MySQL Connector/Net Developer Guide

Abstract

This manual describes how to install and configure MySQL Connector/Net, the driver that enables .NET applications to communicate with MySQL servers, and how to use it to develop database applications.

For notes detailing the changes in each release of Connector/Net, see MySQL Connector/Net Release Notes.

For legal information, see the Legal Notices.

For help with using MySQL, please visit either the MySQL Forums or MySQL Mailing Lists, where you can discuss your issues with other MySQL users.

For additional documentation on MySQL products, including translations of the documentation into other languages, and downloadable versions in variety of formats, including HTML and PDF formats, see the MySQL Documentation Library.

Licensing information.   This product may include third-party software, used under license. If you are using a *Commercial* release of MySQL Connector/Net, see this document for licensing information, including licensing information relating to third-party software that may be included in this Commercial release. If you are using a *Community* release of MySQL Connector/Net, see this document for licensing information, including licensing information relating to third-party software that may be included in this Community release.

Document generated on: 2017-05-25 (revision: 52332)

Table of Contents

Preface and Legal Notices

This manual describes how to install, configure, and develop database applications using MySQL Connector/Net, the driver that allows .NET applications to communicate with MySQL servers.

Legal Notices

You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how Oracle disseminates it (that is, electronically for download on a Web site with the software) or on a CD-ROM or similar medium, provided however that the documentation is disseminated together with the software on the same medium. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of Oracle. Oracle and/or its affiliates reserve any and all rights to this documentation not expressly granted above.

Documentation Accessibility

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

Access to Oracle Support

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

Chapter 1 Introduction to MySQL Connector/Net

Connector/Net lets you easily develop .NET applications that require secure, high-performance data connectivity with MySQL. It implements the required ADO.NET interfaces and integrates into ADO.NET-aware tools. Developers can build applications using their choice of .NET languages. Connector/Net is a fully managed ADO.NET driver written in 100% pure C#. It does not use the MySQL C client library.

For notes detailing the changes in each release of Connector/Net, see MySQL Connector/Net Release Notes.

Connector/Net includes full support for:

- Features provided by MySQL Server up to and including MySQL Server 5.7.

- Large-packet support for sending and receiving rows and `BLOB` values up to 2 gigabytes in size.

- Protocol compression, which enables compressing the data stream between the client and server.

- Connections using TCP/IP sockets, named pipes, or shared memory on Windows.

- Connections using TCP/IP sockets or Unix sockets on Unix.

- The Open Source Mono framework developed by Novell.

- Microsoft Entity Framework.

- Microsoft Windows RT.

This document is intended as a user's guide to Connector/Net and includes a full syntax reference. Syntax information is also included within the `ConnectorNET.chm` file included with the Connector/Net distribution.

If you are using MySQL 5.0 or later, and Visual Studio as your development environment, you can also use the MySQL Visual Studio Plugin. The plugin acts as a DDEX (Data Designer Extensibility) provider: you can use the data design tools within Visual Studio to manipulate the schema and objects within a MySQL database. For more information, see MySQL for Visual Studio.

> **Note**
>
> From Connector/Net 5.1 through 6.6, the Visual Studio Plugin is part of the main Connector/Net package. Starting with 6.7, the Visual Studio Plugin has been separated out and renamed to MySQL for Visual Studio. For release information, see MySQL for Visual Studio Release Notes.

MySQL Connector/Net supports full versions of Visual Studio 2008, 2010, 2012, 2013, 2015, and 2017, although the extent of support may be limited depending on your versions of MySQL Connector/Net and Visual Studio. For details, see MySQL for Visual Studio.

MySQL Connector/Net 6.8 (and earlier) does not support Express versions of Microsoft products, including Microsoft Visual Web Developer. MySQL Connector/Net 6.9, 6.10, and 7.0 do provide support for these products.

Key Topics

- For connection string properties when using the `MySqlConnection` class, see Chapter 6, *Connector/Net Connection String Options Reference*.

Chapter 2 Connector/Net Versions

There are several versions of Connector/Net available:

- `Connector/Net 7.0` includes support for the X DevAPI and has expanded cross-platform support to Linux and OS X when using Microsoft .NET Core framework.

- `Connector/Net 6.10` includes Entity Framework Core 1.1 support and enables compression in the .NET Core version of the driver for enhanced cross-platform application development.

- `Connector/Net 6.9` includes new features such as a MySQL Personalization provider, SiteMap Web provider, a simple Membership Web provider, and support for MySQL for Visual Studio 1.2.

- `Connector/Net 6.8` includes new features such as Entity Framework 6 support, added idempotent script for Entity Framework 6 migrations, changed EF migration history table to use a single column as primary key, removed installer validation when MySQL for Visual Studio is installed, and support for MySQL for Visual Studio 1.1.

- `Connector/Net 6.7` includes new features such as Entity Framework 5 support, built-in Load Balancing (to be used with a back end implementing either MySQL Replication or MySQL Clustering), a Memcached client (compatible with Innodb Memcached plugin) and support for Windows Runtime (WinRT) to write store apps. This version also removes all features related to Visual Studio Integration, which are provided in a separate product, MySQL for Visual Studio.

  This version of Connector/Net is no longer supported.

- `Connector/Net 6.6` includes new features such as stored procedure debugging in Microsoft Visual Studio, support for pluggable authentication including the ability to write your own authentication plugins, Entity Framework 4.3 Code First support, and enhancements to partial trust support to allow hosting services to deploy applications without installing the Connector/Net library in the GAC.

  This version of Connector/Net is no longer supported.

- `Connector/Net 6.5` includes new features such as interceptor classes for exceptions and commands, support for the MySQL 5.6+ fractional seconds feature, better partial-trust support, and better IntelliSense, including auto-completion when editing stored procedures or `.mysql` files.

  This version of Connector/Net is no longer supported.

- `Connector/Net 6.4` includes new features such as support for Windows authentication (when connecting to MySQL Server 5.5+), table caching on the client side, simple connection fail-over support, and improved SQL generation from the Entity Framework provider.

  This version of Connector/Net is no longer supported.

- `Connector/Net 6.3` includes new features such as integration with Visual Studio 2010, such as the availability of DDL T4 template for Entity Framework, and a custom MySQL SQL Editor. Other features include refactored transaction scope: Connector/Net now supports nested transactions in a scope where they use the same connection string.

  This version of Connector/Net is no longer supported.

- `Connector/Net 6.2` includes new features such as a new logging system and client SSL certificates.

  This version of Connector/Net is no longer supported.

- `Connector/Net 6.1` includes new features such as the MySQL Website Configuration Tool, and a Session State Provider.

This version of Connector/Net is no longer supported.

- `Connector/Net 6.0` includes support for UDF schema collection, Initial Entity Framework, and use of the traditional SQL Server buttons in Visual Studio for keys, indexes, and so on.

  This version of Connector/Net is no longer supported.

- `Connector/Net 5.2` includes support for a new membership/role provider, Compact Framework 2.0, a new stored procedure parser and improvements to `GetSchema`. Connector/Net 5.2 also includes the Visual Studio Plugin as a standard installable component.

  This version of Connector/Net is no longer supported.

- `Connector/Net 5.1` includes support for a new membership/role provider, Compact Framework 2.0, a new stored procedure parser and improvements to `GetSchema`. Connector/Net 5.1 also includes the Visual Studio Plugin as a standard installable component.

  This version of Connector/Net is no longer supported.

- `Connector/Net 5.0` includes full support for the ADO.Net 2.0 interfaces and subclasses, includes support for the usage advisor and performance monitor (PerfMon) hooks.

  This version of Connector/Net is no longer supported.

- `Connector/Net 1.0` includes full compatibility with the ADO.NET driver interface.

  This version of Connector/Net is no longer supported.

The following table shows the .NET Framework version required, and the MySQL Server version supported by Connector/Net:

Table 2.1 Connector/Net Requirements for Related Products

Connector/Net version	ADO.NET version supported	.NET Framework version required	MySQL Server version supported	Currently supported
7.0	2.x+	4.5.x for VS 2013 / 2015 / 2017, Core 1.1 for VS 2015 / 2017	5.7, 5.6, 5.5, 5.1	Yes
6.10	2.x+	4.5.2 for VS 2013 / 2015 / 2017, Core 1.1 for VS 2015 / 2017	5.7, 5.6, 5.5, 5.1	Yes
6.9	2.x+	3.5+ for VS 2008, 4.x+ for VS 2010 / 2012 / 2013, .NET RT for VS 2012 / 2013	5.7, 5.6, 5.5, 5.1, 5.0	Yes
6.8	2.x+	3.5+ for VS 2008, 4.x+ for VS 2010 / 2012 / 2013, .NET RT for VS 2012/2013	5.7, 5.6, 5.5, 5.1, 5.0	Yes

Connector/Net version	ADO.NET version supported	.NET Framework version required	MySQL Server version supported	Currently supported
6.7	2.x+	2.x+ for VS 2008, 4.x+ for VS 2010 / 2012 / 2013, .NET RT for VS 2012/2013	5.7, 5.6, 5.5, 5.1, 5.0	No
6.6	2.x+	2.x+ for VS 2008, 4.x+ for VS 2010 / 2012 / 2013	5.7, 5.6, 5.5, 5.1, 5.0	No
6.5	2.x+	2.x+ for VS 2008, 4.x+ for VS 2010	5.7, 5.6, 5.5, 5.1, 5.0	No
6.4	2.x+	2.x+, 4.x+ for VS 2010	5.6, 5.5, 5.1, 5.0	No
6.3	2.x+	2.x+, 4.x+ for VS 2010	5.6, 5.5, 5.1, 5.0	No
6.2	2.x+	2.x+	5.6, 5.5, 5.1, 5.0, 4.1	No
6.1	2.x+	2.x+	5.6, 5.5, 5.1, 5.0, 4.1	No
6.0	2.x+	2.x+	5.5, 5.1, 5.0, 4.1	No
5.2	2.x+	2.x+	5.5, 5.1, 5.0, 4.1	No
5.1	2.x+	2.x+	5.5, 5.1, 5.0, 4.1, 4.0	No
5.0	2.x+	2.x+	5.0, 4.1, 4.0	No
1.0	1.x	1.x	5.0, 4.1, 4.0	No

Note

Version numbers for MySQL products are formatted as X.Y.Z, where Z=0 indicates Alpha, Z=1 indicates Beta, Z=2 indicates Beta or RC, Z>3 indicates RC or GA, and Z=>4 indicates GA. However, Windows tools (Control Panel, properties display) may show the version numbers as XX.YY.ZZ. For example, the official MySQL formatted version number 5.0.9 may be displayed by Windows tools as 5.00.09. The two versions are the same; only the number display format is different.

Chapter 3 Connector/Net Installation

Table of Contents

MySQL Connector/Net runs on any platform that supports the .NET Framework. The .NET Framework is primarily supported on recent versions of Microsoft Windows, and is supported on Linux through the Open Source Mono platform.

Connector/Net is available for download from the MySQL Installer, as a standalone .msi, or from the MySQL NuGet repository.

3.1 Installing Connector/Net on Windows

On Microsoft Windows, you can install either through a binary installation process using a Connector/Net MSI, choose the MySQL Connector/Net product from the MySQL Installer, using Nuget, or by downloading and using the Source.

Before installing, ensure that your system is up to date, including installing the latest version of the .NET Framework.

3.1.1 Installing Connector/Net Using MySQL Installer

MySQL Installer provides an easy to use, wizard-based installation experience for all MySQL software on Windows. It can be used to install and upgrade your MySQL Connector/Net installation.

To use, download and install MySQL Installer.

After executing MySQL Installer, choose and install the MySQL Connector/Net product.

3.1.2 Installing Connector/Net Using the Standalone Installer

You install MySQL Connector/Net through a Windows Installer (`.msi`) installation package, which can install Connector/Net on supported Windows operating systems. The MSI package is a file named `mysql-connector-net-version.msi`, where `version` indicates the Connector/Net version.

> **Note**
>
> Using the central MySQL Installer is recommended, instead of the standalone package that is documented in this section. The MySQL Installer is available for download at MySQL Installer.

To install Connector/Net:

1. Double-click the MSI installer file, and click **Next** to start the installation.

Figure 3.1 Connector/Net Installation: Welcome
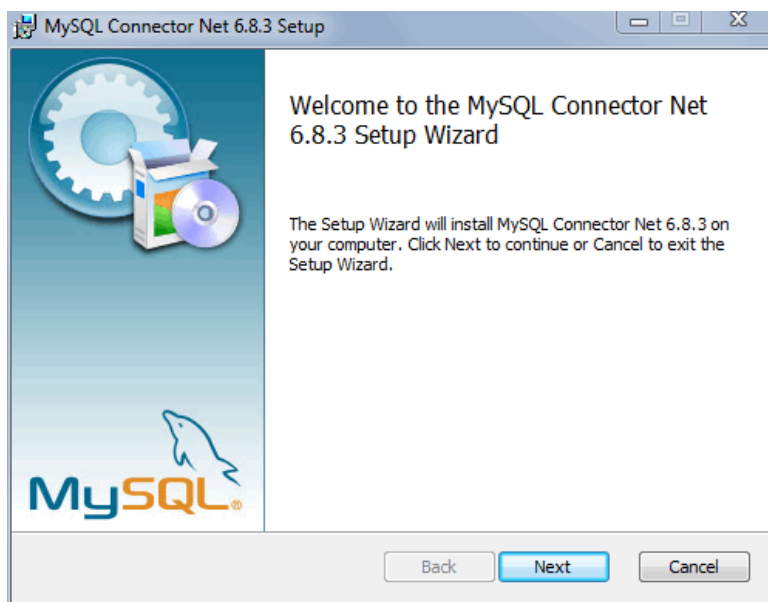
2. You must choose the type of installation to perform.

Figure 3.2 Connector/Net Installation: Choose Setup Type
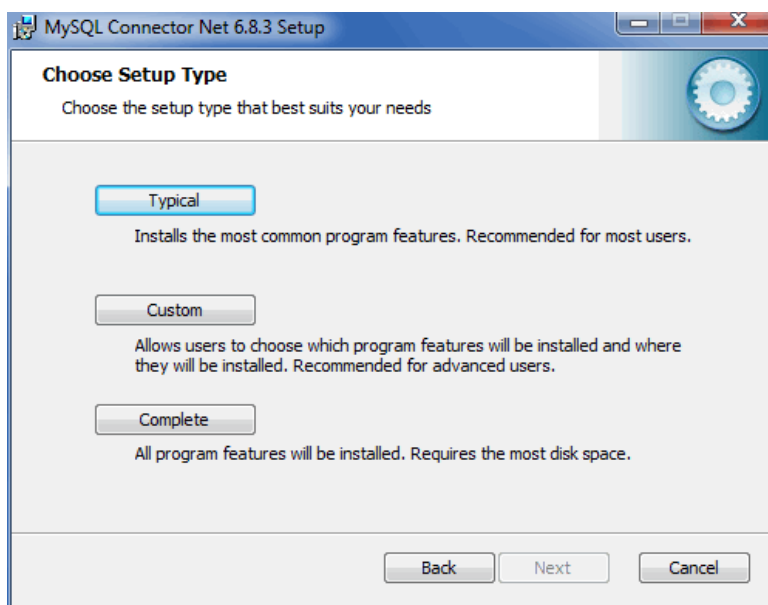
For most situations, the Typical installation is suitable. Click the **Typical** button and proceed to Step 5. A Complete installation installs all the available files. To conduct a Complete installation, click the **Complete** button and proceed to step 5. To customize your installation, including choosing the components to install and some installation options, click the **Custom** button and proceed to Step 3.

The Connector/Net installer will register the connector within the Global Assembly Cache (GAC) - this will make the Connector/Net component available to all applications, not just those where you explicitly

reference the Connector/Net component. The installer will also create the necessary links in the Start menu to the documentation and release notes.

3. If you have chosen a custom installation, you can select the individual components to install, including the core interface component, supporting documentation (a CHM file) samples and examples, and the source code. Select the items, and their installation level, and then click **Next** to continue the installation.

Figure 3.3 Connector/Net Installation: Custom Setup
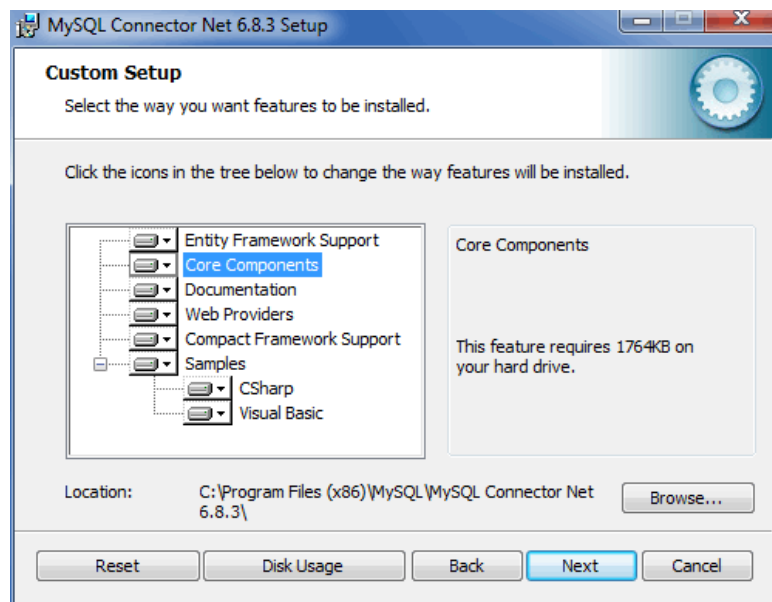
4. You will be given a final opportunity to confirm the installation. Click **Install** to copy and install the files onto your machine.

Figure 3.4 Connector/Net Installation: Ready To Begin
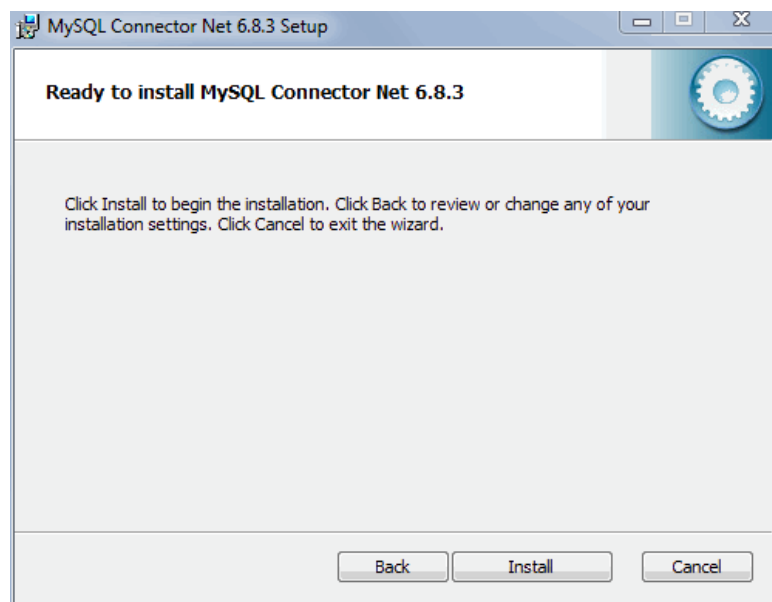
5. Once the installation has been completed, click **Finish** to exit the installer.

Figure 3.5 Connector/Net Installation: Finish Installation
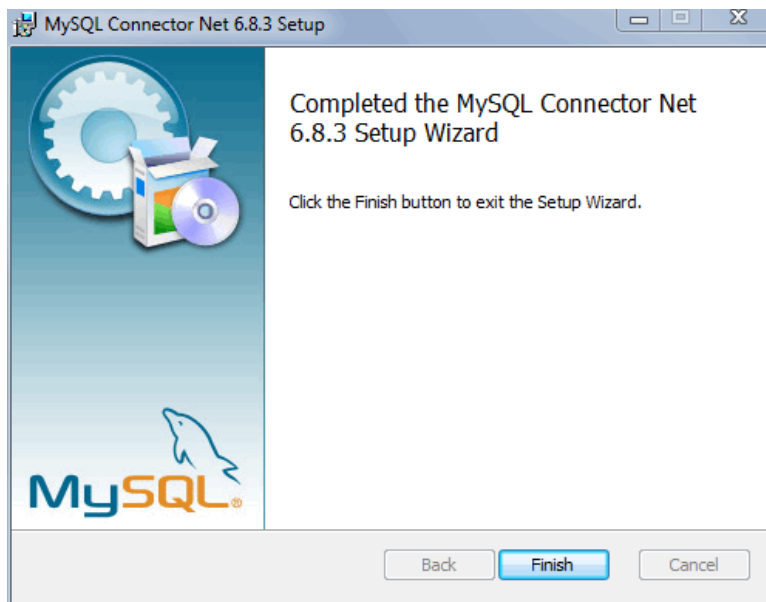
Unless you choose otherwise, Connector/Net is installed in `C:\Program Files (x86)\MySQL\MySQL Connector Net X.X.X`, where `X.X.X` is replaced with the version of Connector/Net you are installing. New installations do not overwrite existing versions of Connector/Net.

Depending on your installation type, the installed components will include some or all of the following components:

- `Assemblies`: Connector/Net MySQL libraries for different versions of the .NET environment.

- `Documentation`: Connector/Net documentation in CHM format.

- `Samples`: Sample code and applications that use the Connector/Net component.

You may also use the `/quiet` or `/q` command-line option with the `msiexec` tool to install the Connector/Net package automatically (using the default options) with no notification to the user. Using this method the user cannot select options. Additionally, no prompts, messages or dialog boxes will be displayed.

```
C:\> msiexec /package connector-net.msi /quiet
```

To provide a progress bar to the user during automatic installation, use the `/passive` option.

3.2 Installing Connector/Net on Unix with Mono

There is no installer available for installing the Connector/Net component on your Unix installation. Before installing, ensure that you have a working Mono project installation. To test whether your system has Mono installed, enter:

```
shell> mono --version
```

The version of the Mono JIT compiler is displayed.

To compile C# source code, make sure a Mono C# compiler is installed.

> **Note**
>
> There are three Mono C# compilers available: `mcs`, which accesses the 1.0-profile libraries, `gmcs`, which accesses the 2.0-profile libraries, and `dmcs`, which accesses the 4.0-profile libraries.

To install Connector/Net on Unix/Mono:

1. Download the `mysql-connector-net-version-noinstall.zip` and extract the contents to a directory of your choice, for example: `~/connector-net/`.

2. In the directory where you unzipped the connector to, change into the `bin` subdirectory. Ensure the file `MySql.Data.dll` is present. This filename is case-sensitive.

3. You must register the Connector/Net component, `MySql.Data`, in the Global Assembly Cache (GAC). In the current directory enter the `gacutil` command:

```
root-shell> gacutil /i MySql.Data.dll
```

   This will register `MySql.Data` into the GAC. You can check this by listing the contents of `/usr/lib/mono/gac`, where you will find `MySql.Data` if the registration has been successful.

You are now ready to compile your application. You must ensure that when you compile your application you include the Connector/Net component using the `-r:` command-line option. For example:

```
shell> gmcs -r:System.dll -r:System.Data.dll -r:MySql.Data.dll HelloWorld.cs
```

The referenced assemblies depend on the requirements of the application, but applications using MySQL Connector/Net must provide `-r:MySql.Data` at a minimum.

You can further check your installation by running the compiled program, for example:

```
shell> mono HelloWorld.exe
```

3.3 Installing Connector/Net from the Source Code

Source packages of Connector/Net are available for download from http://dev.mysql.com/downloads/connector/net/.

The file contains the following directories:

- `Documentation`: Source files to build the documentation into the compiled HTML (CHM) format, code examples, collection files, and licenses for third-party components.

- `Samples`: Source files for several example applications.

- `Source\MySQL.ConnectorInstaller`: Source files to build the Connector/Net installer program.

- `Source\MySQL.Data`: Source files for the core data provider.

- `Source\MySQL.Data.Entity`: Source files for the Entity Framework.

- `Source\MySQL.Web`: Source files for the web providers, including the membership/role/profile providers that are used in ASP.NET web sites.

- `Tests`: Test cases for Connector/Net.

Building the Source Code on Microsoft Windows

The following procedure can be used to build the connector on Microsoft Windows.

- Navigate to the root of the source code tree.

- A Microsoft Visual Studio solution file named `MySqlClient.sln` is available to build the connector. Click this file to load the solution into Visual Studio.

  `MySqlClient.sln` must be compiled with VS 2008, VS 2010, or VS 2012. Also, depending on the version, the dependencies to build it include Visual Studio SDK, NUnit, Entity Framework, and ANTLR Integration for Visual Studio.

- Select **Build**, **Build Solution** from the main menu to build the solution.

Building the Source Code on Unix

Support for building Connector/Net on Mono/Unix is not available.

Chapter 4 Connector/Net Tutorials

Table of Contents

The following tutorials illustrate how to develop MySQL programs using technologies such as Visual Studio, C#, ASP.NET, and the .NET and Mono frameworks. Work through the first tutorial to verify that you have the right software components installed and configured, then choose other tutorials to try depending on the features you intend to use in your applications.

4.1 Tutorial: An Introduction to Connector/Net Programming

This section provides a gentle introduction to programming with Connector/Net. The example code is written in C#, and is designed to work on both Microsoft .NET Framework and Mono.

This tutorial is designed to get you up and running with Connector/Net as quickly as possible, it does not go into detail on any particular topic. However, the following sections of this manual describe each of the topics introduced in this tutorial in more detail. In this tutorial you are encouraged to type in and run the code, modifying it as required for your setup.

This tutorial assumes you have MySQL and Connector/Net already installed. It also assumes that you have installed the `world` database sample, which can be downloaded from the MySQL Documentation page. You can also find details on how to install the database on the same page.

> **Note**
>
> Before compiling the example code, make sure that you have added References to your project as required. The References required are `System`, `System.Data` and `MySql.Data`.

4.1.1 The MySqlConnection Object

For your Connector/Net application to connect to a MySQL database, it must establish a connection by using a `MySqlConnection` object.

The `MySqlConnection` constructor takes a connection string as one of its parameters. The connection string provides necessary information to make the connection to the MySQL database. The connection string is discussed more fully in Section 5.1, "Connecting to MySQL Using Connector/Net". For a list of supported connection string options, see Chapter 6, *Connector/Net Connection String Options Reference*.

The following code shows how to create a connection object:

```
using System;
using System.Data;

using MySql.Data;
using MySql.Data.MySqlClient;

public class Tutorial1
{
    public static void Main()
    {
        string connStr = "server=localhost;user=root;database=world;port=3306;password=******;";
        MySqlConnection conn = new MySqlConnection(connStr);
        try
        {
            Console.WriteLine("Connecting to MySQL...");
            conn.Open();
            // Perform database operations
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }
        conn.Close();
        Console.WriteLine("Done.");
    }
}
```

When the `MySqlConnection` constructor is invoked, it returns a connection object, which is used for subsequent database operations. Open the connection before any other operations take place. Before the application exits, close the connection to the database by calling `Close` on the connection object.

Sometimes an attempt to perform an `Open` on a connection object can fail, generating an exception that can be handled using standard exception handling code.

In this section you have learned how to create a connection to a MySQL database, and open and close the corresponding connection object.

4.1.2 The MySqlCommand Object

Once a connection has been established with the MySQL database, the next step is do carry out the desired database operations. This can be achieved through the use of the `MySqlCommand` object.

You will see how to create a `MySqlCommand` object. Once it has been created, there are three main methods of interest that you can call:

- **ExecuteReader** - used to query the database. Results are usually returned in a `MySqlDataReader` object, created by `ExecuteReader`.

- **ExecuteNonQuery** - used to insert and delete data.

- **ExecuteScalar** - used to return a single value.

Once a `MySqlCommand` object has been created, you will call one of the above methods on it to carry out a database operation, such as perform a query. The results are usually returned into a `MySqlDataReader`

object, and then processed, for example the results might be displayed. The following code demonstrates how this could be done.

```
using System;
using System.Data;

using MySql.Data;
using MySql.Data.MySqlClient;

public class Tutorial2
{
    public static void Main()
    {
        string connStr = "server=localhost;user=root;database=world;port=3306;password=******;";
        MySqlConnection conn = new MySqlConnection(connStr);
        try
        {
            Console.WriteLine("Connecting to MySQL...");
            conn.Open();

            string sql = "SELECT Name, HeadOfState FROM Country WHERE Continent='Oceania'";
            MySqlCommand cmd = new MySqlCommand(sql, conn);
            MySqlDataReader rdr = cmd.ExecuteReader();

            while (rdr.Read())
            {
                Console.WriteLine(rdr[0]+" -- "+rdr[1]);
            }
            rdr.Close();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }

        conn.Close();
        Console.WriteLine("Done.");
    }
}
```

When a connection has been created and opened, the code then creates a `MySqlCommand` object. Then the SQL query to be executed is passed to the `MySqlCommand` constructor. The `ExecuteReader` method is then used to generate a `MySqlReader` object. The `MySqlReader` object contains the results generated by the SQL executed on the command object. Once the results have been obtained in a `MySqlReader` object, the results can be processed. In this case, the information is printed out by a `while` loop. Finally, the `MySqlReader` object is disposed of by running its `Close` method on it.

In the next example, you will see how to use the `ExecuteNonQuery` method.

The procedure for performing an `ExecuteNonQuery` method call is simpler, as there is no need to create an object to store results. This is because `ExecuteNonQuery` is only used for inserting, updating and deleting data. The following example illustrates a simple update to the `Country` table:

```
using System;
using System.Data;

using MySql.Data;
using MySql.Data.MySqlClient;

public class Tutorial3
{
    public static void Main()
```

```
    {
        string connStr = "server=localhost;user=root;database=world;port=3306;password=******;";
        MySqlConnection conn = new MySqlConnection(connStr);
        try
        {
            Console.WriteLine("Connecting to MySQL...");
            conn.Open();

            string sql = "INSERT INTO Country (Name, HeadOfState, Continent) VALUES ('Disneyland','Mickey Mous
            MySqlCommand cmd = new MySqlCommand(sql, conn);
            cmd.ExecuteNonQuery();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }

        conn.Close();
        Console.WriteLine("Done.");
    }
}
```

The query is constructed, the command object created and the `ExecuteNonQuery` method called on the command object. You can access your MySQL database with the `mysql` command interpreter and verify that the update was carried out correctly.

Finally, you will see how the `ExecuteScalar` method can be used to return a single value. Again, this is straightforward, as a `MySqlDataReader` object is not required to store results, a simple variable will do. The following code illustrates how to use `ExecuteScalar`:

```
using System;
using System.Data;

using MySql.Data;
using MySql.Data.MySqlClient;

public class Tutorial4
{
    public static void Main()
    {
        string connStr = "server=localhost;user=root;database=world;port=3306;password=******;";
        MySqlConnection conn = new MySqlConnection(connStr);
        try
        {
            Console.WriteLine("Connecting to MySQL...");
            conn.Open();

            string sql = "SELECT COUNT(*) FROM Country";
            MySqlCommand cmd = new MySqlCommand(sql, conn);
            object result = cmd.ExecuteScalar();
            if (result != null)
            {
                int r = Convert.ToInt32(result);
                Console.WriteLine("Number of countries in the world database is: " + r);
            }

        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }

        conn.Close();
        Console.WriteLine("Done.");
    }
```

```
}
```

This example uses a simple query to count the rows in the `Country` table. The result is obtained by calling `ExecuteScalar` on the command object.

4.1.3 Working with Decoupled Data

Previously, when using `MySqlDataReader`, the connection to the database was continually maintained, unless explicitly closed. It is also possible to work in a manner where a connection is only established when needed. For example, in this mode, a connection could be established to read a chunk of data, the data could then be modified by the application as required. A connection could then be reestablished only if and when the application writes data back to the database. This decouples the working data set from the database.

This decoupled mode of working with data is supported by Connector/Net. There are several parts involved in allowing this method to work:

- **Data Set** - The Data Set is the area in which data is loaded to read or modify it. A `DataSet` object is instantiated, which can store multiple tables of data.

- **Data Adapter** - The Data Adapter is the interface between the Data Set and the database itself. The Data Adapter is responsible for efficiently managing connections to the database, opening and closing them as required. The Data Adapter is created by instantiating an object of the `MySqlDataAdapter` class. The `MySqlDataAdapter` object has two main methods: `Fill` which reads data into the Data Set, and `Update`, which writes data from the Data Set to the database.

- **Command Builder** - The Command Builder is a support object. The Command Builder works in conjunction with the Data Adapter. When a `MySqlDataAdapter` object is created, it is typically given an initial SELECT statement. From this `SELECT` statement the Command Builder can work out the corresponding `INSERT`, `UPDATE` and `DELETE` statements that would be required to update the database. To create the Command Builder, an object of the class `MySqlCommandBuilder` is created.

Each of these classes will now be discussed in more detail.

Instantiating a DataSet object

A `DataSet` object can be created simply, as shown in the following example code snippet:

```
DataSet dsCountry;
...
dsCountry = new DataSet();
```

Although this creates the `DataSet` object, it has not yet filled it with data. For that, a Data Adapter is required.

Instantiating a MySqlDataAdapter object

The `MySqlDataAdapter` can be created as illustrated by the following example:

```
MySqlDataAdapter daCountry;
...
string sql = "SELECT Code, Name, HeadOfState FROM Country WHERE Continent='North America'";
daCountry = new MySqlDataAdapter (sql, conn);
```

> **Note**
>
> The `MySqlDataAdapter` is given the SQL specifying the data to work with.

Instantiating a MySqlCommandBuilder object

Once the `MySqlDataAdapter` has been created, it is necessary to generate the additional statements required for inserting, updating and deleting data. There are several ways to do this, but in this tutorial you will see how this can most easily be done with `MySqlCommandBuilder`. The following code snippet illustrates how this is done:

```
MySqlCommandBuilder cb = new MySqlCommandBuilder(daCountry);
```

> **Note**
>
> The `MySqlDataAdapter` object is passed as a parameter to the command builder.

Filling the Data Set

To do anything useful with the data from your database, you need to load it into a Data Set. This is one of the jobs of the `MySqlDataAdapter` object, and is carried out with its `Fill` method. The following example code illustrates this:

```
DataSet dsCountry;
...
dsCountry = new DataSet();
...
daCountry.Fill(dsCountry, "Country");
```

The `Fill` method is a `MySqlDataAdapter` method, and the Data Adapter knows how to establish a connection with the database and retrieve the required data, and then populate the Data Set when the `Fill` method is called. The second parameter "Country" is the table in the Data Set to update.

Updating the Data Set

The data in the Data Set can now be manipulated by the application as required. At some point, changes to data will need to be written back to the database. This is achieved through a `MySqlDataAdapter` method, the `Update` method.

```
daCountry.Update(dsCountry, "Country");
```

Again, the Data Set and the table within the Data Set to update are specified.

Working Example

The interactions between the `DataSet`, `MySqlDataAdapter` and `MySqlCommandBuilder` classes can be a little confusing, so their operation can perhaps be best illustrated by working code.

In this example, data from the `world` database is read into a Data Grid View control. Here, the data can be viewed and changed before clicking an update button. The update button then activates code to write changes back to the database. The code uses the principles explained above. The application was built using the Microsoft Visual Studio to place and create the user interface controls, but the main code that uses the key classes described above is shown below, and is portable.

```
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;

using MySql.Data;
using MySql.Data.MySqlClient;

namespace WindowsFormsApplication5
{
    public partial class Form1 : Form
    {
        MySqlDataAdapter daCountry;
        DataSet dsCountry;

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

            string connStr = "server=localhost;user=root;database=world;port=3306;password=******;";
            MySqlConnection conn = new MySqlConnection(connStr);
            try
            {
                label2.Text = "Connecting to MySQL...";

                string sql = "SELECT Code, Name, HeadOfState FROM Country WHERE Continent='North America'"
                daCountry = new MySqlDataAdapter (sql, conn);
                MySqlCommandBuilder cb = new MySqlCommandBuilder(daCountry);

                dsCountry = new DataSet();
                daCountry.Fill(dsCountry, "Country");
                dataGridView1.DataSource = dsCountry;
                dataGridView1.DataMember = "Country";
            }
            catch (Exception ex)
            {
                label2.Text = ex.ToString();
            }
        }

        private void button1_Click(object sender, EventArgs e)
        {
            daCountry.Update(dsCountry, "Country");
            label2.Text = "MySQL Database Updated!";
        }

    }
}
```

The application running is shown below:

Figure 4.1 World Database Application
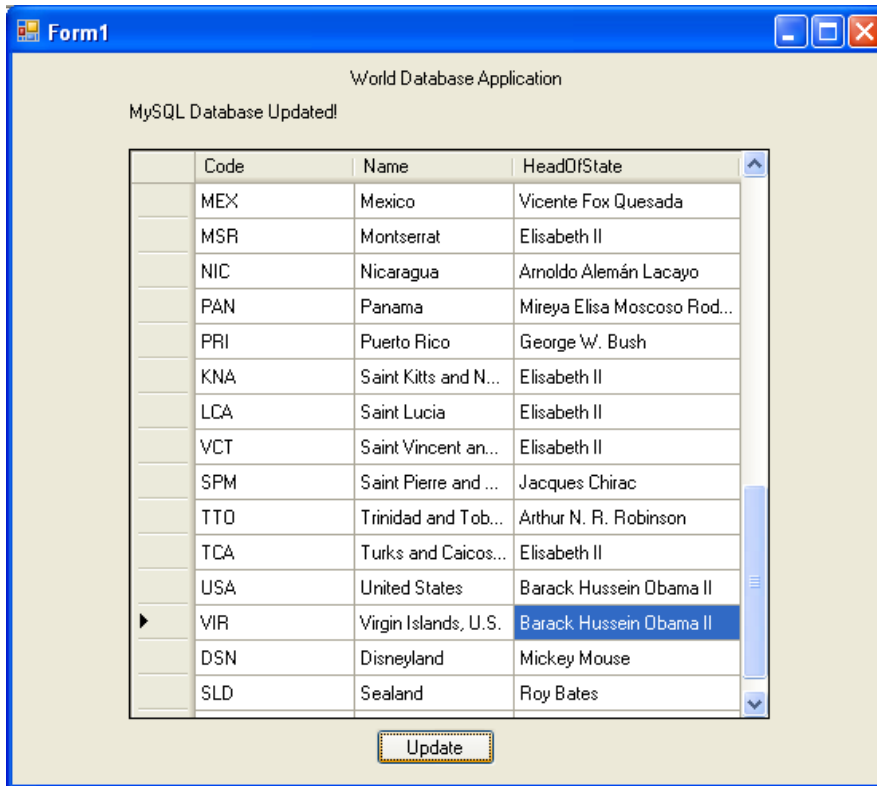
4.1.4 Working with Parameters

This part of the tutorial shows you how to use parameters in your Connector/Net application.

Although it is possible to build SQL query strings directly from user input, this is not advisable as it does not prevent erroneous or malicious information being entered. It is safer to use parameters as they will be processed as field data only. For example, imagine the following query was constructed from user input:

```
string sql = "SELECT Name, HeadOfState FROM Country WHERE Continent = "+user_continent;
```

If the string `user_continent` came from a Text Box control, there would potentially be no control over the string entered by the user. The user could enter a string that generates a runtime error, or in the worst case actually harms the system. When using parameters it is not possible to do this because a parameter is only ever treated as a field parameter, rather than an arbitrary piece of SQL code.

The same query written using a parameter for user input is:

```
string sql = "SELECT Name, HeadOfState FROM Country WHERE Continent = @Continent";
```

> **Note**
>
> The parameter is preceded by an '@' symbol to indicate it is to be treated as a parameter.

As well as marking the position of the parameter in the query string, it is necessary to add a parameter to the Command object. This is illustrated by the following code snippet:

```
cmd.Parameters.AddWithValue("@Continent", "North America");
```

In this example the string "North America" is supplied as the parameter value statically, but in a more practical example it would come from a user input control.

A further example illustrates the complete process:

```
using System;
using System.Data;

using MySql.Data;
using MySql.Data.MySqlClient;

public class Tutorial5
{
    public static void Main()
    {
        string connStr = "server=localhost;user=root;database=world;port=3306;password=******;";
        MySqlConnection conn = new MySqlConnection(connStr);
        try
        {
            Console.WriteLine("Connecting to MySQL...");
            conn.Open();

            string sql = "SELECT Name, HeadOfState FROM Country WHERE Continent=@Continent";
            MySqlCommand cmd = new MySqlCommand(sql, conn);

            Console.WriteLine("Enter a continent e.g. 'North America', 'Europe': ");
            string user_input = Console.ReadLine();

            cmd.Parameters.AddWithValue("@Continent", user_input);

            MySqlDataReader rdr = cmd.ExecuteReader();

            while (rdr.Read())
            {
                Console.WriteLine(rdr["Name"]+" --- "+rdr["HeadOfState"]);
            }
            rdr.Close();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }

        conn.Close();
        Console.WriteLine("Done.");
    }
}
```

In this part of the tutorial you have see how to use parameters to make your code more secure.

4.1.5 Working with Stored Procedures

This section illustrates how to work with stored procedures. Putting database-intensive operations into stored procedures lets you define an API for your database application. You can reuse this API across multiple applications and multiple programming languages. This technique avoids duplicating database code, saving time and effort when you make updates due to schema changes, tune the performance of queries, or add new database operations for logging, security, and so on. Before working through this tutorial, familiarize yourself with the CREATE PROCEDURE and CREATE FUNCTION statements that create different kinds of stored routines.

For the purposes of this tutorial, you will create a simple stored procedure to see how it can be called from Connector/Net. In the MySQL Client program, connect to the `world` database and enter the following stored procedure:

```
DELIMITER //
CREATE PROCEDURE country_hos
(IN con CHAR(20))
BEGIN
  SELECT Name, HeadOfState FROM Country
  WHERE Continent = con;
END //
DELIMITER ;
```

Test that the stored procedure works as expected by typing the following into the `mysql` command interpreter:

```
CALL country_hos('Europe');
```

> **Note**
>
> The stored routine takes a single parameter, which is the continent to restrict your search to.

Having confirmed that the stored procedure is present and correct, you can see how to access it from Connector/Net.

Calling a stored procedure from your Connector/Net application is similar to techniques you have seen earlier in this tutorial. A `MySqlCommand` object is created, but rather than taking an SQL query as a parameter, it takes the name of the stored procedure to call. Set the `MySqlCommand` object to the type of stored procedure, as shown by the following code snippet:

```
string rtn = "country_hos";
MySqlCommand cmd = new MySqlCommand(rtn, conn);
cmd.CommandType = CommandType.StoredProcedure;
```

In this case, the stored procedure requires you to pass a parameter. This can be achieved using the techniques seen in the previous section on parameters, Section 4.1.4, "Working with Parameters", as shown in the following code snippet:

```
cmd.Parameters.AddWithValue("@con", "Europe");
```

The value of the parameter `@con` could more realistically have come from a user input control, but for simplicity it is set as a static string in this example.

At this point, everything is set up and you can call the routine using techniques also learned in earlier sections. In this case, the `ExecuteReader` method of the `MySqlCommand` object is used.

Complete working code for the stored procedure example is shown below:

```
using System;
using System.Data;

using MySql.Data;
using MySql.Data.MySqlClient;

public class Tutorial6
{
    public static void Main()
```

```
    {
        string connStr = "server=localhost;user=root;database=world;port=3306;password=******;";
        MySqlConnection conn = new MySqlConnection(connStr);
        try
        {
            Console.WriteLine("Connecting to MySQL...");
            conn.Open();

            string rtn = "country_hos";
            MySqlCommand cmd = new MySqlCommand(rtn, conn);
            cmd.CommandType = CommandType.StoredProcedure;

            cmd.Parameters.AddWithValue("@con", "Europe");

            MySqlDataReader rdr = cmd.ExecuteReader();
            while (rdr.Read())
            {
                Console.WriteLine(rdr[0] + " --- " + rdr[1]);
            }
            rdr.Close();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }

        conn.Close();
        Console.WriteLine("Done.");
    }
}
```

In this section, you have seen how to call a stored procedure from Connector/Net. For the moment, this concludes our introductory tutorial on programming with Connector/Net.

4.2 Tutorial: MySQL Connector/Net ASP.NET Membership and Role Provider

Many web sites feature the facility for the user to create a user account. They can then log into the web site and enjoy a personalized experience. This requires that the developer creates database tables to store user information, along with code to gather and process this data. This represents a burden on the developer, and there is the possibility for security issues to creep into the developed code. However, ASP.NET 2.0 introduced the Membership system. This system is designed around the concept of Membership, Profile and Role Providers, which together provide all of the functionality to implement a user system, that previously would have to have been created by the developer from scratch.

Currently, MySQL Connector/Net provides Membership, Role, Profile and Session State Providers.

This tutorial shows you how to set up your ASP.NET web application to use the MySQL Connector/Net Membership and Role Providers. It assumes that you have MySQL Server installed, along with MySQL Connector/Net and Microsoft Visual Studio. This tutorial was tested with MySQL Connector/Net 6.0.4 and Microsoft Visual Studio 2008 Professional Edition. It is recommended you use 6.0.4 or above for this tutorial.

1. Create a new database in the MySQL Server using the MySQL Command-Line Client program (mysql), or other suitable tool. It does not matter what name is used for the database, but record it. You specify it in the connection string constructed later in this tutorial. This database contains the tables, automatically created for you later, used to store data about users and roles.

