be used
                            if a mongod is currently accessing the same path
  --directoryperdb          if dbpath specified, each db is in a separate
                            directory
  --journal                 enable journaling
  -d [ --db ] arg           database to use
  -c [ --collection ] arg   collection to use (some commands)
  --objcheck                validate object before inserting
  --filter arg              filter to apply before inserting
  --drop                    drop each collection before import
  --oplogReplay             replay oplog for point-in-time restore
  --keepIndexVersion        don't upgrade indexes to newest version

C:\mongo\bin>_
```

Figure 83: Getting information about mongorestore parameters

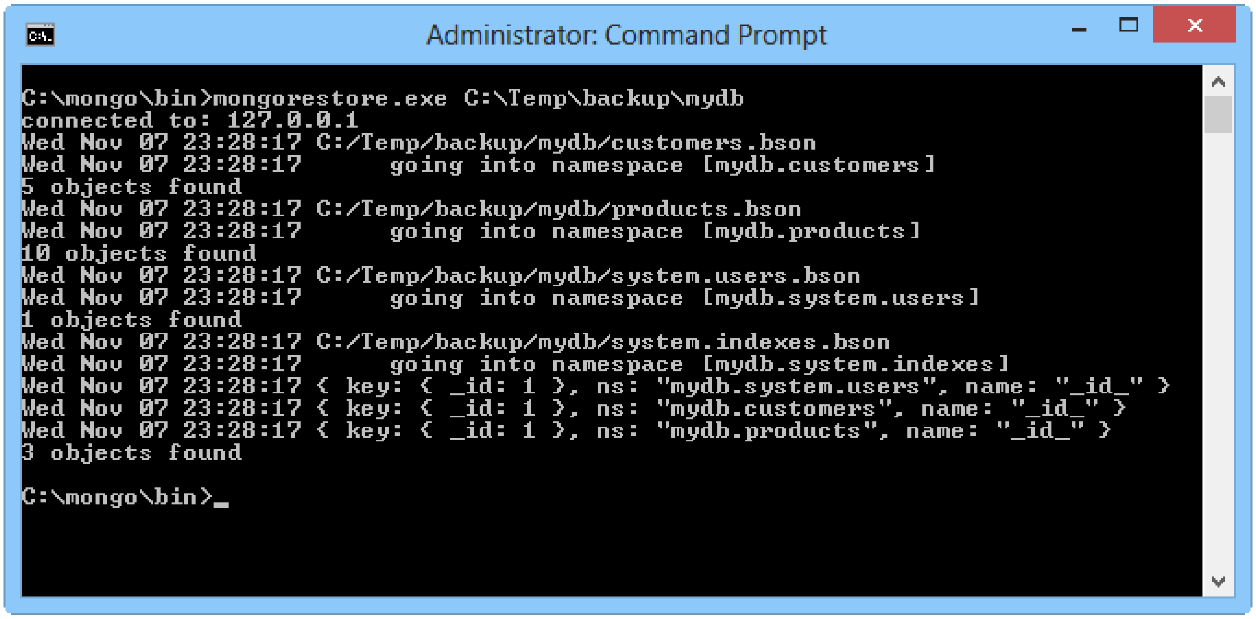
The mongorestore parameters are summarized in the following code example.

```
usage: ./mongorestore [options] [directory or filename to restore from]
options:
  --help                    produce help message
  -v [ --verbose ]          be more verbose (include multiple times for more
                            verbosity e.g. -vvvvv)
  --version                 print the program's version and exit
  -h [ --host ] arg         mongo host to connect to ( <set name>/s1,s2 for
                            sets)
  --port arg                server port. Can also use --host hostname:port
  --ipv6                    enable IPv6 support (disabled by default)
  -u [ --username ] arg     username
  -p [ --password ] arg     password
  --dbpath arg              directly access mongod database files in the given
                            path, instead of connecting to a mongod server -
                            needs to lock the data directory, so cannot be used
                            if a mongod is currently accessing the same path
  --directoryperdb          if dbpath specified, each db is in a separate
                            directory
  --journal                 enable journaling
  -d [ --db ] arg           database to use
  -c [ --collection ] arg collection to use (some commands)
  --objcheck                validate object before inserting
  --filter arg            filter to apply before inserting
  --drop                  drop each collection before import
  --oplogReplay           replay oplog for point-in-time restore
  --keepIndexVersion      don't upgrade indexes to newest version
```

The following code example can be used to restore the **mydb** database located in the **c:\temp\local** folder.