2. Create a new ASP.NET Web Site in Visual Studio. If you are not sure how to do this, refer to Section 4.7, "Tutorial: Databinding in ASP.NET Using LINQ on Entities", which demonstrates how to create a simple ASP.NET web site.

3. Add References to `MySql.Data` and `MySql.Web` to the web site project.

4. Locate the `machine.config` file on your system, which is the configuration file for the .NET Framework.

5. Search the `machine.config` file to find the membership provider `MySQLMembershipProvider`.

6. Add the attribute `autogenerateschema="true"`. The appropriate section should now resemble the following:

> **Note**
>
> For the sake of brevity, some information is excluded.

```
<membership>
 <providers>
   <add name="AspNetSqlMembershipProvider"
     type="System.Web.Security.SqlMembershipProvider"
     ...
     connectionStringName="LocalSqlServer"
     ... />
   <add name="MySQLMembershipProvider"
     autogenerateschema="true"
     type="MySql.Web.Security.MySQLMembershipProvider, MySql.Web, Version=6.0.4.0, Culture=neutral, PublicK
     connectionStringName="LocalMySqlServer"
     ... />
 </providers>
</membership>
```

> **Note**
>
> The connection string, `LocalMySqlServer`, connects to the MySQL server that contains the membership database.

The `autogenerateschema="true"` attribute will cause MySQL Connector/Net to silently create, or upgrade, the schema on the database server, to contain the required tables for storing membership information.

7. It is now necessary to create the connection string referenced in the previous step. Load the web site's `web.config` file into Visual Studio.

8. Locate the section marked `<connectionStrings>`. Add the following connection string information:

```
<connectionStrings>
  <remove name="LocalMySqlServer"/>
  <add name="LocalMySqlServer"
       connectionString="Datasource=localhost;Database=users;uid=root;pwd=password;"
       providerName="MySql.Data.MySqlClient"/>
</connectionStrings>
```

The database specified is the one created in the first step. You could alternatively have used an existing database.

9. At this point build the solution to ensure no errors are present. This can be done by selecting **Build**, **Build Solution** from the main menu, or pressing **F6**.

10. ASP.NET supports the concept of locally and remotely authenticated users. With local authentication the user is validated using their Windows credentials when they attempt to access the web site. This can be useful in an Intranet environment. With remote authentication, a user is prompted for their

login details when accessing the web site, and these credentials are checked against the membership information stored in a database server such as MySQL Server. You will now see how to choose this form of authentication.

Start the ASP.NET Web Site Administration Tool. This can be done quickly by clicking the small hammer/Earth icon in the Solution Explorer. You can also launch this tool by selecting **Website**, **ASP.NET Configuration** from the main menu.

11. In the ASP.NET Web Site Administration Tool click the **Security** tab.

12. Now click the **User Authentication Type** link.

13. Select the **From the internet** radio button. The web site will now need to provide a form to allow the user to enter their login details. These will be checked against membership information stored in the MySQL database.

Figure 4.2 Authentication Type
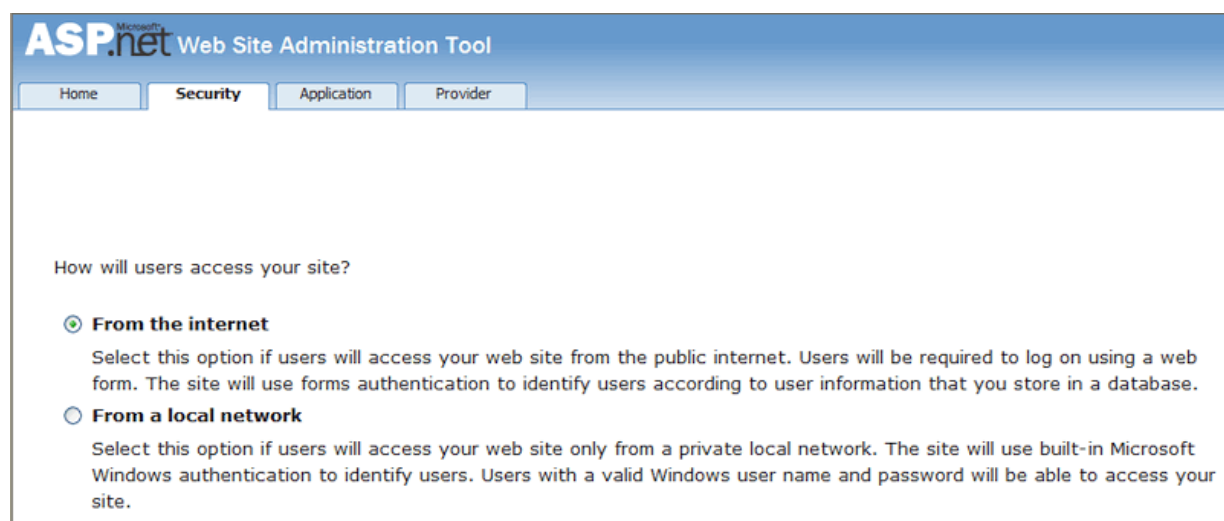
14. You now need to specify the Role and Membership Provider to be used. Click the **Provider** tab.

15. Click the **Select a different provider for each feature (advanced)** link.

16. Now select the **MySQLMembershipProvider** and the **MySQLRoleProvider** radio buttons.

Figure 4.3 Select Membership and Role Provider
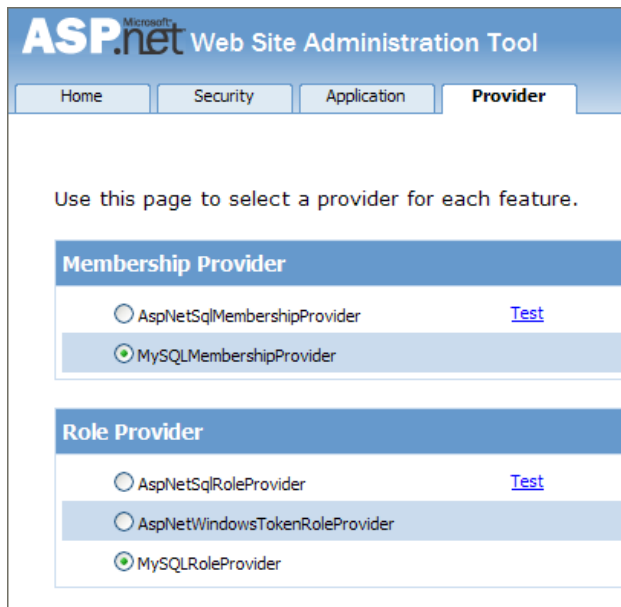
17. In Visual Studio, rebuild the solution by selecting **Build**, **Rebuild Solution** from the main menu.

18. Check that the necessary schema has been created. This can be achieved using the `mysql` command interpreter.

Figure 4.4 Membership and Role Provider Tables
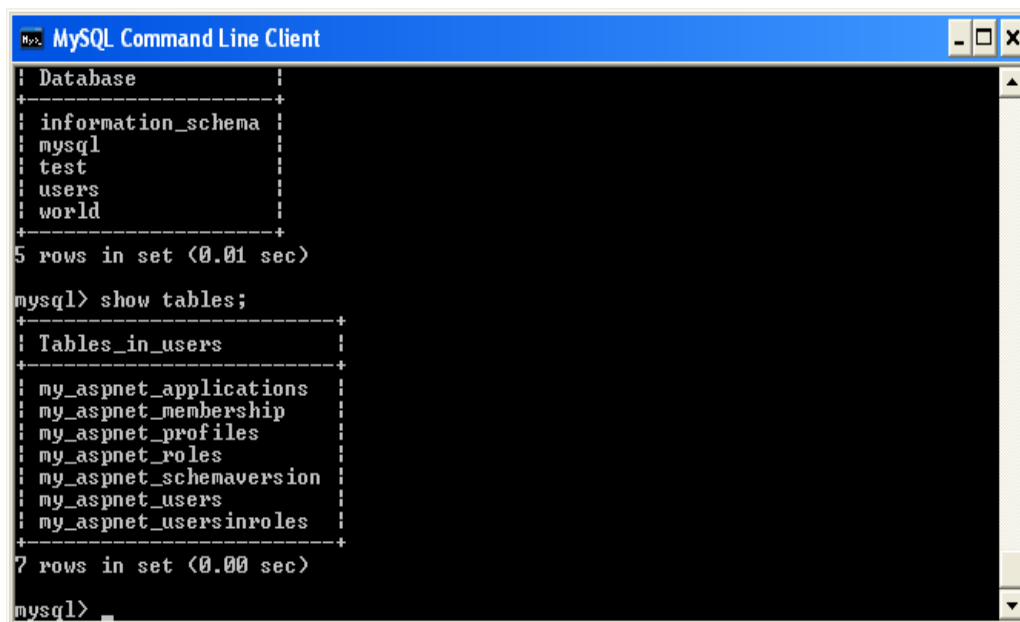
19. Assuming all is present and correct, you can now create users and roles for your web application. The easiest way to do this is with the ASP.NET Web Site Administration Tool. However, many web applications contain their own modules for creating roles and users. For simplicity, the ASP.NET Web Site Administration Tool will be used in this tutorial.

20. In the ASP.NET Web Site Administration Tool, click the **Security** tab. Now that both the Membership and Role Provider are enabled, you will see links for creating roles and users. Click the **Create or Manage Roles** link.

Figure 4.5 Security Tab
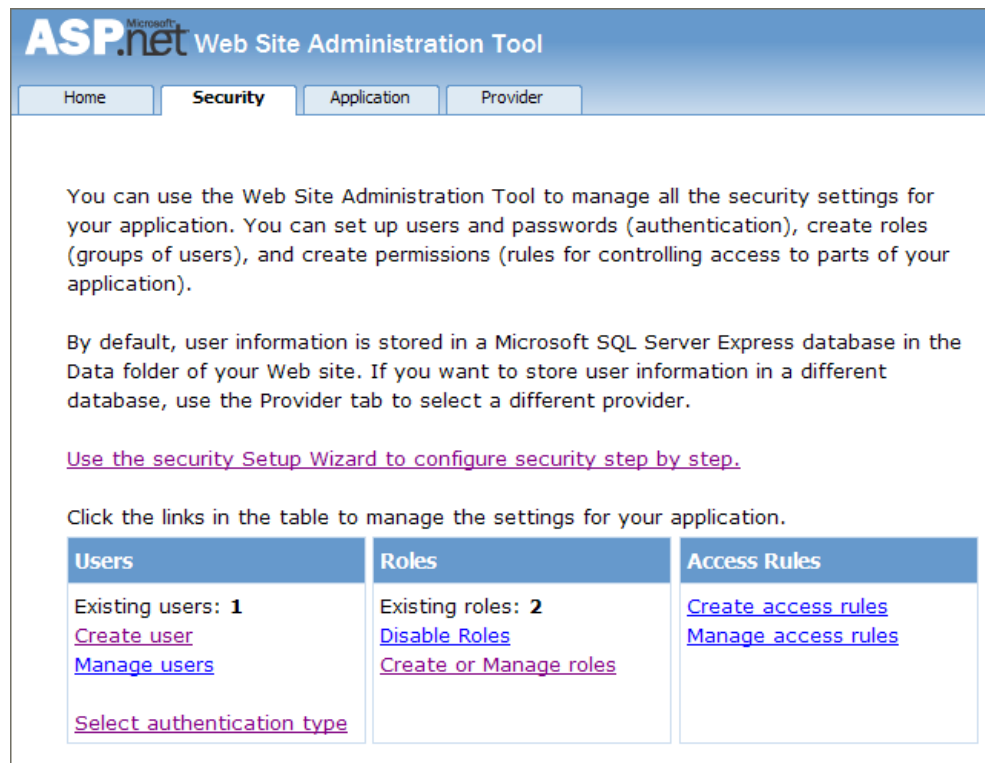
21. You can now enter the name of a new Role and click **Add Role** to create the new Role. Create new Roles as required.

22. Click the **Back** button.

23. Click the **Create User** link. You can now fill in information about the user to be created, and also allocate that user to one or more Roles.

Figure 4.6 Create User

24. Using the `mysql` command interpreter, you can check that your database has been correctly populated with the Membership and Role data.

Figure 4.7 Membership and Roles Table Contents

In this tutorial, you have seen how to set up the MySQL Connector/Net Membership and Role Providers for use in your ASP.NET web application.

4.3 Tutorial: MySQL Connector/Net ASP.NET Profile Provider

This tutorial shows you how to use the MySQL Profile Provider to store user profile information in a MySQL database. The tutorial uses MySQL Connector/Net 6.1.1, MySQL Server 5.1 and Microsoft Visual Studio 2008 Professional Edition.

Many modern web sites allow the user to create a personal profile. This requires a significant amount of code, but ASP.NET reduces this considerable by including the functionality in its Profile classes. The Profile Provider provides an abstraction between these classes and a data source. The MySQL Profile Provider enables profile data to be stored in a MySQL database. This enables the profile properties to be written to a persistent store, and be retrieved when required. The Profile Provider also enables profile data to be managed effectively, for example it enables profiles that have not been accessed since a specific date to be deleted.

The following steps show you how you can select the MySQL Profile Provider.

1. Create a new ASP.NET web project.

2. Select the MySQL Website Configuration tool. Due to a bug in 6.1.1 you may have to first connect to a server in Server Explorer before the tool's icon will display in the toolbar of the Solution Explorer.

3. In the MySQL Website Configuration tool navigate through the tool to the Profiles page.

4. Select the **Use MySQL to manage my profiles** check box.

5. Select the **Autogenerate Schema** check box.

6. Click the **Edit...** button and configure a connection string for the database that will be used to store user profile information.

7. Navigate to the last page of the tool and click **Finish** to save your changes and exit the tool.

At this point you are now ready to start using the MySQL Profile Provider. With the following steps you can carry out a preliminary test of your installation.

1. Open your `web.config` file.

2. Add a simple profile such as the following:

```
<system.web>
  <anonymousIdentification enabled="true"/>
  <profile defaultProvider="MySQLProfileProvider">
    ...
    <properties>
      <add name="Name" allowAnonymous="true"/>
      <add name="Age" allowAnonymous="true" type="System.UInt16"/>
      <group name="UI">
        <add name="Color" allowAnonymous="true" defaultValue="Blue"/>
        <add name="Style" allowAnonymous="true" defaultValue="Plain"/>
      </group>
    </properties>
  </profile>
  ...
```

Setting `anonymousIdentification` to true enables unauthenticated users to use profiles. They are identified by a GUID in a cookie rather than by a user name.

Now that the simple profile has been defined in `web.config`, the next step is to write some code to test the profile.

1. In Design View design a simple page with the following controls:

Figure 4.8 Simple Profile Application
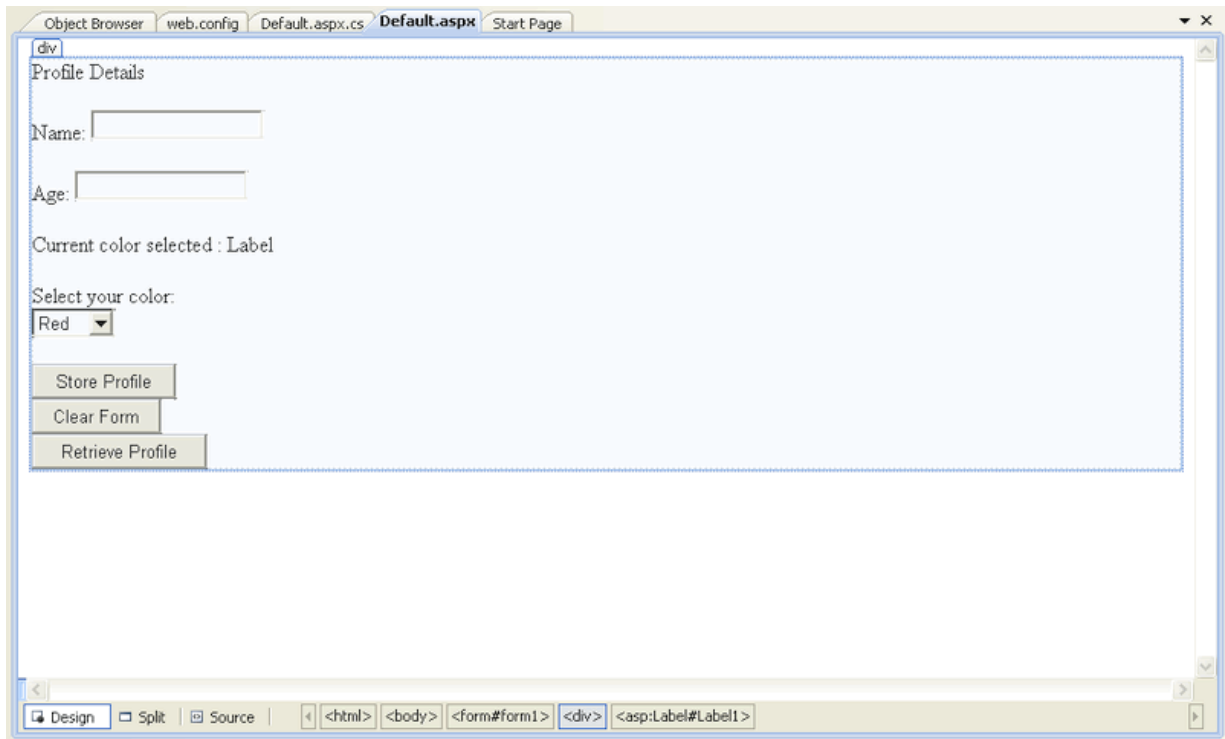
These will allow the user to enter some profile information. The user can also use the buttons to save their profile, clear the page, and restore their profile data.

2. In the Code View add code as follows:

```
...
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        TextBox1.Text = Profile.Name;
        TextBox2.Text = Profile.Age.ToString();
        Label1.Text = Profile.UI.Color;
    }
}

// Store Profile
protected void Button1_Click(object sender, EventArgs e)
{
    Profile.Name = TextBox1.Text;
    Profile.Age = UInt16.Parse(TextBox2.Text);
}

// Clear Form
protected void Button2_Click(object sender, EventArgs e)
{
    TextBox1.Text = "";
    TextBox2.Text = "";
    Label1.Text = "";
}
```

```
// Retrieve Profile
protected void Button3_Click(object sender, EventArgs e)
{
    TextBox1.Text = Profile.Name;
    TextBox2.Text = Profile.Age.ToString();
    Label1.Text = Profile.UI.Color;
}

protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
{
    Profile.UI.Color = DropDownList1.SelectedValue;
}
...
```

3. Save all files and build the solution to check that no errors have been introduced.

4. Run the application.

5. Enter your name, age, and select a color from the listbox. Now store this information in your profile by clicking **Store Profile**.

   Not selecting a color from the listbox uses the default color, *Blue*, that was specified in the `web.config` file.

6. Click **Clear Form** to clear text from the textboxes and the label that displays your chosen color.

7. Now click **Retrieve Profile** to restore your profile data from the MySQL database.

8. Now exit the browser to terminate the application.

9. Run the application again, which also restores your profile information from the MySQL database.

In this tutorial you have seen how to using the MySQL Profile Provider with MySQL Connector/Net.

4.4 Tutorial: Web Parts Personalization Provider

MySQL Connector/Net provides a Web Parts Personalization provider that allows you to use a MySQL server to store personalization data.

> **Note**
>
> This feature was added in MySQL Connector/Net 6.9.0.

This tutorial demonstrates how to configure the Web Parts Personalization provider using MySQL Connector/Net.

Requirements

- An ASP.NET web site or web application with a membership provider

- .NET Framework 3.0 or above

- MySQL Server 5.5 or above

Configuring MySQL Web Parts Personalization Provider

Tasks include:

- Add References to *MySql.Data* and *MySql.Web* to the web site or web application project.

- Include a MySQL Connector/Net personalization provider into the *system.web* section in the `web.config` file:

```
<webParts>
 <personalization defaultProvider="MySQLPersonalizationProvider">
  <providers>
    <clear/>
    <add name="MySQLPersonalizationProvider"
      type="MySql.Web.Personalization.MySqlPersonalizationProvider,
        MySql.Web, Version=6.9.3.0, Culture=neutral,
        PublicKeyToken=c5687fc88969c44d"
      connectionStringName="LocalMySqlServer"
      applicationName="/" />
  </providers>
  <authorization>
    <allow verbs="modifyState" users="*" />
    <allow verbs="enterSharedScope" users="*"/>
  </authorization>
 </personalization>
</webParts>
```

Creating Web Part Controls

- Create a Web application using MySQL Connector/Net ASP.NET Membership. For information about doing this, see Section 4.2, "Tutorial: MySQL Connector/Net ASP.NET Membership and Role Provider".

- Create a new ASP.NET page.

- Change to the *Design* view.

- From the **Toolbox**, drag a **WebPartManager** control to the page.

- Now define an HTML table with 3 columns and 1 row.

- In the left and middle cells, from the **WebParts Toolbox**, drag and drop a *WebPartZone* control into each cell.

- In the right column, from the **WebParts Toolbox**, drag and drop a *CatalogZone* with *PageCatalogPart* and *EditorZone* controls.

- Add controls to the *WebPartZone*, it will look similar to:

```
<table>
  <tr>
    <td>
      <asp:WebPartZone ID="LeftZone" runat="server" HeaderText="Left Zone">
        <ZoneTemplate>
          <asp:Label ID="Label1" runat="server" title="Left Zone">
            <asp:BulletedList ID="BulletedList1" runat="server">
              <asp:ListItem Text="Item 1"></asp:ListItem>
              <asp:ListItem Text="Item 2"></asp:ListItem>
              <asp:ListItem Text="Item 3"></asp:ListItem>
            </asp:BulletedList>
          </asp:Label>
        </ZoneTemplate>
      </asp:WebPartZone>
    </td>
    <td>
      <asp:WebPartZone ID="MainZone" runat="server" HeaderText="Main Zone">
        <ZoneTemplate>
```

```
            <asp:Label ID="Label11" runat="server" title="Main Zone">
              <h2>This is the Main Zone</h2>
            </asp:Label>
          </ZoneTemplate>
        </asp:WebPartZone>
      </td>
      <td>
        <asp:CatalogZone ID="CatalogZone1" runat="server">
          <ZoneTemplate>
            <asp:PageCatalogPart ID="PageCatalogPart1" runat="server" />
          </ZoneTemplate>
        </asp:CatalogZone>
        <asp:EditorZone ID="EditorZone1" runat="server">
          <ZoneTemplate>
            <asp:LayoutEditorPart ID="LayoutEditorPart1" runat="server" />
            <asp:AppearanceEditorPart ID="AppearanceEditorPart1" runat="server" />
          </ZoneTemplate>
        </asp:EditorZone>
      </td>
    </tr>
</table>
```

- Outside of the HTML table, add a *DropDownList*, *2 Buttons*, and a *Label* as follows:

```
<asp:DropDownList ID="DisplayModes" runat="server" AutoPostBack="True"
   OnSelectedIndexChanged="DisplayModes_SelectedIndexChanged">
</asp:DropDownList>
<asp:Button ID="ResetButton" runat="server" Text="Reset"
   OnClick="ResetButton_Click" />
<asp:Button ID="ToggleButton" runat="server" OnClick="ToggleButton_Click"
   Text="Toggle Scope" />
<asp:Label ID="ScopeLabel" runat="server"></asp:Label>
```

- The following code will fill the *DropDownList* for the *DisplayModes*, show the current scope, reset the personalization state, toggle the scope (between user and the shared scope), and change the display mode:

```
public partial class WebPart : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    if (!IsPostBack)
    {
      foreach (WebPartDisplayMode mode in WebPartManager1.SupportedDisplayModes)
      {
        if (mode.IsEnabled(WebPartManager1))
        {
          DisplayModes.Items.Add(mode.Name);
        }
      }
    }
    ScopeLabel.Text = WebPartManager1.Personalization.Scope.ToString();
  }

  protected void ResetButton_Click(object sender, EventArgs e)
  {
    if (WebPartManager1.Personalization.IsEnabled &&
      WebPartManager1.Personalization.IsModifiable)
```

```
      {
         WebPartManager1.Personalization.ResetPersonalizationState();
      }
   }

   protected void ToggleButton_Click(object sender, EventArgs e)
   {
      WebPartManager1.Personalization.ToggleScope();
   }

   protected void DisplayModes_SelectedIndexChanged(object sender, EventArgs e)
   {
      var mode = WebPartManager1.SupportedDisplayModes[DisplayModes.SelectedValue];
      if (mode != null && mode.IsEnabled(WebPartManager1))
      {
         WebPartManager1.DisplayMode = mode;
      }
   }
}
```

Testing Web Part Changes

Run the application and open the Web Part page, and it should look like this:

Figure 4.9 Web Parts Page
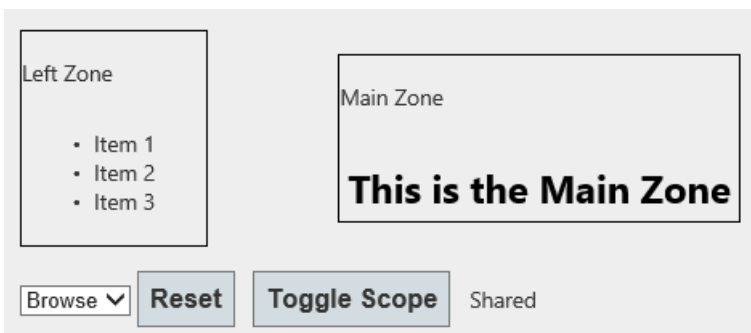
The first time when the user is not authenticated, by default the scope is *Shared*. The user must be authenticated to change settings on the web part controls.

In the following screenshot shows an authenticated user that is able to customize the controls:

Figure 4.10 Authenticated User Controls
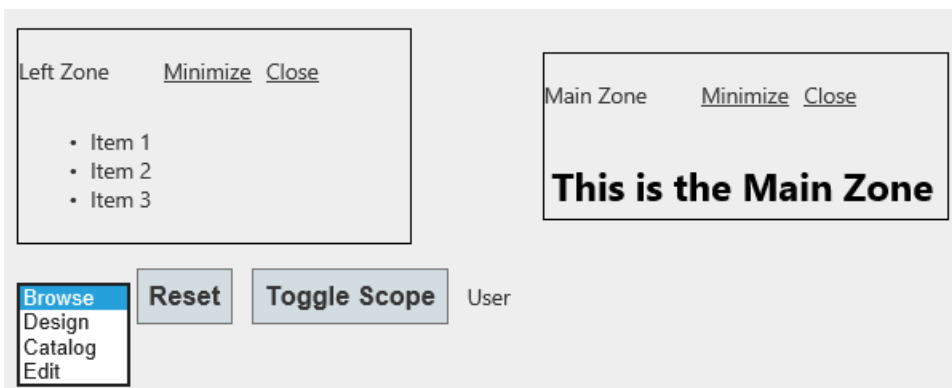
You can see the *DropDownList* and all of the display modes for the current user.

Click *Toggle Scope* to switch the application to the shared scope:

Figure 4.11 Toggle Scope
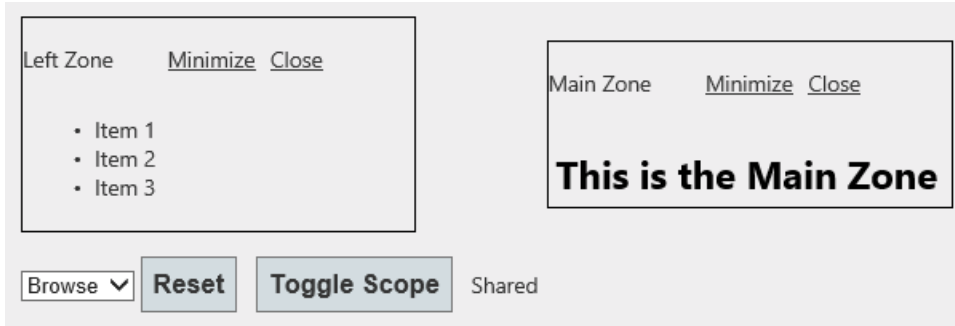
Now you can personalize the zones using the *Edit* or *Catalog* display modes at a specific user or all users level.

Figure 4.12 Personalize Zones

4.5 Tutorial: Simple Membership Web Provider

This section documents the ability to use a Simple Membership Provider on MVC 4 templates. The configuration OAuth compatible for the application to login using external credentials from third party providers like Google, Facebook, Twitter or others.

This tutorial creates an application using a Simple Membership Provider, and then adds third-party (google) OAuth authentication support.

> **Note**
>
> This feature was added in MySQL Connector/Net 6.9.0.

4.5.1 Requirements

- Connector/Net 6.9.x, or greater

- .NET Framework 4.0, or greater

- Visual Studio 2012, or greater

- MVC 4

4.5.2 Creating and Configuring a New Project

- Open Visual Studio and create a new project, of **MVC 4 Web Application** type, and configure the project to use .NET Framework 4.5.

Figure 4.13 Simple Membership: New Project
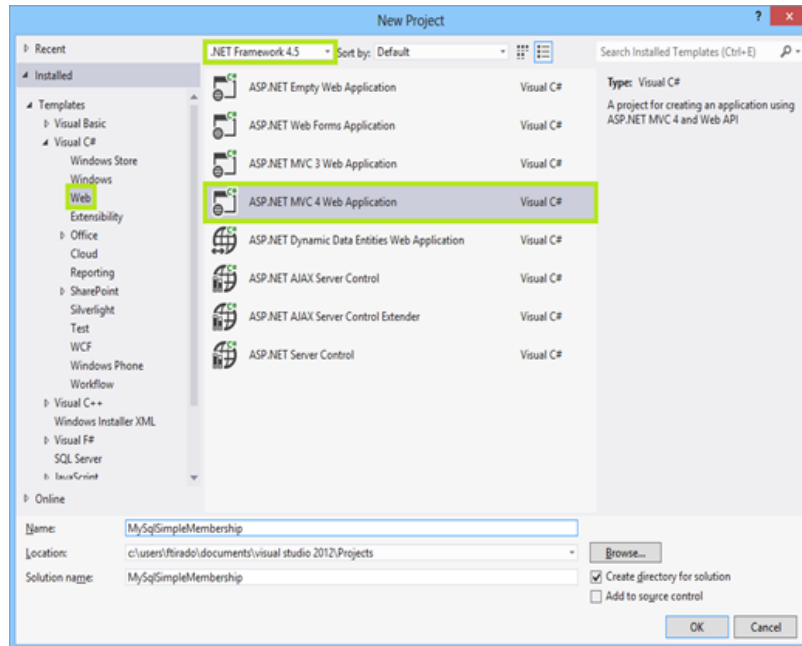
- Choose the template and view engine that you like. This tutorial is using the **Internet Application Template** with the Razor view engine. Optionally, here you can add a unit test project by checking **Create a unit test project**.

Figure 4.14 Simple Membership: Choose Template and Engine
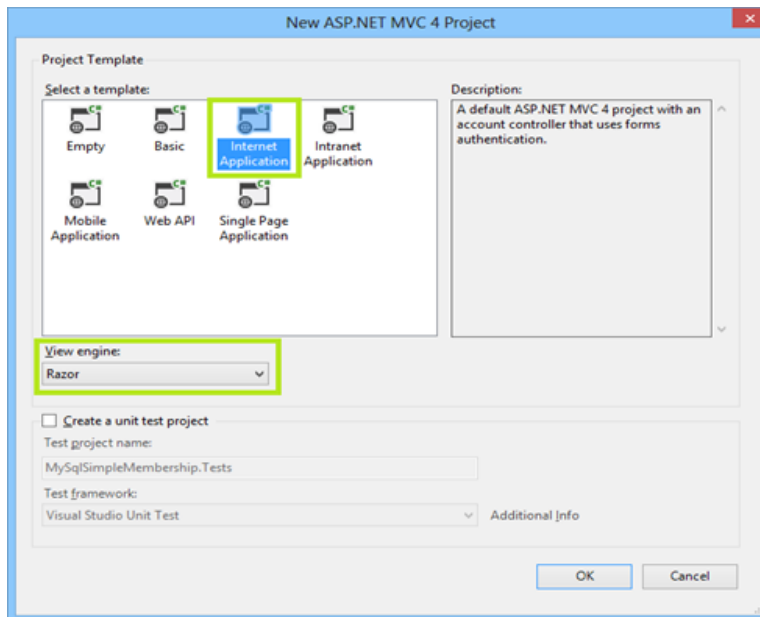
- Add references to the `MySql.Data`, `MySql.Data.Entities`, and `MySql.Web` assemblies. The assemblies chosen must match the .NET Framework and Entity Framework versions added to the project by the template.

- Add a valid MySQL connection string to the `web.config` file, similar to:

```
<add
  name="MyConnection"
  connectionString="server=localhost;UserId=root;password=pass;database=MySqlSimpleMembership;logging=tr
  providerName="MySql.Data.MySqlClient"/>
```

- Under the <system.data> node, add configuration information similar to the following:

```
<membership defaultProvider="MySqlSimpleMembershipProvider">
<providers>
<clear/>
<add
  name="MySqlSimpleMembershipProvider"
  type="MySql.Web.Security.MySqlSimpleMembershipProvider,MySql.Web,Version=6.9.2.0,Culture=neutral,Publi
  applicationName="MySqlSimpleMembershipTest"
  description="MySQLdefaultapplication"
  connectionStringName="MyConnection"
  userTableName="MyUserTable"
  userIdColumn="MyUserIdColumn"
  userNameColumn="MyUserNameColumn"
  autoGenerateTables="True"/>
</providers>
</membership>
```

- Update the above configuration with valid values for the following properties: connectionStringName, userTableName, userIdColumn, userNameColumn, and autoGenerateTables.

- **userTableName**: Name of the table to store the user information. This table is independent from the schema generated by the provider, and it can be changed in the future.

- **userId**: Name of the column that stores the ID for the records in the **userTableName**.

- **userName**: Name of the column that stores the name/user for the records in the **userTableName**.

- **connectionStringName**: This property must match a connection string defined in `web.config` file.

- **autoGenerateTables**: This must be set to `false` if the table to handle the credentials already exists.

- Update your **DBContext** class with the connection string name configured.

- Go to the SimpleMembershipInitializer class contained in the InitializeSimpleMembershipAttribute.cs file that is located in the Filters/ folder. Look for the method call "WebSecurity.InitializeDatabaseConnection", and update the parameters with the configuration for connectionStringName, userTableName, userIdColumn and userNameColumn.

- If the database configured in the connection string does not exist, then create it.

- After running the web application, the following screen is displayed on success:

Figure 4.15 Simple Membership: Generated Home Page
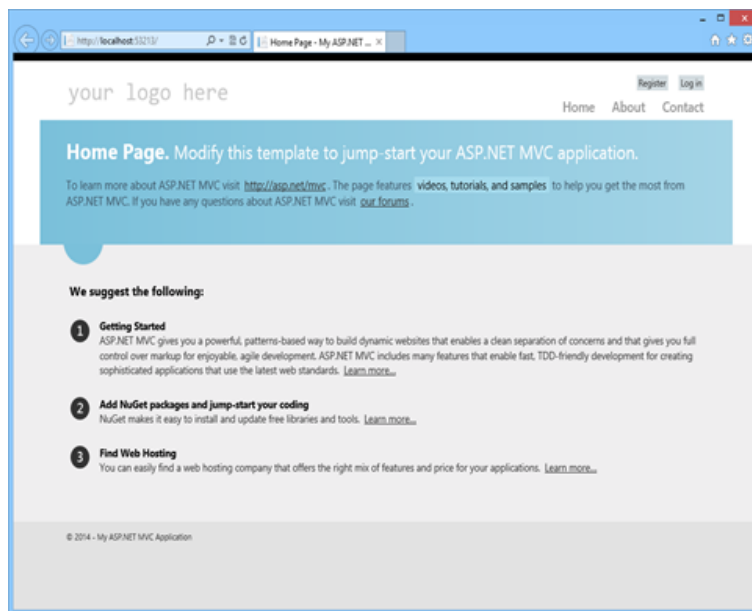
- If the application executed with success, then the generated schema will be similar to the following:

Figure 4.16 Simple Membership: Generated Tables
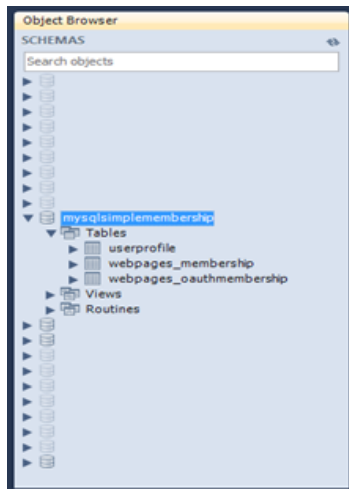
- To create a user, click **Register** on the generated web page. Type the desired user name and password, and then execute the registration form. This redirects you to the home page with the newly created user logged in, as shown near the top-right side of the page:

Figure 4.17 Simple Membership: Logged In
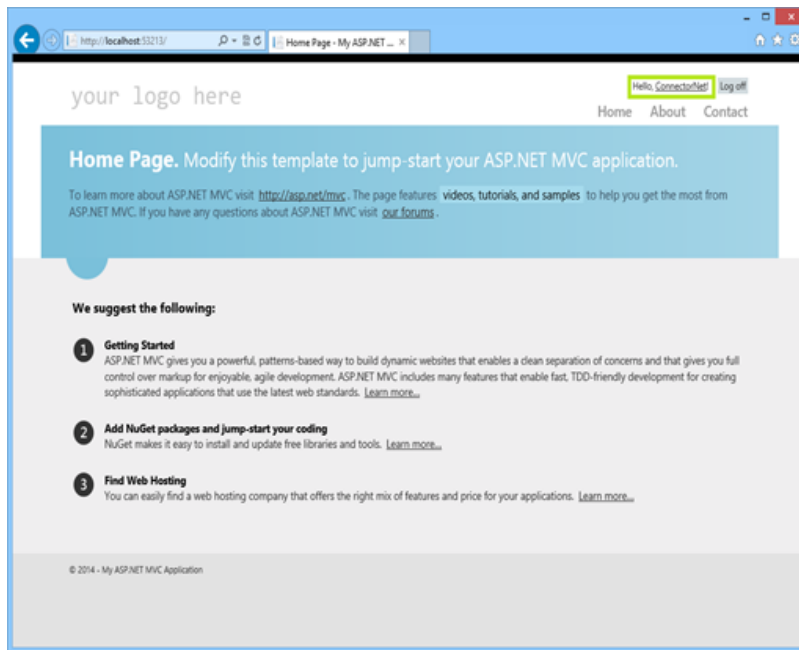
- The data for the newly created user can be located in the `UserProfile` and `Webpages_Membership` tables.

4.5.3 Adding OAuth Authentication to a Project

OAuth is another authentication option for websites that use the Simple Membership Provider. A user can be validated using an external account like Facebook, Twitter, Google, and others. The following steps enable authentication using a Google account in the application.

- Go to the class `AuthConfig.cs` in the `App_Start` folder.

- As we are using google, find the **RegisterAuth** method and uncomment the last line where it calls **OauthWebSecurity.RegisterGoogleClient**.

- Run the application. Once the application is running, click **Log in** to open the log in page. Then, click **Google** under **Use another service to log in**.

Figure 4.18 Simple Membership with OAuth: Google Service
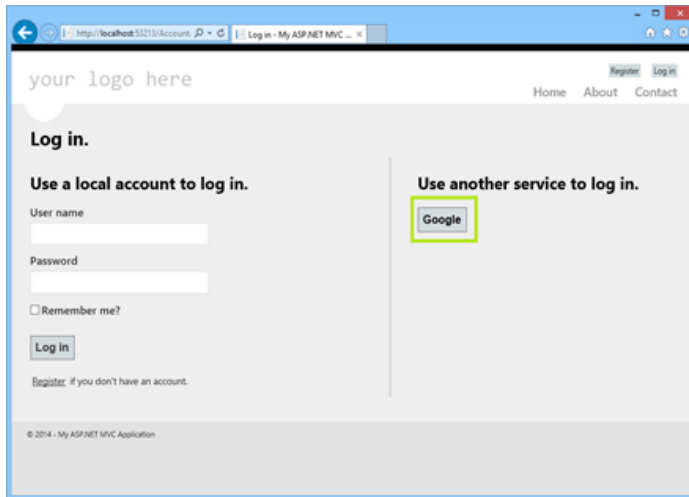
- This redirects to the Google login page (at google.com), and requests you to sign in with your Google account information.

Figure 4.19 Simple Membership with OAuth: Google Service Login
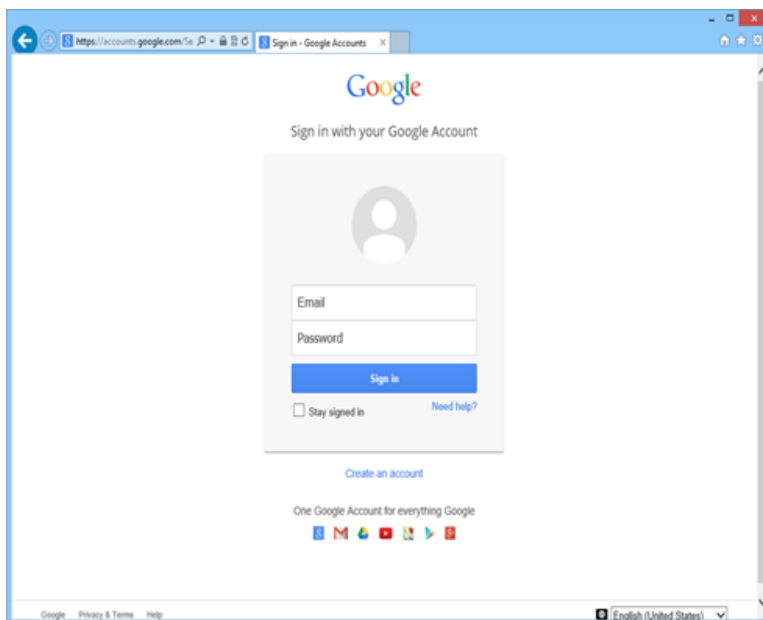
- After submitting the correct credentials, a message requests permission for your application to access the user's information. Read the description and then click **Accept** to allow the quoted actions, and to redirect back to your application's login page.

Figure 4.20 Simple Membership with OAuth: Google Service Approval
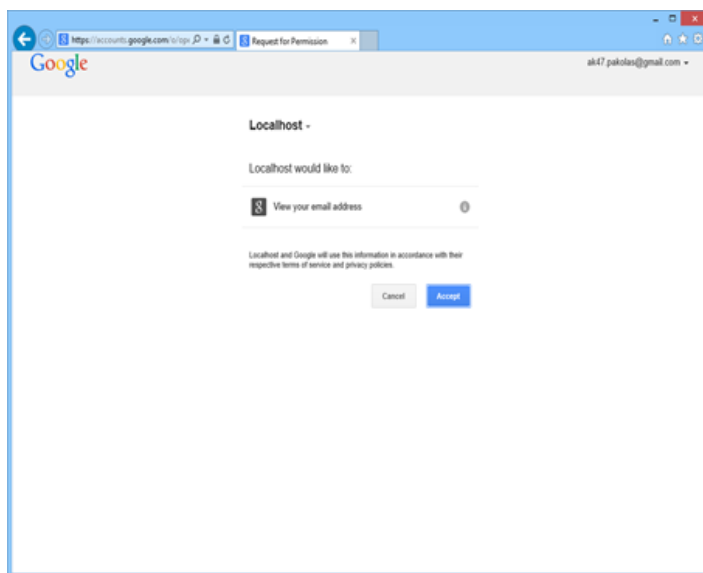
- Now your application can register the account. The **User name** field will be filled in with the appropriate information (in this case, the email address that is associated with the Google account). Click **Register** to register the user with your application.

Figure 4.21 Simple Membership with OAuth: Google Service Registration
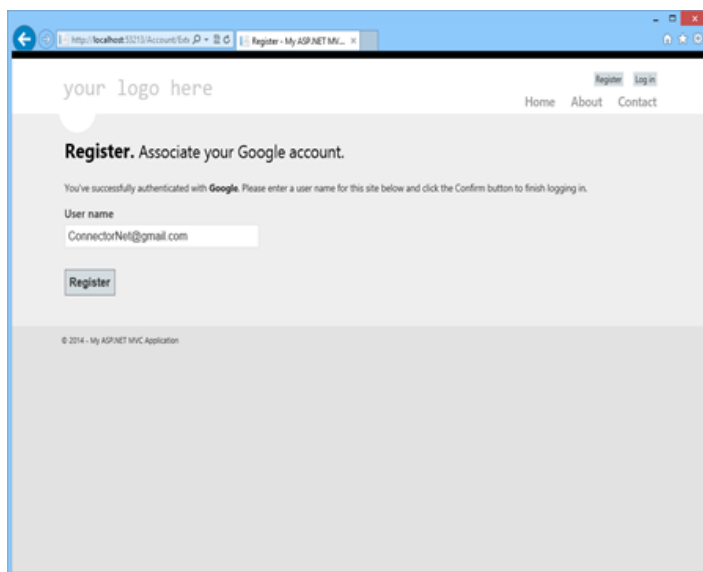
- Now the new user is logged into the application from an external source using OAuth.

Figure 4.22 Simple Membership with OAuth: Google Service Logged In
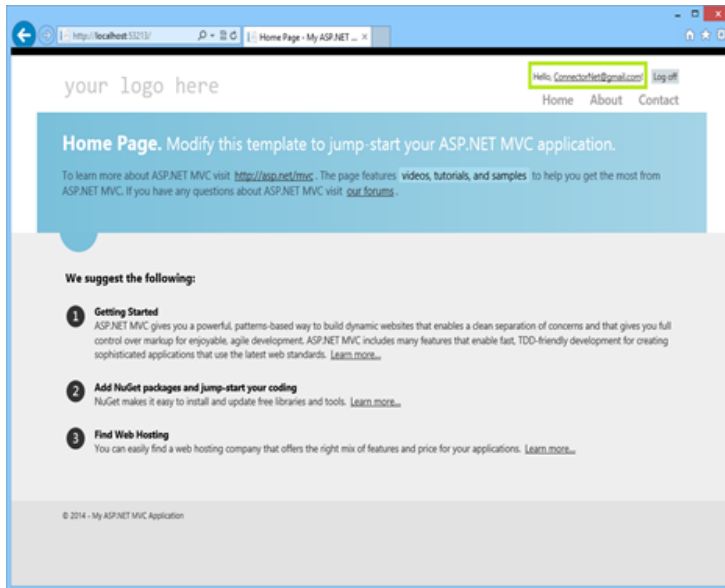
- Information about the new user is stored in the `UserProfile` and `Webpages_OauthMembership` tables.

To use another external option to authenticate users, you must enable the client in the same class where we enabled the Google provider in this tutorial. Typically, providers require you to register your application before allowing OAuth authentication, and once registered they typically provide a token/key and an ID that must be used when registering the provider in the application.

4.6 Tutorial: Using an Entity Framework Entity as a Windows Forms Data Source

In this tutorial you will learn how to create a Windows Forms Data Source from an Entity in an Entity Data Model. This tutorial assumes that you have installed the `world` database sample, which can be downloaded from the MySQL Documentation page. You can also find details on how to install the database on the same page. It will also be convenient for you to create a connection to the `world` database after it is installed. For instructions on how to do this, see C.

Creating a new Windows Forms application

The first step is to create a new Windows Forms application.

1. In Visual Studio, select **File**, **New**, **Project** from the main menu.

2. Choose the **Windows Forms Application** installed template. Click **OK**. The solution is created.

Adding an Entity Data Model

You will now add an Entity Data Model to your solution.

1. In the Solution Explorer, right-click your application and select **Add**, **New Item...**. From **Visual Studio installed templates** select **ADO.NET Entity Data Model**. Click **Add**.

Figure 4.23 Add Entity Data Model
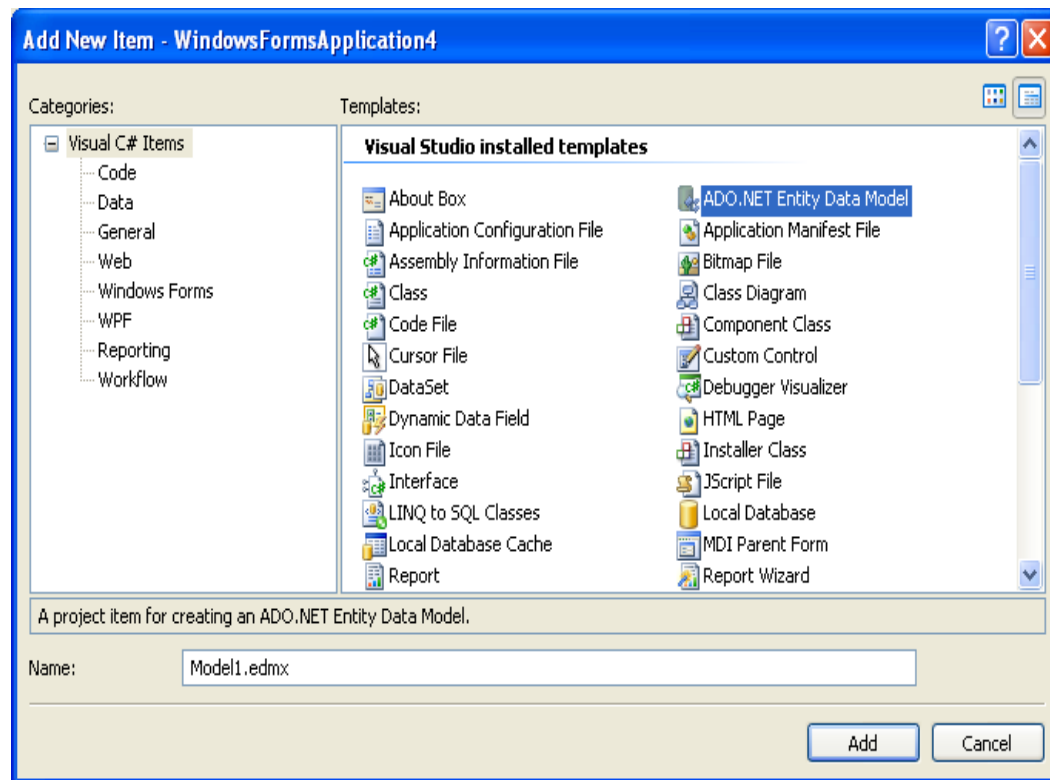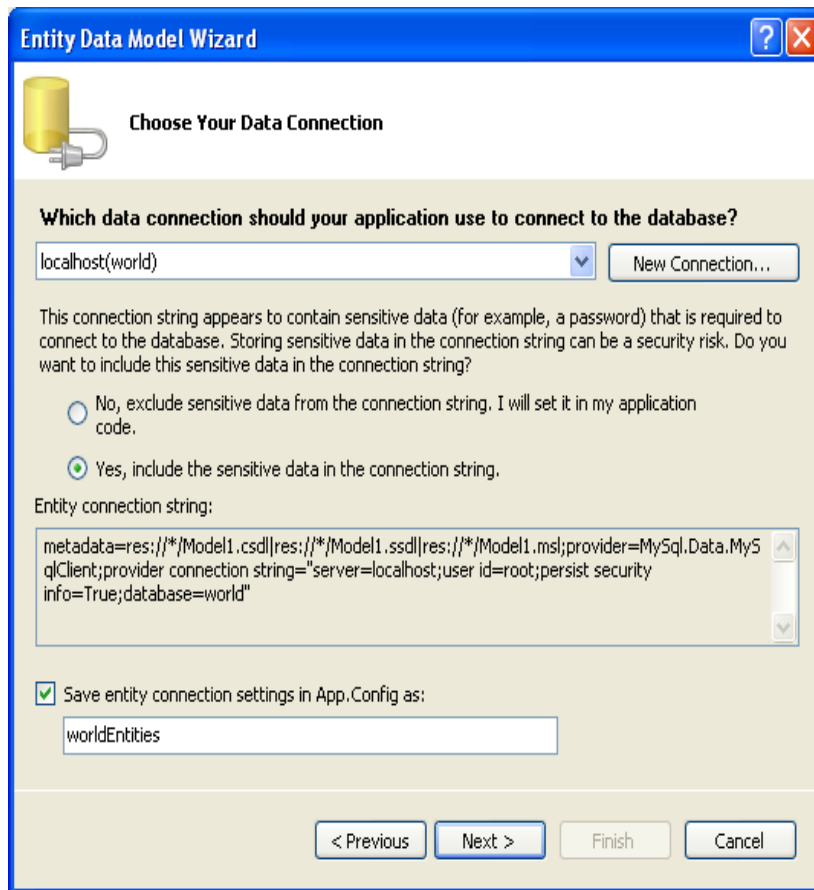
2. You will now see the Entity Data Model Wizard. You will use the wizard to generate the Entity Data Model from the `world` database sample. Select the icon **Generate from database**. Click **Next**.

Figure 4.24 Entity Data Model Wizard Screen 1
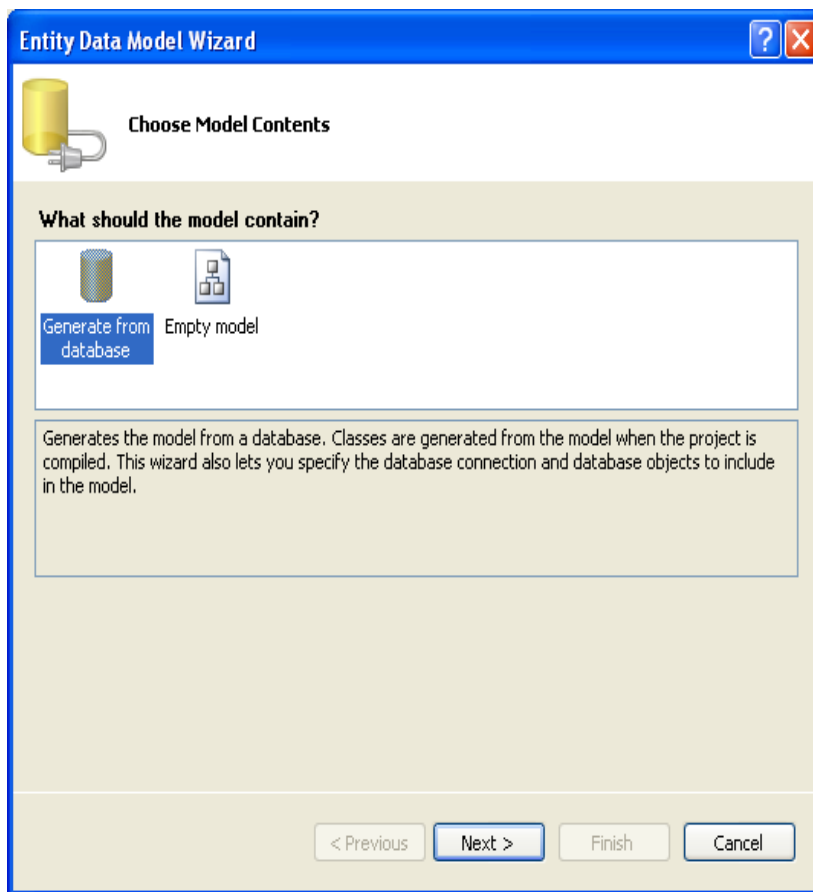
3. You can now select the connection you made earlier to the `world` database. If you have not already done so, you can create the new connection at this time by clicking **New Connection...**. For further instructions on creating a connection to a database see Making a Connection.

Figure 4.25 Entity Data Model Wizard Screen 2

4. Make a note of the entity connection settings to be used in App.Config, as these will be used later to write the necessary control code.

5. Click **Next**.

6. The Entity Data Model Wizard connects to the database. You are then presented with a tree structure of the database. From this you can select the object you would like to include in your model. If you had created Views and Stored Routines these will be displayed along with any tables. In this example you just need to select the tables. Click **Finish** to create the model and exit the wizard.

Figure 4.26 Entity Data Model Wizard Screen 3
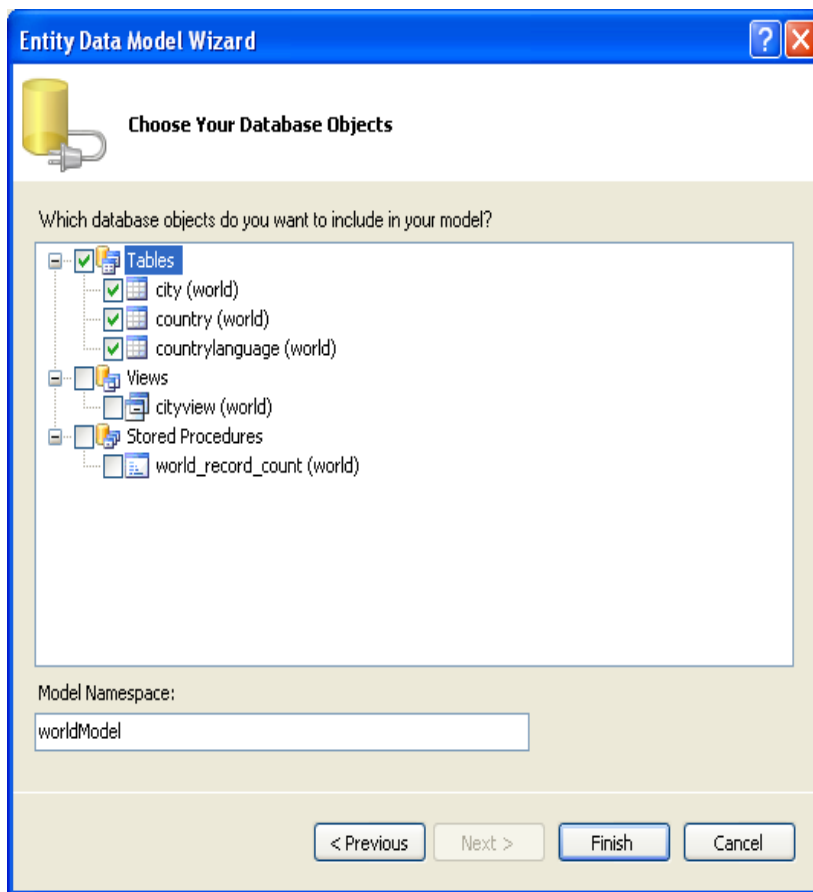
7. Visual Studio will generate the model and then display it.

Figure 4.27 Entity Data Model Diagram
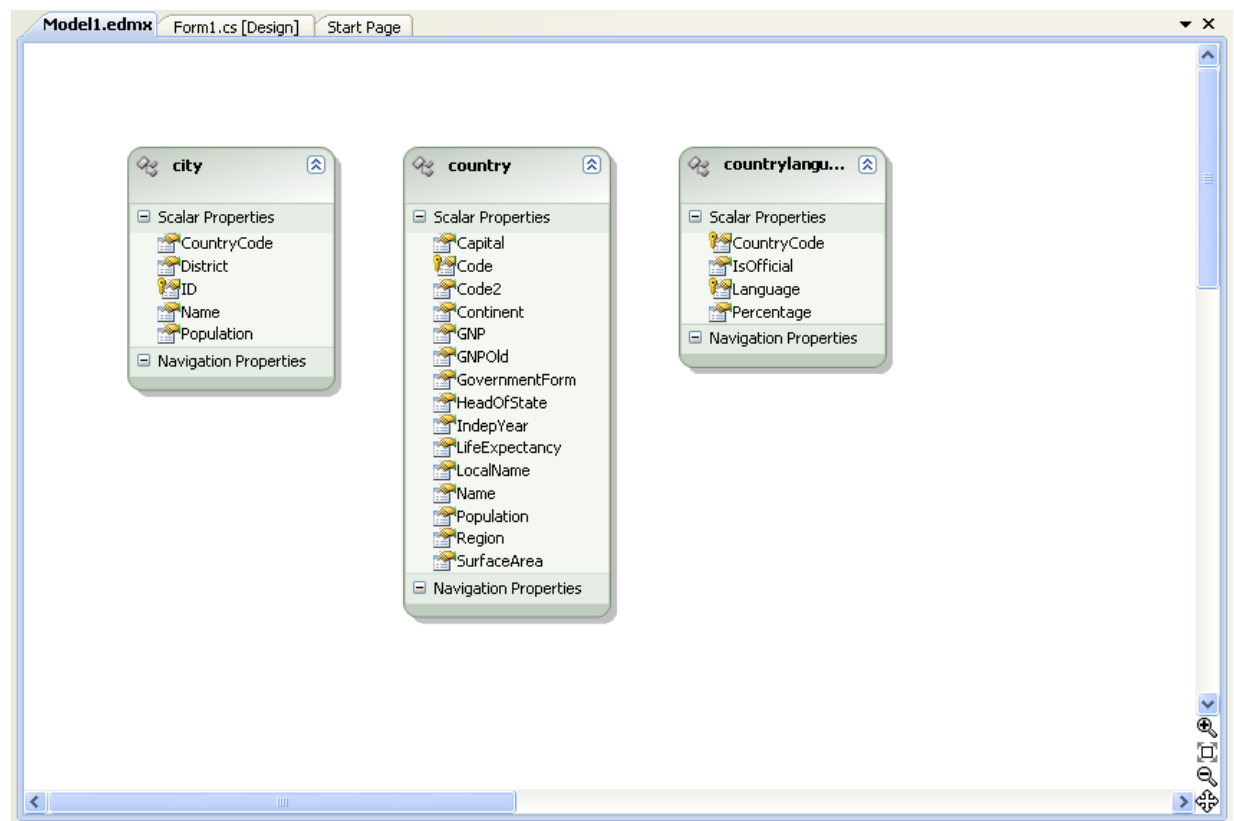
8. From the Visual Studio main menu select **Build**, **Build Solution**, to ensure that everything compiles correctly so far.

Adding a new Data Source

You will now add a new Data Source to your project and see how it can be used to read and write to the database.

1. From the Visual Studio main menu select **Data**, **Add New Data Source....** You will be presented with the Data Source Configuration Wizard.

Figure 4.28 Entity Data Source Configuration Wizard Screen 1
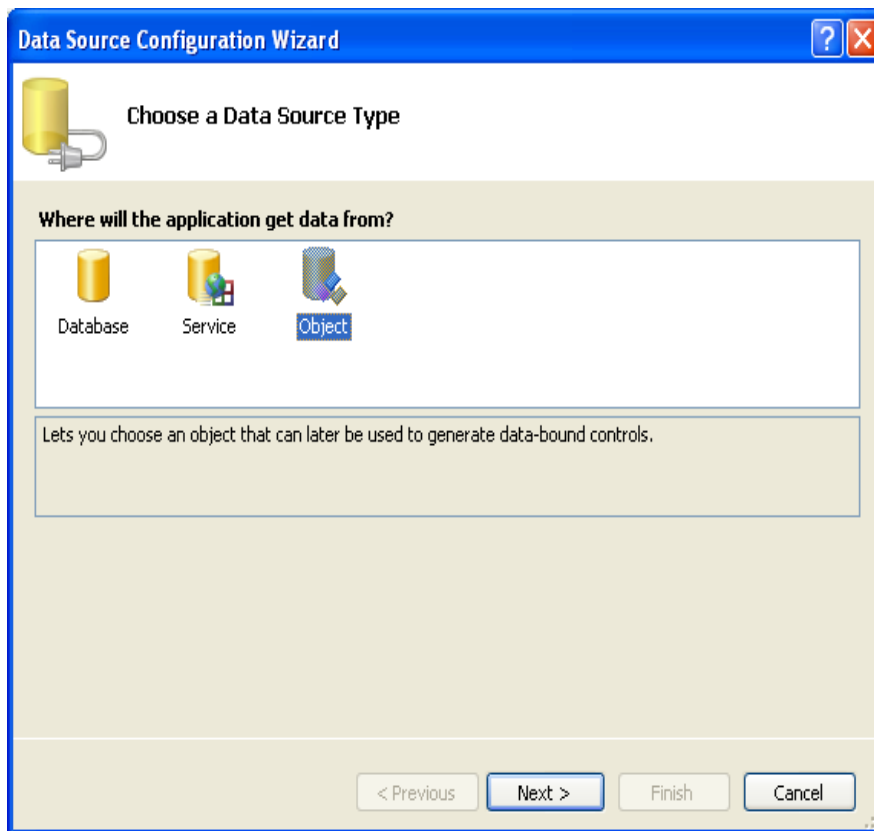
2. Select the **Object** icon. Click **Next**.

3. You will now select the Object to bind to. Expand the tree. In this tutorial, you will select the city table. Once the city table has been selected click **Next**.

Figure 4.29 Entity Data Source Configuration Wizard Screen 2
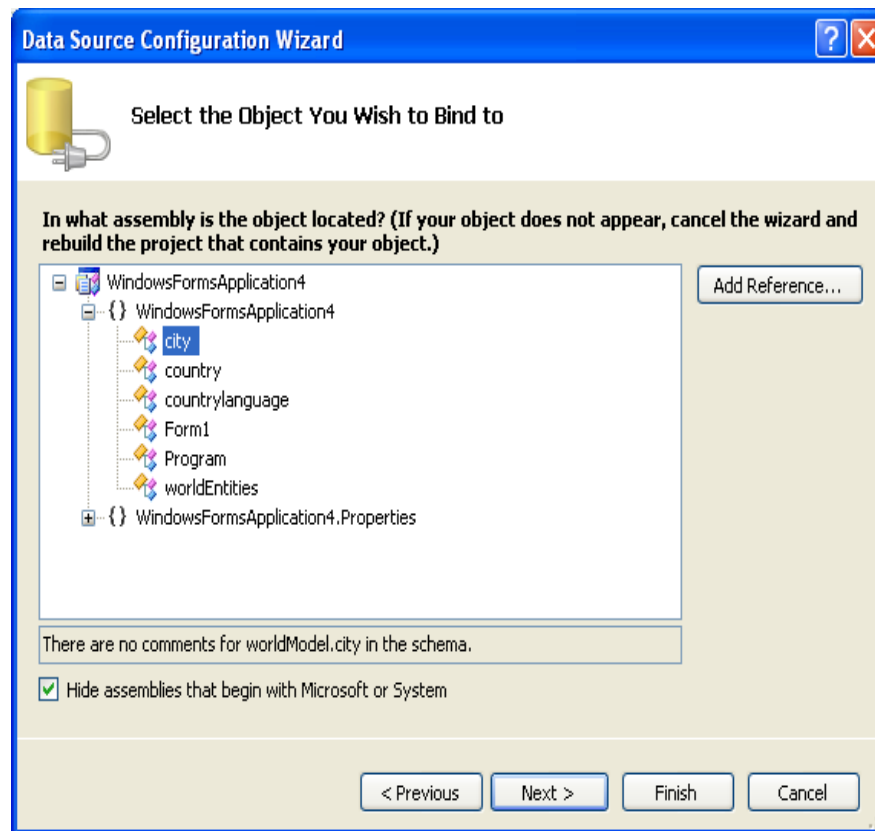
4. The wizard will confirm that the city object is to be added. Click **Finish**.

Figure 4.30 Entity Data Source Configuration Wizard Screen 3
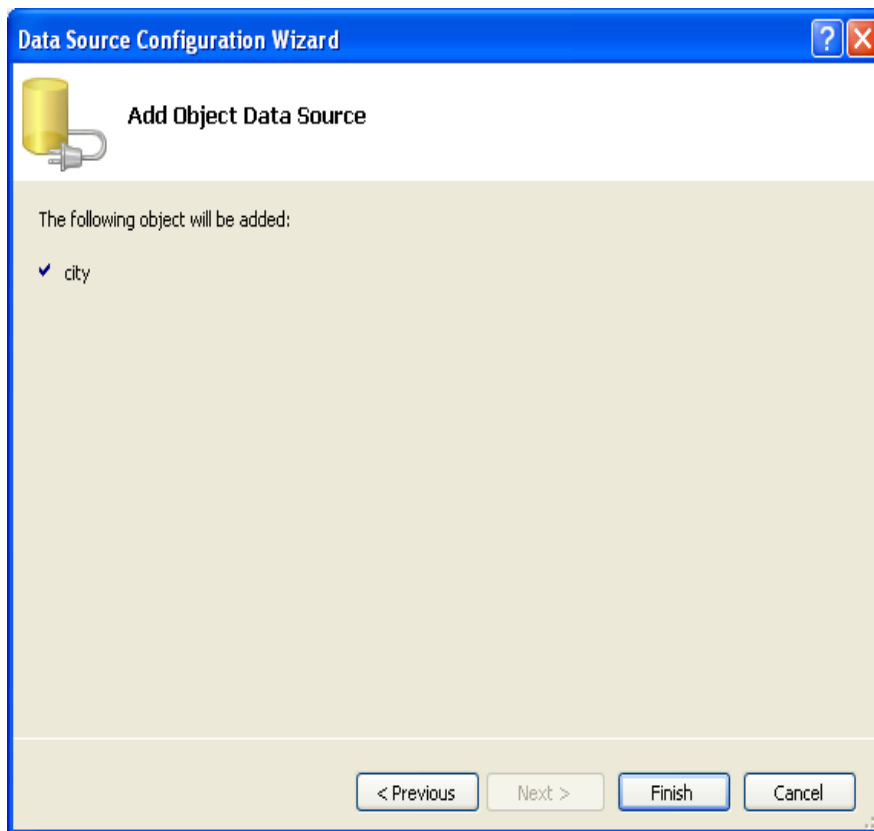
5. The city object will be display in the Data Sources panel. If the Data Sources panel is not displayed, select **Data**, **Show Data Sources** from the Visual Studio main menu. The docked panel will then be displayed.

Figure 4.31 Data Sources
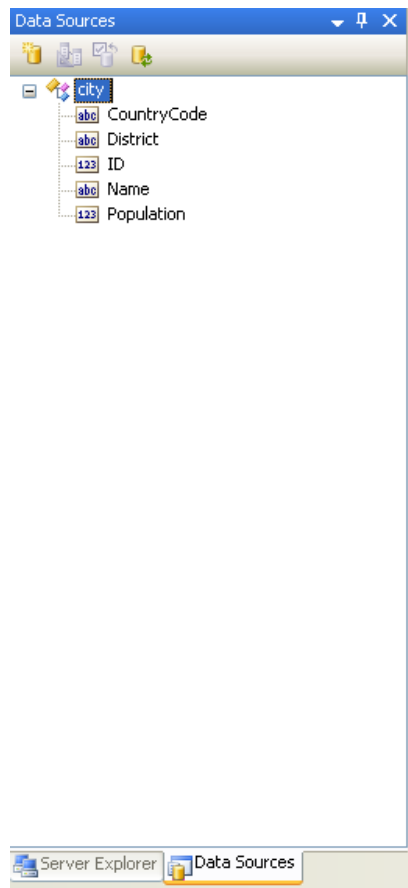
Using the Data Source in a Windows Form

You will now learn how to use the Data Source in a Windows Form.

1. In the Data Sources panel select the Data Source you just created and drag and drop it onto the Form Designer. By default the Data Source object will be added as a Data Grid View control.

> **Note**
>
> The Data Grid View control is bound to `cityBindingSource`, and the Navigator control is bound to `cityBindingNavigator`.

Figure 4.32 Data Form Designer
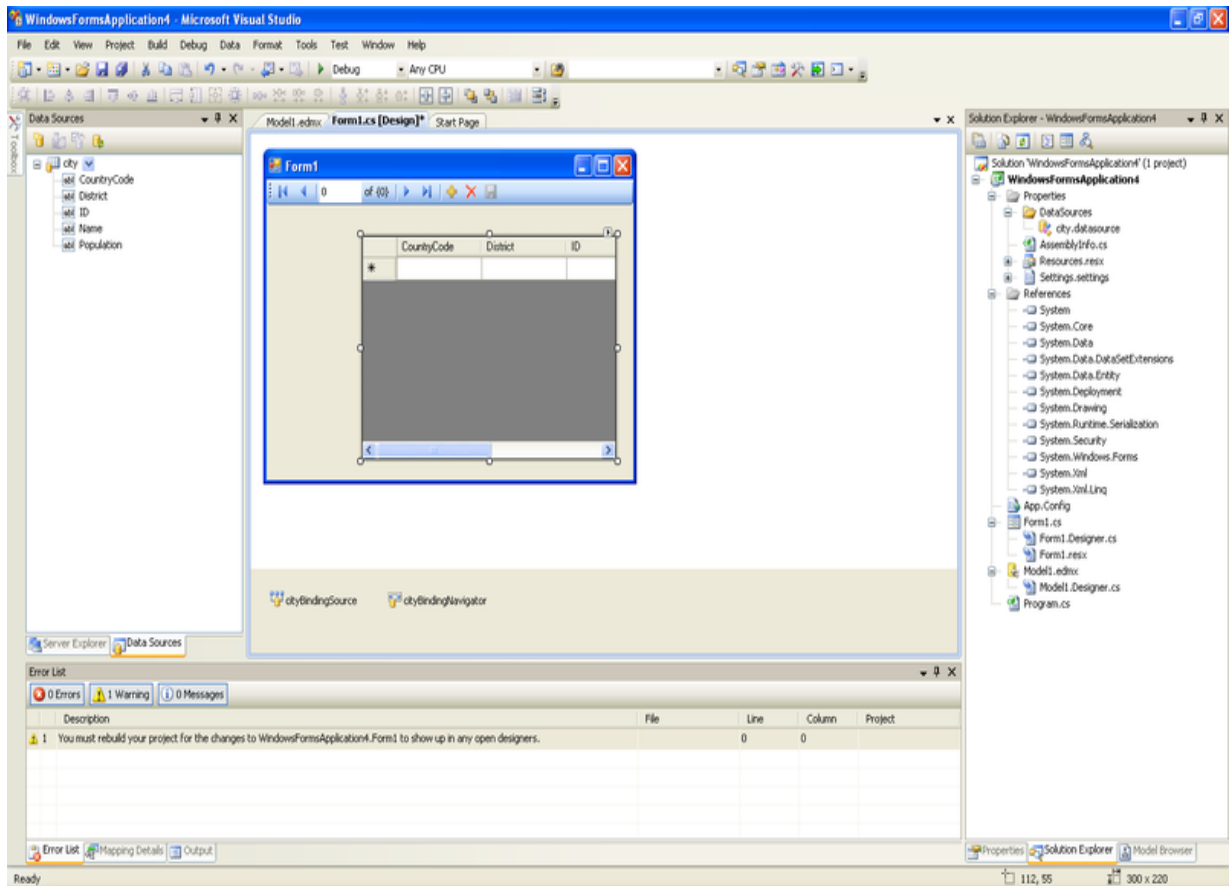
2. Save and rebuild the solution before continuing.

Adding Code to Populate the Data Grid View

You are now ready to add code to ensure that the Data Grid View control will be populated with data from the City database table.

1. Double-click the form to access its code.

2. Add code to instantiate the Entity Data Model's EntityContainer object and retrieve data from the database to populate the control.

Figure 4.33 Adding Code to the Form
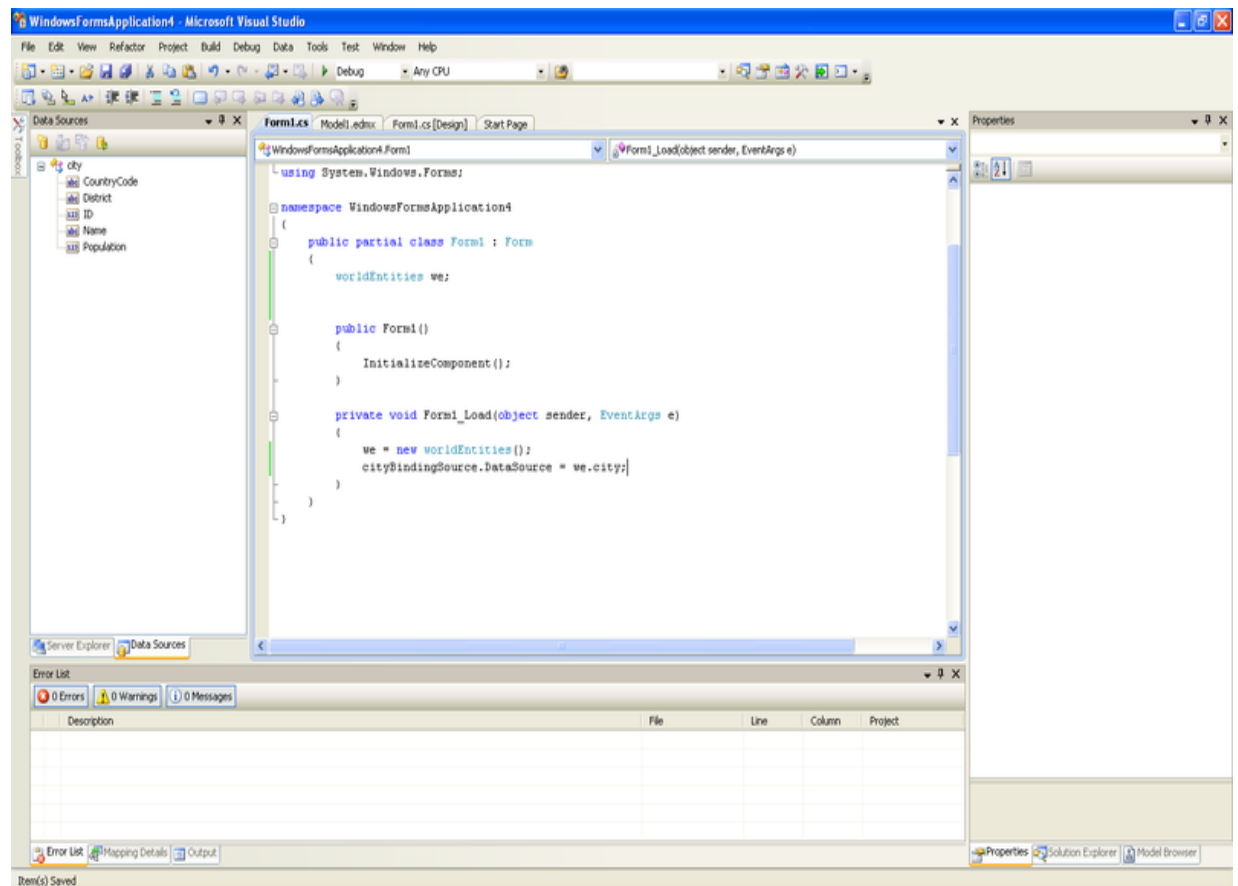
3. Save and rebuild the solution.

4. Run the solution. Ensure the grid is populated and you can navigate the database.

Figure 4.34 The Populated Grid Control
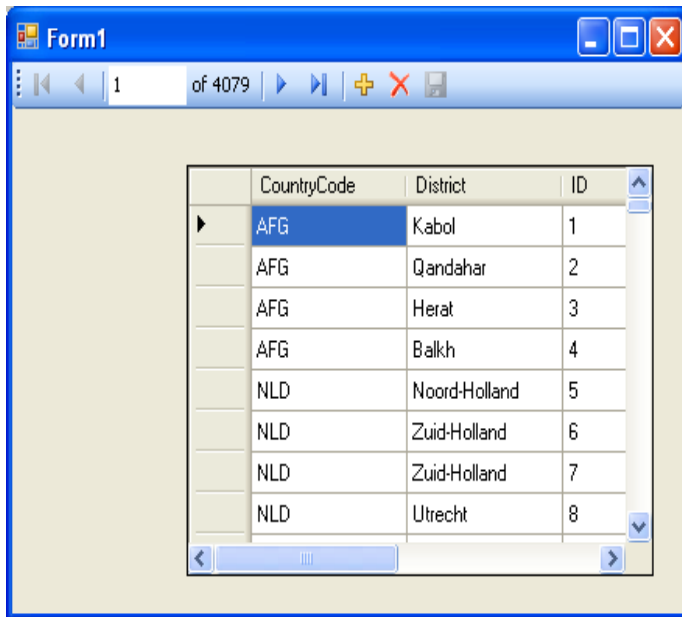
Adding Code to Save Changes to the Database

You will now add code to enable you to save changes to the database.

The Binding source component ensures that changes made in the Data Grid View control are also made to the Entity classes bound to it. However, that data needs to be saved back from the entities to the database itself. This can be achieved by the enabling of the Save button in the Navigator control, and the addition of some code.

1. In the Form Designer, click the Save icon in the Form toolbar and ensure that its Enabled property is set to True.

Figure 4.35 Save Button Enabled
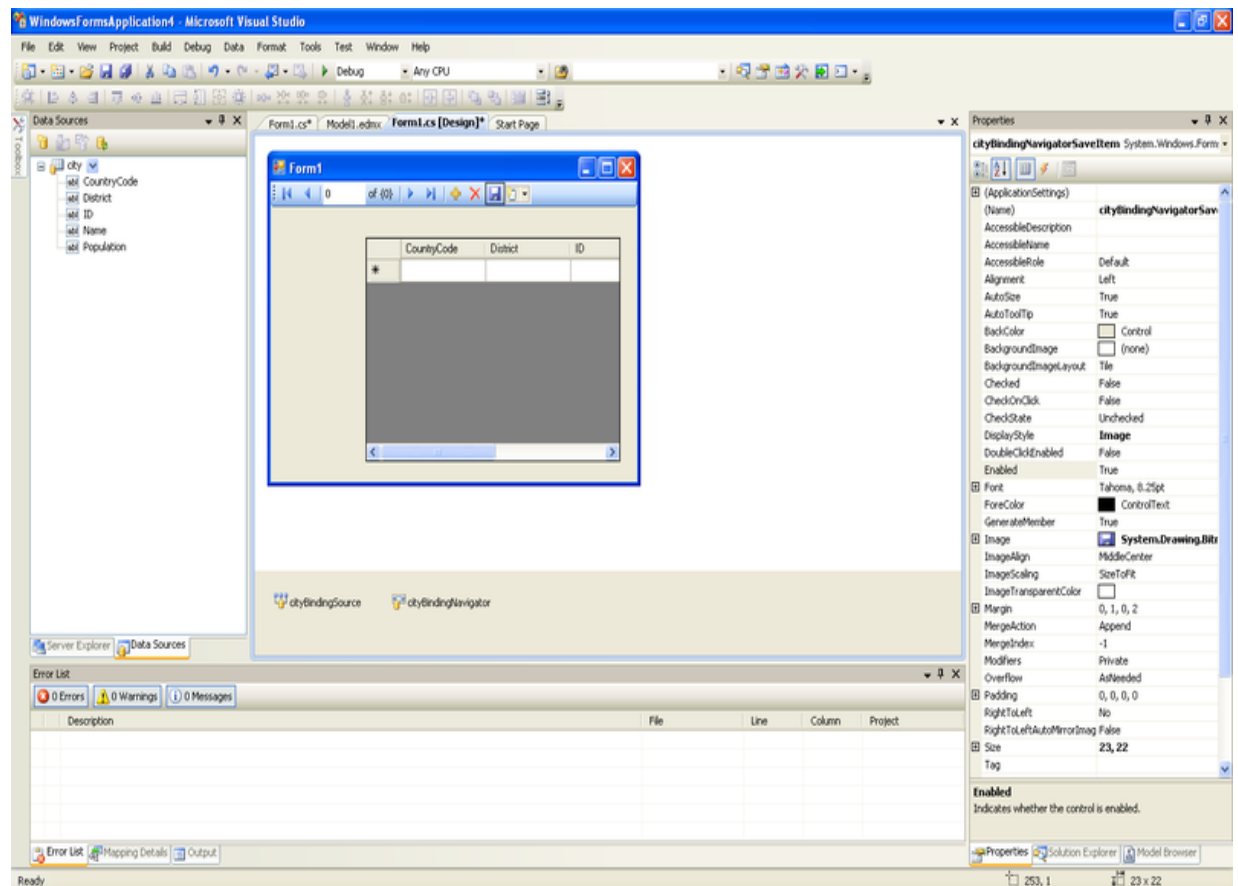
2. Double-click the Save icon in the Form toolbar to display its code.

3. You now need to add code to ensure that data is saved to the database when the save button is clicked in the application.

Figure 4.36 Adding Save Code to the Form
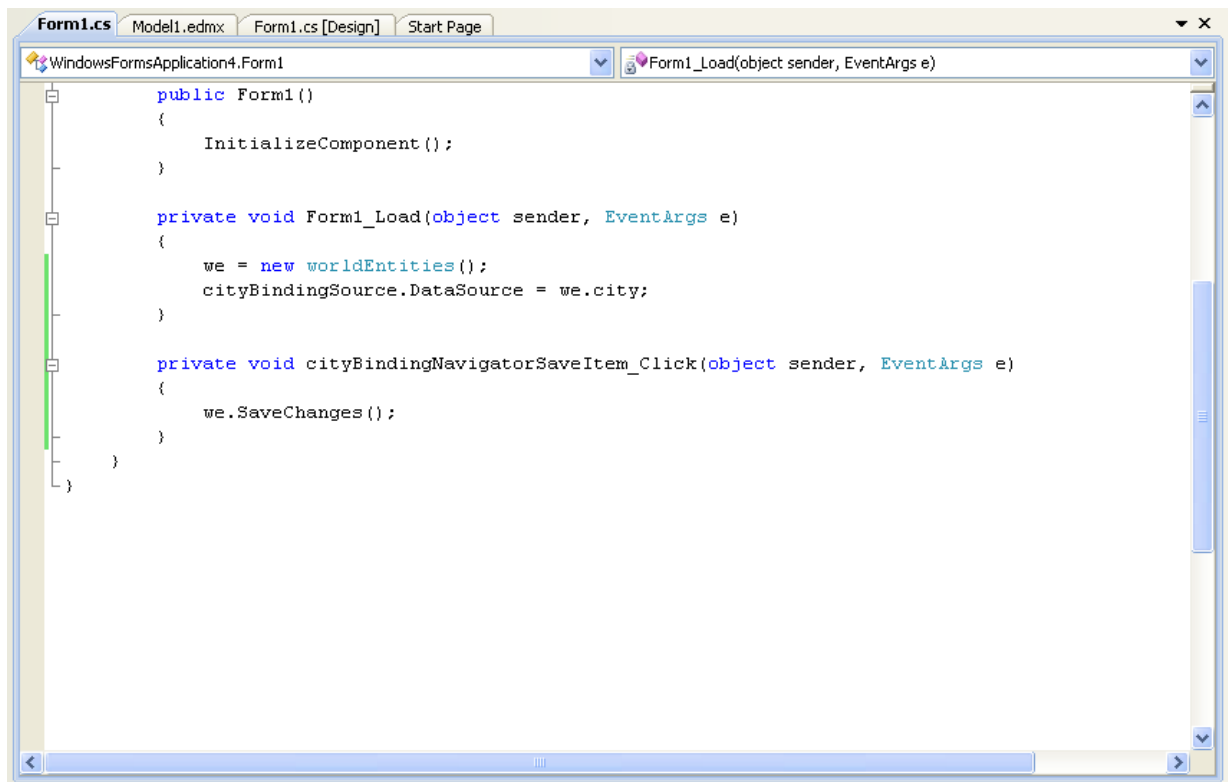
4. Once the code has been added, save the solution and rebuild it. Run the application and verify that changes made in the grid are saved.

4.7 Tutorial: Databinding in ASP.NET Using LINQ on Entities

In this tutorial you create an ASP.NET web page that binds LINQ queries to entities using the Entity Framework mapping.

If you have not already done so, install the `world` database sample prior to attempting this tutorial. See the tutorial Section 4.6, "Tutorial: Using an Entity Framework Entity as a Windows Forms Data Source" for instructions on downloading and installing this database.

Creating an ASP.NET web site

In this part of the tutorial, you create an ASP.NET web site. The web site uses the `world` database. The main web page features a drop down list from which you can select a country. Data about that country's cities is then displayed in a grid view control.

1. From the Visual Studio main menu select **File**, **New**, **Web Site...**.

2. From the Visual Studio installed templates select **ASP.NET Web Site**. Click **OK**. You will be presented with the Source view of your web page by default.

3. Click the Design view tab situated underneath the Source view panel.

Figure 4.37 The Design Tab
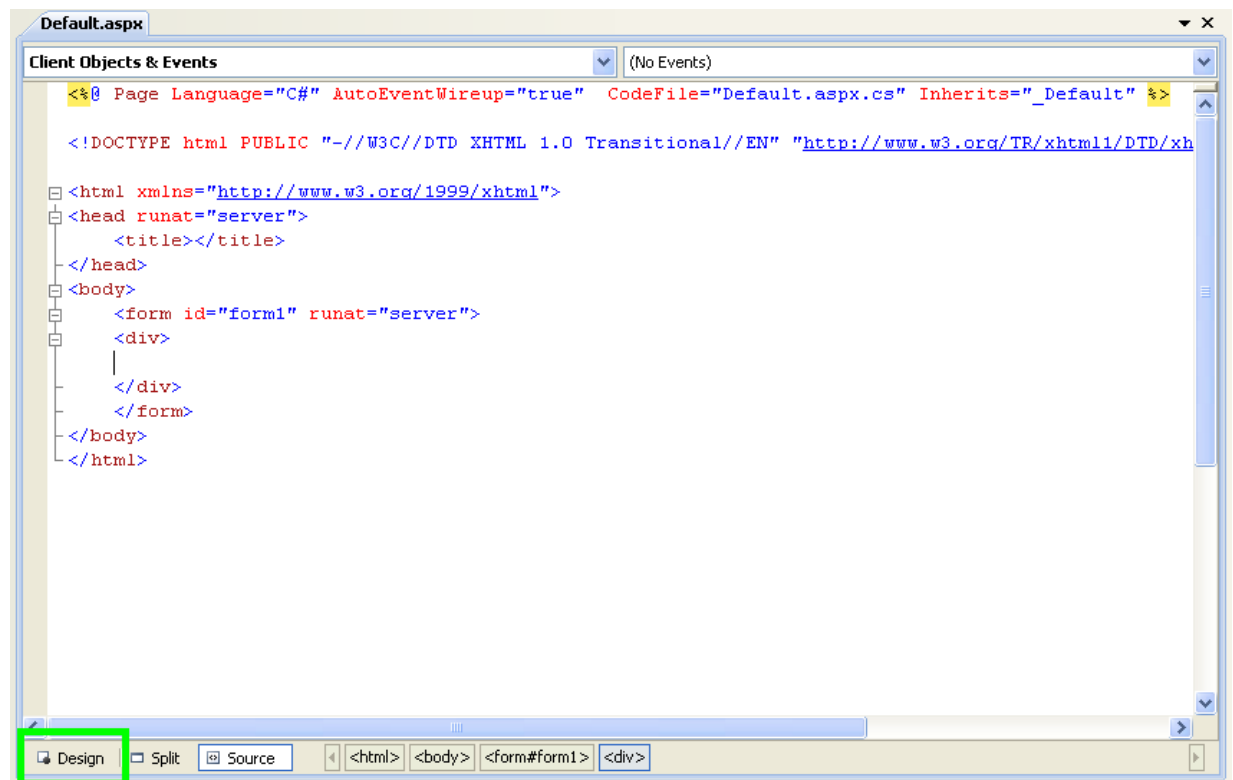
4. In the Design view panel, enter some text to decorate the blank web page.