```
mongorestore.exe C:\Temp\backup\mydb
```

Running this code will produce the response shown in Figure 84.



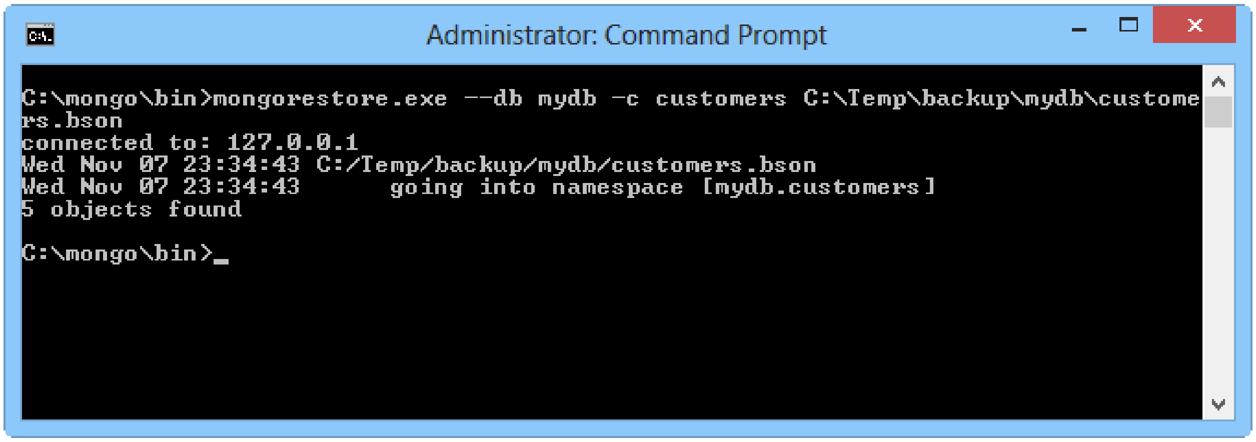
```
C:\mongo\bin>mongorestore.exe C:\Temp\backup\mydb
connected to: 127.0.0.1
Wed Nov 07 23:28:17 C:/Temp/backup/mydb/customers.bson
Wed Nov 07 23:28:17      going into namespace [mydb.customers]
5 objects found
Wed Nov 07 23:28:17 C:/Temp/backup/mydb/products.bson
Wed Nov 07 23:28:17      going into namespace [mydb.products]
10 objects found
Wed Nov 07 23:28:17 C:/Temp/backup/mydb/system.users.bson
Wed Nov 07 23:28:17      going into namespace [mydb.system.users]
1 objects found
Wed Nov 07 23:28:17 C:/Temp/backup/mydb/system.indexes.bson
Wed Nov 07 23:28:17      going into namespace [mydb.system.indexes]
Wed Nov 07 23:28:17 { key: { _id: 1 }, ns: "mydb.system.users", name: "_id_" }
Wed Nov 07 23:28:17 { key: { _id: 1 }, ns: "mydb.customers", name: "_id_" }
Wed Nov 07 23:28:17 { key: { _id: 1 }, ns: "mydb.products", name: "_id_" }
3 objects found

C:\mongo\bin>_
```

Figure 84: Executing restore database mydb

You can specify what kind of collection will be restored to MongoDB. For instance, if you want to restore the file customers.bson to the **customers** collection in the database **mydb**, you can use the following code example.

```
mongorestore.exe --db mydb -c customers C:\Temp\backup\mydb\custome
rs.bson
```

The response is shown in Figure 85.



```
C:\mongo\bin>mongorestore.exe --db mydb -c customers C:\Temp\backup\mydb\custome
rs.bson
connected to: 127.0.0.1
Wed Nov 07 23:34:43 C:/Temp/backup/mydb/customers.bson
Wed Nov 07 23:34:43      going into namespace [mydb.customers]
5 objects found

C:\mongo\bin>_
```

Figure 85: Restoring data with a specific collection type