5. Click Toolbox. From the list of controls select **DropDownList**. Drag and drop the control to a location beneath the text on your web page.

Figure 4.38 Drop Down List
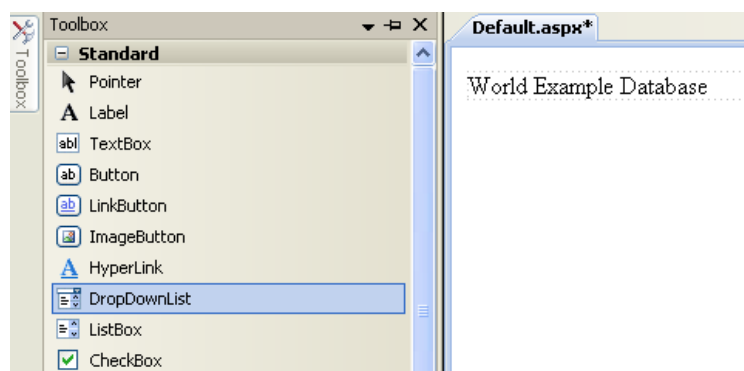
6. From the **DropDownList** control's context menu, ensure that the **Enable AutoPostBack** check box is enabled. This will ensure the control's event handler is called when an item is selected. The user's choice will in turn be used to populate the **GridView** control.

Figure 4.39 Enable AutoPostBack
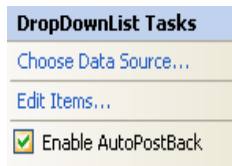
7. From the Toolbox select the **GridView** control.

Figure 4.40 Grid View Control
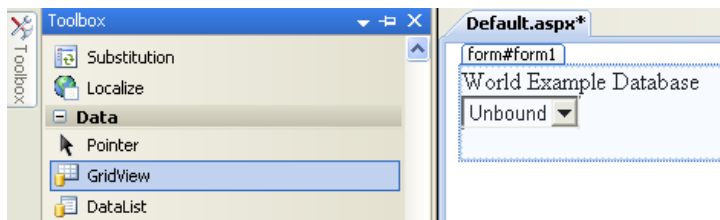
Drag and drop the Grid Vew control to a location just below the Drop Down List you already placed.

Figure 4.41 Placed Grid Vew Control
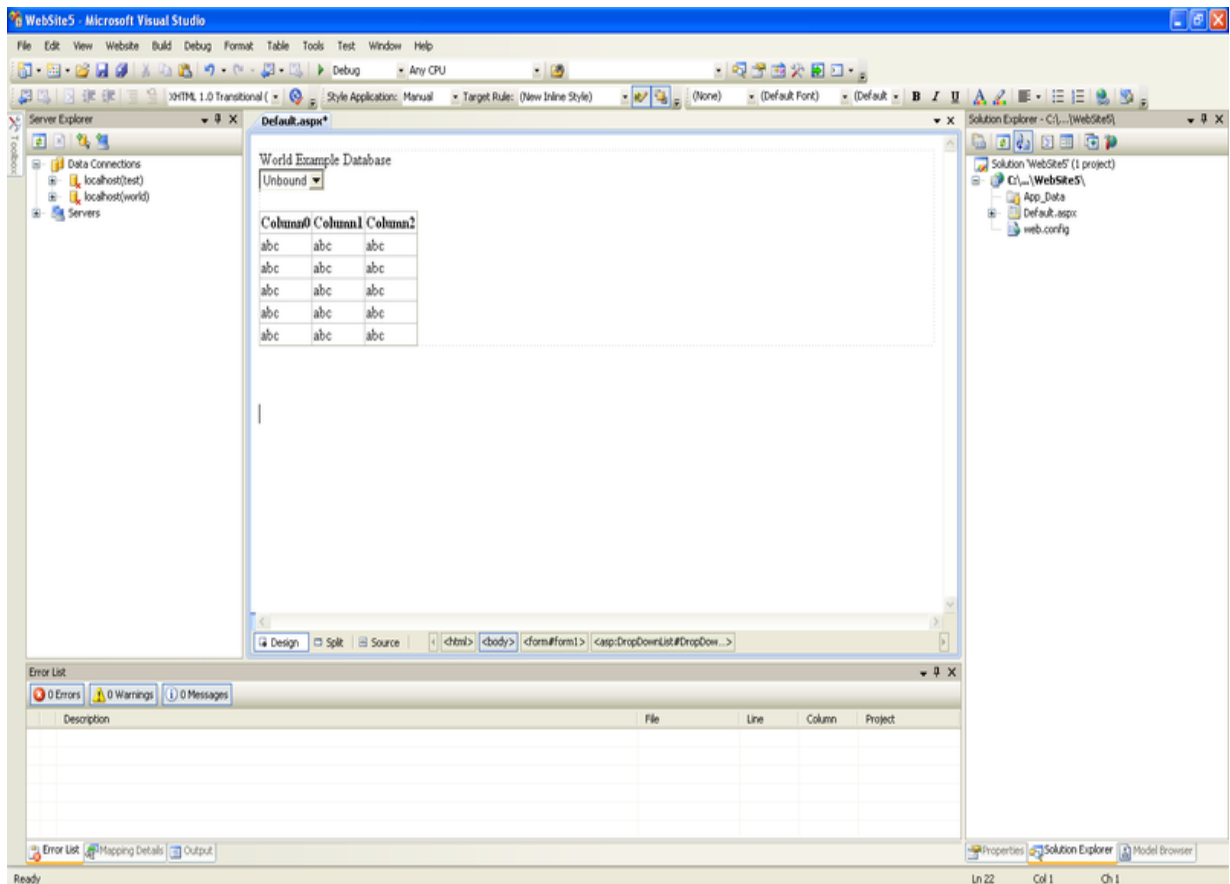
8. At this point it is recommended that you save your solution, and build the solution to ensure that there are no errors.

9. If you run the solution you will see that the text and drop down list are displayed, but the list is empty. Also, the grid view does not appear at all. Adding this functionality is described in the following sections.

At this stage you have a web site that will build, but further functionality is required. The next step will be to use the Entity Framework to create a mapping from the `world` database into entities that you can control programmatically.

Creating an ADO.NET Entity Data Model

In this stage of the tutorial you will add an ADO.NET Entity Data Model to your project, using the `world` database at the storage level. The procedure for doing this is described in the tutorial Section 4.6, "Tutorial: Using an Entity Framework Entity as a Windows Forms Data Source", and so will not be repeated here.

Populating a Drop Data List Box with using the results of a entity LINQ query

In this part of the tutorial you will write code to populate the DropDownList control. When the web page loads the data to populate the list will be achieved by using the results of a LINQ query on the model created previously.

1. In the Design view panel, double-click any blank area. This brings up the `Page_Load` method.

2. Modify the relevant section of code according to the following listing:

```
...
public partial class _Default : System.Web.UI.Page
{
    worldModel.worldEntities we;

    protected void Page_Load(object sender, EventArgs e)
    {
        we = new worldModel.worldEntities();

        if (!IsPostBack)
        {
            var countryQuery = from c in we.country
                               orderby c.Name
                               select new { c.Code, c.Name };
            DropDownList1.DataValueField = "Code";
            DropDownList1.DataTextField = "Name";
            DropDownList1.DataSource = countryQuery;
            DataBind();
        }
    }
...
```

The list control only needs to be populated when the page first loads. The conditional code ensures that if the page is subsequently reloaded, the list control is not repopulated, which would cause the user selection to be lost.

3. Save the solution, build it and run it. You should see the list control has been populated. You can select an item, but as yet the grid view control does not appear.

At this point you have a working Drop Down List control, populated by a LINQ query on your entity data model.

Populating a Grid View control using an entity LINQ query

In the last part of this tutorial you will populate the Grid View Control using a LINQ query on your entity data model.

1. In the Design view, double-click the **DropDownList** control. This causes its `SelectedIndexChanged` code to be displayed. This method is called when a user selects an item in the list control and thus fires an AutoPostBack event.

2. Modify the relevant section of code accordingly to the following listing:

```
...
    protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
    {
        var cityQuery = from c in we.city
                        where c.CountryCode == DropDownList1.SelectedValue
                        orderby c.Name
                        select new { c.Name, c.Population, c.CountryCode };
        GridView1.DataSource = cityQuery;
        DataBind();
    }
...
```

The grid view control is populated from the result of the LINQ query on the entity data model.

3. As a check compare your code to that shown in the following screenshot:

Figure 4.42 Source Code
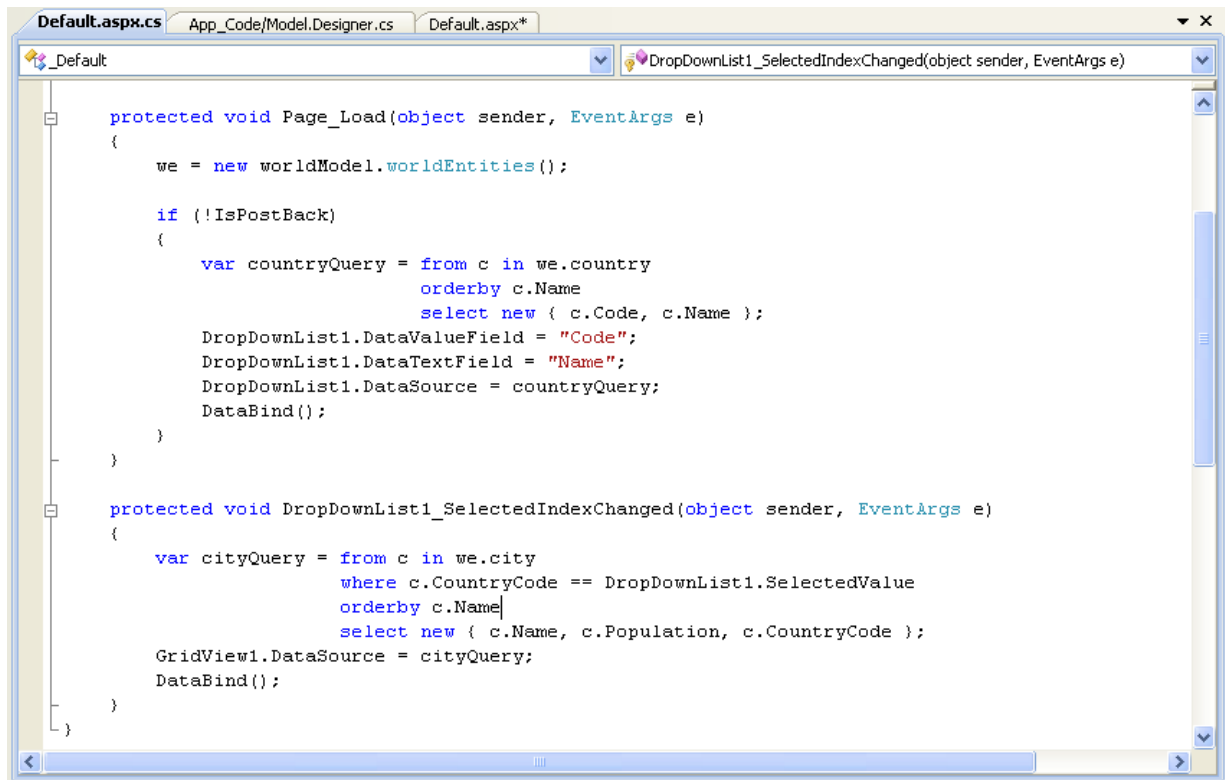
4. Save, build and run the solution. As you select a country you will see its cities are displayed in the grid view control.

Figure 4.43 The Working Web Site
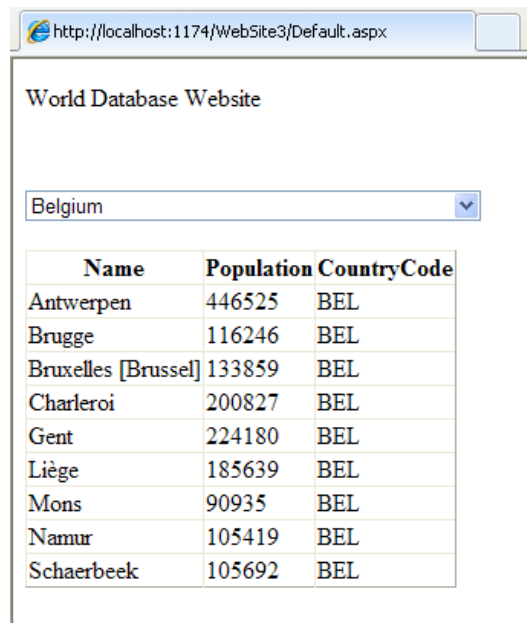
In this tutorial you have seen how to create an ASP.NET web site, you have also seen how you can access a MySQL database using LINQ queries on an entity data model.

4.8 Tutorial: Using SSL with MySQL Connector/Net

In this tutorial you will learn how you can use MySQL Connector/Net to connect to a MySQL server configured to use SSL. Support for SSL client certificates was added with MySQL Connector/Net 6.2.

MySQL Server uses the PEM format for certificates and private keys. This tutorial will use the test certificates from the server test suite by way of example. You can obtain the MySQL Server source code from MySQL Downloads. The certificates can be found in the directory `./mysql-test/std_data`.

To carry out the steps in this tutorial, you must have Open SSL installed. This can be downloaded for Microsoft Windows at no charge from Shining Light Productions.

Further details on the connection string options used in this tutorial can be found at Chapter 6, *Connector/ Net Connection String Options Reference*.

Configuring the MySQL Server to use SSL

1. In the MySQL Server configuration file, set the SSL parameters as follows:

```
ssl-ca=path/to/repo/mysql-test/std_data/cacert.pem
ssl-cert=path/to/repo/mysql-test/std_data/server-cert.pem
ssl-key=path/to/repo/mysql-test/std_data/server-key.pem
```

   Adjust the directories according to the location in which you installed the MySQL source code.

2. In this step you create a test user and set the user to require SSL.

   Using the MySQL Command-Line Client, connect as `root` and create the user `sslclient`.

3. To set privileges and requirements, issue the following command:

```
GRANT ALL PRIVILEGES ON *.* TO sslclient@'%' REQUIRE SSL;
```

Creating a certificate file to use with the .NET client

1. The .NET client does not use the PEM file format, as .NET does not support this format natively. You will be using test client certificates from the same server repository, for the purposes of this example. Convert these to PFX format first. This format is also known as PKCS#12. An article describing this procedure can be found at the Citrix website. From the directory *server-repository-root/ mysql-test/std_data*, issue the following command:

```
openssl pkcs12 -export -in client-cert.pem -inkey client-key.pem -certfile cacert.pem -out client.pfx
```

2. When asked for an export password, enter the password "pass". The file `client.pfx` will be generated. This file is used in the remainder of the tutorial.

Connecting to the server using a file-based certificate

1. You will use PFX file, `client.pfx` you created in the previous step to authenticate the client. The following example demonstrates how to connect using the `SSL Mode`, `CertificateFile` and `CertificatePassword` connection string options:

```
using (MySqlConnection connection = new MySqlConnection(
  "database=test;user=sslclient;" +
  "CertificateFile=H:\\git\\mysql-trunk\\mysql-test\\std_data\\client.pfx" +
  "CertificatePassword=pass;" +
  "SSL Mode=Required "))
{
    connection.Open();
}
```

The path to the certificate file will need to be changed to reflect your individual installation.

Connecting to the server using a store-based certificate

1. The first step is to import the PFX file, `client.pfx`, into the Personal Store. Double-click the file in Windows explorer. This launches the Certificate Import Wizard.

2. Follow the steps dictated by the wizard, and when prompted for the password for the PFX file, enter "pass".

3. Click **Finish** to close the wizard and import the certificate into the personal store.

Examine certificates in the Personal Store

1. Start the Microsoft Management Console by entering `mmc.exe` at a command prompt.

2. Select **File**, **Add/Remove snap-in**. Click **Add**. Select **Certificates** from the list of available snap-ins in the dialog.

3. Click **Add** button in the dialog, and select the **My user account** radio button. This is used for personal certificates.

4. Click the **Finish** button.

5. Click **OK** to close the Add/Remove Snap-in dialog.

6. You will now have **Certificates – Current User** displayed in the left panel of the Microsoft Management Console. Expand the Certificates - Current User tree item and select **Personal**, **Certificates**. The

right-hand panel will display a certificate issued to MySQL. This is the certificate that was previously imported. Double-click the certificate to display its details.

7. After you have imported the certificate to the Personal Store, you can use a more succint connection string to connect to the database, as illustrated by the following code:

```
using (MySqlConnection connection = new MySqlConnection(
    "database=test;user=sslclient;" +
    "Certificate Store Location=CurrentUser;" +
    "SSL Mode=Required"))
{
    connection.Open();
}
```

Certificate Thumbprint Parameter

If you have a large number of certificates in your store, and many have the same Issuer, this can be a source of confusion and result in the wrong certificate being used. To alleviate this situation, there is an optional Certificate Thumbprint parameter that can additionally be specified as part of the connection string. As mentioned before, you can double-click a certificate in the Microsoft Management Console to display the certificate's details. When the Certificate dialog is displayed click the **Details** tab and scroll down to see the thumbprint. The thumbprint will typically be a number such as `#47 94 36 00 9a 40 f3 01 7a 14 5c f8 47 9e 76 94 d7 aa de f0`. This thumbprint can be used in the connection string, as the following code illustrates:

```
using (MySqlConnection connection = new MySqlConnection(
      "database=test;user=sslclient;" +
      "Certificate Store Location=CurrentUser;" +
      "Certificate Thumbprint=479436009a40f3017a145cf8479e7694d7aadef0;"+
      "SSL Mode=Required"))
{
    connection.Open();
}
```

Spaces in the thumbprint parameter are optional and the value is case-insensitive.

4.9 Tutorial: Using MySqlScript

This tutorial teaches you how to use the `MySqlScript` class. This class enables you to execute a series of statements. Depending on the circumstances, this can be more convenient than using the `MySqlCommand` approach.

Further details of the `MySqlScript` class can be found in the reference documentation supplied with MySQL Connector/Net.

To run the example programs in this tutorial, set up a simple test database and table using the `mysql` Command-Line Client or MySQL Workbench. Commands for the `mysql` Command-Line Client are given here:

```
CREATE DATABASE TestDB;
USE TestDB;
CREATE TABLE TestTable (id INT NOT NULL PRIMARY KEY
  AUTO_INCREMENT, name VARCHAR(100));
```

The main method of the `MySqlScript` class is the `Execute` method. This method causes the script (sequence of statements) assigned to the Query property of the MySqlScript object to be executed.

The Query property can be set through the `MySqlScript` constructor or by using the Query property. `Execute` returns the number of statements executed.

The `MySqlScript` object will execute the specified script on the connection set using the Connection property. Again, this property can be set directly or through the `MySqlScript` constructor. The following code snippets illustrate this:

```
string sql = "SELECT * FROM TestTable";
...
MySqlScript script = new MySqlScript(conn, sql);
...
MySqlScript script = new MySqlScript();
script.Query = sql;
script.Connection = conn;
...
script.Execute();
```

The MySqlScript class has several events associated with it. There are:

1. Error - generated if an error occurs.

2. ScriptCompleted - generated when the script successfully completes execution.

3. StatementExecuted - generated after each statement is executed.

It is possible to assign event handlers to each of these events. These user-provided routines are called back when the connected event occurs. The following code shows how the event handlers are set up.

```
script.Error += new MySqlScriptErrorEventHandler(script_Error);
script.ScriptCompleted += new EventHandler(script_ScriptCompleted);
script.StatementExecuted += new MySqlStatementExecutedEventHandler(script_StatementExecuted);
```

In VisualStudio, you can save typing by using tab completion to fill out stub routines. Start by typing, for example, "script.Error +=". Then press **TAB**, and then press **TAB** again. The assignment is completed, and a stub event handler created. A complete working example is shown below:

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

using System.Data;
using MySql.Data;
using MySql.Data.MySqlClient;

namespace MySqlScriptTest
{
    class Program
    {
        static void Main(string[] args)
        {
            string connStr = "server=localhost;user=root;database=TestDB;port=3306;password=******;";
            MySqlConnection conn = new MySqlConnection(connStr);

            try
            {
                Console.WriteLine("Connecting to MySQL...");
                conn.Open();

                string sql = "INSERT INTO TestTable(name) VALUES ('Superman');" +
                             "INSERT INTO TestTable(name) VALUES ('Batman');" +
```

```
                           "INSERT INTO TestTable(name) VALUES ('Wolverine');" +
                           "INSERT INTO TestTable(name) VALUES ('Storm');";

            MySqlScript script = new MySqlScript(conn, sql);

            script.Error += new MySqlScriptErrorEventHandler(script_Error);
            script.ScriptCompleted += new EventHandler(script_ScriptCompleted);
            script.StatementExecuted += new MySqlStatementExecutedEventHandler(script_StatementExecuted

            int count = script.Execute();

            Console.WriteLine("Executed " + count + " statement(s).");
            Console.WriteLine("Delimiter: " + script.Delimiter);
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }

        conn.Close();
        Console.WriteLine("Done.");
    }

    static void script_StatementExecuted(object sender, MySqlScriptEventArgs args)
    {
        Console.WriteLine("script_StatementExecuted");
    }

    static void script_ScriptCompleted(object sender, EventArgs e)
    {
        /// EventArgs e will be EventArgs.Empty for this method
        Console.WriteLine("script_ScriptCompleted!");
    }

    static void script_Error(Object sender, MySqlScriptErrorEventArgs args)
    {
        Console.WriteLine("script_Error: " + args.Exception.ToString());
    }
  }
}
```

In the `script_ScriptCompleted` event handler, the `EventArgs` parameter `e` will be `EventArgs.Empty`. In the case of the `ScriptCompleted` event there is no additional data to be obtained, which is why the event object is `EventArgs.Empty`.

4.9.1 Using Delimiters with MySqlScript

Depending on the nature of the script, you may need control of the delimiter used to separate the statements that will make up a script. The most common example of this is where you have a multi-statement stored routine as part of your script. In this case if the default delimiter of ";" is used you will get an error when you attempt to execute the script. For example, consider the following stored routine:

```
CREATE PROCEDURE test_routine()
BEGIN
    SELECT name FROM TestTable ORDER BY name;
    SELECT COUNT(name) FROM TestTable;
END
```

This routine actually needs to be executed on the MySQL Server as a single statement. However, with the default delimiter of ";", the `MySqlScript` class would interpret the above as two statements, the first being:

65

```
CREATE PROCEDURE test_routine()
BEGIN
    SELECT name FROM TestTable ORDER BY name;
```

Executing this as a statement would generate an error. To solve this problem `MySqlScript` supports the ability to set a different delimiter. This is achieved through the Delimiter property. For example, you could set the delimiter to "??", in which case the above stored routine would no longer generate an error when executed. Multiple statements can be delimited in the script, so for example, you could have a three statement script such as:

```
string sql = "DROP PROCEDURE IF EXISTS test_routine??" +
             "CREATE PROCEDURE test_routine() " +
             "BEGIN " +
             "SELECT name FROM TestTable ORDER BY name;" +
             "SELECT COUNT(name) FROM TestTable;" +
             "END??" +
             "CALL test_routine()";
```

You can change the delimiter back at any point by setting the Delimiter property. The following code shows a complete working example:

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

using MySql.Data;
using MySql.Data.MySqlClient;

namespace ConsoleApplication8
{
    class Program
    {
        static void Main(string[] args)
        {
            string connStr = "server=localhost;user=root;database=TestDB;port=3306;password=******;";
            MySqlConnection conn = new MySqlConnection(connStr);

            try
            {
                Console.WriteLine("Connecting to MySQL...");
                conn.Open();

                string sql =    "DROP PROCEDURE IF EXISTS test_routine??" +
                                "CREATE PROCEDURE test_routine() " +
                                "BEGIN " +
                                "SELECT name FROM TestTable ORDER BY name;" +
                                "SELECT COUNT(name) FROM TestTable;" +
                                "END??" +
                                "CALL test_routine()";

                MySqlScript script = new MySqlScript(conn);

                script.Query = sql;
                script.Delimiter = "??";
                int count = script.Execute();
                Console.WriteLine("Executed " + count + " statement(s)");
                script.Delimiter = ";";
                Console.WriteLine("Delimiter: " + script.Delimiter);
                Console.WriteLine("Query: " + script.Query);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.ToString());
```

```
            }

            conn.Close();
            Console.WriteLine("Done.");
        }
    }
}
```

4.10 Tutorial: Generating MySQL DDL from an Entity Framework Model

In this tutorial, you will learn how to create MySQL DDL from an Entity Framework model. Use Visual Studio 2010 and MySQL Connector/Net 6.3 to carry out this tutorial.

1. Create a new console application in Visual Studio 2010.

2. Using the **Solution Explorer**, add a reference to `MySql.Data.Entity`.

3. From the **Solution Explorer** select **Add**, **New Item**. In the **Add New Item** dialog select **Online Templates**. Select **ADO.NET Entity Data Model** and click **Add**. The **Entity Data Model** dialog will be displayed.

4. In the **Entity Data Model** dialog select **Empty Model**. Click **Finish**. A blank model will be created.

5. Create a simple model. A single Entity will do for the purposes of this tutorial.

6. In the **Properties** panel select **ConceptualEntityModel** from the drop-down listbox.

7. In the **Properties** panel, locate the **DDL Generation Template** in the category **Database Script Generation**.

8. For the **DDL Generation** property select **SSDLToMySQL.tt(VS)** from the drop-down listbox.

9. Save the solution.

10. Right-click an empty space in the model design area. The context-sensitive menu will be displayed.

11. From the context-sensitive menu select **Generate Database from Model**. The **Generate Database Wizard** dialog will be displayed.

12. In the **Generate Database Wizard** dialog select an existing connection, or create a new connection to a server. Select an appropriate radio button to show or hide sensitive data. For the purposes of this tutorial you can select **Yes** (although you might skip this for commercial applications).

13. Click **Next**. MySQL compatible DDL code will be generated. Click **Finish** to exit the wizard.

You have seen how to create MySQL DDL code from an Entity Framework model.

Chapter 5 Connector/Net Programming

Table of Contents

Connector/Net comprises several classes that are used to connect to the database, execute queries and statements, and manage query results.

The following are the major classes of Connector/Net:

- `MySqlCommand`: Represents an SQL statement to execute against a MySQL database.

- `MySqlCommandBuilder`: Automatically generates single-table commands used to reconcile changes made to a DataSet with the associated MySQL database.

- `MySqlConnection`: Represents an open connection to a MySQL Server database.

- `MySqlDataAdapter`: Represents a set of data commands and a database connection that are used to fill a data set and update a MySQL database.

- `MySqlDataReader`: Provides a means of reading a forward-only stream of rows from a MySQL database.

- `MySqlException`: The exception that is thrown when MySQL returns an error.

- `MySqlHelper`: Helper class that makes it easier to work with the provider.

- `MySqlTransaction`: Represents an SQL transaction to be made in a MySQL database.

In the following sections, you will learn about some common use cases for Connector/Net, including BLOB handling, date handling, and using Connector/Net with common tools such as Crystal Reports.

5.1 Connecting to MySQL Using Connector/Net

All interaction between a .NET application and the MySQL server is routed through a `MySqlConnection` object. Before your application can interact with the server, it must instantiate, configure, and open a `MySqlConnection` object.

Even when using the `MySqlHelper` class, a `MySqlConnection` object is created by the helper class.

This section describes how to connect to MySQL using the `MySqlConnection` object.

5.2 Creating a Connector/Net Connection String

The `MySqlConnection` object is configured using a connection string. A connection string contains several key/value pairs, separated by semicolons. In each key/value pair, the option name and its corresponding value are joined by an equal sign. For the list of option names to use in the connection string, see Chapter 6, *Connector/Net Connection String Options Reference*.

The following is a sample connection string:

```
Server=127.0.0.1;Uid=root;Pwd=12345;Database=test;
```

In this example, the `MySqlConnection` object is configured to connect to a MySQL server at `127.0.0.1`, with a user name of `root` and a password of `12345`. The default database for all statements will be the `test` database.

> **Note**
>
> Using the '`@`' symbol for parameters is now the preferred approach, although the old pattern of using '`?`' is still supported. To avoid conflicts when using the '`@`' symbol in combination with user variables, see the `Allow User Variables` connection string option in Chapter 6, *Connector/Net Connection String Options Reference*. The `Old Syntax` connection string option has now been deprecated.

5.2.1 Opening a Connection

Once you have created a connection string it can be used to open a connection to the MySQL server.

The following code is used to create a `MySqlConnection` object, assign the connection string, and open the connection.

Connector/Net can also connect using the native Windows authentication plugin. See Section 5.5, "Using the Windows Native Authentication Plugin" for details.

You can further extend the authentication mechanism by writing your own authentication plugin. See Section 5.6, "Writing a Custom Authentication Plugin" for details.

Visual Basic Example

```vb
Dim conn As New MySql.Data.MySqlClient.MySqlConnection
Dim myConnectionString as String

myConnectionString = "server=127.0.0.1;" _
            & "uid=root;" _
            & "pwd=12345;" _
            & "database=test;"

Try
  conn.ConnectionString = myConnectionString
  conn.Open()

Catch ex As MySql.Data.MySqlClient.MySqlException
  MessageBox.Show(ex.Message)
End Try
```

C# Example

```csharp
MySql.Data.MySqlClient.MySqlConnection conn;
string myConnectionString;

myConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn = new MySql.Data.MySqlClient.MySqlConnection();
    conn.ConnectionString = myConnectionString;
    conn.Open();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message);
}
```

You can also pass the connection string to the constructor of the `MySqlConnection` class:

Visual Basic Example

```vb
Dim myConnectionString as String

myConnectionString = "server=127.0.0.1;" _
            & "uid=root;" _
            & "pwd=12345;" _
            & "database=test;"

Try
    Dim conn As New MySql.Data.MySqlClient.MySqlConnection(myConnectionString)
    conn.Open()
Catch ex As MySql.Data.MySqlClient.MySqlException
  MessageBox.Show(ex.Message)
```

```
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
string myConnectionString;

myConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn = new MySql.Data.MySqlClient.MySqlConnection(myConnectionString);
    conn.Open();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message);
}
```

Once the connection is open it can be used by the other Connector/Net classes to communicate with the MySQL server.

5.2.2 Handling Connection Errors

Because connecting to an external server is unpredictable, it is important to add error handling to your .NET application. When there is an error connecting, the MySqlConnection class will return a MySqlException object. This object has two properties that are of interest when handling errors:

- Message: A message that describes the current exception.

- Number: The MySQL error number.

When handling errors, you can your application's response based on the error number. The two most common error numbers when connecting are as follows:

- 0: Cannot connect to server.

- 1045: Invalid user name and/or password.

The following code shows how to adapt the application's response based on the actual error:

Visual Basic Example

```
Dim myConnectionString as String

myConnectionString = "server=127.0.0.1;" _
          & "uid=root;" _
          & "pwd=12345;" _
          & "database=test;"

Try
    Dim conn As New MySql.Data.MySqlClient.MySqlConnection(myConnectionString)
    conn.Open()
Catch ex As MySql.Data.MySqlClient.MySqlException
    Select Case ex.Number
        Case 0
            MessageBox.Show("Cannot connect to server. Contact administrator")
        Case 1045
            MessageBox.Show("Invalid username/password, please try again")
    End Select
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
string myConnectionString;

myConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    conn = new MySql.Data.MySqlClient.MySqlConnection(myConnectionString);
    conn.Open();
}
    catch (MySql.Data.MySqlClient.MySqlException ex)
{
    switch (ex.Number)
    {
        case 0:
            MessageBox.Show("Cannot connect to server.  Contact administrator");
            break;
        case 1045:
            MessageBox.Show("Invalid username/password, please try again");
            break;
    }
}
```

> **Important**
>
> If you are using multilanguage databases then you must specify the character set
> in the connection string. If you do not specify the character set, the connection
> defaults to the `latin1` charset. You can specify the character set as part of the
> connection string, for example:

```
MySqlConnection myConnection = new MySqlConnection("server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;Charset=latin1;");
```

5.2.3 Using GetSchema on a Connection

The `GetSchema()` method of the connection object can be used to retrieve schema information about the
database currently connected to. The schema information is returned in the form of a `DataTable`. The
schema information is organized into a number of collections. Different forms of the `GetSchema()` method
can be used depending on the information required. There are three forms of the `GetSchema()` method:

- `GetSchema()` - This call will return a list of available collections.

- `GetSchema(String)` - This call returns information about the collection named in the string parameter.
  If the string "MetaDataCollections" is used then a list of all available collections is returned. This is the
  same as calling `GetSchema()` without any parameters.

- `GetSchema(String, String[])` - In this call the first string parameter represents the collection
  name, and the second parameter represents a string array of restriction values. Restriction values
  limit the amount of data that will be returned. Restriction values are explained in more detail in the
  Microsoft .NET documentation.

5.2.3.1 Collections

The collections can be broadly grouped into two types: collections that are common to all data providers,
and collections specific to a particular provider.

Common

The following collections are common to all data providers:

- MetaDataCollections

- DataSourceInformation

- DataTypes

- Restrictions

- ReservedWords

Provider-specific

The following are the collections currently provided by MySQL Connector/Net, in addition to the common collections above:

- Databases

- Tables

- Columns

- Users

- Foreign Keys

- IndexColumns

- Indexes

- Foreign Key Columns

- UDF

- Views

- ViewColumns

- Procedure Parameters

- Procedures

- Triggers

Example Code

A list of available collections can be obtained using the following code:

```
using System;
using System.Data;
using System.Text;
using MySql.Data;
using MySql.Data.MySqlClient;

namespace ConsoleApplication2
{
    class Program
    {

        private static void DisplayData(System.Data.DataTable table)
        {
            foreach (System.Data.DataRow row in table.Rows)
            {
```

```
                foreach (System.Data.DataColumn col in table.Columns)
                {
                    Console.WriteLine("{0} = {1}", col.ColumnName, row[col]);
                }
                Console.WriteLine("============================");
            }
        }

        static void Main(string[] args)
        {

            string connStr = "server=localhost;user=root;database=world;port=3306;password=******;";
            MySqlConnection conn = new MySqlConnection(connStr);

            try
            {
                Console.WriteLine("Connecting to MySQL...");
                conn.Open();

                DataTable table = conn.GetSchema("MetaDataCollections");
                //DataTable table = conn.GetSchema("UDF");
                DisplayData(table);

                conn.Close();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.ToString());
            }
            Console.WriteLine("Done.");
        }
    }
}
```

Further information on the `GetSchema()` method and schema collections can be found in the Microsoft .NET documentation.

5.3 Using MySqlCommand

A `MySqlCommand` has the `CommandText` and `CommandType` properties associated with it. The `CommandText` will be handled differently depending on the setting of `CommandType`. `CommandType` can be one of:

1. Text - An SQL text command (default)

2. StoredProcedure - The name of a Stored Procedure

3. TableDirect - The name of a table (new in Connector/Net 6.2)

The default `CommandType`, `Text`, is used for executing queries and other SQL commands. Some example of this can be found in the following section Section 4.1.2, "The MySqlCommand Object".

If `CommandType` is set to `StoredProcedure`, set `CommandText` to the name of the Stored Procedure to access.

If `CommandType` is set to `TableDirect`, all rows and columns of the named table will be returned when you call one of the Execute methods. In effect, this command performs a `SELECT *` on the table specified. The `CommandText` property is set to the name of the table to query. This is illustrated by the following code snippet:

```
...
MySqlCommand cmd = new MySqlCommand();
```

```
cmd.CommandText = "mytable";
cmd.Connection = someConnection;
cmd.CommandType = CommandType.TableDirect;
MySqlDataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
    Console.WriteLn(reader[0], reader[1]...);
}
...
```

Examples of using the CommandType of StoredProcedure can be found in the section Section 5.10, "Accessing Stored Procedures with Connector/Net".

Commands can have a timeout associated with them. This is useful as you may not want a situation were a command takes up an excessive amount of time. A timeout can be set using the `CommandTimeout` property. The following code snippet sets a timeout of one minute:

```
MySqlCommand cmd = new MySqlCommand();
cmd.CommandTimeout = 60;
```

The default value is 30 seconds. Avoid a value of 0, which indicates an indefinite wait. To change the default command timeout, use the connection string option `Default Command Timeout`.

Prior to MySQL Connector/Net 6.2, `MySqlCommand.CommandTimeout` included user processing time, that is processing time not related to direct use of the connector. Timeout was implemented through a .NET Timer, that triggered after `CommandTimeout` seconds. This timer consumed a thread.

MySQL Connector/Net 6.2 introduced timeouts that are aligned with how Microsoft handles `SqlCommand.CommandTimeout`. This property is the cumulative timeout for all network reads and writes during command execution or processing of the results. A timeout can still occur in the `MySqlReader.Read` method after the first row is returned, and does not include user processing time, only IO operations. The 6.2 implementation uses the underlying stream timeout facility, so is more efficient in that it does not require the additional timer thread as was the case with the previous implementation.

Further details on this can be found in the relevant Microsoft documentation.

5.4 Using Connector/Net with Connection Pooling

The Connector/Net supports connection pooling for better performance and scalability with database-intensive applications. This is enabled by default. You can turn it off or adjust its performance characteristics using the connection string options `Pooling`, `Connection Reset`, `Connection Lifetime`, `Cache Server Properties`, `Max Pool Size` and `Min Pool Size`. See Section 5.2, "Creating a Connector/Net Connection String" for further information.

Connection pooling works by keeping the native connection to the server live when the client disposes of a `MySqlConnection`. Subsequently, if a new `MySqlConnection` object is opened, it will be created from the connection pool, rather than creating a new native connection. This improves performance.

Guidelines

To work as designed, it is best to let the connection pooling system manage all connections. Do not create a globally accessible instance of `MySqlConnection` and then manually open and close it. This interferes with the way the pooling works and can lead to unpredictable results or even exceptions.

One approach that simplifies things is to avoid manually creating a `MySqlConnection` object. Instead use the overloaded methods that take a connection string as an argument. Using this approach, Connector/Net will automatically create, open, close and destroy connections, using the connection pooling system for best performance.

Typed Datasets and the `MembershipProvider` and `RoleProvider` classes use this approach. Most classes that have methods that take a `MySqlConnection` as an argument, also have methods that take a connection string as an argument. This includes `MySqlDataAdapter`.

Instead of manually creating `MySqlCommand` objects, you can use the static methods of the `MySqlHelper` class. These take a connection string as an argument, and they fully support connection pooling.

Resource Usage

Starting with MySQL Connector/Net 6.2, there is a background job that runs every three minutes and removes connections from pool that have been idle (unused) for more than three minutes. The pool cleanup frees resources on both client and server side. This is because on the client side every connection uses a socket, and on the server side every connection uses a socket and a thread.

Prior to this change, connections were never removed from the pool, and the pool always contained the peak number of open connections. For example, a web application that peaked at 1000 concurrent database connections would consume 1000 threads and 1000 open sockets at the server, without ever freeing up those resources from the connection pool. Connections, no matter how old, will not be closed if the number of connections in the pool is less than or equal to the value set by the `Min Pool Size` connection string parameter.

5.5 Using the Windows Native Authentication Plugin

Connector/Net applications can authenticate to a MySQL server using the Windows Native Authentication Plugin as of Connector/Net 6.4.4 and MySQL 5.5.16/5.6.10. Users who have logged in to Windows can connect from MySQL client programs to the server based on the information in their environment without specifying an additional password. For background and usage information about the authentication plugin, see, Windows Pluggable Authentication.

The interface matches the MySql.Data.MySqlClient object. To enable, pass in `Integrated Security` to the connection string with a value of `yes` or `sspi`.

Passing in a user ID is optional. When Windows authentication is set up, a MySQL user is created and configured to be used by Windows authentication. By default, this user ID is named `auth_windows`, but can be defined using a different name. If the default name is used, then passing the user ID to the connection string from Connector/Net is optional, because it will use the `auth_windows` user. Otherwise, the name must be passed to the connection string using the standard user ID element.

5.6 Writing a Custom Authentication Plugin

Advanced users with special security requirements can create their own authentication plugins for Connector/Net applications. You can extend the handshake protocol, adding custom logic. This capability requires Connector/Net 6.6.3 or higher, and MySQL 5.5.16 or higher. For background and usage information about MySQL authentication plugins, see, Authentication Plugins and Writing Authentication Plugins.

To write a custom authentication plugin, you will need a reference to the assembly `MySql.Data.dll`. The classes relevant for writing authentication plugins are available at the namespace `MySql.Data.MySqlClient.Authentication`.

How the Custom Authentication Plugin Works

At some point during handshake, the internal method

```
void Authenticate(bool reset)
```

of `MySqlAuthenticationPlugin` is called. This method in turns calls several overridable methods of the current plugin.

Creating the Authentication Plugin Class

You put the authentication plugin logic inside a new class derived from `MySql.Data.MySqlClient.Authentication.MySqlAuthenticationPlugin`. The following methods are available to be overridden:

```
protected virtual void CheckConstraints()
protected virtual void AuthenticationFailed(Exception ex)
protected virtual void AuthenticationSuccessful()
protected virtual byte[] MoreData(byte[] data)
protected virtual void AuthenticationChange()
public abstract string PluginName { get; }
public virtual string GetUsername()
public virtual object GetPassword()
protected byte[] AuthData;
```

The following is a brief explanation of each one:

```
/// <summary>
/// This method must check authentication method specific constraints in the
environment and throw an Exception
/// if the conditions are not met. The default implementation does nothing.
/// </summary>
protected virtual void CheckConstraints()

/// <summary>
/// This method, called when the authentication failed, provides a chance to
plugins to manage the error
/// the way they consider decide (either showing a message, logging it, etc.).
/// The default implementation wraps the original exception in a MySqlException
with an standard message and rethrows it.
/// </summary>
/// <param name="ex">The exception with extra information on the error.</param>
protected virtual void AuthenticationFailed(Exception ex)

/// <summary>
/// This method is invoked when the authentication phase was successful accepted
by the server.
/// Derived classes must override this if they want to be notified of such
condition.
/// </summary>
/// <remarks>The default implementation does nothing.</remarks>
protected virtual void AuthenticationSuccessful()

/// <summary>
/// This method provides a chance for the plugin to send more data when the
server requests so during the
/// authentication phase. This method will be called at least once, and more
than one depending upon whether the
/// server response packets have the 0x01 prefix.
/// </summary>
/// <param name="data">The response data from the server, during the
authentication phase the first time is called is null, in
subsequent calls contains the server response.</param>
/// <returns>The data generated by the plugin for server consumption.</returns>
/// <remarks>The default implementation always returns null.</remarks>
protected virtual byte[] MoreData(byte[] data)

/// <summary>
/// The plugin name.
```

```
/// </summary>
public abstract string PluginName { get; }

/// <summary>
/// Gets the user name to send to the server in the authentication phase.
/// </summary>
/// <returns>An string with the user name</returns>
/// <remarks>Default implementation returns the UserId passed from the
connection string.</remarks>
public virtual string GetUsername()

/// <summary>
/// Gets the password to send to the server in the authentication phase. This
can be a string or a
/// </summary>
/// <returns>An object, can be byte[], string or null, with the password.
</returns>
/// <remarks>Default implementation returns null.</remarks>
public virtual object GetPassword()

/// <summary>
/// The authentication data passed when creating the plugin.
/// For example in mysql_native_password this is the seed to encrypt the
password.
/// </summary>
protected byte[] AuthData;
```

Sample Authentication Plugin

Here is an example showing how to create the authentication plugin, then enable it by means of a configuration file. Follow these steps:

1. Create a console app, adding a reference to `MySql.Data.dll`.

2. Design the main program as follows:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using MySql.Data.MySqlClient;

namespace AuthPluginTest
{
  class Program
  {
    static void Main(string[] args)
    {
      // Customize the connection string as necessary.
      MySqlConnection con = new MySqlConnection("server=localhost;
database=test; user id=myuser; password=mypass");
      con.Open();
      con.Close();
    }
  }
}
```

3. Create your plugin class. In this example, we add an "alternative" implementation of the Native password plugin by just using the same code from the original plugin. We name our class `MySqlNativePasswordPlugin2`:

```
using System.IO;
using System;
using System.Text;
using System.Security.Cryptography;
using MySql.Data.MySqlClient.Authentication;
using System.Diagnostics;

namespace AuthPluginTest
{
  public class MySqlNativePasswordPlugin2 : MySqlAuthenticationPlugin
  {
    public override string PluginName
    {
      get { return "mysql_native_password"; }
    }

    public override object GetPassword()
    {
      Debug.WriteLine("Calling MySqlNativePasswordPlugin2.GetPassword");
      return Get411Password(Settings.Password, AuthData);
    }

    /// <summary>
    /// Returns a byte array containing the proper encryption of the
    /// given password/seed according to the new 4.1.1 authentication scheme.
    /// </summary>
    /// <param name="password"></param>
    /// <param name="seed"></param>
    /// <returns></returns>
    private byte[] Get411Password(string password, byte[] seedBytes)
    {
      // if we have no password, then we just return 1 zero byte
      if (password.Length == 0) return new byte[1];

      SHA1 sha = new SHA1CryptoServiceProvider();

      byte[] firstHash = sha.ComputeHash(Encoding.Default.GetBytes(password));
      byte[] secondHash = sha.ComputeHash(firstHash);

      byte[] input = new byte[seedBytes.Length + secondHash.Length];
      Array.Copy(seedBytes, 0, input, 0, seedBytes.Length);
      Array.Copy(secondHash, 0, input, seedBytes.Length, secondHash.Length);
      byte[] thirdHash = sha.ComputeHash(input);

      byte[] finalHash = new byte[thirdHash.Length + 1];
      finalHash[0] = 0x14;
      Array.Copy(thirdHash, 0, finalHash, 1, thirdHash.Length);

      for (int i = 1; i < finalHash.Length; i++)
        finalHash[i] = (byte)(finalHash[i] ^ firstHash[i - 1]);
      return finalHash;
    }
  }
}
```

4. Notice that the plugin implementation just overrides `GetPassword`, and provides an implementation to encrypt the password using the 4.1 protocol. We also put the following line in the `GetPassword` body:

```
Debug.WriteLine("Calling MySqlNativePasswordPlugin2.GetPassword");
```

to provide confirmation that the plugin was effectively used. (You could also put a breakpoint on that method.)

5. Enable the new plugin in the configuration file:

```xml
<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="MySQL" type="MySql.Data.MySqlClient.MySqlConfiguration,
MySql.Data"/>
  </configSections>
  <MySQL>
    <AuthenticationPlugins>
      <add name="mysql_native_password"
type="AuthPluginTest.MySqlNativePasswordPlugin2, AuthPluginTest"></add>
    </AuthenticationPlugins>
  </MySQL>
<startup><supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
</startup></configuration>
```

6. Run the application. In Visual Studio, you will see the message `Calling MySqlNativePasswordPlugin2.GetPassword` in the debug window.

7. Continue enhancing the authentication logic, overriding more methods if you required.

5.7 MySQL Fabric Support

Connector/Net supports MySQL Fabric as a Replication/Load balancing plugin.

> **Note**
>
> This feature was added in MySQL Connector/Net 6.9.4.

The following steps are required to use MySQL Fabric with Connector/Net:

System Requirements

Confirm that you have the required Connector/Net and Fabric versions installed:

• Connector/Net 6.9.4 or newer

• MySQL Fabric 1.5.0 or newer

Set up the MySQL Fabric Plugin

First, add **MySql.Data** and **MySql.Fabric.Plugin** to the project references:

Figure 5.1 MySQL Fabric Project References
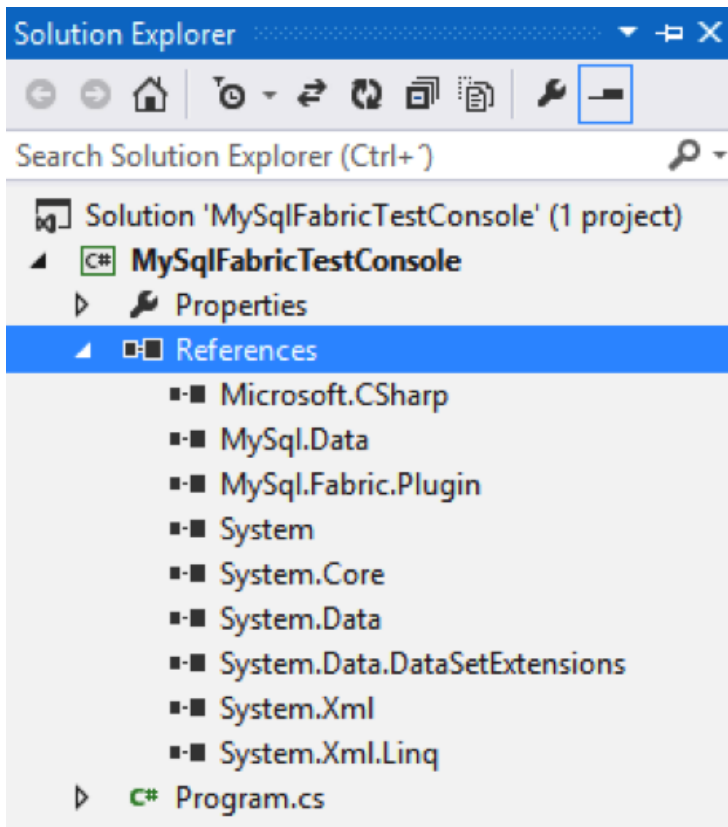
Second, add a configuration section with the Fabric connection to the `App.config` configuration file. For example:

```
<configuration>
 <configSections>
  <section name="MySQL" type="MySql.Data.MySqlClient.MySqlConfiguration, MySql.Data,
    Version=6.9.4.0, Culture=neutral, PublicKeyToken=c5687fc88969c44d"/>
 </configSections>
 <MySQL>
  <Replication>
   <ServerGroups>
    <Group name="Fabric" groupType="MySql.Fabric.FabricServerGroup, MySql.Fabric.Plugin">
     <Servers>
      <Server name="fabric" connectionstring="server=localhost;port=32275;uid=admin;password=adminpass;"/>
     </Servers>
    </Group>
   </ServerGroups>
  </Replication>
 </MySQL>
</configuration>
```

Notice that the Fabric connection is set in the Server node:

```
<Server name="fabric" connectionstring="server=localhost;port=32275;uid=admin;password=adminpass;"/>
```

Connector/Net only supports the MySQL protocol for connecting to MySQL Fabric, so the correct port must be used.

Using MySQL Fabric Groups

The MySQL Fabric group is used with a **MySqlConnection** that contains the server name specified in the `App.config` file, and a username and password for connecting to the servers defined in the group.

A Fabric extension method is used to specify the group and mode:

```
MySqlConnection conn = new MySqlConnection(connectionString);

conn.SetFabricProperties(groupId: "my_group", mode: FabricServerModeEnum.Read_Write);
```

The following example shows how to store and retrieve information in a specific Fabric group:

> **Note**
>
> The initial MySQL Fabric configuration for this example is defined in the MySQL
> Fabric documentation at Example: Fabric and Replication.

```
using System;
using MySql.Data.MySqlClient;
using MySql.Fabric;

namespace FabricTest
{
  class Program
  {
    public const string connectionString = "server=fabric;uid=appuser;password=pass;";

    static void Main(string[] args)
    {
      RunFabricTest();
    }

    static string AddEmployee(MySqlConnection conn, int emp_no, string first_name, string last_name)
    {
      conn.SetFabricProperties(groupId: "my_group", mode: FabricServerModeEnum.Read_Write);

      MySqlCommand cmd = new MySqlCommand("USE employees", conn);
      cmd.ExecuteNonQuery();

      cmd.CommandText = "INSERT INTO employees VALUES (@emp_no, @first_name, @last_name)";
      cmd.Parameters.Add("emp_no", emp_no);
      cmd.Parameters.Add("first_name", first_name);
      cmd.Parameters.Add("last_name", last_name);
      cmd.ExecuteNonQuery();

      cmd.CommandText = "SELECT @@global.gtid_executed";
      cmd.Parameters.Clear();
      using (MySqlDataReader reader = cmd.ExecuteReader())
      {
        while (reader.Read())
        {
          Console.WriteLine("Transactions executed on the master " + reader.GetValue(0));
        }
        return reader.GetString(0);
      }
    }

    static void FindEmployee(MySqlConnection conn, int emp_no, string gtid_executed)
    {
```

```
      conn.SetFabricProperties(groupId: "my_group", mode: FabricServerModeEnum.Read_only);

    MySqlCommand cmd = new MySqlCommand("", conn);
    cmd.CommandText = "SELECT WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS(@gtid_executed, 0)";
    cmd.Parameters.Add("gtid_executed", gtid_executed);
    using (MySqlDataReader reader = cmd.ExecuteReader())
    {
      while (reader.Read())
      {
        Console.WriteLine("Had to synchronize " + reader.GetValue(0) + " transactions.");
      }
    }

    cmd.CommandText = "USE employees";
    cmd.Parameters.Clear();
    cmd.ExecuteNonQuery();

    cmd.CommandText = "SELECT first_name, last_name FROM employees ";
    cmd.CommandText += " WHERE emp_no = @emp_no";
    cmd.Parameters.Clear();
    cmd.Parameters.Add("emp_no", emp_no);
    using (MySqlDataReader reader = cmd.ExecuteReader())
    {
      while (reader.Read())
      {
        object[] values = new object[reader.FieldCount];
        reader.GetValues(values);
        Console.WriteLine("Retrieved {0}", string.Join(",", values));
      }
    }
  }

  static void RunFabricTest()
  {
    using (MySqlConnection conn = new MySqlConnection(connectionString))
    {
      string gtid_executed;
      conn.SetFabricProperties(groupId: "my_group", mode: FabricServerModeEnum.Read_Write);
      conn.Open();

      MySqlCommand cmd = new MySqlCommand("", conn);
      cmd.CommandText = "CREATE DATABASE IF NOT EXISTS employees;";
      cmd.ExecuteNonQuery();

      cmd.CommandText = "USE employees;";
      cmd.ExecuteNonQuery();

      cmd.CommandText = "DROP TABLE IF EXISTS employees;";
      cmd.ExecuteNonQuery();

      cmd.CommandText = "CREATE TABLE employees(";
      cmd.CommandText += "   emp_no INT, ";
      cmd.CommandText += "   first_name CHAR(40), ";
      cmd.CommandText += "   last_name CHAR(40)";
      cmd.CommandText += ");";
      cmd.ExecuteNonQuery();

      gtid_executed = AddEmployee(conn, 12, "John", "Doe");
      FindEmployee(conn, 12, gtid_executed);
    }
  }
}
}
```

Using Ranged Sharding

Sharding with Connector/Net requires you to specify the table, key, and scope for each executed query.

```
MySqlConnection con = new MySqlConnection(connectionString);

con.SetFabricProperties(table: "employees.employees", key: empId.ToString(),
  mode: FabricServerModeEnum.Read_Write, scope: FabricScopeEnum.Local);

MySqlCommand cmd = new MySqlCommand(
  string.Format("insert into employees(emp_no, first_name, last_name) values ({0}, '{1}', '{2}')",
  empId, firstName, lastName), con);
cmd.ExecuteScalar();
```

You can use the following MySQL Fabric configuration to execute the code example:

> **Note**
>
> For related MySQL Fabric documentation, see Sharding Scenario.

```
using System;
using System.Collections.Generic;
using MySql.Data.MySqlClient;
using MySql.Fabric;

namespace FabricTest
{
  class ShardTest
  {
    public const string connectionString = "server=fabric;uid=appuser;password=pass;";

    public static void test_shard_range()
    {
      using (MySqlConnection con = new MySqlConnection(connectionString))
      {
        con.SetFabricProperties(groupId: "group_id-global", mode: FabricServerModeEnum.Read_Write,
          scope: FabricScopeEnum.Global);
        con.Open();

        MySqlCommand cmd = new MySqlCommand("create database if not exists employees", con);
        cmd.ExecuteScalar();

        cmd.CommandText = "use employees";
        cmd.ExecuteScalar();

        cmd.CommandText = "drop table if exists employees";
        cmd.ExecuteScalar();

        cmd.CommandText =
@"create table employees (
    emp_no int,
    first_name char( 40 ),
    last_name char( 40 )
)";
        cmd.ExecuteScalar();

        string gtid = prepare_synchronization(con);

        string[] first_names = { "John", "Buffalo", "Michael", "Kate", "Deep", "Genesis" };
        string[] last_names = { "Doe", "Bill", "Jackson", "Bush", "Purple" };
        List<int> list_emp_no = new List<int>();

        con.SetFabricProperties(scope: FabricScopeEnum.Local);
```

```csharp
      for (int i = 0; i < 10; i++)
      {
        int empId = pick_shard_key();
        list_emp_no.Add(empId);
        add_employee(con, empId, first_names[empId % first_names.Length], last_names[empId % last_names.Leng
      }

      for (int i = 0; i < list_emp_no.Count; i++)
      {
        int empId = list_emp_no[ i ];
        find_employee(con, empId, gtid);
      }
    }
  }

  public static int pick_shard_key()
  {
    Random r = new Random();
    int shard = r.Next(0, 2);
    int shard_range = shard * 10000;
    shard_range = (shard != 0) ? shard_range : shard_range + 1;
    int shift_within_shard = r.Next(0, 99999);
    return shard_range + shift_within_shard;
  }

  public static void add_employee(MySqlConnection con, int empId, string firstName, string lastName, string
  {
    con.SetFabricProperties( table: "employees.employees", key: empId.ToString(), mode: FabricServerModeEnum
    synchronize(con, gtid);
    MySqlCommand cmd = new MySqlCommand(
      string.Format("insert into employees( emp_no, first_name, last_name ) values ( {0}, '{1}', '{2}' )",
      empId, firstName, lastName), con);
    cmd.ExecuteScalar();
  }

  public static void find_employee(MySqlConnection con, int empId, string gtid)
  {
    con.SetFabricProperties(table: "employees.employees", key: empId.ToString(),
      mode: FabricServerModeEnum.Read_only);
    synchronize(con, gtid);
    MySqlCommand cmd = new MySqlCommand(string.Format("
      select first_name, last_name from employees where emp_no = {0}", empId), con);
    using (MySqlDataReader r = cmd.ExecuteReader())
    {
      while (r.Read())
      {
        Console.WriteLine("( {0}, {1} )", r.GetString(0), r.GetString(1));
      }
    }
  }

  public static string prepare_synchronization(MySqlConnection con)
  {
    string gtid_executed = "";
    MySqlCommand cmd = new MySqlCommand("select @@global.gtid_executed", con);
    gtid_executed = ( string )cmd.ExecuteScalar();
    return gtid_executed;
  }

  public static void synchronize(MySqlConnection con, string gtid_executed)
  {
    MySqlCommand cmd = new MySqlCommand( string.Format( "SELECT WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS('{0}', 0)"
      gtid_executed ), con);
    cmd.ExecuteScalar();
  }
}
```

```
}
```

5.8 Using Connector/Net with Table Caching

This feature exists with Connector/Net versions 6.4 and above.

Table caching is a feature that can be used to cache slow-changing datasets on the client side. This is useful for applications that are designed to use readers, but still want to minimize trips to the server for slow-changing tables.

This feature is transparent to the application, and is disabled by default.

Configuration

- To enable table caching, add `'table cache = true'` to the connection string.

- Optionally, specify the `'Default Table Cache Age'` connection string option, which represents the number of seconds a table is cached before the cached data is discarded. The default value is `60`.

- You can turn caching on and off and set caching options at runtime, on a per-command basis.

5.9 Using the Connector/Net with Prepared Statements

Use of prepared statements can provide significant performance improvements on queries that are executed more than once.

> **Note**
>
> Prepared statement support is available with MySQL 4.1 and above.

Prepared execution is faster than direct execution for statements executed more than once, primarily because the query is parsed only once. In the case of direct execution, the query is parsed every time it is executed. Prepared execution also can provide a reduction of network traffic because for each execution of the prepared statement, it is necessary only to send the data for the parameters.

Another advantage of prepared statements is that, with server-side prepared statements enabled, it uses a binary protocol that makes data transfer between client and server more efficient.

> **Note**
>
> Enable server-side prepapared statements (they are disabled by default) by passing in "IgnorePrepare=false" to your connection string.

5.9.1 Preparing Statements in Connector/Net

To prepare a statement, create a command object and set the `.CommandText` property to your query.

After entering your statement, call the `.Prepare` method of the `MySqlCommand` object. After the statement is prepared, add parameters for each of the dynamic elements in the query.

After you enter your query and enter parameters, execute the statement using the `.ExecuteNonQuery()`, `.ExecuteScalar()`, or `.ExecuteReader` methods.

For subsequent executions, you need only modify the values of the parameters and call the execute method again, there is no need to set the `.CommandText` property or redefine the parameters.

Visual Basic Example

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand

conn.ConnectionString = strConnection

Try
   conn.Open()
   cmd.Connection = conn

   cmd.CommandText = "INSERT INTO myTable VALUES(NULL, @number, @text)"
   cmd.Prepare()

   cmd.Parameters.AddWithValue("@number", 1)
   cmd.Parameters.AddWithValue("@text", "One")

   For i = 1 To 1000
       cmd.Parameters("@number").Value = i
       cmd.Parameters("@text").Value = "A string value"

       cmd.ExecuteNonQuery()
     Next
Catch ex As MySqlException
    MessageBox.Show("Error " & ex.Number & " has occurred: " & ex.Message, "Error", MessageBoxButtons.OK, Mess
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();

conn.ConnectionString = strConnection;

try
{
    conn.Open();
    cmd.Connection = conn;

    cmd.CommandText = "INSERT INTO myTable VALUES(NULL, @number, @text)";
    cmd.Prepare();

    cmd.Parameters.AddWithValue("@number", 1);
    cmd.Parameters.AddWithValue("@text", "One");

    for (int i=1; i <= 1000; i++)
    {
        cmd.Parameters["@number"].Value = i;
        cmd.Parameters["@text"].Value = "A string value";

        cmd.ExecuteNonQuery();
    }
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

5.10 Accessing Stored Procedures with Connector/Net

MySQL server version 5 and up supports stored procedures with the SQL 2003 stored procedure syntax.

A stored procedure is a set of SQL statements that is stored in the server. Clients make a single call to the stored procedure, passing parameters that can influence the procedure logic and query conditions, rather than issuing individual hardcoded SQL statements.

Stored procedures can be particularly useful in situations such as the following:

- Stored procedures can act as an API or abstraction layer, allowing multiple client applications to perform the same database operations. The applications can be written in different languages and run on different platforms. The applications do not need to hardcode table and column names, complicated queries, and so on. When you extend and optimize the queries in a stored procedure, all the applications that call the procedure automatically receive the benefits.

- When security is paramount, stored procedures keep applications from directly manipulating tables, or even knowing details such as table and column names. Banks, for example, use stored procedures for all common operations. This provides a consistent and secure environment, and procedures can ensure that each operation is properly logged. In such a setup, applications and users would not get any access to the database tables directly, but can only execute specific stored procedures.

Connector/Net supports the calling of stored procedures through the `MySqlCommand` object. Data can be passed in and out of a MySQL stored procedure through use of the `MySqlCommand.Parameters` collection.

> **Note**
>
> When you call a stored procedure, the command object makes an additional `SELECT` call to determine the parameters of the stored procedure. You must ensure that the user calling the procedure has the `SELECT` privilege on the `mysql.proc` table to enable them to verify the parameters. Failure to do this will result in an error when calling the procedure.

This section will not provide in-depth information on creating Stored Procedures. For such information, please refer to http://dev.mysql.com/doc/mysql/en/stored-routines.html.

A sample application demonstrating how to use stored procedures with Connector/Net can be found in the `Samples` directory of your Connector/Net installation.

5.10.1 Using Stored Routines from Connector/Net

Stored procedures in MySQL can be created using a variety of tools. First, stored procedures can be created using the `mysql` command-line client. Second, stored procedures can be created using MySQL Workbench. Finally, stored procedures can be created using the `.ExecuteNonQuery` method of the `MySqlCommand` object.

Unlike the command-line and GUI clients, you are not required to specify a special delimiter when creating stored procedures in Connector/Net.

To call a stored procedure using Connector/Net, you create a `MySqlCommand` object and pass the stored procedure name as the `.CommandText` property. You then set the `.CommandType` property to `CommandType.StoredProcedure`.

After the stored procedure is named, you create one `MySqlCommand` parameter for every parameter in the stored procedure. `IN` parameters are defined with the parameter name and the object containing the value, `OUT` parameters are defined with the parameter name and the data type that is expected to be returned. All parameters need the parameter direction defined.

After defining the parameters, you call the stored procedure by using the
`MySqlCommand.ExecuteNonQuery()` method.

Once the stored procedure is called, the values of the output parameters can be retrieved by using the
`.Value` property of the `MySqlConnector.Parameters` collection.

> **Note**
>
> When a stored procedure is called using `MySqlCommand.ExecuteReader`, and
> the stored procedure has output parameters, the output parameters are only set
> after the `MySqlDataReader` returned by `ExecuteReader` is closed.

The following C# example code demonstrates the use of stored procedures. It assumes the database
'employees' has already been created:

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

using System.Data;
using MySql.Data;
using MySql.Data.MySqlClient;

namespace UsingStoredRoutines
{
    class Program
    {
        static void Main(string[] args)
        {
            MySqlConnection conn = new MySqlConnection();
            conn.ConnectionString = "server=localhost;user=root;database=employees;port=3306;password=******;"
            MySqlCommand cmd = new MySqlCommand();

            try
            {
                Console.WriteLine("Connecting to MySQL...");
                conn.Open();
                cmd.Connection = conn;
                cmd.CommandText = "DROP PROCEDURE IF EXISTS add_emp";
                cmd.ExecuteNonQuery();
                cmd.CommandText = "DROP TABLE IF EXISTS emp";
                cmd.ExecuteNonQuery();
                cmd.CommandText = "CREATE TABLE emp (empno INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY, f
                cmd.ExecuteNonQuery();

                cmd.CommandText = "CREATE PROCEDURE add_emp(" +
                                   "IN fname VARCHAR(20), IN lname VARCHAR(20), IN bday DATETIME, OUT empno INT
                                   "BEGIN INSERT INTO emp(first_name, last_name, birthdate) " +
                                   "VALUES(fname, lname, DATE(bday)); SET empno = LAST_INSERT_ID(); END";

                cmd.ExecuteNonQuery();
            }
            catch (MySqlException ex)
            {
                Console.WriteLine ("Error " + ex.Number + " has occurred: " + ex.Message);
            }
            conn.Close();
            Console.WriteLine("Connection closed.");
            try
            {
                Console.WriteLine("Connecting to MySQL...");
                conn.Open();
                cmd.Connection = conn;
```

```
                cmd.CommandText = "add_emp";
                cmd.CommandType = CommandType.StoredProcedure;

                cmd.Parameters.AddWithValue("@lname", "Jones");
                cmd.Parameters["@lname"].Direction = ParameterDirection.Input;

                cmd.Parameters.AddWithValue("@fname", "Tom");
                cmd.Parameters["@fname"].Direction = ParameterDirection.Input;

                cmd.Parameters.AddWithValue("@bday", "1940-06-07");
                cmd.Parameters["@bday"].Direction = ParameterDirection.Input;

                cmd.Parameters.AddWithValue("@empno", MySqlDbType.Int32);
                cmd.Parameters["@empno"].Direction = ParameterDirection.Output;

                cmd.ExecuteNonQuery();

                Console.WriteLine("Employee number: "+cmd.Parameters["@empno"].Value);
                Console.WriteLine("Birthday: " + cmd.Parameters["@bday"].Value);
            }
            catch (MySql.Data.MySqlClient.MySqlException ex)
            {
                Console.WriteLine("Error " + ex.Number + " has occurred: " + ex.Message);
            }
            conn.Close();
            Console.WriteLine("Done.");
        }
    }
}
```

The following code shows the same application in Visual Basic:

```
Imports System
Imports System.Collections.Generic
Imports System.Linq
Imports System.Text

Imports System.Data
Imports MySql.Data
Imports MySql.Data.MySqlClient

Module Module1

    Sub Main()
        Dim conn As New MySqlConnection()
        conn.ConnectionString = "server=localhost;user=root;database=world;port=3306;password=******;"
        Dim cmd As New MySqlCommand()

        Try
            Console.WriteLine("Connecting to MySQL...")
            conn.Open()
            cmd.Connection = conn
            cmd.CommandText = "DROP PROCEDURE IF EXISTS add_emp"
            cmd.ExecuteNonQuery()
            cmd.CommandText = "DROP TABLE IF EXISTS emp"
            cmd.ExecuteNonQuery()
            cmd.CommandText = "CREATE TABLE emp (empno INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY, f
            cmd.ExecuteNonQuery()

            cmd.CommandText = "CREATE PROCEDURE add_emp(" & "IN fname VARCHAR(20), IN lname VARCHAR(20), I

            cmd.ExecuteNonQuery()
        Catch ex As MySqlException
            Console.WriteLine(("Error " & ex.Number & " has occurred: ") + ex.Message)
```

```
            End Try
            conn.Close()
            Console.WriteLine("Connection closed.")
            Try
                Console.WriteLine("Connecting to MySQL...")
                conn.Open()
                cmd.Connection = conn

                cmd.CommandText = "add_emp"
                cmd.CommandType = CommandType.StoredProcedure

                cmd.Parameters.AddWithValue("@lname", "Jones")
                cmd.Parameters("@lname").Direction = ParameterDirection.Input

                cmd.Parameters.AddWithValue("@fname", "Tom")
                cmd.Parameters("@fname").Direction = ParameterDirection.Input

                cmd.Parameters.AddWithValue("@bday", "1940-06-07")
                cmd.Parameters("@bday").Direction = ParameterDirection.Input

                cmd.Parameters.AddWithValue("@empno", MySqlDbType.Int32)
                cmd.Parameters("@empno").Direction = ParameterDirection.Output

                cmd.ExecuteNonQuery()

                Console.WriteLine("Employee number: " & cmd.Parameters("@empno").Value)
                Console.WriteLine("Birthday: " & cmd.Parameters("@bday").Value)
            Catch ex As MySql.Data.MySqlClient.MySqlException
                Console.WriteLine(("Error " & ex.Number & " has occurred: ") + ex.Message)
            End Try
            conn.Close()
            Console.WriteLine("Done.")

        End Sub

End Module
```

5.11 Handling BLOB Data With Connector/Net

One common use for MySQL is the storage of binary data in BLOB columns. MySQL supports four different BLOB data types: TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB, all described in The BLOB and TEXT Types and Data Type Storage Requirements.

Data stored in a BLOB column can be accessed using Connector/Net and manipulated using client-side code. There are no special requirements for using Connector/Net with BLOB data.

Simple code examples will be presented within this section, and a full sample application can be found in the Samples directory of the Connector/Net installation.

5.11.1 Preparing the MySQL Server

The first step is using MySQL with BLOB data is to configure the server. Let's start by creating a table to be accessed. In my file tables, I usually have four columns: an AUTO_INCREMENT column of appropriate size (UNSIGNED SMALLINT) to serve as a primary key to identify the file, a VARCHAR column that stores the file name, an UNSIGNED MEDIUMINT column that stores the size of the file, and a MEDIUMBLOB column that stores the file itself. For this example, I will use the following table definition:

```
CREATE TABLE file(
file_id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
file_name VARCHAR(64) NOT NULL,
```

```
file_size MEDIUMINT UNSIGNED NOT NULL,
file MEDIUMBLOB NOT NULL);
```

After creating a table, you might need to modify the `max_allowed_packet` system variable. This variable determines how large of a packet (that is, a single row) can be sent to the MySQL server. By default, the server only accepts a maximum size of 1MB from the client application. If you intend to exceed 1MB in your file transfers, increase this number.

The `max_allowed_packet` option can be modified using the MySQL Workbench **Server Administration** screen. Adjust the Maximum permitted option in the **Data / Memory size** section of the Networking tab to an appropriate setting. After adjusting the value, click the **Apply** button and restart the server using the `Startup / Shutdown` screen of MySQL Workbench. You can also adjust this value directly in the `my.cnf` file (add a line that reads `max_allowed_packet=xxM`), or use the `SET max_allowed_packet=xxM;` syntax from within MySQL.

Try to be conservative when setting `max_allowed_packet`, as transfers of BLOB data can take some time to complete. Try to set a value that will be adequate for your intended use and increase the value if necessary.

5.11.2 Writing a File to the Database

To write a file to a database, we need to convert the file to a byte array, then use the byte array as a parameter to an `INSERT` query.

The following code opens a file using a FileStream object, reads it into a byte array, and inserts it into the `file` table:

Visual Basic Example

```vbnet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand

Dim SQL As String

Dim FileSize As UInt32
Dim rawData() As Byte
Dim fs As FileStream

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"

Try
    fs = New FileStream("c:\image.png", FileMode.Open, FileAccess.Read)
    FileSize = fs.Length

    rawData = New Byte(FileSize) {}
    fs.Read(rawData, 0, FileSize)
    fs.Close()

    conn.Open()

    SQL = "INSERT INTO file VALUES(NULL, @FileName, @FileSize, @File)"

    cmd.Connection = conn
    cmd.CommandText = SQL
    cmd.Parameters.AddWithValue("@FileName", strFileName)
    cmd.Parameters.AddWithValue("@FileSize", FileSize)
    cmd.Parameters.AddWithValue("@File", rawData)
```

```
    cmd.ExecuteNonQuery()

    MessageBox.Show("File Inserted into database successfully!", _
    "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk)

    conn.Close()
Catch ex As Exception
    MessageBox.Show("There was an error: " & ex.Message, "Error", _
        MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();

string SQL;
UInt32 FileSize;
byte[] rawData;
FileStream fs;

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    fs = new FileStream(@"c:\image.png", FileMode.Open, FileAccess.Read);
    FileSize = fs.Length;

    rawData = new byte[FileSize];
    fs.Read(rawData, 0, FileSize);
    fs.Close();

    conn.Open();

    SQL = "INSERT INTO file VALUES(NULL, @FileName, @FileSize, @File)";

    cmd.Connection = conn;
    cmd.CommandText = SQL;
    cmd.Parameters.AddWithValue("@FileName", strFileName);
    cmd.Parameters.AddWithValue("@FileSize", FileSize);
    cmd.Parameters.AddWithValue("@File", rawData);

    cmd.ExecuteNonQuery();

    MessageBox.Show("File Inserted into database successfully!",
        "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

    conn.Close();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

The `Read` method of the `FileStream` object is used to load the file into a byte array which is sized according to the `Length` property of the `FileStream` object.

After assigning the byte array as a parameter of the `MySqlCommand` object, the `ExecuteNonQuery` method is called and the `BLOB` is inserted into the `file` table.

5.11.3 Reading a BLOB from the Database to a File on Disk

Once a file is loaded into the `file` table, we can use the `MySqlDataReader` class to retrieve it.

The following code retrieves a row from the `file` table, then loads the data into a `FileStream` object to be written to disk:

Visual Basic Example

```
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myData As MySqlDataReader
Dim SQL As String
Dim rawData() As Byte
Dim FileSize As UInt32
Dim fs As FileStream

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"

SQL = "SELECT file_name, file_size, file FROM file"

Try
    conn.Open()

    cmd.Connection = conn
    cmd.CommandText = SQL

    myData = cmd.ExecuteReader

    If Not myData.HasRows Then Throw New Exception("There are no BLOBs to save")

    myData.Read()

    FileSize = myData.GetUInt32(myData.GetOrdinal("file_size"))
    rawData = New Byte(FileSize) {}

    myData.GetBytes(myData.GetOrdinal("file"), 0, rawData, 0, FileSize)

    fs = New FileStream("C:\newfile.png", FileMode.OpenOrCreate, FileAccess.Write)
    fs.Write(rawData, 0, FileSize)
    fs.Close()

    MessageBox.Show("File successfully written to disk!", "Success!", MessageBoxButtons.OK, MessageBoxIcon

    myData.Close()
    conn.Close()
Catch ex As Exception
    MessageBox.Show("There was an error: " & ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Err
End Try
```

C# Example

```
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
MySql.Data.MySqlClient.MySqlDataReader myData;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
```

```
string SQL;
UInt32 FileSize;
byte[] rawData;
FileStream fs;

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

SQL = "SELECT file_name, file_size, file FROM file";

try
{
    conn.Open();

    cmd.Connection = conn;
    cmd.CommandText = SQL;

    myData = cmd.ExecuteReader();

    if (! myData.HasRows)
        throw new Exception("There are no BLOBs to save");

    myData.Read();

    FileSize = myData.GetUInt32(myData.GetOrdinal("file_size"));
    rawData = new byte[FileSize];

    myData.GetBytes(myData.GetOrdinal("file"), 0, rawData, 0, (int)FileSize);

    fs = new FileStream(@"C:\newfile.png", FileMode.OpenOrCreate, FileAccess.Write);
    fs.Write(rawData, 0, (int)FileSize);
    fs.Close();

    MessageBox.Show("File successfully written to disk!",
        "Success!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

    myData.Close();
    conn.Close();
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show("Error " + ex.Number + " has occurred: " + ex.Message,
        "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

After connecting, the contents of the `file` table are loaded into a `MySqlDataReader` object. The `GetBytes` method of the `MySqlDataReader` is used to load the `BLOB` into a byte array, which is then written to disk using a `FileStream` object.

The `GetOrdinal` method of the MySqlDataReader can be used to determine the integer index of a named column. Use of the GetOrdinal method prevents errors if the column order of the `SELECT` query is changed.

5.12 Asynchronous methods

The Task-based Asynchronous Pattern (TAP) is a pattern for asynchrony in the .NET Framework. It is based on the Task and Task<TResult> types in the System.Threading.Tasks namespace, which are used to represent arbitrary asynchronous operations.

Async-Await are new keywords introduced to work with the TAP. The **Async modifier** is used to specify that a method, lambda expression, or anonymous method is asynchronous. The **Await** operator is applied to a task in an asynchronous method to suspend the execution of the method until the awaited task completes.

Requirements

- **Async-Await** support requires .NET Framework 4.5 or greater
- **TAP** support requires .NET Framework 4.0 or greater
- Connector/Net 6.9 or greater

Methods

These methods can be used with either TAP or Async-Await.

- Namespace MySql.Data.Entity
  - Class EFMySqlCommand
    - Task PrepareAsync()
    - Task PrepareAsync(CancellationToken)
- Namespace MySql.Data
  - Class MySqlBulkLoader
    - Task<int> LoadAsync()
    - Task<int> LoadAsync(CancellationToken)
  - Class MySqlConnection
    - Task<MySqlTransaction> BeginTransactionAsync()
    - Task<MySqlTransaction> BeginTransactionAsync (CancellationToken)
    - Task<MySqlTransaction> BeginTransactionAsync(IsolationLevel)
    - Task<MySqlTransaction> BeginTransactionAsync (IsolationLevel , CancellationToken)
    - Task ChangeDatabaseAsync(string)
    - Task ChangeDatabaseAsync(string, CancellationToken)
    - Task CloseAsync()
    - Task CloseAsync(CancellationToken)
    - Task ClearPoolAsync(MySqlConnection)
    - Task ClearPoolAsync(MySqlConnection, CancellationToken)
    - Task ClearAllPoolsAsync()
    - Task ClearAllPoolsAsync(CancellationToken)
    - Task<MySqlSchemaCollection> GetSchemaCollection(string, string[])
    - Task<MySqlSchemaCollection> GetSchemaCollection(string, string[], CancellationToken)
  - Class MySqlDataAdapter

- Task<int> FillAsync(DataSet)

- Task<int> FillAsync(DataSet, CancellationToken)

- Task<int> FillAsync(DataTable)

- Task<int> FillAsync(DataTable, CancellationToken)

- Task<int> FillAsync(DataSet, string)

- Task<int> FillAsync(DataSet, string, CancellationToken)

- Task<int> FillAsync(DataTable, IDataReader)

- Task<int> FillAsync(DataTable, IDataReader, CancellationToken)

- Task<int> FillAsync(DataTable, IDbCommand, CommandBehavior)

- Task<int> FillAsync(DataTable, IDbCommand, CommandBehavior, CancellationToken)

- Task<int> FillAsync(int, int, params DataTable[])

- Task<int> FillAsync(int, int, params DataTable[], CancellationToken)

- Task<int> FillAsync(DataSet, int, int, string)

- Task<int> FillAsync(DataSet, int, int, string, CancellationToken)

- Task<int> FillAsync(DataSet, string, IDataReader, int, int)

- Task<int> FillAsync(DataSet, string, IDataReader, int, int, CancellationToken)

- Task<int> FillAsync(DataTable[], int, int, IDbCommand, CommandBehavior)

- Task<int> FillAsync(DataTable[], int, int, IDbCommand, CommandBehavior, CancellationToken)

- Task<int> FillAsync(DataSet, int, int, string, IDbCommand, CommandBehavior)

- Task<int> FillAsync(DataSet, int, int, string, IDbCommand, CommandBehavior, CancellationToken)

- Task<DataTable[]> FillSchemaAsync(DataSet, SchemaType)

- Task<DataTable[]> FillSchemaAsync(DataSet, SchemaType, CancellationToken)

- Task<DataTable[]> FillSchemaAsync(DataSet, SchemaType, string)

- Task<DataTable[]> FillSchemaAsync(DataSet, SchemaType, string, CancellationToken)

- Task<DataTable[]> FillSchemaAsync(DataSet, SchemaType, string, IDataReader)

- Task<DataTable[]> FillSchemaAsync(DataSet, SchemaType, string, IDataReader, CancellationToken)

- Task<DataTable[]> FillSchemaAsync(DataSet, SchemaType, IDbCommand, string, CommandBehavior)

- Task<DataTable[]> FillSchemaAsync(DataSet, SchemaType, IDbCommand, string, CommandBehavior, CancellationToken)

- Task<DataTable> FillSchemaAsync(DataTable, SchemaType)

- Task<DataTable> FillSchemaAsync(DataTable, SchemaType, CancellationToken)

- Task<DataTable> FillSchemaAsync(DataTable, SchemaType, IDataReader)

- Task<DataTable> FillSchemaAsync(DataTable, SchemaType, IDataReader, CancellationToken)

- Task<DataTable> FillSchemaAsync(DataTable, SchemaType, IDbCommand, CommandBehavior)

- Task<DataTable> FillSchemaAsync(DataTable, SchemaType, IDbCommand, CommandBehavior, CancellationToken)

- Task<int> UpdateAsync(DataRow[])

- Task<int> UpdateAsync(DataRow[], CancellationToken)

- Task<int> UpdateAsync(DataSet)

- Task<int> UpdateAsync(DataSet, CancellationToken)

- Task<int> UpdateAsync(DataTable)

- Task<int> UpdateAsync(DataTable, CancellationToken)

- Task<int> UpdateAsync(DataRow[], DataTableMapping, CancellationToken)

- Task<int> UpdateAsync(DataSet, string)

- Task<int> UpdateAsync(DataSet, string, CancellationToken)

- Class MySqlHelper

  - Task<DataRow> ExecuteDataRowAsync(string, string, params MySqlParameter[])

  - Task<DataRow> ExecuteDataRowAsync(string, string, CancellationToken, params MySqlParameter[])

  - Task<int> ExecuteNonQueryAsync(MySqlConnection, string, params MySqlParameter[])

  - Task<int> ExecuteNonQueryAsync(MySqlConnection, string, CancellationToken, params MySqlParameter[])

  - Task<int> ExecuteNonQueryAsync(string, string, params MySqlParameter[])

  - Task<int> ExecuteNonQueryAsync(string, string, CancellationToken, params MySqlParameter[])

  - Task<DataSet> ExecuteDatasetAsync(string, string)

  - Task<DataSet> ExecuteDatasetAsync(string, string, CancellationToken)

  - Task<DataSet> ExecuteDatasetAsync(string, string, CancellationToken, params MySqlParameter[])

  - Task<DataSet> ExecuteDatasetAsync(MySqlConnection, string)

- Task<DataSet> ExecuteDatasetAsync(MySqlConnection, string, CancellationToken)

- Task<DataSet> ExecuteDatasetAsync(MySqlConnection, string, params MySqlParameter[])

- Task<DataSet> ExecuteDatasetAsync(MySqlConnection, string, CancellationToken, params MySqlParameter[])

- Task UpdateDataSetAsync(string, string, DataSet, string)

- Task UpdateDataSetAsync(string, string, DataSet, string, CancellationToken)

- Task<MySqlDataReader> ExecuteReaderAsync(MySqlConnection, MySqlTransaction, string, MySqlParameter[], bool)

- Task<MySqlDataReader> ExecuteReaderAsync(MySqlConnection, MySqlTransaction, string, MySqlParameter[], bool, CancellationToken)

- Task<MySqlDataReader> ExecuteReaderAsync(string, string)

- Task<MySqlDataReader> ExecuteReaderAsync(string, string, CancellationToken)

- Task<MySqlDataReader> ExecuteReaderAsync(MySqlConnection, string)

- Task<MySqlDataReader> ExecuteReaderAsync(MySqlConnection, string, CancellationToken)

- Task<MySqlDataReader> ExecuteReaderAsync(string, string, params MySqlParameter[])

- Task<MySqlDataReader> ExecuteReaderAsync(string, string, CancellationToken, params MySqlParameter[])

- Task<MySqlDataReader> ExecuteReaderAsync(MySqlConnection, string, params MySqlParameter[])

- Task<MySqlDataReader> ExecuteReaderAsync(MySqlConnection, string, CancellationToken, params MySqlParameter[])

- Task<object> ExecuteScalarAsync(string, string)

- Task<object> ExecuteScalarAsync(string, string, CancellationToken)

- Task<object> ExecuteScalarAsync(string, string, params MySqlParameter[])

- Task<object> ExecuteScalarAsync(string, string, CancellationToken, params MySqlParameter[])

- Task<object> ExecuteScalarAsync(MySqlConnection, string)

- Task<object> ExecuteScalarAsync(MySqlConnection, string, CancellationToken)

- Task<object> ExecuteScalarAsync(MySqlConnection, string, params MySqlParameter[])

- Task<object> ExecuteScalarAsync(MySqlConnection, string, CancellationToken, params MySqlParameter[])

- Class MySqlScript

  - Task<int> ExecuteAsync()

- Task<int> ExecuteAsync(CancellationToken)

In addition to the methods listed above, the following are methods inherited from the .NET Framework:

- Namespace MySql.Data.Entity

  - Class EFMySqlCommand

    - Task<DbDataReader> ExecuteDbDataReaderAsync(CommandBehaviour, CancellationToken)

    - Task<int> ExecuteNonQueryAsync()

    - Task<int> ExecuteNonQueryAsync(CancellationToken)

    - Task<DbDataReader> ExecuteReaderAsync()

    - Task<DbDataReader> ExecuteReaderAsync(CancellationToken)

    - Task<DbDataReader> ExecuteReaderAsync(CommandBehaviour)

    - Task<DbDataReader> ExecuteReaderAsync(CommandBehaviour, CancellationToken)

    - Task<object> ExecuteScalarAsync()

    - Task<object> ExecuteScalarAsync(CancellationToken)

- Namespace MySql.Data

  - Class MySqlCommand

    - Task<DbDataReader> ExecuteDbDataReaderAsync(CommandBehaviour, CancellationToken)

    - Task<int> ExecuteNonQueryAsync()

    - Task<int> ExecuteNonQueryAsync(CancellationToken)

    - Task<DbDataReader> ExecuteReaderAsync()

    - Task<DbDataReader> ExecuteReaderAsync(CancellationToken)

    - Task<DbDataReader> ExecuteReaderAsync(CommandBehaviour)

    - Task<DbDataReader> ExecuteReaderAsync(CommandBehaviour, CancellationToken)

    - Task<object> ExecuteScalarAsync()

    - Task<object> ExecuteScalarAsync(CancellationToken)

  - Class MySqlConnection

    - Task OpenAsync()

    - Task OpenAsync(CancellationToken)

  - Class MySqlDataReader

    - Task<T> GetFieldValueAsync<T>(int)

- Task<T> GetFieldValueAsync<T>(int, CancellationToken)

- Task<bool> IsDBNullAsync(int)

- Task<bool> IsDBNullAsync(int, CancellationToken)

- Task<bool> NextResultAsync()

- Task<bool> NextResultAsync(CancellationToken)

- Task<bool> ReadAsync()

- Task<bool> ReadAsync(CancellationToken)

Examples

The following examples demonstrate how to use the Asynchronous methods:

In this example, a method has the "async" modifier because the method "await" call made applies to the method LoadAsync. The method returns a Task which contains information about the result of the awaited method. Returning Task is like having a void method, but you should not use "async void" if your method is not a top level access method, like an event.

```
public async Task BulkLoadAsync()
{
  MySqlConnection myConn = new MySqlConnection("MyConnectionString");
  MySqlBulkLoader loader = new MySqlBulkLoader(myConn);

  loader.TableName        = "BulkLoadTest";
  loader.FileName         = @"c:\MyPath\MyFile.txt";
  loader.Timeout          = 0;

  var result              = await loader.LoadAsync();

}
```

In this example, an "async void" method is used with "await" for the ExecuteNonQueryAsync method, to correspond to the onclick event of a button. This is why the method does not return a Task.

```
private async void myButton_Click()
{
  MySqlConnection myConn = new MySqlConnection("MyConnectionString");
  MySqlCommand proc      = new MySqlCommand("MyAsyncSpTest", myConn);

  proc.CommandType        = CommandType.StoredProcedure;

  int result              = await proc.ExecuteNonQueryAsync();
}
```

5.13 Using the Connector/Net Interceptor Classes

An interceptor is a software design pattern that provides a transparent way to extend or modify some aspect of a program, similar to a user exit. No recompiling is required. With Connector/Net, the interceptors

are enabled and disabled by updating the connection string to refer to different sets of interceptor classes that you instantiate.

Connector/Net includes the following interceptor classes:

- The `BaseCommandInterceptor` lets you perform additional operations when a program issues a SQL command. For example, you can examine the SQL statement for logging or debugging purposes, substitute your own result set to implement a caching mechanism, and so on. Depending on the use case, your code can supplement the SQL command or replace it entirely.

  The `BaseCommandInterceptor` class has these methods that you can override:

  ```
  public virtual bool ExecuteScalar(string sql, ref object returnValue);
  public virtual bool ExecuteNonQuery(string sql, ref int returnValue);
  public virtual bool ExecuteReader(string sql, CommandBehavior behavior, ref MySqlDataReader returnValue)
  public virtual void Init(MySqlConnection connection);
  ```

  If your interceptor overrides one of the `Execute...` methods, set the `returnValue` output parameter and return `true` if you handled the event, or `false` if you did not handle the event. The SQL command is processed normally only when all command interceptors return `false`.

  The connection passed to the `Init` method is the connection that is attached to this interceptor.

- The `BaseExceptionInterceptor` lets you perform additional operations when a program encounters an SQL exception. The exception interception mechanism is modeled after the Connector/J model. You can code an interceptor class and connect it to an existing program without recompiling, and intercept exceptions when they are created. You can then change the exception type and optionally attach information to it. This capability lets you turn on and off logging and debugging code without hardcoding anything in the application. This technique applies to exceptions raised at the SQL level, not to lower-level system or I/O errors.

  You develop an exception interceptor first by creating a subclass of the `BaseExceptionInterceptor` class. You must override the `InterceptException()` method. You can also override the `Init()` method to do some one-time initialization.

  Each exception interceptor has 2 methods:

  ```
  public abstract Exception InterceptException(Exception exception,
    MySqlConnection connection);
  public virtual void Init(MySqlConnection connection);
  ```

  The connection passed to `Init()` is the connection that is attached to this interceptor.

  Each interceptor is required to override `InterceptException` and return an exception. It can return the exception it is given, or it can wrap it in a new exception. We currently do not offer the ability to suppress the exception.

Here are examples of using the FQN (fully qualified name) on the connection string:

```
MySqlConnection c1 = new MySqlConnection(@"server=localhost;pooling=false;
commandinterceptors=CommandApp.MyCommandInterceptor,CommandApp");

MySqlConnection c2 = new MySqlConnection(@"server=localhost;pooling=false;
exceptioninterceptors=ExceptionStackTraceTest.MyExceptionInterceptor,ExceptionStackTraceTest");
```

In this example, the command interceptor is called `CommandApp.MyCommandInterceptor` and exists in the `CommandApp` assembly. The exception interceptor is called

`ExceptionStackTraceTest.MyExceptionInterceptor` and exists in the
`ExceptionStackTraceTest` assembly.

To shorten the connection string, you can register your exception interceptors in your `app.config` or
`web.config` file like this:

```
<configSections>
<section name="MySQL" type="MySql.Data.MySqlClient.MySqlConfiguration,
MySql.Data"/>
</configSections>
<MySQL>
<CommandInterceptors>
  <add name="myC" type="CommandApp.MyCommandInterceptor,CommandApp" />
</CommandInterceptors>
</MySQL>

<configSections>
<section name="MySQL" type="MySql.Data.MySqlClient.MySqlConfiguration,
MySql.Data"/>
</configSections>
<MySQL>
<ExceptionInterceptors>
  <add name="myE"
type="ExceptionStackTraceTest.MyExceptionInterceptor,ExceptionStackTraceTest" />
</ExceptionInterceptors>
</MySQL>
```

Once you have done that, your connection strings can look like these:

```
MySqlConnection c1 = new MySqlConnection(@"server=localhost;pooling=false;
commandinterceptors=myC");

MySqlConnection c2 = new MySqlConnection(@"server=localhost;pooling=false;
exceptioninterceptors=myE");
```

5.14 Handling Date and Time Information in Connector/Net

MySQL and the .NET languages handle date and time information differently, with MySQL allowing dates
that cannot be represented by a .NET data type, such as '`0000-00-00 00:00:00`'. These differences
can cause problems if not properly handled.

The following sections demonstrate how to properly handle date and time information when using
Connector/Net.

5.14.1 Fractional Seconds

Connector/Net 6.5 and higher support the fractional seconds feature introduced in MySQL 5.6.4. Fractional
seconds could always be specified in a date literal or passed back and forth as parameters and return
values, but the fractional part was always stripped off when stored in a table column. In MySQL 5.6.4 and
higher, the fractional part is now preserved in data stored and retrieved through SQL. For fractional second
handling in MySQL 5.6.4 and higher, see Fractional Seconds in Time Values. For the behavior of fractional
seconds prior to MySQL 5.6.4, see Fractional Seconds in Time Values.

To use the more precise date and time types, specify a value from 1 to 6 when creating the table
column, for example `TIME(3)` or `DATETIME(6)`, representing the number of digits of precision after

the decimal point. Specifying a precision of 0 leaves the fractional part out entirely. In your C# or Visual Basic code, refer to the `Millisecond` member to retrieve the fractional second value from the `MySqlDateTime` object returned by the `GetMySqlDateTime` function. The `DateTime` object returned by the `GetDateTime` function also contains the fractional value, but only the first 3 digits.

For related code examples, see the following blog post: https://blogs.oracle.com/MySqlOnWindows/entry/milliseconds_value_support_on_datetime

5.14.2 Problems when Using Invalid Dates

The differences in date handling can cause problems for developers who use invalid dates. Invalid MySQL dates cannot be loaded into native .NET `DateTime` objects, including `NULL` dates.

Because of this issue, .NET `DataSet` objects cannot be populated by the `Fill` method of the `MySqlDataAdapter` class as invalid dates will cause a `System.ArgumentOutOfRangeException` exception to occur.

5.14.3 Restricting Invalid Dates

The best solution to the date problem is to restrict users from entering invalid dates. This can be done on either the client or the server side.

Restricting invalid dates on the client side is as simple as always using the .NET `DateTime` class to handle dates. The `DateTime` class will only allow valid dates, ensuring that the values in your database are also valid. The disadvantage of this is that it is not useful in a mixed environment where .NET and non .NET code are used to manipulate the database, as each application must perform its own date validation.

Users of MySQL 5.0.2 and higher can use the new `traditional` SQL mode to restrict invalid date values. For information on using the `traditional` SQL mode, see Server SQL Modes.

5.14.4 Handling Invalid Dates

Although it is strongly recommended that you avoid the use of invalid dates within your .NET application, it is possible to use invalid dates by means of the `MySqlDateTime` data type.

The `MySqlDateTime` data type supports the same date values that are supported by the MySQL server. The default behavior of Connector/Net is to return a .NET DateTime object for valid date values, and return an error for invalid dates. This default can be modified to cause Connector/Net to return `MySqlDateTime` objects for invalid dates.

To instruct Connector/Net to return a `MySqlDateTime` object for invalid dates, add the following line to your connection string:

```
Allow Zero Datetime=True
```

The `MySqlDateTime` class can still be problematic. The following are some known issues:

1. Data binding for invalid dates can still cause errors (zero dates like 0000-00-00 do not seem to have this problem).

2. The `ToString` method return a date formatted in the standard MySQL format (for example, `2005-02-23 08:50:25`). This differs from the `ToString` behavior of the .NET DateTime class.

3. The `MySqlDateTime` class supports NULL dates, while the .NET DateTime class does not. This can cause errors when trying to convert a MySQLDateTime to a DateTime if you do not check for NULL first.

Because of the known issues, the best recommendation is still to use only valid dates in your application.

5.14.5 Handling NULL Dates

The .NET `DateTime` data type cannot handle `NULL` values. As such, when assigning values from a query to a `DateTime` variable, you must first check whether the value is in fact `NULL`.

When using a `MySqlDataReader`, use the `.IsDBNull` method to check whether a value is `NULL` before making the assignment:

Visual Basic Example

```
If Not myReader.IsDBNull(myReader.GetOrdinal("mytime")) Then
    myTime = myReader.GetDateTime(myReader.GetOrdinal("mytime"))
Else
    myTime = DateTime.MinValue
End If
```

C# Example

```
if (! myReader.IsDBNull(myReader.GetOrdinal("mytime")))
    myTime = myReader.GetDateTime(myReader.GetOrdinal("mytime"));
else
    myTime = DateTime.MinValue;
```

`NULL` values will work in a data set and can be bound to form controls without special handling.

5.15 Using the MySqlBulkLoader Class

MySQL Connector/Net features a bulk loader class that wraps the MySQL statement `LOAD DATA INFILE`. This gives MySQL Connector/Net the ability to load a data file from a local or remote host to the server. The class concerned is `MySqlBulkLoader`. This class has various methods, the main one being `load` to cause the specified file to be loaded to the server. Various parameters can be set to control how the data file is processed. This is achieved through setting various properties of the class. For example, the field separator used, such as comma or tab, can be specified, along with the record terminator, such as newline.

The following code shows a simple example of using the `MySqlBulkLoader` class. First an empty table needs to be created, in this case in the `test` database:

```
CREATE TABLE Career (
       Name VARCHAR(100) NOT NULL,
       Age INTEGER,
       Profession VARCHAR(200)
);
```

A simple tab-delimited data file is also created (it could use any other field delimiter such as comma):

```
Table Career in Test Database
Name   Age   Profession
```

```
Tony   47   Technical Writer
Ana   43   Nurse
Fred   21   IT Specialist
Simon   45   Hairy Biker
```

The first three lines need to be ignored with this test file, as they do not contain table data. This can be achieved using the `NumberOfLinesToSkip` property. This file can then be loaded and used to populate the `Career` table in the `test` database:

```csharp
using System;
using System.Text;
using MySql.Data;
using MySql.Data.MySqlClient;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {

            string connStr = "server=localhost;user=root;database=test;port=3306;password=******;";
            MySqlConnection conn = new MySqlConnection(connStr);

            MySqlBulkLoader bl = new MySqlBulkLoader(conn);
            bl.TableName = "Career";
            bl.FieldTerminator = "\t";
            bl.LineTerminator = "\n";
            bl.FileName = "c:/career_data.txt";
            bl.NumberOfLinesToSkip = 3;

            try
            {
                Console.WriteLine("Connecting to MySQL...");
                conn.Open();

                // Upload data from file
                int count = bl.Load();
                Console.WriteLine(count + " lines uploaded.");

                string sql = "SELECT Name, Age, Profession FROM Career";
                MySqlCommand cmd = new MySqlCommand(sql, conn);
                MySqlDataReader rdr = cmd.ExecuteReader();

                while (rdr.Read())
                {
                    Console.WriteLine(rdr[0] + " -- " + rdr[1] + " -- " + rdr[2]);
                }

                rdr.Close();

                conn.Close();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.ToString());
            }
            Console.WriteLine("Done.");
        }
    }
}
```

Further information on `LOAD DATA INFILE` can be found in LOAD DATA INFILE Syntax. Further information on `MySqlBulkLoader` can be found in the reference documentation that was included with your connector.

5.16 Using the MySQL Connector/Net Trace Source Object

MySQL Connector/Net 6.2 introduced support for .NET 2.0 compatible tracing, using `TraceSource` objects.

The .NET 2.0 tracing architecture consists of four main parts:

- *Source* - This is the originator of the trace information. The source is used to send trace messages. The name of the source provided by MySQL Connector/Net is `mysql`.

- *Switch* - This defines the level of trace information to emit. Typically, this is specified in the `app.config` file, so that it is not necessary to recompile an application to change the trace level.

- *Listener* - Trace listeners define where the trace information will be written to. Supported listeners include, for example, the Visual Studio Output window, the Windows Event Log, and the console.

- *Filter* - Filters can be attached to listeners. Filters determine the level of trace information that will be written. While a switch defines the level of information that will be written to all listeners, a filter can be applied on a per-listener basis, giving finer grained control of trace information.

To use tracing **MySql.Data.MySqlClient.MySqlTrace** can be used as a TraceSource for MySQL Connector/Net and the connection string must include *"Logging=True"*.

To enable trace messages, configure a trace switch. Trace switches have associated with them a trace level enumeration, these are **Off**, **Error**, **Warning**, **Info**, and **Verbose**.

```
MySqlTrace.Switch.Level = SourceLevels.Verbose;
```

This sets the trace level to **Verbose**, meaning that all trace messages will be written.

It is convenient to be able to change the trace level without having to recompile the code. This is achieved by specifying the trace level in application configuration file, `app.config`. You then simply need to specify the desired trace level in the configuration file and restart the application. The trace source is configured within the `system.diagnostics` section of the file. The following XML snippet illustrates this:

```
<configuration>
  ...
  <system.diagnostics>
    <sources>
      <source name="mysql" switchName="MySwitch"
              switchType="System.Diagnostics.SourceSwitch" />
      ...
    </sources>
    <switches>
      <add name="MySwitch" value="Verbose"/>
      ...
    </switches>
  </system.diagnostics>
  ...
</configuration>
```

By default, trace information is written to the Output window of Microsoft Visual Studio. There are a wide range of listeners that can be attached to the trace source, so that trace messages can be written out to various destinations. You can also create custom listeners to allow trace messages to be written to other destinations as mobile devices and web services. A commonly used example of a listener is `ConsoleTraceListener`, which writes trace messages to the console.

To add a listener at runtime, use code such as the following:

```
ts.Listeners.Add(new ConsoleTraceListener());
```

Then, call methods on the trace source object to generate trace information. For example, the `TraceInformation()`, `TraceEvent()`, or `TraceData()` methods can be used.

5.16.1 Viewing MySQL Trace Information

This section describes how to set up your application to view MySQL trace information.

The first thing you need to do is create a suitable `app.config` file for your application. An example is shown in the following code:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.diagnostics>
    <sources>
      <source name="mysql" switchName="SourceSwitch"
        switchType="System.Diagnostics.SourceSwitch" >
        <listeners>
          <add name="console" />
          <remove name ="Default" />
        </listeners>
      </source>
    </sources>
    <switches>
      <!-- You can set the level at which tracing is to occur -->
      <add name="SourceSwitch" value="Verbose" />
      <!-- You can turn tracing off -->
      <!--add name="SourceSwitch" value="Off" -->
    </switches>
    <sharedListeners>
      <add name="console"
        type="System.Diagnostics.ConsoleTraceListener"
        initializeData="false"/>
    </sharedListeners>
  </system.diagnostics>
</configuration>
```

This ensures a suitable trace source is created, along with a switch. The switch level in this case is set to `Verbose` to display the maximum amount of information.

In the application the only other step required is to add `logging=true` to the connection string. An example application could be:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;
using MySql.Data;
using MySql.Data.MySqlClient;
using MySql.Web;


namespace ConsoleApplication1
{
```

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

            string connStr = "server=localhost;user=root;database=world;port=3306;password=******;logging=true
            MySqlConnection conn = new MySqlConnection(connStr);
            try
            {
                Console.WriteLine("Connecting to MySQL...");
                conn.Open();

                string sql = "SELECT Name, HeadOfState FROM Country WHERE Continent='Oceania'";
                MySqlCommand cmd = new MySqlCommand(sql, conn);
                MySqlDataReader rdr = cmd.ExecuteReader();

                while (rdr.Read())
                {
                    Console.WriteLine(rdr[0] + " -- " + rdr[1]);
                }

                rdr.Close();

                conn.Close();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.ToString());
            }
            Console.WriteLine("Done.");
        }
    }
}
```

This simple application will then generate the following output:

```
Connecting to MySQL...
mysql Information: 1 : 1: Connection Opened: connection string = 'server=localhost;User Id=root;database=world
;password=******;logging=True'
mysql Information: 3 : 1: Query Opened: SHOW VARIABLES
mysql Information: 4 : 1: Resultset Opened: field(s) = 2, affected rows = -1, inserted id = -1
mysql Information: 5 : 1: Resultset Closed. Total rows=272, skipped rows=0, size (bytes)=7058
mysql Information: 6 : 1: Query Closed
mysql Information: 3 : 1: Query Opened: SHOW COLLATION
mysql Information: 4 : 1: Resultset Opened: field(s) = 6, affected rows = -1, inserted id = -1
mysql Information: 5 : 1: Resultset Closed. Total rows=127, skipped rows=0, size (bytes)=4102
mysql Information: 6 : 1: Query Closed
mysql Information: 3 : 1: Query Opened: SET character_set_results=NULL
mysql Information: 4 : 1: Resultset Opened: field(s) = 0, affected rows = 0, inserted id = 0
mysql Information: 5 : 1: Resultset Closed. Total rows=0, skipped rows=0, size (bytes)=0
mysql Information: 6 : 1: Query Closed
mysql Information: 10 : 1: Set Database: world
mysql Information: 3 : 1: Query Opened: SELECT Name, HeadOfState FROM Country WHERE Continent='Oceania'
mysql Information: 4 : 1: Resultset Opened: field(s) = 2, affected rows = -1, inserted id = -1
American Samoa -- George W. Bush
Australia -- Elisabeth II
...
Wallis and Futuna -- Jacques Chirac
Vanuatu -- John Bani
United States Minor Outlying Islands -- George W. Bush
mysql Information: 5 : 1: Resultset Closed. Total rows=28, skipped rows=0, size (bytes)=788
mysql Information: 6 : 1: Query Closed
Done.
mysql Information: 2 : 1: Connection Closed
```

The first number displayed in the trace message corresponds to the MySQL event type:

Event	Description
1	ConnectionOpened: connection string
2	ConnectionClosed:
3	QueryOpened: mysql server thread id, query text
4	ResultOpened: field count, affected rows (-1 if select), inserted id (-1 if select)
5	ResultClosed: total rows read, rows skipped, size of resultset in bytes
6	QueryClosed:
7	StatementPrepared: prepared sql, statement id
8	StatementExecuted: statement id, mysql server thread id
9	StatementClosed: statement id
10	NonQuery: [varies]
11	UsageAdvisorWarning: usage advisor flag. NoIndex = 1, BadIndex = 2, SkippedRows = 3, SkippedColumns = 4, FieldConversion = 5.
12	Warning: level, code, message
13	Error: error number, error message

The second number displayed in the trace message is the connection count.

Although this example uses the `ConsoleTraceListener`, any of the other standard listeners could have been used. Another possibility is to create a custom listener that uses the information passed using the `TraceEvent` method. For example, a custom trace listener could be created to perform active monitoring of the MySQL event messages, rather than simply writing these to an output device.

It is also possible to add listeners to the MySQL Trace Source at runtime. This can be done with the following code:

```
MySqlTrace.Listeners.Add(new ConsoleTraceListener());
```

MySQL Connector/Net 6.3.2 introduced the ability to switch tracing on and off at runtime. This can be achieved using the calls `MySqlTrace.EnableQueryAnalyzer(string host, int postInterval)` and `MySqlTrace.DisableQueryAnalyzer()`. The parameter `host` is the URL of the MySQL Enterprise Monitor server to monitor. The parameter `postInterval` is how often to post the data to MySQL Enterprise Monitor, in seconds.

5.16.2 Building Custom Listeners

To build custom listeners that work with the MySQL Connector/Net Trace Source, it is necessary to understand the key methods used, and the event data formats used.

The main method involved in passing trace messages is the `TraceSource.TraceEvent` method. This has the prototype:

```
public void TraceEvent(
    TraceEventType eventType,
    int id,
    string format,
    params Object[] args
```

```
)
```

This trace source method will process the list of attached listeners and call the listener's `TraceListener.TraceEvent` method. The prototype for the `TraceListener.TraceEvent` method is as follows:

```
public virtual void TraceEvent(
    TraceEventCache eventCache,
    string source,
    TraceEventType eventType,
    int id,
    string format,
    params Object[] args
)
```

The first three parameters are used in the standard as defined by Microsoft. The last three parameters contain MySQL-specific trace information. Each of these parameters is now discussed in more detail.

`int id`

This is a MySQL-specific identifier. It identifies the MySQL event type that has occurred, resulting in a trace message being generated. This value is defined by the `MySqlTraceEventType` public enum contained in the MySQL Connector/Net code:

```
public enum MySqlTraceEventType : int
{
    ConnectionOpened = 1,
    ConnectionClosed,
    QueryOpened,
    ResultOpened,
    ResultClosed,
    QueryClosed,
    StatementPrepared,
    StatementExecuted,
    StatementClosed,
    NonQuery,
    UsageAdvisorWarning,
    Warning,
    Error
}
```

The MySQL event type also determines the contents passed using the parameter `params Object[] args`. The nature of the `args` parameters are described in further detail in the following material.

`string format`

This is the format string that contains zero or more format items, which correspond to objects in the args array. This would be used by a listener such as `ConsoleTraceListener` to write a message to the output device.

`params Object[] args`

This is a list of objects that depends on the MySQL event type, `id`. However, the first parameter passed using this list is always the driver id. The driver id is a unique number that is incremented each time the connector is opened. This enables groups of queries on the same connection to be identified. The parameters that follow driver id depend on the MySQL event id, and are as follows:

MySQL-specific event type	Arguments (params Object[] args)
ConnectionOpened	Connection string

MySQL-specific event type	Arguments (params Object[] args)
ConnectionClosed	No additional parameters
QueryOpened	mysql server thread id, query text
ResultOpened	field count, affected rows (-1 if select), inserted id (-1 if select)
ResultClosed	total rows read, rows skipped, size of resultset in bytes
QueryClosed	No additional parameters
StatementPrepared	prepared sql, statement id
StatementExecuted	statement id, mysql server thread id
StatementClosed	statement id
NonQuery	Varies
UsageAdvisorWarning	usage advisor flag. NoIndex = 1, BadIndex = 2, SkippedRows = 3, SkippedColumns = 4, FieldConversion = 5.
Warning	level, code, message
Error	error number, error message

This information will allow you to create custom trace listeners that can actively monitor the MySQL-specific events.

5.17 Binary/Nonbinary Issues

There are certain situations where MySQL will return incorrect metadata about one or more columns. More specifically, the server will sometimes report that a column is binary when it is not and vice versa. In these situations, it becomes practically impossible for the connector to be able to correctly identify the correct metadata.

Some examples of situations that may return incorrect metadata are:

- Execution of `SHOW PROCESSLIST`. Some of the columns will be returned as binary even though they only hold string data.

- When a temporary table is used to process a resultset, some columns may be returned with incorrect binary flags.

- Some server functions such `DATE_FORMAT` will incorrectly return the column as binary.

With the availability of `BINARY` and `VARBINARY` data types, it is important that we respect the metadata returned by the server. However, we are aware that some existing applications may break with this change, so we are creating a connection string option to enable or disable it. By default, Connector/Net 5.1 respects the binary flags returned by the server. You might need to make small changes to your application to accommodate this change.

In the event that the changes required to your application would be too large, adding `'respect binary flags=false'` to your connection string causes the connector to use the prior behavior: any column that is marked as string, regardless of binary flags, will be returned as string. Only columns that are specifically marked as a `BLOB` will be returned as `BLOB`.

5.18 Character Set Considerations for Connector/Net

Treating Binary Blobs As UTF8

Before the introduction of The utf8mb4 Character Set (4-Byte UTF-8 Unicode Encoding) (in MySQL 5.5.3), MySQL did not support 4-byte UTF8 sequences. This makes it difficult to represent some multibyte languages such as Japanese. To try and alleviate this, Connector/Net supports a mode where binary blobs can be treated as strings.

To do this, you set the `'Treat Blobs As UTF8'` connection string keyword to `yes`. This is all that needs to be done to enable conversion of all binary blobs to UTF8 strings. To convert only some of your BLOB columns, you can make use of the `'BlobAsUTF8IncludePattern'` and `'BlobAsUTF8ExcludePattern'` keywords. Set these to a regular expression pattern that matches the column names to include or exclude respectively.

When the regular expression patterns both match a single column, the include pattern is applied before the exclude pattern. The result, in this case, would be that the column would be excluded. Also, be aware that this mode does not apply to columns of type `BINARY` or `VARBINARY` and also do not apply to nonbinary `BLOB` columns.

This mode only applies to reading strings out of MySQL. To insert 4-byte UTF8 strings into blob columns, use the .NET `Encoding.GetBytes` function to convert your string to a series of bytes. You can then set this byte array as a parameter for a `BLOB` column.

5.19 Using Connector/Net with Crystal Reports

Crystal Reports is a common tool used by Windows application developers to perform reporting and document generation. In this section we will show how to use Crystal Reports XI with MySQL and Connector/Net.

5.19.1 Creating a Data Source

When creating a report in Crystal Reports there are two options for accessing the MySQL data while designing your report.

The first option is to use Connector/ODBC as an ADO data source when designing your report. You will be able to browse your database and choose tables and fields using drag and drop to build your report. The disadvantage of this approach is that additional work must be performed within your application to produce a data set that matches the one expected by your report.

The second option is to create a data set in VB.NET and save it as XML. This XML file can then be used to design a report. This works quite well when displaying the report in your application, but is less versatile at design time because you must choose all relevant columns when creating the data set. If you forget a column you must re-create the data set before the column can be added to the report.

The following code can be used to create a data set from a query and write it to disk:

Visual Basic Example

```
Dim myData As New DataSet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=world"
```

```
Try
    conn.Open()
    cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " _
        & "country.name, country.population, country.continent " _
        & "FROM country, city ORDER BY country.continent, country.name"
    cmd.Connection = conn

    myAdapter.SelectCommand = cmd
    myAdapter.Fill(myData)

    myData.WriteXml("C:\dataset.xml", XmlWriteMode.WriteSchema)
Catch ex As Exception
    MessageBox.Show(ex.Message, "Report could not be created", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

C# Example

```
DataSet myData = new DataSet();
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
MySql.Data.MySqlClient.MySqlDataAdapter myAdapter;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
myAdapter = new MySql.Data.MySqlClient.MySqlDataAdapter();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
  "pwd=12345;database=test;";

try
{
  cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " +
  "country.name, country.population, country.continent " +
  "FROM country, city ORDER BY country.continent, country.name";
  cmd.Connection = conn;

  myAdapter.SelectCommand = cmd;
  myAdapter.Fill(myData);

  myData.WriteXml(@"C:\dataset.xml", XmlWriteMode.WriteSchema);
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
  MessageBox.Show(ex.Message, "Report could not be created",
  MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

The resulting XML file can be used as an ADO.NET XML datasource when designing your report.

If you choose to design your reports using Connector/ODBC, it can be downloaded from dev.mysql.com.

5.19.2 Creating the Report

For most purposes, the Standard Report wizard helps with the initial creation of a report. To start the wizard, open Crystal Reports and choose the **New > Standard Report** option from the File menu.

The wizard first prompts you for a data source. If you use Connector/ODBC as your data source, use the OLEDB provider for ODBC option from the OLE DB (ADO) tree instead of the ODBC (RDO) tree when choosing a data source. If using a saved data set, choose the ADO.NET (XML) option and browse to your saved data set.

The remainder of the report creation process is done automatically by the wizard.

After the report is created, choose the Report Options... entry of the File menu. Un-check the Save Data With Report option. This prevents saved data from interfering with the loading of data within our application.

5.19.3 Displaying the Report

To display a report we first populate a data set with the data needed for the report, then load the report and bind it to the data set. Finally we pass the report to the crViewer control for display to the user.

The following references are needed in a project that displays a report:

- CrystalDecisions.CrystalReports.Engine

- CrystalDecisions.ReportSource

- CrystalDecisions.Shared

- CrystalDecisions.Windows.Forms

The following code assumes that you created your report using a data set saved using the code shown in Section 5.19.1, "Creating a Data Source", and have a crViewer control on your form named `myViewer`.

Visual Basic Example

```
Imports CrystalDecisions.CrystalReports.Engine
Imports System.Data
Imports MySql.Data.MySqlClient

Dim myReport As New ReportDocument
Dim myData As New DataSet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter

conn.ConnectionString = _
    "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=test"

Try
    conn.Open()

    cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " _
        & "country.name, country.population, country.continent " _
        & "FROM country, city ORDER BY country.continent, country.name"
    cmd.Connection = conn

    myAdapter.SelectCommand = cmd
    myAdapter.Fill(myData)

    myReport.Load(".\world_report.rpt")
    myReport.SetDataSource(myData)
    myViewer.ReportSource = myReport
Catch ex As Exception
    MessageBox.Show(ex.Message, "Report could not be created", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

C# Example

```
using CrystalDecisions.CrystalReports.Engine;
using System.Data;
using MySql.Data.MySqlClient;

ReportDocument myReport = new ReportDocument();
DataSet myData = new DataSet();
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
MySql.Data.MySqlClient.MySqlDataAdapter myAdapter;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
myAdapter = new MySql.Data.MySqlClient.MySqlDataAdapter();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    cmd.CommandText = "SELECT city.name AS cityName, city.population AS CityPopulation, " +
        "country.name, country.population, country.continent " +
        "FROM country, city ORDER BY country.continent, country.name";
    cmd.Connection = conn;

    myAdapter.SelectCommand = cmd;
    myAdapter.Fill(myData);

    myReport.Load(@".\world_report.rpt");
    myReport.SetDataSource(myData);
    myViewer.ReportSource = myReport;
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
    MessageBox.Show(ex.Message, "Report could not be created",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

A new data set it generated using the same query used to generate the previously saved data set. Once the data set is filled, a ReportDocument is used to load the report file and bind it to the data set. The ReportDocument is the passed as the ReportSource of the crViewer.

This same approach is taken when a report is created from a single table using Connector/ODBC. The data set replaces the table used in the report and the report is displayed properly.

When a report is created from multiple tables using Connector/ODBC, a data set with multiple tables must be created in our application. This enables each table in the report data source to be replaced with a report in the data set.

We populate a data set with multiple tables by providing multiple SELECT statements in our MySqlCommand object. These SELECT statements are based on the SQL query shown in Crystal Reports in the Database menu's Show SQL Query option. Assume the following query:

```
SELECT `country`.`Name`, `country`.`Continent`, `country`.`Population`, `city`.`Name`, `city`.`Population`
FROM `world`.`country` `country` LEFT OUTER JOIN `world`.`city` `city` ON `country`.`Code`=`city`.`CountryC
ORDER BY `country`.`Continent`, `country`.`Name`, `city`.`Name`
```

This query is converted to two SELECT queries and displayed with the following code:

Visual Basic Example

```
Imports CrystalDecisions.CrystalReports.Engine
Imports System.Data
```

```vbnet
Imports MySql.Data.MySqlClient

Dim myReport As New ReportDocument
Dim myData As New DataSet
Dim conn As New MySqlConnection
Dim cmd As New MySqlCommand
Dim myAdapter As New MySqlDataAdapter

conn.ConnectionString = "server=127.0.0.1;" _
    & "uid=root;" _
    & "pwd=12345;" _
    & "database=world"

Try
    conn.Open()
    cmd.CommandText = "SELECT name, population, countrycode FROM city ORDER BY countrycode, name; " _
        & "SELECT name, population, code, continent FROM country ORDER BY continent, name"
    cmd.Connection = conn

    myAdapter.SelectCommand = cmd
    myAdapter.Fill(myData)

    myReport.Load(".\world_report.rpt")
    myReport.Database.Tables(0).SetDataSource(myData.Tables(0))
    myReport.Database.Tables(1).SetDataSource(myData.Tables(1))
    myViewer.ReportSource = myReport
Catch ex As Exception
    MessageBox.Show(ex.Message, "Report could not be created", MessageBoxButtons.OK, MessageBoxIcon.Error)
End Try
```

C# Example

```csharp
using CrystalDecisions.CrystalReports.Engine;
using System.Data;
using MySql.Data.MySqlClient;

ReportDocument myReport = new ReportDocument();
DataSet myData = new DataSet();
MySql.Data.MySqlClient.MySqlConnection conn;
MySql.Data.MySqlClient.MySqlCommand cmd;
MySql.Data.MySqlClient.MySqlDataAdapter myAdapter;

conn = new MySql.Data.MySqlClient.MySqlConnection();
cmd = new MySql.Data.MySqlClient.MySqlCommand();
myAdapter = new MySql.Data.MySqlClient.MySqlDataAdapter();

conn.ConnectionString = "server=127.0.0.1;uid=root;" +
    "pwd=12345;database=test;";

try
{
    cmd.CommandText = "SELECT name, population, countrycode FROM city ORDER " +
        "BY countrycode, name; SELECT name, population, code, continent FROM " +
        "country ORDER BY continent, name";
    cmd.Connection = conn;

    myAdapter.SelectCommand = cmd;
    myAdapter.Fill(myData);

    myReport.Load(@".\world_report.rpt");
    myReport.Database.Tables(0).SetDataSource(myData.Tables(0));
    myReport.Database.Tables(1).SetDataSource(myData.Tables(1));
    myViewer.ReportSource = myReport;
}
catch (MySql.Data.MySqlClient.MySqlException ex)
{
```

```
    MessageBox.Show(ex.Message, "Report could not be created",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

It is important to order the `SELECT` queries in alphabetic order, as this is the order the report will expect its source tables to be in. One SetDataSource statement is needed for each table in the report.

This approach can cause performance problems because Crystal Reports must bind the tables together on the client-side, which will be slower than using a pre-saved data set.

5.20 ASP.NET Provider Model

MySQL Connector/Net includes support for the ASP.NET 2.0 provider model. This model enables application developers to focus on the business logic of their application instead of having to recreate such boilerplate items as membership and roles support.

MySQL Connector/Net supplies the following providers:

- Membership Provider

- Role Provider

- Profile Provider

- Session State Provider (MySQL Connector/Net 6.1 and later)

The following tables show the supported providers, their default provider and the corresponding MySQL provider.

Membership Provider

Default Provider	MySQL Provider
System.Web.Security.SqlMembershipProvider	MySql.Web.Security.MySQLMembershipProvider

Role Provider

Default Provider	MySQL Provider
System.Web.Security.SqlRoleProvider	MySql.Web.Security.MySQLRoleProvider

Profile Provider

Default Provider	MySQL Provider
System.Web.Profile.SqlProfileProvider	MySql.Web.Profile.MySQLProfileProvider

SessionState Provider

Default Provider	MySQL Provider
System.Web.SessionState.InProcSessionStateStore	MySql.Web.SessionState.MySqlSessionStateStore

Note

The MySQL Session State provider uses slightly different capitalization on the class name compared to the other MySQL providers.

Installing The Providers

The installation of Connector/Net 5.1 or later will install the providers and register them in your machine's .NET configuration file, `machine.config`. The additional entries created will result in the `system.web` section appearing similar to the following code:

```
<system.web>
  <processModel autoConfig="true" />
  <httpHandlers />
  <membership>
    <providers>
      <add name="AspNetSqlMembershipProvider" type="System.Web.Security.SqlMembershipProvider, System.Web, Ver
      <add name="MySQLMembershipProvider" type="MySql.Web.Security.MySQLMembershipProvider, MySql.Web, Version
    </providers>
  </membership>
  <profile>
    <providers>
      <add name="AspNetSqlProfileProvider" connectionStringName="LocalSqlServer" applicationName="/" type="Sys
      <add name="MySQLProfileProvider" type="MySql.Web.Profile.MySQLProfileProvider, MySql.Web, Version=6.1.1.
    </providers>
  </profile>
  <roleManager>
    <providers>
      <add name="AspNetSqlRoleProvider" connectionStringName="LocalSqlServer" applicationName="/" type="System
      <add name="AspNetWindowsTokenRoleProvider" applicationName="/" type="System.Web.Security.WindowsTokenRol
      <add name="MySQLRoleProvider" type="MySql.Web.Security.MySQLRoleProvider, MySql.Web, Version=6.1.1.0, Cu
    </providers>
  </roleManager>
</system.web>
```

Each provider type can have multiple provider implementations. The default provider can also be set here using the `defaultProvider` attribute, but usually this is set in the `web.config` file either manually or by using the ASP.NET configuration tool.

At time of writing, the `MySqlSessionStateStore` is not added to `machine.config` at install time, and so add the following:

```
<sessionState>
  <providers>
    <add name="MySqlSessionStateStore" type="MySql.Web.SessionState.MySqlSessionStateStore, MySql.Web, Version
  </providers>
</sessionState>
```

The SessionState Provider uses the `customProvider` attribute, rather than `defaultProvider`, to set the provider as the default. A typical `web.config` file might contain:

```
  <system.web>
      <membership defaultProvider="MySQLMembershipProvider" />
      <roleManager defaultProvider="MySQLRoleProvider" />
      <profile defaultProvider="MySQLProfileProvider" />
      <sessionState customProvider="MySqlSessionStateStore" />
      <compilation debug="false">
        ...
```

This sets the MySQL Providers as the defaults to be used in this web application.

The providers are implemented in the file `mysql.web.dll` and this file can be found in your MySQL Connector/Net installation folder. There is no need to run any type of SQL script to set up the database schema, as the providers create and maintain the proper schema automatically.

Using The Providers

The easiest way to start using the providers is to use the ASP.NET configuration tool that is available on the Solution Explorer toolbar when you have a website project loaded.

In the web pages that open, you can select the MySQL membership and roles providers by picking a custom provider for each area.

When the provider is installed, it creates a dummy connection string named `LocalMySqlServer`. Although this has to be done so that the provider will work in the ASP.NET configuration tool, you override this connection string in your `web.config` file. You do this by first removing the dummy connection string and then adding in the proper one, as shown in the following example:

```
<connectionStrings>
  <remove name="LocalMySqlServer"/>
  <add name="LocalMySqlServer" connectionString="server=xxx;uid=xxx;pwd=xxx;database=xxx;"/>
</connectionStrings>
```

> **Note**
>
> You must specify the database in this connection.

Rather than manually editing configuration files, consider using the MySQL Website Configuration tool in MySQL for Visual Studio to configure your desired provider setup. The tool modifies your `website.config` file to the desired configuration. A tutorial on doing this is available in the following section MySQL Website Configuration Tool.

A tutorial demonstrating how to use the Membership and Role Providers can be found in the following section Section 4.2, "Tutorial: MySQL Connector/Net ASP.NET Membership and Role Provider".

Deployment

To use the providers on a production server, distribute the `MySql.Data` and the `MySql.Web` assemblies, and either register them in the remote systems Global Assembly Cache or keep them in your application's `bin/` directory.

5.21 Working with Partial Trust / Medium Trust

.NET applications operate under a given trust level. Normal desktop applications operate under full trust, while web applications that are hosted in shared environments are normally run under the partial trust level (also known as "medium trust"). Some hosting providers host shared applications in their own app pools and allow the application to run under full trust, but this configuration is relatively rare. The Connector/Net support for partial trust has improved over time to simplify the configuration and deployment process for hosting providers.

5.21.1 Evolution of Partial Trust Support Across Connector/Net Versions

The partial trust support for Connector/Net has improved rapidly throughout the 6.5.x and 6.6.x versions. The latest enhancements do require some configuration changes in existing deployments. Here is a summary of the changes for each version.

6.6.4 and Above: Library Can Be Inside or Outside GAC

Now you can install the `MySql.Data.dll` library in the Global Assembly Cache (GAC) as explained in Section 5.21.2, "Configuring Partial Trust with Connector/Net Library Installed in GAC", or in a `bin` or `lib` folder inside the project or solution as explained in Section 5.21.3, "Configuring Partial Trust with Connector/Net Library Not Installed in GAC". If the library is not in the GAC, the only protocol supported is TCP/IP.

6.5.1 and Above: Partial Trust Requires Library in the GAC

Connector/Net 6.5 fully enables our provider to run in a partial trust environment when the library is installed in the Global Assembly Cache (GAC). The new `MySqlClientPermission` class, derived from the .NET `DBDataPermission` class, helps to simplify the permission setup.

5.0.8 / 5.1.3 and Above: Partial Trust Requires Socket Permissions

Starting with these versions, Connector/Net can be used under partial trust hosting that has been modified to allow the use of sockets for communication. By default, partial trust does not include `SocketPermission`. Connector/Net uses sockets to talk with the MySQL server, so the hosting provider must create a new trust level that is an exact clone of partial trust but that has the following permissions added:

- `System.Net.SocketPermission`

- `System.Security.Permissions.ReflectionPermission`

- `System.Net.DnsPermission`

- `System.Security.Permissions.SecurityPermission`

Prior to 5.0.8 / 5.1.3: Partial Trust Not Supported

Connector/Net versions prior to 5.0.8 and 5.1.3 were not compatible with partial trust hosting.

5.21.2 Configuring Partial Trust with Connector/Net Library Installed in GAC

If the library is installed in the GAC, you must include the connection option `includesecurityasserts=true` in your connection string. This is a new requirement as of Connector/Net 6.6.4.

The following list shows steps and code fragments needed to run a Connector/Net application in a partial trust environment. For illustration purposes, we use the Pipe Connections protocol in this example.

1. Install Connector/Net: version 6.6.1 or higher, or 6.5.4 or higher.

2. After installing the library, make the following configuration changes:

   In the `SecurityClasses` section, add a definition for the `MySqlClientPermission` class, including the version to use.

   ```
   <configuration>
     <mscorlib>
       <security>
         <policy>
           <PolicyLevel version="1">
             <SecurityClasses>
               ....
               <SecurityClass Name="MySqlClientPermission" Description="MySql.Data.MySqlClient.MySqlClientPerm
   ```

   Scroll down to the `ASP.Net` section:

   ```
   <PermissionSet  class="NamedPermissionSet"  version="1"  Name="ASP.Net">
   ```

Add a new entry for the detailed configuration of the `MySqlClientPermission` class:

```
<IPermission class="MySqlClientPermission" version="1" Unrestricted="true"/>
```

> **Note**
>
> This configuration is the most generalized way that includes all keywords.

3. Configure the MySQL server to accept pipe connections, by adding the `--enable-named-pipe` option on the command line. If you need more information about this, see Installing MySQL on Microsoft Windows.

4. Confirm that the hosting provider has installed the Connector/Net library (`MySql.Data.dll`) in the GAC.

5. Optionally, the hosting provider can avoid granting permissions globally by using the new `MySqlClientPermission` class in the trust policies. (The alternative is to globally enable the permissions `System.Net.SocketPermission`, `System.Security.Permissions.ReflectionPermission`, `System.Net.DnsPermission`, and `System.Security.Permissions.SecurityPermission`.)

6. Create a simple web application using Visual Studio 2010.

7. Add the reference in your application for the `MySql.Data.MySqlClient` library.

8. Edit your `web.config` file so that your application runs using a Medium trust level:

```
<system.web>
  <trust level="Medium"/>
</system.web>
```

9. Add the `MySql.Data.MySqlClient` namespace to your server-code page.

10. Define the connection string, in slightly different ways depending on the Connector/Net version.

   Only for 6.6.4 or later: To use the connections inside any web application that will run in Medium trust, add the new `includesecurityasserts` option to the connection string. `includesecurityasserts=true` that makes the library request the following permissions when required: `SocketPermissions`, `ReflectionPermissions`, `DnsPermissions`, `SecurityPermissions` among others that are not granted in Medium trust levels.

   For Connector/Net 6.6.3 or earlier: No special setting for security is needed within the connection string.

```
MySqlConnectionStringBuilder myconnString = new MySqlConnectionStringBuilder("server=localhost;User Id=
myconnString.PipeName = "MySQL55";
myconnString.ConnectionProtocol = MySqlConnectionProtocol.Pipe;
// Following attribute is a new requirement when the library is in the GAC.
// Could also be done by adding includesecurityasserts=true; to the string literal
// in the constructor above.
// Not needed with Connector/Net 6.6.3 and earlier.
myconnString.IncludeSecurityAsserts = true;
```

11. Define the `MySqlConnection` to use:

```
MySqlConnection myconn = new MySqlConnection(myconnString.ConnectionString);
myconn.Open();
```

12. Retrieve some data from your tables:

```
MySqlCommand cmd = new MySqlCommand("Select * from products", myconn);
MySqlDataAdapter da = new MySqlDataAdapter(cmd);
DataSet1 tds = new DataSet1();
da.Fill(tds, tds.Tables[0].TableName);
GridView1.DataSource = tds;
GridView1.DataBind();
myconn.Close()
```

13. Run the program. It should execute successfully, without requiring any special code or encountering any security problems.

5.21.3 Configuring Partial Trust with Connector/Net Library Not Installed in GAC

When deploying a web application to a Shared Hosted environment, where this environment is configured to run all their .NET applications under a partial or medium trust level, you might not be able to install the Connector/Net library in the GAC. Instead, you put a reference to the library in the `bin` or `lib` folder inside the project or solution. In this case, you configure the security in a different way than when the library is in the GAC.

Connector/Net is commonly used by applications that run in Windows environments where the default communication for the protocol is used via sockets or by TCP/IP. For this protocol to operate is necessary have the required socket permissions in the web configuration file as follows:

1. Open the medium trust policy web configuration file, which should be under this folder:

```
%windir%\Microsoft.NET\Framework\{version}\CONFIG\web_mediumtrust.config
```

Use `Framework64` in the path instead of `Framework` if you are using a 64-bit installation of the framework.

2. Locate the `SecurityClasses` tag:

```
<SecurityClass Name="SocketPermission"
Description="System.Net.SocketPermission, System, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
```

3. Scroll down and look for the following `PermissionSet`:

```
<PermissionSet version="1" Name="ASP.Net">
```

4. Add the following inside this `PermissionSet`:

```
<IPermission class="SocketPermission" version="1" Unrestricted="true" />
```

This configuration lets you use the driver with the default Windows protocol TCP/IP without having any security issues. This approach only supports the TCP/IP protocol, so you cannot use any other type of connection.

Also, since the `MySQLClientPermissions` class is not added to the medium trust policy, you cannot use it. This configuration is the minimum required in order to work with Connector/Net without the GAC.

Chapter 6 Connector/Net Connection String Options Reference

For usage information about connection strings, see Section 5.2, "Creating a Connector/Net Connection String". The first table list options that apply generally to all server configurations. The options related to systems using a connection pool are split into a separate table.

General Options

Table 6.1 Connector/Net Connection String Options - General

Name	Default	Description
`Allow Batch`, `AllowBatch`	true	When true, multiple SQL statements can be sent with one command execution. Note: starting with MySQL 4.1.1, batch statements should be separated by the server-defined separator character. Statements sent to earlier versions of MySQL should be separated by ';'.
`Allow User Variables`, `AllowUserVariables`	false	Setting this to `true` indicates that the provider expects user variables in the SQL. This option was added in Connector/Net version 5.2.2.
`Allow Zero Datetime`, `AllowZeroDateTime`	false	If set to `True`, `MySqlDataReader.GetValue()` returns a `MySqlDateTime` object for date or datetime columns that have disallowed values, such as zero datetime values, and a `System.DateTime` object for valid values. If set to `False` (the default setting) it causes a `System.DateTime` object to be returned for all valid values and an exception to be thrown for disallowed values, such as zero datetime values.
`Auto Enlist`, `AutoEnlist`	true	If `AutoEnlist` is set to `true`, which is the default, a connection opened using `TransactionScope` participates in this scope, it commits when the scope commits and rolls back if `TransactionScope` does not commit. However, this feature is considered security sensitive and therefore cannot be used in a medium trust environment.
`BlobAsUTF8ExcludePattern`	null	A POSIX-style regular expression that matches the names of BLOB columns that do not contain UTF-8 character data. See Section 5.18, "Character Set Considerations for Connector/Net" for usage details.
`BlobAsUTF8IncludePattern`	null	A POSIX-style regular expression that matches the names of BLOB columns containing UTF-8 character data. See Section 5.18, "Character Set Considerations for Connector/Net" for usage details.
`Certificate File`, `CertificateFile`	null	This option specifies the path to a certificate file in PKCS #12 format (`.pfx`). For an example of usage, see Section 4.8, "Tutorial: Using SSL with MySQL Connector/Net". Was introduced with 6.2.1.
`Certificate Password`, `CertificatePassword`	null	Specifies a password that is used in conjunction with a certificate specified using the option `CertificateFile`. For an example of usage, see Section 4.8, "Tutorial: Using SSL with MySQL Connector/Net". Was introduced with 6.2.1.

Name	Default	Description
`Certificate Store Location`, `CertificateStoreLocation`	null	Enables you to access a certificate held in a personal store, rather than use a certificate file and password combination. For an example of usage, see Section 4.8, "Tutorial: Using SSL with MySQL Connector/Net". Was introduced with 6.2.1.
`Certificate Thumbprint`, `CertificateThumbprint`	null	Specifies a certificate thumbprint to ensure correct identification of a certificate contained within a personal store. For an example of usage, see Section 4.8, "Tutorial: Using SSL with MySQL Connector/Net". Was introduced with 6.2.1.
`CharSet`, `Character Set`, `CharacterSet`		Specifies the character set that should be used to encode all queries sent to the server. Resultsets are still returned in the character set of the result data.
`Check Parameters`, `CheckParameters`	true	Indicates if stored routine parameters should be checked against the server.
`Command Interceptors`, `CommandInterceptors`		The list of interceptors that can intercept SQL command operations.
`Connect Timeout`, `Connection Timeout`, `ConnectionTimeout`	15	The length of time (in seconds) to wait for a connection to the server before terminating the attempt and generating an error.
`Connect_Attrs`		Passes a predefined set of key/value pairs containing connection-specific data items to the server, for application-defined purposes. connector/Net automatically transmits the following attributes: `_pid`, `_client_version`, `_platform`, `_program_name`, `_os`, `_os_details`, and `_thread`. See Performance Schema Connection Attribute Tables for the Performance Schema tables to query on the server side to make use of these attributes.
`Convert Zero Datetime`, `ConvertZeroDateTime`	false	True to have `MySqlDataReader.GetValue()` and `MySqlDataReader.GetDateTime()` return `DateTime.MinValue` for date or datetime columns that have disallowed values.
`Default Command Timeout`, `DefaultCommandTimeout`	30	Sets the default value of the command timeout to be used. This does not supersede the individual command timeout property on an individual command object. If you set the command timeout property, that will be used. This option was added in Connector/Net 5.1.4
`Default Table Cache Age`, `DefaultTableCacheAge`	60	Specifies how long a `TableDirect` result should be cached, in seconds. For usage information about table caching, see Section 5.8, "Using Connector/Net with Table Caching". This option was added in Connector/Net 6.4.
`enableSessionExpireCallback`	false	When set to `true`, causes the session-expiry scanner to raise the `session_end` event before deleting the session data stored in the `my_aspnet_sessions` table, when a session times out. Enable this option to write additional application-specific cleanup code to handle the

Name	Default	Description
		`session_end` event of the `global.asax` class, before the stored data of the session gets deleted. Within the session_end method, any other required cleanup can be done. This option was added in Connector/Net 6.4.5; it is not yet available in Connector/Net 6.5.x releases.
`Encrypt`, `UseSSL`	false	For Connector/Net 5.0.3 and later, when `true`, SSL encryption is used for all data sent between the client and server if the server has a certificate installed. Recognized values are `true`, `false`, `yes`, and `no`. In versions before 5.0.3, this option had no effect. From version 6.2.1, this option is deprecated and is replaced by `SSL Mode`. The option still works if used. If this option is set to true, it is equivalent to `SSL Mode = Preferred`.
`Exception Interceptors`, `ExceptionInterceptors`		The list of interceptors that can triage thrown `MySqlException` exceptions.
`Functions Return String`, `FunctionsReturnString`	false	Causes the connector to return binary/varbinary values as strings, if they do not have a tablename in the metadata.
`Host`, `Server`, `Data Source`, `DataSource`, `Address`, `Addr`, `Network Address`	localhost	The name or network address of the instance of MySQL to which to connect. Multiple hosts can be specified separated by commas. This can be useful where multiple MySQL servers are configured for replication and you are not concerned about the precise server you are connecting to. No attempt is made by the provider to synchronize writes to the database, so take care when using this option. In Unix environment with Mono, this can be a fully qualified path to a MySQL socket file. With this configuration, the Unix socket is used instead of the TCP/IP socket. Currently, only a single socket name can be given, so accessing MySQL in a replicated environment using Unix sockets is not currently supported.
`Ignore Prepare`, `IgnorePrepare`	true	When true, instructs the provider to ignore any calls to `MySqlCommand.Prepare()`. This option is provided to prevent issues with corruption of the statements when used with server-side prepared statements. If you use server-side prepare statements, set this option to false. This option was added in Connector/Net 5.0.3 and Connector/Net 1.0.9.
`includesecurityasserts`, `include security asserts`	false	Must be set to `true` when using the `MySQLClientPermissions` class in a partial trust environment, with the library installed in the GAC of the hosting environment. This requirement is new for partial-trust applications in Connector/Net 6.6.4 and higher. See Section 5.21, "Working with Partial Trust / Medium Trust" for details.
`Initial Catalog`, `Database`	mysql	The case-sensitive name of the database to use initially.
`Interactive`, `Interactive Session`, `InteractiveSession`	false	If set to true, the client is interactive. An interactive client is one where the server variable `CLIENT_INTERACTIVE` is set. If an interactive client is set, the `wait_timeout`

Name	Default	Description
		variable is set to the value of `interactive_timeout`. The client will then time out after this period of inactivity. For more details, see Server System Variables in the MySQL Reference Manual.
`Integrated Security`, `IntegratedSecurity`	no	Use Windows authentication when connecting to server. By default, it is turned off. To enable, specify a value of `yes`. (You can also use the value `sspi` as an alternative to `yes`.) For details, see Section 5.5, "Using the Windows Native Authentication Plugin". This option was introduced in Connector/Net 6.4.4.
`Keep Alive`, `Keepalive`	0	For TCP connections, idle connection time measured in seconds, before the first keepalive packet is sent. A value of 0 indicates that keepalive is not used. Before Connector/Net 6.6.7/6.7.5/6.8.4, this value was measured in milliseconds.
`Logging`	false	When true, various pieces of information is output to any configured TraceListeners. See Section 5.16, "Using the MySQL Connector/Net Trace Source Object" for further details.
`Old Guids`, `OldGuids`	false	This option was introduced in Connector/Net 6.1.1. The back-end representation of a GUID type was changed from `BINARY(16)` to `CHAR(36)`. This was done to allow developers to use the server function `UUID()` to populate a GUID table - `UUID()` generates a 36-character string. Developers of older applications can add `'Old Guids=true'` to the connection string to use a GUID of data type `BINARY(16)`.
`Old Syntax`, `OldSyntax`, `Use Old Syntax`, `UseOldSyntax`	false	This option was deprecated in Connector/Net 5.2.2 and removed in Connector/Net 6.10.2. All code should now be written using the '@' symbol as the parameter marker.
`Password`, `pwd`		The password for the MySQL account being used.
`Persist Security Info`, `PersistSecurityInfo`	false	When set to `false` or `no` (strongly recommended), security-sensitive information, such as the password, is not returned as part of the connection if the connection is open or has ever been in an open state. Resetting the connection string resets all connection string values, including the password. Recognized values are `true`, `false`, `yes`, and `no`.
`Pipe Name`, `Pipe`, `PipeName`	mysql	When set to the name of a named pipe, the `MySqlConnection` attempts to connect to MySQL on that named pipe. This setting only applies to the Windows platform.
`Port`	3306	The port MySQL is using to listen for connections. This value is ignored if Unix socket is used.
`Procedure Cache Size`, `ProcedureCacheSize`, `procedure cache`, `procedurecache`	25	Sets the size of the stored procedure cache. By default, Connector/Net stores the metadata (input/output data types) about the last 25 stored procedures used. To disable the stored procedure cache, set the value to zero

Name	Default	Description
		(0). This option was added in Connector/Net 5.0.2 and Connector/Net 1.0.9.
`Protocol`, `Connection Protocol`, `ConnectionProtocol`	socket	Specifies the type of connection to make to the server. Values can be: `socket` or `tcp` for a socket connection, `pipe` for a named pipe connection, `unix` for a Unix socket connection, `memory` to use MySQL shared memory.
`Replication`	false	Indicates if this connection is to use replicated servers.
`Respect Binary Flags`, `RespectBinaryFlags`	true	Setting this option to `false` means that Connector/Net ignores a column's binary flags as set by the server. This option was added in Connector/Net version 5.1.3.
`Shared Memory Name`, `SharedMemoryName`	MYSQL	The name of the shared memory object to use for communication if the connection protocol is set to `memory`.
`Sql Server Mode`, `sqlservermode`	false	Allow SQL Server syntax. When set to `true`, enables Connector/Net to support square brackets around symbols instead of backticks. This enables Visual Studio wizards that bracket symbols with [] to work with Connector/Net. This option incurs a performance hit, so should only be used if necessary. This option was added in version 6.3.1.
`SSL Mode`, `SslMode`	None	This option has the following values: • **None** - do not use SSL. • **Preferred** - use SSL if the server supports it, but allow connection in all cases. • **Required** - Always use SSL. Deny connection if server does not support SSL. • **VerifyCA** - Always use SSL. Validate the CA but tolerate name mismatch. • **VerifyFull** - Always use SSL. Fail if the host name is not correct. This option was introduced in MySQL Connector/Net 6.2.1.
`Table Cache`, `tablecache`, `tablecaching`	false	Enables or disables caching of `TableDirect` commands. A value of `true` enables the cache while `false` disables it. For usage information about table caching, see Section 5.8, "Using Connector/Net with Table Caching". This option was added in Connector/Net 6.4.
`Treat BLOBs as UTF8`, `TreatBlobsAsUTF8`	false	
`Treat Tiny As Boolean`, `TreatTinyAsBoolean`	true	Setting this value to `false` causes `TINYINT(1)` to be treated as an `INT`. See Numeric Type Overview for a further explanation of the `TINYINT` and `BOOL` data types.

Name	Default	Description
`Use Affected Rows`, `UseAffectedRows`	false	When `true`, the connection reports changed rows instead of found rows. This option was added in Connector/Net version 5.2.6.
`Use Procedure Bodies`, `UseProcedureBodies`, `procedure bodies`	true	When set to `true`, the default value, MySQL Connector/Net expects the body of the procedure to be viewable. This enables it to determine the parameter types and order. Set the option to `false` when the user connecting to the database does not have the `SELECT` privileges for the `mysql.proc` (stored procedures) table, or cannot view `INFORMATION_SCHEMA.ROUTINES`. In this case, MySQL Connector/Net cannot determine the types and order of the parameters, and must be alerted to this fact by setting this option to `false`. When set to `false`, MySQL Connector/Net does not rely on this information being available when the procedure is called. Because MySQL Connector/Net will not be able to determine this information, explicitly set the types of all the parameters before the call and add the parameters to the command in the same order as they appear in the procedure definition. This option was added in MySQL Connector/Net 5.0.4 and MySQL Connector/Net 1.0.10.
`User Id`, `UserID`, `Username`, `Uid`, `User name`, `User`		The MySQL login account being used.
`Compress`, `Use Compression`, `UseCompression`	false	Setting this option to `true` enables compression of packets exchanged between the client and the server. This exchange is defined by the MySQL client/server protocol. Compression is used if both client and server support ZLIB compression, and the client has requested compression using this option. A compressed packet header is: packet length (3 bytes), packet number (1 byte), and Uncompressed Packet Length (3 bytes). The Uncompressed Packet Length is the number of bytes in the original, uncompressed packet. If this is zero, the data in this packet has not been compressed. When the compression protocol is in use, either the client or the server may compress packets. However, compression will not occur if the compressed length is greater than the original length. Thus, some packets will contain compressed data while other packets will not.
`Use Usage Advisor`, `Usage Advisor`, `UseUsageAdvisor`	false	Logs inefficient database operations.
`Use Performance Monitor`, `UsePerformanceMonitor`, `userperfmon`, `perfmon`	false	Indicates that performance counters should be updated during execution.

Connection Pooling Options

The following table lists the valid names for options related to connection pooling within the
`ConnectionString`. For more information about connection pooling, see Section 5.4, "Using Connector/
Net with Connection Pooling".

Table 6.2 Connector/Net Connection String Options - Connection Pooling

Name	Default	Description
`Cache Server Properties`, `CacheServerProperties`	false	Specifies whether server variable settings are updated by a `SHOW VARIABLES` command each time a pooled connection is returned. Enabling this setting speeds up connections in a connection pool environment. Your application is not informed of any changes to configuration variables made by other connections. This option was added in Connector/Net 6.3.
`Connection Lifetime`, `ConnectionLifeTime`	0	When a connection is returned to the pool, its creation time is compared with the current time, and the connection is destroyed if that time span (in seconds) exceeds the value specified by `Connection Lifetime`. This is useful in clustered configurations to force load balancing between a running server and a server just brought online. A value of zero (0) causes pooled connections to have the maximum connection timeout.
`Connection Reset`, `ConnectionReset`	false	If true, the connection state is reset when it is retrieved from the pool. The default value of false avoids making an additional server round trip when obtaining a connection, but the connection state is not reset.
`Maximum Pool Size`, `Max Pool Size`, `MaximumPoolsize`, `maxpoolsize` (only for Connector/Net 6.7 and later)	100	The maximum number of connections allowed in the pool.
`Minimum Pool Size`, `Min Pool Size`, `MinimumPoolSize`, `minpoolsize` (only for Connector/Net 6.7 and later)	0	The minimum number of connections allowed in the pool.
`Pooling`	true	When `true`, the `MySqlConnection` object is drawn from the appropriate pool, or if necessary, is created and added to the appropriate pool. Recognized values are `true`, `false`, `yes`, and `no`.

Chapter 7 Connector/Net Support for Windows Store

Starting with 6.7, Connector/Net fully supports building Windows Store apps. Windows Store are based on .NET but use a very restrictive subset of that functionality. The main difference is the complete lack of the ADO.Net data subsystem. Below is a list of the differences between Connector/Net 6.7 and Connector/Net RT 6.7. Please note that we are looking at ways of bringing these functionalities back in future versions.

- No support for MySqlDataAdapter. MySqlCommand and MySqlDataReader are supported.

- Connector/Net RT does not support SSL connections or Windows authentication. Also, SHA256 is not currently supported.

- Connector/Net RT only supports TCP connections. Named pipe and shared memory connections are not supported.

- Connector/Net RT doesn't support tracing.

  This version of Connector/Net is no longer supported.

- Connector/Net RT doesn't support load balancing. Command and Exception interceptors are supported however.

  This version of Connector/Net is no longer supported.

- MySqlConnection.GetSchema methods do not return DataTable types. Instead they return a new object called MySqlSchemaCollection. You can query this object for the schema information. Normal Connector/Net will include support for returning schema information in both DataTable and MySqlSchemaCollection format.

  This version of Connector/Net is no longer supported.

- Some contructors and other apis on supported classes may have been removed or altered. Any API that used DataTable, DataSet, or DataRow has been altered or removed.

  This version of Connector/Net is no longer supported.

Using Connector/Net RT is easy. Simply create a Windows Store application using Visual Studio and then reference the MySql.Data.RT.dll assembly in your project. The code you write should be exactly the same as for normal Connector/Net (including using same namespace MySql.Data.MySqlClient) except for the differences listed above.

Chapter 8 EF5 Support

Support for Entity Framework 5 (EF5) is based on the introduction of .NET Framework 4.5, which is the default target framework in Microsoft Visual Studio 2012, or higher. Using this framework version is a requirement for applying any of the features in EF5 to your code.

This chapter provides a detailed description of each one of the features.

EF5 Features

Spatial Data Type (Geometry type support). Spatial data types support is backed up by the server capabilities, which are documented at: Extensions for Spatial Data There are different types for spatial data. MySQL supports the following types that you can instantiate and use:

`Point`, `LineString`, `Polygon`, `GeometryCollection`, `MultiPoint`, `MultiLineString`, and `MultiPolygon`.

In MySQL Connector/Net, the different types can be managed with the `Geometry` type. Before Connector/Net 6.7, the geometry types did not have a MySql type inside the driver. Common practice was to use a binary type; however, this is no longer needed. Now, all the specific operations for any geometry type can be performed by using this new class.

An example of how to write a point data is shown here using Connector/Net 6.7 and MySQL 5.6.7 or higher.

```
   //Storing a geometry point
MySqlConnection conn = new MySqlConnection("server=localhost;userid=root;database=testgeo;");
   conn.Open();
MySqlCommand cmd = new MySqlCommand("CREATE TABLE Test (v Geometry NOT NULL)");
cmd.Connection = conn;
   cmd.ExecuteNonQuery();

   cmd = new MySqlCommand("INSERT INTO Test VALUES(GeomFromText(?v))", conn);
cmd.Parameters.Add("?v",MySqlDbType.String);
cmd.Parameters[0].Value = "POINT(47.37-122.21)";
cmd.ExecuteNonQuery();

   cmd.CommandText = "SELECT AsText(v) FROM Test";
using(MySqlDataReader reader = cmd.ExecuteReader())
{
 reader.Read();
 varval = reader.GetValue(0);
 Console.WriteLine(val);
}
conn.Close();
```

Output:

POINT(47.37, -122.21)

To read a set of rows that contains a Geometry column can be achieved by using an appropriate `MySqlDataReader`. Example:

```
   cmd.CommandText = "SELECT v FROM Test";
using(MySqlDataReader reader = cmd.ExecuteReader())
{
 reader.Read();
 var val = reader.GetMySqlGeometry(0);
```

```
  var valWithName = reader.GetMySqlGeometry("v");
  Console.WriteLine(val.ToString());
  // output : ("POINT(47.37 -122.21)"
  Console.WriteLine(valWithName.ToString());
  // output("POINT(47.37 -122.21)"
}
```

A geometry point also can contain a Spatial Reference Identifier (SRID) value. This value can also be stored using a `Geometry` type.

Example:

```
   MySqlGeometry v = new MySqlGeometry(47.37, -122.21, 101);
var par =new MySqlParameter("?v", MySqlDbType.Geometry);
par.Value = v;

   MySqlCommand cmd = new MySqlCommand("INSERT INTO Test VALUES(?v)", conn);
cmd.Parameters.Add(par);
cmd.ExecuteNonQuery();

   cmd.CommandText = "SELECT SRID(v) FROM Test";

   using (MySqlDataReader reader = cmd.ExecuteReader())
{
   reader.Read(); var val = reader.GetString(0);
   Console.WriteLine(val); // output "101"
}
```

Other types of values, besides `Point`, can be stored at the `Geometry` type:

- `LineString`

- `MultiLineString`

- `Polygon`

- `MultiPolygon`

- `GeometryCollection`

Spatial data is supported in Model First, Database First, and Code First approaches.

Using spatial types with an Entity Framework Model is one of the new features of Connector/Net 6.7.

Spatial types can be used with any of the strategies used to create the data layer of any application: Database First, Code First or Model First. Entity Framework support two main types for spatial data: `DbGeometry` and `DBGeography`. The second one is not supported by MySQL Connector/Net, because MySQL server does not have any equivalent type to map to this type. So, all the examples will use the `DbGeometry` type.

Example of Usage in Code First

An entity that contains a Geometry column can be defined as follows:

```
   public class Distributor { public int DistributorId { get; set; }
   public string Name { get; set; } public DbGeometry point { get; set;
   } }

   public class DistributorsContext : DbContext { public
   DbSet<Distributor> Distributors { get; set; } }
```

Creating the database:

```
using (DistributorsContext context = new DistributorsContext()) {
context.Database.Delete(); context.Database.Create();
context.Distributors.Add(new Distributor() { Name = "Graphic
Design Institute", point =
DbGeometry.FromText("POINT(-122.336106 47.605049)",101),
}); context.SaveChanges();

var result = (from u in context.Distributors select
u.point.CoordinateSystemId).ToList();

foreach (var item in result)
Console.WriteLine("CoordinateSystemId " + item);
```

Output:

CoordinateSystemId 101

Spatial-Supported Functions

There are some useful functions that takes Geometry values:

- `SpatialDimension`

- `SpatialEnvelope`

- `IsSimpleGeometry`

- `SpatialTypeName`

- `CoordinateSystemId`

- `Point`

- `XCoordinate`

- `YCoordinate`

- `GeometryFromText`

- `SpatialContains`

- `AsText`

- `SpatialBuffer`

- `SpatialDifference`

- `SpatialIntersection`

Enumeration Support

- Enumeration types are used to define a set of named constants that may be assigned to a numeric value.

- By default the underlying type of each element in the enumeration is `int`. But it can be specified as another numeric type by using a colon. For example, enum Months : byte { Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec };

- When a value is not specified in the defined list, the values are automatically incremented by 1. In the previous example, Months.Jan has a default value of 0.

To use enumeration types from within a class defined by Entity Framework:

(In Visual Studio 2012 and using the Database First approach.)

1. Create a new project (make sure .NET Framework is target in the New Project dialog.).

2. Add a new Entity Model.

3. Select the Existing Database option.

4. Select the tables to be imported in the model.

5. In order to create an enumeration type, an integer column should exists inside of a table.

6. In the Model designer select the table to use.

7. Right click the column of integer type and select **Create enum type**.

   At this point, each LINQ to Entities query can use the enumeration type directly.

Example:

```
  using (var db = new DistributorsEntities()) {
db.Database.CreateIfNotExists();
db.distributors.Add(new distributor { Type = DistributorType.Regional });
db.distributors.Add(new distributor { Type = DistributorType.Reseller });
db.distributors.Add(new distributor { Type = DistributorType.Zone });
db.SaveChanges();

var testQ = (from d in db.distributors select d).FirstOrDefault();
foreach (var item in testQ)
  Console.WriteLine(item.Type);
}
```

Output:

Regional Reseller Zone

Other important features and improvements included in Entity Framework are:

- All LINQ queries are now automatically compiled and cached to improve query performance.

- Multiple Diagrams for a model is supported.

- Enhancement Table per type in SQL Generation.

- Performance Enhancements.

- The `StoreGeneratedPattern` for key columns can now be set on an entity Properties window and this value will propagate from the entity model down to the stored definition. The stored `Generated Pattern` attribute allows you to control how the Entity Framework synchronizes database column values and entity property values.

Chapter 9 EF6 Support

MySQL Connector/Net 6.8 integrates support for Entity Framework 6.0 (EF6), but also offers support for Entity Framework 5 (EF 5). This chapter explains the new features in Entity Framework 6 implemented in MySQL Connector/Net 6.8.

Requirements for Entity Framework 6.0 Support

- MySQL Connector/Net 6.8.x

- MySQL Server 5.1 or higher

- Entity Framework 6 assemblies

- .NET Framework 4.0 or higher

Configuration

To configure MySQL Connector/Net support for EF6:

1. An important first step is editing the configuration sections in the `App.config` file to add the connection string and the MySQL Connector/Net provider for EF6:

```
<connectionStrings>
    <add name="MyContext" providerName="MySql.Data.MySqlClient"
        connectionString="server=localhost;port=3306;database=mycontext;uid=root;password=********"/>
</connectionStrings>
<entityFramework>
    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory, EntityFrame
    <providers>
        <provider invariantName="MySql.Data.MySqlClient"
            type="MySql.Data.MySqlClient.MySqlProviderServices, MySql.Data.Entity.EF6"/>
        <provider invariantName="System.Data.SqlClient"
            type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer"/>
    </providers>
</entityFramework>
```

2. Apply the reference for `MySql.Data.Entity`.

   - Installing the MySql.Data.Entity NuGet package adds this reference automatically to `app.config` or `web.config` during the installation. For more information about the MySql.Data.Entity NuGet package and its uses, see https://www.nuget.org/packages/MySql.Data.Entity/.

     Proceed to step 3.

   - Otherwise, add the reference for `MySql.Data.Entity.EF6` assembly into the project. Depending on the .NET Framework used, the assembly is to be taken from either the `v4.0` or the `v4.5` folder).

     Unless MySQL Connector/Net was installed with the standalone MSI or MySQL Installer, which adds the reference, insert the following data provider information into the `app.config` or `web.config` file:

```
<system.data>
  <DbProviderFactories>
    <remove invariant="MySql.Data.MySqlClient" />
    <add name="MySQL Data Provider" invariant="MySql.Data.MySqlClient" description=".Net Framework D
        type="MySql.Data.MySqlClient.MySqlClientFactory, MySql.Data, Version=6.9.9.0, Culture=neutr
  </DbProviderFactories>
```

```
</system.data>
```

> **Important**
>
> Always update the version number to match the one in the `MySql.Data.dll` assembly.

3. Set the new `DbConfiguration` class for MySql. This step is optional but highly recommended, because it adds all the dependency resolvers for MySql classes. This can be done in three ways:

   - Adding the `DbConfigurationTypeAttribute` on the context class:

   ```
   [DbConfigurationType(typeof(MySqlEFConfiguration))]
   ```

   - Calling `DbConfiguration.SetConfiguration(new MySqlEFConfiguration())` at the application startup.

   - Set the `DbConfiguration` type in the configuration file:

   ```
   <entityFramework codeConfigurationType="MySql.Data.Entity.MySqlEFConfiguration, MySql.Data.Entity.EF6">
   ```

   It is also possible to create a custom `DbConfiguration` class and add the dependency resolvers needed.

EF6 Features

Following are the new features in Entity Framework 6 implemented in MySQL Connector/Net 6.8:

- *Async Query and Save* adds support for the task-based asynchronous patterns that have been introduced since .NET 4.5. The new asynchronous methods supported by MySQL Connector/Net are:

  - `ExecuteNonQueryAsync`

  - `ExecuteScalarAsync`

  - `PrepareAsync`

- *Connection Resiliency / Retry Logic* enables automatic recovery from transient connection failures. To use this feature, add to the `OnCreateModel` method:

```
SetExecutionStrategy(MySqlProviderInvariantName.ProviderName, () => new MySqlExecutionStrategy());
```

- *Code-Based Configuration* gives you the option of performing configuration in code, instead of performing it in a configuration file, as it has been done traditionally.

- *Dependency Resolution* introduces support for the Service Locator. Some pieces of functionality that can be replaced with custom implementations have been factored out. To add a dependency resolver, use:

```
AddDependencyResolver(new MySqlDependencyResolver());
```

The following resolvers can be added:

- *DbProviderFactory -> MySqlClientFactory*

- *IDbConnectionFactory -> MySqlConnectionFactory*

- *MigrationSqlGenerator -> MySqlMigrationSqlGenerator*

- *DbProviderServices -> MySqlProviderServices*

- *IProviderInvariantName -> MySqlProviderInvariantName*

- *IDbProviderFactoryResolver -> MySqlProviderFactoryResolver*

- *IManifestTokenResolver -> MySqlManifestTokenResolver*

- *IDbModelCacheKey -> MySqlModelCacheKeyFactory*

- *IDbExecutionStrategy -> MySqlExecutionStrategy*

- *Interception/SQL logging* provides low-level building blocks for interception of Entity Framework operations with simple SQL logging built on top:

```
myContext.Database.Log = delegate(string message) { Console.Write(message); };
```

- *DbContext can now be created with a DbConnection that is already opened*, which enables scenarios where it would be helpful if the connection could be open when creating the context (such as sharing a connection between components when you cannot guarantee the state of the connection)

```
[DbConfigurationType(typeof(MySqlEFConfiguration))]
class JourneyContext : DbContext
{
  public DbSet<MyPlace> MyPlaces { get; set; }

  public JourneyContext()
    : base()
  {

  }

  public JourneyContext(DbConnection existingConnection, bool contextOwnsConnection)
    : base(existingConnection, contextOwnsConnection)
  {

  }
}

using (MySqlConnection conn = new MySqlConnection("<connectionString>"))
{
  conn.Open();
  ...

  using (var context = new JourneyContext(conn, false))
  {
    ...
  }
}
```

- *Improved Transaction Support* provides support for a transaction external to the framework as well as improved ways of creating a transaction within the Entity Framework. Starting with Entity Framework 6, `Database.ExecuteSqlCommand()` will wrap by default the command in a transaction if one was not already present. There are overloads of this method that allow users to override this behavior if wished. Execution of stored procedures included in the model through APIs such as `ObjectContext.ExecuteFunction()` does the same. It is also possible to pass an existing transaction to the context.

- *DbSet.AddRange/RemoveRange* provides an optimized way to add or remove multiple entities from a set.

Code First Features

Following are new Code First features supported by MySQL Connector/Net:

- *Code First Mapping to Insert/Update/Delete Stored Procedures* supported:

```
modelBuilder.Entity<EntityType>().MapToStoredProcedures();
```

- *Idempotent migrations scripts* allow you to generate an SQL script that can upgrade a database at any version up to the latest version. To do so, run the *Update-Database -Script -SourceMigration: $InitialDatabase* command in Package Manager Console.

- *Configurable Migrations History Table* allows you to customize the definition of the migrations history table.

Example for Using EF6

Model:

```
using MySql.Data.Entity;
using System.Data.Common;
using System.Data.Entity;

namespace EF6
{
  // Code-Based Configuration and Dependency resolution
  [DbConfigurationType(typeof(MySqlEFConfiguration))]
  public class Parking : DbContext
  {
    public DbSet<Car> Cars { get; set; }

    public Parking()
      : base()
    {

    }

    // Constructor to use on a DbConnection that is already opened
    public Parking(DbConnection existingConnection, bool contextOwnsConnection)
      : base(existingConnection, contextOwnsConnection)
    {

    }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
      base.OnModelCreating(modelBuilder);
      modelBuilder.Entity<Car>().MapToStoredProcedures();
    }
  }

  public class Car
  {
    public int CarId { get; set; }

    public string Model { get; set; }

    public int Year { get; set; }
```

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

Program:

```
using MySql.Data.MySqlClient;
using System;
using System.Collections.Generic;

namespace EF6
{
  class Example
  {
    public static void ExecuteExample()
    {
      string connectionString = "server=localhost;port=3305;database=parking;uid=root;";

      using (MySqlConnection connection = new MySqlConnection(connectionString))
      {
        // Create database if not exists
        using (Parking contextDB = new Parking(connection, false))
        {
          contextDB.Database.CreateIfNotExists();
        }

        connection.Open();
        MySqlTransaction transaction = connection.BeginTransaction();

        try
        {
          // DbConnection that is already opened
          using (Parking context = new Parking(connection, false))
          {

            // Interception/SQL logging
            context.Database.Log = (string message) => { Console.WriteLine(message); };

            // Passing an existing transaction to the context
            context.Database.UseTransaction(transaction);

            // DbSet.AddRange
            List<Car> cars = new List<Car>();

            cars.Add(new Car { Manufacturer = "Nissan", Model = "370Z", Year = 2012 });
            cars.Add(new Car { Manufacturer = "Ford", Model = "Mustang", Year = 2013 });
            cars.Add(new Car { Manufacturer = "Chevrolet", Model = "Camaro", Year = 2012 });
            cars.Add(new Car { Manufacturer = "Dodge", Model = "Charger", Year = 2013 });

            context.Cars.AddRange(cars);

            context.SaveChanges();
          }

          transaction.Commit();
        }
        catch
        {
          transaction.Rollback();
          throw;
        }
      }
    }
  }
}
```

Chapter 10 Connector/Net API Reference

Table of Contents

This section of the manual contains a complete reference to the Connector/Net ADO.NET component, automatically generated from the embedded documentation.

10.1 MySql.Data.MySqlClient Namespace

Namespace hierarchy

Classes

Class	Description
BaseCommandInterceptor	Provides a means of enhancing or replacing SQL commands through the connection string rather than recompiling.
BaseExceptionInterceptor	Provides a means of enabling and disabling exception handling through the connection string rather than recompiling.
MySqlClientPermission	Derived from the .NET DBDataPermission class. For usage information, see Section 5.21, "Working with Partial Trust / Medium Trust".
MySqlCommand	Represents an SQL statement to execute against a MySQL database. This class cannot be inherited.
MySqlCommandBuilder	Automatically generates single-table commands used to reconcile changes made to a DataSet with the associated MySQL database. This class cannot be inherited.
MySqlConnection	Represents an open connection to a MySQL Server database. This class cannot be inherited.
MySqlDataAdapter	Represents a set of data commands and a database connection that are used to fill a dataset and update a MySQL database. This class cannot be inherited.
MySqlDataReader	Provides a means of reading a forward-only stream of rows from a MySQL database. This class cannot be inherited.
MySqlError	Collection of error codes that can be returned by the server
MySqlException	The exception that is thrown when MySQL returns an error. This class cannot be inherited.
MySqlHelper	Helper class that makes it easier to work with the provider.
MySqlInfoMessageEventArgs	Provides data for the InfoMessage event. This class cannot be inherited.
MySqlParameter	Represents a parameter to a MySqlCommand, and optionally, its mapping to DataSetcolumns. This class cannot be inherited.
MySqlParameterCollection	Represents a collection of parameters relevant to a MySqlCommand as well as their respective mappings to columns in a DataSet. This class cannot be inherited.
MySqlRowUpdatedEventArgs	Provides data for the RowUpdated event. This class cannot be inherited.
MySqlRowUpdatingEventArgs	Provides data for the RowUpdating event. This class cannot be inherited.
MySqlTransaction	Represents an SQL transaction to be made in a MySQL database. This class cannot be inherited.

Delegates

Delegate	Description
MySqlInfoMessageEventHandler	Represents the method that will handle the InfoMessage event of a MySqlConnection.
MySqlRowUpdatedEventHandler	Represents the method that will handle the RowUpdatedevent of a MySqlDataAdapter.
MySqlRowUpdatingEventHandler	Represents the method that will handle the RowUpdatingevent of a MySqlDataAdapter.

Enumerations

Enumeration	Description
MySqlDbType	Specifies MySQL-specific data type of a field, property, for use in a MySqlParameter.
MySqlErrorCode	

10.1.1 MySql.Data.MySqlClientHierarchy

See Also

MySql.Data.MySqlClient Namespace

10.1.2 BaseCommandInterceptor Class

The `BaseCommandInterceptor` class has these methods that you can override:

```
public virtual bool ExecuteScalar(string sql, ref object returnValue);
public virtual bool ExecuteNonQuery(string sql, ref int returnValue);
public virtual bool ExecuteReader(string sql, CommandBehavior behavior, ref MySqlDataReader returnValue);
public virtual void Init(MySqlConnection connection);
```

If your interceptor overrides one of the `Execute...` methods, set the `returnValue` output parameter and return `true` if you handled the event, or `false` if you did not handle the event. The SQL command is processed normally only when all command interceptors return `false`.

The connection passed to the `Init` method is the connection that is attached to this interceptor.

For full usage and examples, see Section 5.13, "Using the Connector/Net Interceptor Classes".

10.1.3 BaseExceptionInterceptor Class

You develop an exception interceptor first by creating a subclass of the `BaseExceptionInterceptor` class. You must override the `InterceptException()` method. You can also override the `Init()` method to do some one-time initialization.

Each exception interceptor has 2 methods:

```
public abstract Exception InterceptException(Exception exception,
  MySqlConnection connection);
public virtual void Init(MySqlConnection connection);
```

The connection passed to `Init()` is the connection that is attached to this interceptor.

Each interceptor is required to override `InterceptException` and return an exception. It can return the exception it is given, or it can wrap it in a new exception. We currently do not offer the ability to suppress the exception.

For full usage and examples, see Section 5.13, "Using the Connector/Net Interceptor Classes".

10.1.4 MySqlCommand Class

For a list of all members of this type, see MySqlCommand Members.

Syntax: Visual Basic

```
NotInheritable Public Class MySqlCommand_
  Inherits Component_
  Implements IDbCommand, ICloneable
```

Syntax: C#

```
public sealed class MySqlCommand : Component, IDbCommand, ICloneable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlCommand Members, MySql.Data.MySqlClient Namespace

10.1.4.1 MySqlCommand Members

MySqlCommand overview

Public Instance Constructors

MySqlCommand	Overloaded. Initializes a new instance of the MySqlCommand class.

Public Instance Properties

CommandText	
CommandTimeout	
CommandType	
Connection	
Container (inherited from Component)	Gets the IContainer that contains the Component.
IsPrepared	
Parameters	

Site (inherited from Component)	Gets or sets the ISite of the Component.
Transaction	
UpdatedRowSource	

Public Instance Methods

Cancel	Attempts to cancel the execution of a MySqlCommand. This operation is not supported.
CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
CreateParameter	Creates a new instance of a MySqlParameter object.
Dispose (inherited from Component)	Releases all resources used by the Component.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
ExecuteNonQuery	
ExecuteReader	Overloaded.
ExecuteScalar	
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
Prepare	
ToString (inherited from Component)	Returns a String containing the name of the Component, if any. This method should not be overridden.

Public Instance Events

Disposed (inherited from Component)	Adds an event handler to listen to the Disposed event on the component.

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace

MySqlCommand Constructor

Initializes a new instance of the MySqlCommand class.

Overload List

Initializes a new instance of the MySqlCommand class.

- public MySqlCommand();

- public MySqlCommand(string);

- public MySqlCommand(string,MySqlConnection);

- public MySqlCommand(string,MySqlConnection,MySqlTransaction);

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace

MySqlCommand Constructor ()

Initializes a new instance of the MySqlCommand class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlCommand();
```

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace, MySqlCommand Constructor Overload List

MySqlCommand Constructor (String)

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal cmdText As String _
)
```

Syntax: C#

```
public MySqlCommand(
   string cmdText
);
```

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace, MySqlCommand Constructor Overload List

MySqlCommand Constructor (string, MySqlConnection)

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal cmdText As String, _
   ByVal connection As MySqlConnection _
)
```

Syntax: C#

```
public MySqlCommand(
   string cmdText,
   MySqlConnection connection
);
```

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace, MySqlCommand Constructor Overload List

MySqlCommand Constructor (string, MySqlConnection, MySqlTransaction)

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal cmdText As String, _
   ByVal connection As MySqlConnection, _
   ByVal transaction As MySqlTransaction _
)
```

Syntax: C#

```
public MySqlCommand(
   string cmdText,
   MySqlConnection connection,
   MySqlTransaction transaction
);
```

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace, MySqlCommand Constructor Overload List

CommandText Property

Syntax: Visual Basic

```
NotOverridable Public Property CommandText As String _
_
  Implements IDbCommand.CommandText
```

Syntax: C#

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

Implements

IDbCommand.CommandText

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace

CommandTimeout Property

Syntax: Visual Basic

```
NotOverridable Public Property CommandTimeout As Integer _
  Implements IDbCommand.CommandTimeout
```

Syntax: C#

```
public int CommandTimeout {get; set;}
```

Implements

IDbCommand.CommandTimeout

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace

CommandType Property

Syntax: Visual Basic

```
NotOverridable Public Property CommandType As CommandType _
   Implements IDbCommand.CommandType
```

Syntax: C#

```
public System.Data.CommandType CommandType {get; set;}
```

Implements

IDbCommand.CommandType

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace

Connection Property

Syntax: Visual Basic

```
Public Property Connection As MySqlConnection
```

Syntax: C#

```
public MySqlConnection Connection {get; set;}
```

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace

IsPrepared Property

Syntax: Visual Basic

```
Public ReadOnly Property IsPrepared As Boolean
```

Syntax: C#

```
public bool IsPrepared {get;}
```

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace

Parameters Property

Syntax: Visual Basic

```
Public ReadOnly Property Parameters As MySqlParameterCollection
```

Syntax: C#

```
public MySqlParameterCollection Parameters {get;}
```

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace

Transaction Property

Syntax: Visual Basic

```
Public Property Transaction As MySqlTransaction
```

Syntax: C#

```
public MySqlTransaction Transaction {get; set;}
```

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace

UpdatedRowSource Property

Syntax: Visual Basic

```
NotOverridable Public Property UpdatedRowSource As UpdateRowSource _
  Implements IDbCommand.UpdatedRowSource
```

Syntax: C#

```
public System.Data.UpdateRowSource UpdatedRowSource {get; set;}
```

Implements

IDbCommand.UpdatedRowSource

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace

MySqlCommand.Cancel Method

Attempts to cancel the execution of a MySqlCommand. This operation is not supported.

Syntax: Visual Basic

```
NotOverridable Public Sub Cancel() _
  Implements IDbCommand.Cancel
```

Syntax: C#

```
public void Cancel();
```

Implements

IDbCommand.Cancel

Remarks

Cancelling an executing command is currently not supported on any version of MySQL.

Exceptions

Exception Type	Condition
NotSupportedException	This operation is not supported.

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace

MySqlCommand.CreateParameter Method

Creates a new instance of a MySqlParameter object.

Syntax: Visual Basic

```
Public Function CreateParameter() As MySqlParameter
```

Syntax: C#

```
public MySqlParameter CreateParameter();
```

Return Value

A MySqlParameter object.

Remarks

This method is a strongly-typed version of CreateParameter.

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace

MySqlCommand.ExecuteNonQuery Method

Syntax: Visual Basic

```
NotOverridable Public Function ExecuteNonQuery() As Integer _
   Implements IDbCommand.ExecuteNonQuery
```

Syntax: C#

```
public int ExecuteNonQuery();
```

Implements

IDbCommand.ExecuteNonQuery

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace

ExecuteReader Method

Overload List

- public MySqlDataReader ExecuteReader();

- public MySqlDataReader ExecuteReader(CommandBehavior);

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace

MySqlCommand.ExecuteReader Method

Syntax: Visual Basic

```
Overloads Public Function ExecuteReader() As MySqlDataReader
```

Syntax: C#

```
public MySqlDataReader ExecuteReader();
```

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace, MySqlCommand.ExecuteReader Overload List

MySqlCommand.ExecuteReader Method

Syntax: Visual Basic

```
Overloads Public Function ExecuteReader( _
   ByVal behavior As CommandBehavior _
) As MySqlDataReader
```

Syntax: C#

```
public MySqlDataReader ExecuteReader(
   CommandBehavior behavior
);
```

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace, MySqlCommand.ExecuteReader Overload List

MySqlCommand.ExecuteScalar Method

Syntax: Visual Basic

```
NotOverridable Public Function ExecuteScalar() As Object _
  Implements IDbCommand.ExecuteScalar
```

Syntax: C#

```
public object ExecuteScalar();
```

Implements

IDbCommand.ExecuteScalar

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace

MySqlCommand.Prepare Method

Syntax: Visual Basic

```
NotOverridable Public Sub Prepare() _
  Implements IDbCommand.Prepare
```

Syntax: C#

```
public void Prepare();
```

Implements

IDbCommand.Prepare

See Also

MySqlCommand Class, MySql.Data.MySqlClient Namespace

10.1.5 MySqlCommandBuilder Class

For a list of all members of this type, see MySqlCommandBuilder Members .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlCommandBuilder _
  Inherits Component
```

Syntax: C#

```
public sealed class MySqlCommandBuilder : Component
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlCommandBuilder Members, MySql.Data.MySqlClient Namespace

10.1.5.1 MySqlCommandBuilder Members

MySqlCommandBuilder overview

Public Static (Shared) Methods

DeriveParameters	Overloaded. Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySql.

Public Instance Constructors

MySqlCommandBuilder	Overloaded. Initializes a new instance of the MySqlCommandBuilder class.

Public Instance Properties

Container (inherited from Component)	Gets the IContainer that contains the Component.
DataAdapter	
QuotePrefix	
QuoteSuffix	
Site (inherited from Component)	Gets or sets the ISite of the Component.

Public Instance Methods

CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Dispose (inherited from Component)	Releases all resources used by the Component.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetDeleteCommand	

GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetInsertCommand	
GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
GetUpdateCommand	
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
RefreshSchema	
ToString (inherited from Component)	Returns a String containing the name of the Component, if any. This method should not be overridden.

Public Instance Events

Disposed (inherited from Component)	Adds an event handler to listen to the Disposed event on the component.

See Also

MySqlCommandBuilder Class, MySql.Data.MySqlClient Namespace

DeriveParameters Method

Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySql.

Overload List

Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySql.

- public static void DeriveParameters(MySqlCommand);

- public static void DeriveParameters(MySqlCommand,bool);

See Also

MySqlCommandBuilder Class, MySql.Data.MySqlClient Namespace

MySqlCommandBuilder.DeriveParameters Method

Retrieves parameter information from the stored procedure specified in the MySqlCommand and populates the Parameters collection of the specified MySqlCommand object. This method is not currently supported since stored procedures are not available in MySql.

Syntax: Visual Basic

```
Overloads Public Shared Sub DeriveParameters( _
   ByVal command As MySqlCommand _
)
```

Syntax: C#

```
public static void DeriveParameters(
   MySqlCommand command
);
```

Parameters

- command: The MySqlCommand referencing the stored procedure from which the parameter information is to be derived. The derived parameters are added to the Parameters collection of the MySqlCommand.

Exceptions

Exception Type	Condition
InvalidOperationException	The command text is not a valid stored procedure name.

See Also

MySqlCommandBuilder Class, MySql.Data.MySqlClient Namespace, MySqlCommandBuilder.DeriveParameters Overload List

MySqlCommandBuilder.DeriveParameters Method

Syntax: Visual Basic

```
Overloads Public Shared Sub DeriveParameters( _
   ByVal command As MySqlCommand, _
   ByVal useProc As Boolean _
)
```

Syntax: C#

```
public static void DeriveParameters(
   MySqlCommand command,
   bool useProc
);
```

See Also

MySqlCommandBuilder Class, MySql.Data.MySqlClient Namespace, MySqlCommandBuilder.DeriveParameters Overload List

MySqlCommandBuilder Constructor

Initializes a new instance of the MySqlCommandBuilder class.

Overload List

Initializes a new instance of the MySqlCommandBuilder class.

- public MySqlCommandBuilder();

- public MySqlCommandBuilder(MySqlDataAdapter);

- public MySqlCommandBuilder(MySqlDataAdapter,bool);

- public MySqlCommandBuilder(bool);

See Also

MySqlCommandBuilder Class, MySql.Data.MySqlClient Namespace

MySqlCommandBuilder Constructor ()

Initializes a new instance of the MySqlCommandBuilder class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlCommandBuilder();
```

See Also

MySqlCommandBuilder Class, MySql.Data.MySqlClient Namespace, MySqlCommandBuilder Constructor Overload List

MySqlCommandBuilder Constructor (MySqlDataAdapter)

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal adapter As MySqlDataAdapter _
)
```

Syntax: C#

```
public MySqlCommandBuilder(
   MySqlDataAdapter adapter
);
```

See Also

MySqlCommandBuilder Class, MySql.Data.MySqlClient Namespace, MySqlCommandBuilder Constructor Overload List

MySqlCommandBuilder Constructor (MySqlDataAdapter, bool)

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal adapter As MySqlDataAdapter, _
   ByVal lastOneWins As Boolean _
)
```

Syntax: C#

```
public MySqlCommandBuilder(
   MySqlDataAdapter adapter,
   bool lastOneWins
);
```

See Also

MySqlCommandBuilder Class, MySql.Data.MySqlClient Namespace, MySqlCommandBuilder Constructor Overload List

MySqlCommandBuilder Constructor (bool)

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal lastOneWins As Boolean _
)
```

Syntax: C#

```
public MySqlCommandBuilder(
   bool lastOneWins
);
```

See Also

MySqlCommandBuilder Class, MySql.Data.MySqlClient Namespace, MySqlCommandBuilder Constructor Overload List

DataAdapter Property

Syntax: Visual Basic

```
Public Property DataAdapter As MySqlDataAdapter
```

Syntax: C#

```
public MySqlDataAdapter DataAdapter {get; set;}
```

See Also

MySqlCommandBuilder Class, MySql.Data.MySqlClient Namespace

QuotePrefix Property

Syntax: Visual Basic

```
Public Property QuotePrefix As String
```

Syntax: C#

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

See Also

QuoteSuffix Property

Syntax: Visual Basic

```
Public Property QuoteSuffix As String
```

Syntax: C#

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

See Also

MySqlCommandBuilder.GetDeleteCommand Method

Syntax: Visual Basic

```
Public Function GetDeleteCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand GetDeleteCommand();
```

See Also

MySqlCommandBuilder.GetInsertCommand Method

Syntax: Visual Basic

```
Public Function GetInsertCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand GetInsertCommand();
```

See Also

MySqlCommandBuilder.GetUpdateCommand Method

Syntax: Visual Basic

```
Public Function GetUpdateCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand GetUpdateCommand();
```

See Also

MySqlCommandBuilder Class, MySql.Data.MySqlClient Namespace

MySqlCommandBuilder.RefreshSchema Method

Syntax: Visual Basic

```
Public Sub RefreshSchema()
```

Syntax: C#

```
public void RefreshSchema();
```

See Also

MySqlCommandBuilder Class, MySql.Data.MySqlClient Namespace

10.1.6 MySqlConnection Class

For a list of all members of this type, see MySqlConnection Members.

Syntax: Visual Basic

```
NotInheritable Public Class MySqlConnection_
  Inherits Component_
  Implements IDbConnection, ICloneable
```

Syntax: C#

```
public sealed class MySqlConnection : Component, IDbConnection, ICloneable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlConnection Members, MySql.Data.MySqlClient Namespace

10.1.6.1 MySqlConnection Members

MySqlConnection overview

Public Instance Constructors

MySqlConnection	Overloaded. Initializes a new instance of the MySqlConnection class.

Public Instance Properties

ConnectionString	
ConnectionTimeout	
Container (inherited from Component)	Gets the IContainer that contains the Component.
Database	
DataSource	Gets the name of the MySQL server to which to connect.
ServerThread	Returns the ID of the server thread this connection is executing on.
ServerVersion	
Site (inherited from Component)	Gets or sets the ISite of the Component.
State	
UseCompression	Indicates if this connection should use compression when communicating with the server.

Public Instance Methods

BeginTransaction	Overloaded.
ChangeDatabase	
Close	
CreateCommand	
CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Dispose (inherited from Component)	Releases all resources used by the Component.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
Open	
Ping	Ping
ToString (inherited from Component)	Returns a String containing the name of the Component, if any. This method should not be overridden.

Public Instance Events

Disposed (inherited from Component)	Adds an event handler to listen to the Disposed event on the component.
InfoMessage	
StateChange	

See Also

MySqlConnection Class, MySql.Data.MySqlClient Namespace

MySqlConnection Constructor

Initializes a new instance of the MySqlConnection class.

Overload List

Initializes a new instance of the MySqlConnection class.

- public MySqlConnection();

- public MySqlConnection(string);

See Also

MySqlConnection Class, MySql.Data.MySqlClient Namespace

MySqlConnection Constructor

Initializes a new instance of the MySqlConnection class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlConnection();
```

See Also

MySqlConnection Class, MySql.Data.MySqlClient Namespace, MySqlConnection Constructor Overload List

MySqlConnection Constructor (string)

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal connectionString As String _
)
```

Syntax: C#

```
public MySqlConnection(
   string connectionString
```

```
);
```

See Also

MySqlConnection Class, MySql.Data.MySqlClient Namespace, MySqlConnection Constructor Overload List

ConnectionString Property

Syntax: Visual Basic

```
NotOverridable Public Property ConnectionString As String _
   Implements IDbConnection.ConnectionString
```

Syntax: C#

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

Implements

IDbConnection.ConnectionString

See Also

MySqlConnection Class, MySql.Data.MySqlClient Namespace

ConnectionTimeout Property

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property ConnectionTimeout As Integer _
   Implements IDbConnection.ConnectionTimeout
```

Syntax: C#

```
public int ConnectionTimeout {get;}
```

Implements

IDbConnection.ConnectionTimeout

See Also

MySqlConnection Class, MySql.Data.MySqlClient Namespace

Database Property

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property Database As String _
   Implements IDbConnection.Database
```

Syntax: C#

```
public string Database {get;}
```

Implements

IDbConnection.Database

See Also

MySqlConnection Class, MySql.Data.MySqlClient Namespace

DataSource Property

Gets the name of the MySQL server to which to connect.

Syntax: Visual Basic

```
Public ReadOnly Property DataSource As String
```

Syntax: C#

```
public string DataSource {get;}
```

See Also

MySqlConnection Class, MySql.Data.MySqlClient Namespace

ServerThread Property

Returns the ID of the server thread on which this connection is executing.

Syntax: Visual Basic

```
Public ReadOnly Property ServerThread As Integer
```

Syntax: C#

```
public int ServerThread {get;}
```

See Also

MySqlConnection Class, MySql.Data.MySqlClient Namespace

ServerVersion Property

Syntax: Visual Basic

```
Public ReadOnly Property ServerVersion As String
```

Syntax: C#

```
public string ServerVersion {get;}
```

See Also

MySqlConnection Class, MySql.Data.MySqlClient Namespace

State Property

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property State As ConnectionState _
  Implements IDbConnection.State
```

Syntax: C#

```
public System.Data.ConnectionState State {get;}
```

Implements

IDbConnection.State

See Also

MySqlConnection Class, MySql.Data.MySqlClient Namespace

UseCompression Property

Indicates if this connection should use compression when communicating with the server.

Syntax: Visual Basic

```
Public ReadOnly Property UseCompression As Boolean
```

Syntax: C#

```
public bool UseCompression {get;}
```

See Also

MySqlConnection Class, MySql.Data.MySqlClient Namespace

BeginTransaction Method

Overload List

- public MySqlTransaction BeginTransaction();

- public MySqlTransaction BeginTransaction(IsolationLevel);

See Also

MySqlConnection Class, MySql.Data.MySqlClient Namespace

MySqlConnection.BeginTransaction Method

Syntax: Visual Basic

```
Overloads Public Function BeginTransaction() As MySqlTransaction
```

Syntax: C#

```
public MySqlTransaction BeginTransaction();
```

See Also

MySqlConnection Class, MySql.Data.MySqlClient Namespace, MySqlConnection.BeginTransaction Overload List

MySqlConnection.BeginTransaction Method (IsolationLevel)

Syntax: Visual Basic

```
Overloads Public Function BeginTransaction( _
   ByVal iso As IsolationLevel _
) As MySqlTransaction
```

Syntax: C#

```
public MySqlTransaction BeginTransaction(
   IsolationLevel iso
);
```

See Also

MySqlConnection Class, MySql.Data.MySqlClient Namespace, MySqlConnection.BeginTransaction Overload List

MySqlConnection.ChangeDatabase Method

Syntax: Visual Basic

```
NotOverridable Public Sub ChangeDatabase( _
   ByVal databaseName As String _
) _
   Implements IDbConnection.ChangeDatabase
```

Syntax: C#

```
public void ChangeDatabase(
   string databaseName
);
```

Implements

IDbConnection.ChangeDatabase

See Also

MySqlConnection Class, MySql.Data.MySqlClient Namespace

MySqlConnection.Close Method

Syntax: Visual Basic

```
NotOverridable Public Sub Close() _
  Implements IDbConnection.Close
```

Syntax: C#

```
public void Close();
```

Implements

IDbConnection.Close

See Also

[MySqlConnection Class](), [MySql.Data.MySqlClient Namespace]()

MySqlConnection.CreateCommand Method

Syntax: Visual Basic

```
Public Function CreateCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand CreateCommand();
```

See Also

[MySqlConnection Class](), [MySql.Data.MySqlClient Namespace]()

MySqlConnection.Open Method

Syntax: Visual Basic

```
NotOverridable Public Sub Open() _
  Implements IDbConnection.Open
```

Syntax: C#

```
public void Open();
```

Implements

IDbConnection.Open

See Also

[MySqlConnection Class](), [MySql.Data.MySqlClient Namespace]()

MySqlConnection.Ping Method

Ping

Syntax: Visual Basic

```
Public Function Ping() As Boolean
```

Syntax: C#

```
public bool Ping();
```

Return Value

See Also

MySqlConnection Class, MySql.Data.MySqlClient Namespace

MySqlConnection.InfoMessage Event

Syntax: Visual Basic

```
Public Event InfoMessage As MySqlInfoMessageEventHandler
```

Syntax: C#

```
public event MySqlInfoMessageEventHandler InfoMessage;
```

See Also

MySqlConnection Class, MySql.Data.MySqlClient Namespace

MySqlConnection.StateChange Event

Syntax: Visual Basic

```
Public Event StateChange As StateChangeEventHandler
```

Syntax: C#

```
public event StateChangeEventHandler StateChange;
```

See Also

MySqlConnection Class, MySql.Data.MySqlClient Namespace

10.1.7 MySqlDataAdapter Class

For a list of all members of this type, see MySqlDataAdapter Members .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlDataAdapter_
  Inherits DbDataAdapter
```

Syntax: C#

```
public sealed class MySqlDataAdapter : DbDataAdapter
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlDataAdapter Members, MySql.Data.MySqlClient Namespace

10.1.7.1 MySqlDataAdapter Members

MySqlDataAdapter overview

Public Instance Constructors

| MySqlDataAdapter | Overloaded. Initializes a new instance of the MySqlDataAdapter class. |

Public Instance Properties

AcceptChangesDuringFill (inherited from DataAdapter)	Gets or sets a value indicating whether AcceptChangesis called on a DataRowafter it is added to the DataTableduring any of the Fill operations.
AcceptChangesDuringUpdate (inherited from DataAdapter)	Gets or sets whether AcceptChangesis called during a Update.
Container (inherited from Component)	Gets the IContainer that contains the Component.
ContinueUpdateOnError (inherited from DataAdapter)	Gets or sets a value that specifies whether to generate an exception when an error is encountered during a row update.
DeleteCommand	Overloaded.
FillLoadOption (inherited from DataAdapter)	Gets or sets the LoadOption that determines how the adapter fills the DataTablefrom the DbDataReader.
InsertCommand	Overloaded.
MissingMappingAction (inherited from DataAdapter)	Determines the action to take when incoming data does not have a matching table or column.
MissingSchemaAction (inherited from DataAdapter)	Determines the action to take when existing DataSetschema does not match incoming data.
ReturnProviderSpecificTypes (inherited from DataAdapter)	Gets or sets whether the Fillmethod should return provider-specific values or common CLS-compliant values.
SelectCommand	Overloaded.
Site (inherited from Component)	Gets or sets the ISite of the Component.
TableMappings (inherited from DataAdapter)	Gets a collection that provides the master mapping between a source table and a DataTable.

UpdateBatchSize (inherited from DbDataAdapter)	Gets or sets a value that enables or disables batch processing support, and specifies the number of commands that can be executed in a batch.
UpdateCommand	Overloaded.

Public Instance Methods

CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Dispose (inherited from Component)	Releases all resources used by the Component.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
Fill (inherited from DbDataAdapter)	Overloaded. Adds or refreshes rows in the DataSetto match those in the data source using the DataSetname, and creates a DataTablenamed "Table."
FillSchema (inherited from DbDataAdapter)	Overloaded. Configures the schema of the specified DataTablebased on the specified SchemaType.
GetFillParameters (inherited from DbDataAdapter)	Gets the parameters set by the user when executing an SQL `SELECT` statement.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
ResetFillLoadOption (inherited from DataAdapter)	Resets FillLoadOptionto its default state and causes Fillto honor AcceptChangesDuringFill.
ShouldSerializeAcceptChangesDuringFill (inherited from DataAdapter)	Determines whether the AcceptChangesDuringFillproperty should be persisted.
ShouldSerializeFillLoadOption (inherited from DataAdapter)	Determines whether the FillLoadOptionproperty should be persisted.
ToString (inherited from Component)	Returns a String containing the name of the Component, if any. This method should not be overridden.
Update (inherited from DbDataAdapter)	Overloaded. Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the specified DataSet.

Public Instance Events

Disposed (inherited from Component)	Adds an event handler to listen to the Disposed event on the component.

FillError (inherited from DataAdapter)	Returned when an error occurs during a fill operation.
RowUpdated	Occurs during Update after a command is executed against the data source. The attempt to update is made, so the event fires.
RowUpdating	Occurs during Update before a command is executed against the data source. The attempt to update is made, so the event fires.

Protected Internal Instance Properties

FillCommandBehavior (inherited from DbDataAdapter)	Gets or sets the behavior of the command used to fill the data adapter.

See Also

MySqlDataAdapter Class, MySql.Data.MySqlClient Namespace

MySqlDataAdapter Constructor

Initializes a new instance of the MySqlDataAdapter class.

Overload List

Initializes a new instance of the MySqlDataAdapter class.

- public MySqlDataAdapter();

- public MySqlDataAdapter(MySqlCommand);

- public MySqlDataAdapter(string,MySqlConnection);

- public MySqlDataAdapter(string,string);

See Also

MySqlDataAdapter Class, MySql.Data.MySqlClient Namespace

MySqlDataAdapter Constructor ()

Initializes a new instance of the MySqlDataAdapter class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlDataAdapter();
```

See Also

MySqlDataAdapter Class, MySql.Data.MySqlClient Namespace, MySqlDataAdapter Constructor Overload List

MySqlDataAdapter Constructor (MySqlCommand)

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal selectCommand As MySqlCommand _
)
```

Syntax: C#

```
public MySqlDataAdapter(
   MySqlCommand selectCommand
);
```

See Also

MySqlDataAdapter Class, MySql.Data.MySqlClient Namespace, MySqlDataAdapter Constructor Overload List

MySqlDataAdapter Constructor (string, MySqlConnection)

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal selectCommandText As String, _
   ByVal connection As MySqlConnection _
)
```

Syntax: C#

```
public MySqlDataAdapter(
   string selectCommandText,
   MySqlConnection connection
);
```

See Also

MySqlDataAdapter Class, MySql.Data.MySqlClient Namespace, MySqlDataAdapter Constructor Overload List

MySqlDataAdapter Constructor (string, string)

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal selectCommandText As String, _
   ByVal selectConnString As String _
)
```

Syntax: C#

```
public MySqlDataAdapter(
   string selectCommandText,
   string selectConnString
);
```

See Also

MySqlDataAdapter Class, MySql.Data.MySqlClient Namespace, MySqlDataAdapter Constructor Overload List

DeleteCommand Property

Syntax: Visual Basic

```
Overloads Public Property DeleteCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand DeleteCommand {get; set;}
```

See Also

MySqlDataAdapter Class, MySql.Data.MySqlClient Namespace

InsertCommand Property

Syntax: Visual Basic

```
Overloads Public Property InsertCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand InsertCommand {get; set;}
```

See Also

MySqlDataAdapter Class, MySql.Data.MySqlClient Namespace

SelectCommand Property

Syntax: Visual Basic

```
Overloads Public Property SelectCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand SelectCommand {get; set;}
```

See Also

MySqlDataAdapter Class, MySql.Data.MySqlClient Namespace

UpdateCommand Property

Syntax: Visual Basic

```
Overloads Public Property UpdateCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand UpdateCommand {get; set;}
```

MySqlDataAdapter.RowUpdated Event

Occurs during Update after a command is executed against the data source. The attempt to update is made, so the event fires.

Syntax: Visual Basic

```
Public Event RowUpdated As MySqlRowUpdatedEventHandler
```

Syntax: C#

```
public event MySqlRowUpdatedEventHandler RowUpdated;
```

Event Data

The event handler receives an argument of type MySqlRowUpdatedEventArgs containing data related to this event. The following `MySqlRowUpdatedEventArgsproperties` provide information specific to this event.

Property	Description
Command	Gets or sets the MySqlCommand executed when Update is called.
Errors	Gets any errors generated by the .NET Framework data provider when the Commandwas executed.
RecordsAffected	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.
Row	Gets the DataRowsent through an Update.
RowCount	Gets the number of rows processed in a batch of updated records.
StatementType	Gets the type of SQL statement executed.
Status	Gets the UpdateStatus of the Commandproperty.
TableMapping	Gets the DataTableMappingsent through an Update.

MySqlDataAdapter.RowUpdating Event

Occurs during Update before a command is executed against the data source. The attempt to update is made, so the event fires.

Syntax: Visual Basic

```
Public Event RowUpdating As MySqlRowUpdatingEventHandler
```

Syntax: C#

```
public event MySqlRowUpdatingEventHandler RowUpdating;
```

Event Data

The event handler receives an argument of type MySqlRowUpdatingEventArgs containing data related to this event. The following `MySqlRowUpdatingEventArgsproperties` provide information specific to this event.

Property	Description
Command	Gets or sets the MySqlCommand to execute when performing the Update.
Errors	Gets any errors generated by the .NET Framework data provider when the Command executes.
Row	Gets the DataRow that will be sent to the server as part of an insert, update, or delete operation.
StatementType	Gets the type of SQL statement to execute.
Status	Gets or sets the UpdateStatus of the Commandproperty.
TableMapping	Gets the DataTableMapping to send through the Update.

See Also

MySqlDataAdapter Class, MySql.Data.MySqlClient Namespace

10.1.8 MySqlDataReader Class

Provides a means of reading a forward-only stream of rows from a MySQL database. This class cannot be inherited.

For a list of all members of this type, see MySqlDataReader Members.

Syntax: Visual Basic

```
NotInheritable Public Class MySqlDataReader_
  Inherits MarshalByRefObject_
  Implements IEnumerable, IDataReader, IDisposable, IDataRecord
```

Syntax: C#

```
public sealed class MySqlDataReader : MarshalByRefObject, IEnumerable, IDataReader, IDisposable, IDataRecord
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

10.1.8.1 MySqlDataReader Members

MySqlDataReader overview

Public Instance Properties

Depth	Gets a value indicating the depth of nesting for the current row. This method is not supported currently and always returns 0.
FieldCount	Gets the number of columns in the current row.
HasRows	Gets a value indicating whether the MySqlDataReader contains one or more rows.
IsClosed	Gets a value indicating whether the data reader is closed.
Item	Overloaded. Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the MySqlDataReader class.
RecordsAffected	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.

Public Instance Methods

Close	Closes the MySqlDataReader object.
CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBoolean	Gets the value of the specified column as a Boolean.
GetByte	Gets the value of the specified column as a byte.
GetBytes	Reads a stream of bytes from the specified column offset into the buffer an array starting at the given buffer offset.
GetChar	Gets the value of the specified column as a single character.
GetChars	Reads a stream of characters from the specified column offset into the buffer as an array starting at the given buffer offset.
GetDataTypeName	Gets the name of the source data type.
GetDateTime	
GetDecimal	
GetDouble	
GetFieldType	Gets the Type that is the data type of the object.
GetFloat	

GetGuid	
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetInt16	
GetInt32	
GetInt64	
GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetMySqlDateTime	
GetName	Gets the name of the specified column.
GetOrdinal	Gets the column ordinal, given the name of the column.
GetSchemaTable	Returns a DataTable that describes the column metadata of the MySqlDataReader.
GetString	
GetTimeSpan	
GetType (inherited from Object)	Gets the Type of the current instance.
GetUInt16	
GetUInt32	
GetUInt64	
GetValue	Gets the value of the specified column in its native format.
GetValues	Gets all attribute columns in the collection for the current row.
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
IsDBNull	Gets a value indicating whether the column contains non-existent or missing values.
NextResult	Advances the data reader to the next result, when reading the results of batch SQL statements.
Read	Advances the MySqlDataReader to the next record.
ToString (inherited from Object)	Returns a String that represents the current Object.

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

Depth Property

Gets a value indicating the depth of nesting for the current row. This method is not supported currently and always returns 0.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property Depth As Integer _
```

```
    Implements IDataReader.Depth
```

Syntax: C#

```
public int Depth {get;}
```

Implements

IDataReader.Depth

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

FieldCount Property

Gets the number of columns in the current row.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property FieldCount As Integer _
  Implements IDataRecord.FieldCount
```

Syntax: C#

```
public int FieldCount {get;}
```

Implements

IDataRecord.FieldCount

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

HasRows Property

Gets a value indicating whether the MySqlDataReader contains one or more rows.

Syntax: Visual Basic

```
Public ReadOnly Property HasRows As Boolean
```

Syntax: C#

```
public bool HasRows {get;}
```

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

IsClosed Property

Gets a value indicating whether the data reader is closed.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property IsClosed As Boolean _
   Implements IDataReader.IsClosed
```

Syntax: C#

```
public bool IsClosed {get;}
```

Implements

IDataReader.IsClosed

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

Item Property

Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the MySqlDataReader class.

Overload List

Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the MySqlDataReader class.

- public object this[int] {get;}

Gets the value of a column in its native format. In C#, this property is the indexer for the MySqlDataReader class.

- public object this[string] {get;}

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

Item Property (Int32)

Overloaded. Gets the value of a column in its native format. In C#, this property is the indexer for the MySqlDataReader class.

Syntax: Visual Basic

```
NotOverridable Overloads Public Default ReadOnly Property Item( _
   ByVal i As Integer _
) _
   Implements IDataRecord.Item As Object _
   Implements IDataRecord.Item
```

Syntax: C#

```
public object this[
   int i
] {get;}
```

Implements

IDataRecord.Item

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace, MySqlDataReader.Item Overload List

Item Property (String)

Gets the value of a column in its native format. In C#, this property is the indexer for the MySqlDataReader class.

Syntax: Visual Basic

```
NotOverridable Overloads Public Default ReadOnly Property Item( _
   ByVal name As String _
) _
  Implements IDataRecord.Item As Object _
  Implements IDataRecord.Item
```

Syntax: C#

```
public object this[
   string name
] {get;}
```

Implements

IDataRecord.Item

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace, MySqlDataReader.Item Overload List

RecordsAffected Property

Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property RecordsAffected As Integer _
  Implements IDataReader.RecordsAffected
```

Syntax: C#

```
public int RecordsAffected {get;}
```

Implements

IDataReader.RecordsAffected

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.Close Method

Closes the MySqlDataReader object.

```
NotOverridable Public Sub Close() _
  Implements IDataReader.Close
```

Syntax: C#

```
public void Close();
```

Implements

IDataReader.Close

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.GetBoolean Method

Gets the value of the specified column as a Boolean.

Syntax: Visual Basic

```
NotOverridable Public Function GetBoolean( _
   ByVal i As Integer _
) As Boolean _
  Implements IDataRecord.GetBoolean
```

Syntax: C#

```
public bool GetBoolean(
   int i
);
```

Parameters

- i:

Return Value

Implements

IDataRecord.GetBoolean

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.GetByte Method

Gets the value of the specified column as a byte.

Syntax: Visual Basic

```
NotOverridable Public Function GetByte( _
   ByVal i As Integer _
) As Byte _
```

```
   Implements IDataRecord.GetByte
```

Syntax: C#

```
public byte GetByte(
   int i
);
```

Parameters

- `i`:

Return Value

Implements

IDataRecord.GetByte

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.GetBytes Method

Reads a stream of bytes from the specified column offset into the buffer an array starting at the given buffer offset.

Syntax: Visual Basic

```
NotOverridable Public Function GetBytes( _
   ByVal i As Integer, _
   ByVal dataIndex As Long, _
   ByVal buffer As Byte(), _
   ByVal bufferIndex As Integer, _
   ByVal length As Integer _
) As Long _
   Implements IDataRecord.GetBytes
```

Syntax: C#

```
public long GetBytes(
   int i,
   long dataIndex,
   byte[] buffer,
   int bufferIndex,
   int length
);
```

Parameters

- `i`: The zero-based column ordinal.

- `dataIndex`: The index within the field from which to begin the read operation.

- `buffer`: The buffer into which to read the stream of bytes.

- `bufferIndex`: The index for buffer to begin the read operation.

- `length`: The maximum length to copy into the buffer.

Return Value

The actual number of bytes read.

Implements

IDataRecord.GetBytes

See Also

,

MySqlDataReader.GetChar Method

Gets the value of the specified column as a single character.

Syntax: Visual Basic

```
NotOverridable Public Function GetChar( _
   ByVal i As Integer _
) As Char _
  Implements IDataRecord.GetChar
```

Syntax: C#

```
public char GetChar(
   int i
);
```

Parameters

- i:

Return Value

Implements

IDataRecord.GetChar

See Also

,

MySqlDataReader.GetChars Method

Reads a stream of characters from the specified column offset into the buffer as an array starting at the given buffer offset.

Syntax: Visual Basic

```
NotOverridable Public Function GetChars( _
   ByVal i As Integer, _
   ByVal fieldOffset As Long, _
   ByVal buffer As Char(), _
   ByVal bufferoffset As Integer, _
   ByVal length As Integer _
) As Long _
```

```
  Implements IDataRecord.GetChars
```

Syntax: C#

```
public long GetChars(
   int i,
   long fieldOffset,
   char[] buffer,
   int bufferoffset,
   int length
);
```

Parameters

- i:

- fieldOffset:

- buffer:

- bufferoffset:

- length:

Return Value

Implements

IDataRecord.GetChars

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.GetDataTypeName Method

Gets the name of the source data type.

Syntax: Visual Basic

```
NotOverridable Public Function GetDataTypeName( _
   ByVal i As Integer _
) As String _
   Implements IDataRecord.GetDataTypeName
```

Syntax: C#

```
public string GetDataTypeName(
   int i
);
```

Parameters

- i:

Return Value

Implements

IDataRecord.GetDataTypeName

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.GetDateTime Method

Syntax: Visual Basic

```
NotOverridable Public Function GetDateTime( _
   ByVal index As Integer _
) As Date _
   Implements IDataRecord.GetDateTime
```

Syntax: C#

```
public DateTime GetDateTime(
   int index
);
```

Implements

IDataRecord.GetDateTime

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.GetDecimal Method

Syntax: Visual Basic

```
NotOverridable Public Function GetDecimal( _
   ByVal index As Integer _
) As Decimal _
   Implements IDataRecord.GetDecimal
```

Syntax: C#

```
public decimal GetDecimal(
   int index
);
```

Implements

IDataRecord.GetDecimal

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.GetDouble Method

Syntax: Visual Basic

```
NotOverridable Public Function GetDouble( _
   ByVal index As Integer _
) As Double _
  Implements IDataRecord.GetDouble
```

Syntax: C#

```
public double GetDouble(
   int index
);
```

Implements

IDataRecord.GetDouble

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.GetFieldType Method

Gets the Type that is the data type of the object.

Syntax: Visual Basic

```
NotOverridable Public Function GetFieldType( _
   ByVal i As Integer _
) As Type _
  Implements IDataRecord.GetFieldType
```

Syntax: C#

```
public Type GetFieldType(
   int i
);
```

Parameters

- i:

Return Value

Implements

IDataRecord.GetFieldType

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.GetFloat Method

Syntax: Visual Basic

```
NotOverridable Public Function GetFloat( _
   ByVal index As Integer _
) As Single _
```

```
    Implements IDataRecord.GetFloat
```

Syntax: C#

```
public float GetFloat(
    int index
);
```

Implements

IDataRecord.GetFloat

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.GetGuid Method

Syntax: Visual Basic

```
NotOverridable Public Function GetGuid( _
    ByVal index As Integer _
) As Guid _
    Implements IDataRecord.GetGuid
```

Syntax: C#

```
public Guid GetGuid(
    int index
);
```

Implements

IDataRecord.GetGuid

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.GetInt16 Method

Syntax: Visual Basic

```
NotOverridable Public Function GetInt16( _
    ByVal index As Integer _
) As Short _
    Implements IDataRecord.GetInt16
```

Syntax: C#

```
public short GetInt16(
    int index
);
```

Implements

IDataRecord.GetInt16

See Also

MySqlDataReader.GetInt32 Method

Syntax: Visual Basic

```
NotOverridable Public Function GetInt32( _
   ByVal index As Integer _
) As Integer _
  Implements IDataRecord.GetInt32
```

Syntax: C#

```
public int GetInt32(
   int index
);
```

Implements

IDataRecord.GetInt32

See Also

MySqlDataReader.GetInt64 Method

Syntax: Visual Basic

```
NotOverridable Public Function GetInt64( _
   ByVal index As Integer _
) As Long _
  Implements IDataRecord.GetInt64
```

Syntax: C#

```
public long GetInt64(
   int index
);
```

Implements

IDataRecord.GetInt64

See Also

MySqlDataReader.GetMySqlDateTime Method

Syntax: Visual Basic

```
Public Function GetMySqlDateTime( _
   ByVal index As Integer _
) As MySqlDateTime
```

Syntax: C#

```
public MySqlDateTime GetMySqlDateTime(
   int index
);
```

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.GetName Method

Gets the name of the specified column.

Syntax: Visual Basic

```
NotOverridable Public Function GetName( _
   ByVal i As Integer _
) As String _
  Implements IDataRecord.GetName
```

Syntax: C#

```
public string GetName(
   int i
);
```

Parameters

- i:

Return Value

Implements

IDataRecord.GetName

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.GetOrdinal Method

Gets the column ordinal, given the name of the column.

Syntax: Visual Basic

```
NotOverridable Public Function GetOrdinal( _
   ByVal name As String _
) As Integer _
  Implements IDataRecord.GetOrdinal
```

Syntax: C#

```
public int GetOrdinal(
   string name
```

```
);
```

Parameters

- name:

Return Value

Implements

IDataRecord.GetOrdinal

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.GetSchemaTable Method

Returns a DataTable that describes the column metadata of the MySqlDataReader.

Syntax: Visual Basic

```
NotOverridable Public Function GetSchemaTable() As DataTable _
  Implements IDataReader.GetSchemaTable
```

Syntax: C#

```
public DataTable GetSchemaTable();
```

Return Value

Implements

IDataReader.GetSchemaTable

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.GetString Method

Syntax: Visual Basic

```
NotOverridable Public Function GetString( _
   ByVal index As Integer _
) As String _
  Implements IDataRecord.GetString
```

Syntax: C#

```
public string GetString(
   int index
);
```

Implements

IDataRecord.GetString

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.GetTimeSpan Method

Syntax: Visual Basic

```
Public Function GetTimeSpan( _
   ByVal index As Integer _
) As TimeSpan
```

Syntax: C#

```
public TimeSpan GetTimeSpan(
   int index
);
```

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.GetUInt16 Method

Syntax: Visual Basic

```
Public Function GetUInt16( _
   ByVal index As Integer _
) As UInt16
```

Syntax: C#

```
public ushort GetUInt16(
   int index
);
```

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.GetUInt32 Method

Syntax: Visual Basic

```
Public Function GetUInt32( _
   ByVal index As Integer _
) As UInt32
```

Syntax: C#

```
public uint GetUInt32(
   int index
);
```

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.GetUInt64 Method

Syntax: Visual Basic

```
Public Function GetUInt64( _
   ByVal index As Integer _
) As UInt64
```

Syntax: C#

```
public ulong GetUInt64(
   int index
);
```

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.GetValue Method

Gets the value of the specified column in its native format.

Syntax: Visual Basic

```
NotOverridable Public Function GetValue( _
   ByVal i As Integer _
) As Object _
   Implements IDataRecord.GetValue
```

Syntax: C#

```
public object GetValue(
   int i
);
```

Parameters

- i:

Return Value

Implements

IDataRecord.GetValue

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.GetValues Method

Gets all attribute columns in the collection for the current row.

Syntax: Visual Basic

```
NotOverridable Public Function GetValues( _
```

```
   ByVal values As Object() _
) As Integer _
  Implements IDataRecord.GetValues
```

Syntax: C#

```
public int GetValues(
   object[] values
);
```

Parameters

- `values`:

Return Value

Implements

IDataRecord.GetValues

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.IsDBNull Method

Gets a value indicating whether the column contains non-existent or missing values.

Syntax: Visual Basic

```
NotOverridable Public Function IsDBNull( _
   ByVal i As Integer _
) As Boolean _
  Implements IDataRecord.IsDBNull
```

Syntax: C#

```
public bool IsDBNull(
   int i
);
```

Parameters

- `i`:

Return Value

Implements

IDataRecord.IsDBNull

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.NextResult Method

Advances the data reader to the next result, when reading the results of batch SQL statements.

Syntax: Visual Basic

```
NotOverridable Public Function NextResult() As Boolean _
  Implements IDataReader.NextResult
```

Syntax: C#

```
public bool NextResult();
```

Return Value

Implements

IDataReader.NextResult

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

MySqlDataReader.Read Method

Advances the MySqlDataReader to the next record.

Syntax: Visual Basic

```
NotOverridable Public Function Read() As Boolean _
  Implements IDataReader.Read
```

Syntax: C#

```
public bool Read();
```

Return Value

Implements

IDataReader.Read

See Also

MySqlDataReader Class, MySql.Data.MySqlClient Namespace

10.1.9 MySqlDbType Enumeration

Specifies MySQL-specific data type of a field, property, for use in a MySqlParameter.

Syntax: Visual Basic

```
Public Enum MySqlDbType
```

Syntax: C#

```
public enum MySqlDbType
```

Members

Member Name	Description
Newdate	Obsolete. Use Datetime or Date type.
Timestamp	A timestamp. The range is '1970-01-01 00:00:01.000000' to '2038-01-19 03:14:07.999999'. (Fractional seconds can only be stored with a MySQL 5.6.4 or higher database server.)
Time	The range is '-838:59:59.000000' to '838:59:59.000000'. (Fractional seconds can only be stored with a MySQL 5.6.4 or higher database server.)
Date	The supported range is '1000-01-01' to '9999-12-31'.
Datetime	The supported range is '1000-01-01 00:00:00.000000' to '9999-12-31 23:59:59.999999'. (Fractional seconds can only be stored with a MySQL 5.6.4 or higher database server.)
Year	A year in 2- or 4-digit format (default is 4-digit). The allowable values are 1901 to 2155, 0000 in the 4-digit year format, and 1970-2069 if you use the 2-digit format (70-69).
TinyBlob	A BLOB column with a maximum length of 255 ($2^8 - 1$) characters.
Blob	A BLOB column with a maximum length of 65535 ($2^{16} - 1$) characters.
MediumBlob	A BLOB column with a maximum length of 16777215 ($2^{24} - 1$) characters.
LongBlob	A BLOB column with a maximum length of 4294967295 or 4G ($2^{32} - 1$) characters.
Int16	A 16-bit signed integer. The signed range is -32768 to 32767. The unsigned range is 0 to 65535.
Int24	Specifies a 24 (3 byte) signed or unsigned value.
Int32	A 32-bit signed integer.
Int64	A 64-bit signed integer.
Byte	The signed range is -128 to 127. The unsigned range is 0 to 255.
Float	A small (single-precision) floating-point number. Allowable values are -3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38.
Double	A normal-size (double-precision) floating-point number. Allowable values are -1.7976931348623157E+308 to -2.2250738585072014E-308, 0, and 2.2250738585072014E-308 to 1.7976931348623157E+308.
UByte	An 8-bit unsigned value.

UInt16	A 16-bit unsigned value.
UInt24	A 24-bit unsigned value.
UInt32	A 32-bit unsigned value.
UInt64	A 64-bit unsigned value.
Decimal	A fixed precision and scale numeric value between -10^38-1 and 10^38-1.
NewDecimal	New Decimal
Set	A set. A string object that can have zero or more values, each of which must be chosen from the list of values 'value1', 'value2', ... A SET can have a maximum of 64 members.
String	Obsolete. Use VarChar type.
VarChar	A variable-length string containing 0 to 255 characters.
VarString	A variable-length string containing 0 to 65535 characters.
Enum	An enumeration. A string object that can have only one value, chosen from the list of values 'value1', 'value2', ..., NULL or the special "" error value. An ENUM can have a maximum of 65535 distinct values.
Geometry	
Bit	Bit-field data type.
TinyText	A nonbinary string column supporting a maximum length of 255 ($2^8 - 1$) characters.
Text	A nonbinary string column supporting a maximum length of 65535 ($2^{16} - 1$) characters.
MediumText	A nonbinary string column supporting a maximum length of 16777215 ($2^{24} - 1$) characters.
LongText	A nonbinary string column supporting a maximum length of 4294967295 ($2^{32} - 1$) characters.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySql.Data.MySqlClient Namespace

10.1.10 MySqlError Class

Collection of error codes that can be returned by the server

For a list of all members of this type, see MySqlError Members.

Syntax: Visual Basic

```
Public Class MySqlError
```

Syntax: C#

```
public class MySqlError
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlError Members, MySql.Data.MySqlClient Namespace

10.1.10.1 MySqlError Members

MySqlError overview

Public Instance Constructors

MySqlError Constructor	

Public Instance Properties

Code	Error code
Level	Error level
Message	Error message

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

MySqlError Constructor

Syntax: Visual Basic

```
Public Sub New( _
   ByVal level As String, _
   ByVal code As Integer, _
   ByVal message As String _
)
```

Syntax: C#

```
public MySqlError(
   string level,
   int code,
   string message
);
```

Parameters

- `level`:

- `code`:

- `message`:

See Also

Code Property

Error code

Syntax: Visual Basic

```
Public ReadOnly Property Code As Integer
```

Syntax: C#

```
public int Code {get;}
```

See Also

Level Property

Error level

Syntax: Visual Basic

```
Public ReadOnly Property Level As String
```

Syntax: C#

```
public string Level {get;}
```

See Also

MySqlError Class, MySql.Data.MySqlClient Namespace

Message Property

Error message

Syntax: Visual Basic

```
Public ReadOnly Property Message As String
```

Syntax: C#

```
public string Message {get;}
```

See Also

MySqlError Class, MySql.Data.MySqlClient Namespace

10.1.11 MySqlErrorCode Enumeration

Syntax: Visual Basic

```
Public Enum MySqlErrorCode
```

Syntax: C#

```
public enum MySqlErrorCode
```

Members

Member Name	Description
PacketTooLarge	
PasswordNotAllowed	
DuplicateKeyEntry	
HostNotPrivileged	
PasswordNoMatch	
AnonymousUser	
DuplicateKey	
KeyNotFound	
DuplicateKeyName	

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

10.1.12 MySqlException Class

The exception that is thrown when MySQL returns an error. This class cannot be inherited.

For a list of all members of this type, see MySqlException Members .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlException _
  Inherits SystemException
```

Syntax: C#

```
public sealed class MySqlException : SystemException
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlException Members, MySql.Data.MySqlClient Namespace

10.1.12.1 MySqlException Members

MySqlException overview

Public Instance Properties

Data (inherited from Exception)	Gets a collection of key/value pairs that provide additional, user-defined information about the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the `Exceptioninstance` that caused the current exception.
Message (inherited from Exception)	Gets a message that describes the current exception.
Number	Gets a number that identifies the type of error. This number corresponds to the error numbers given in Server Error Codes and Messages.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.

StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Exception)	Gets the runtime type of the current instance.
ToString (inherited from Exception)	Creates and returns a string representation of the current exception.

See Also

MySqlException Class, MySql.Data.MySqlClient Namespace

Number Property

Gets a number that identifies the type of error.

Syntax: Visual Basic

```
Public ReadOnly Property Number As Integer
```

Syntax: C#

```
public int Number {get;}
```

See Also

MySqlException Class, MySql.Data.MySqlClient Namespace

10.1.13 MySqlHelper Class

Helper class that makes it easier to work with the provider.

For a list of all members of this type, see MySqlHelper Members .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlHelper
```

Syntax: C#

```
public sealed class MySqlHelper
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlHelper Members, MySql.Data.MySqlClient Namespace

10.1.13.1 MySqlHelper Members

MySqlHelper overview

Public Static (Shared) Methods

ExecuteDataRow	Executes a single SQL statement and returns the first row of the resultset. A new MySqlConnection object is created, opened, and closed during this method.
ExecuteDataset	Overloaded. Executes a single SQL statement and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.
ExecuteNonQuery	Overloaded. Executes a single command against a MySQL database. The MySqlConnection is assumed to be open when the method is called and remains open after the method completes.
ExecuteReader	Overloaded. Executes a single command against a MySQL database.
ExecuteScalar	Overloaded. Execute a single command against a MySQL database.
UpdateDataSet	Updates the given table with data from the given DataSet

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.

| ToString (inherited from Object) | Returns a String that represents the current Object. |

See Also

MySqlHelper Class, MySql.Data.MySqlClient Namespace

MySqlHelper.ExecuteDataRow Method

Executes a single SQL statement and returns the first row of the resultset. A new MySqlConnection object is created, opened, and closed during this method.

Syntax: Visual Basic

```
Public Shared Function ExecuteDataRow( _
   ByVal connectionString As String, _
   ByVal commandText As String, _
   ParamArray parms As MySqlParameter() _
) As DataRow
```

Syntax: C#

```
public static DataRow ExecuteDataRow(
   string connectionString,
   string commandText,
   params MySqlParameter[]parms
);
```

Parameters

- `connectionString`: Settings to be used for the connection

- `commandText`: Command to execute

- `parms`: Parameters to use for the command

Return Value

DataRow containing the first row of the resultset

See Also

MySqlHelper Class, MySql.Data.MySqlClient Namespace

ExecuteDataset Method

Executes a single SQL statement and returns the resultset in a DataSet. The state of the MySqlConnection object remains unchanged after execution of this method.

Overload List

Executes a single SQL statement and returns the resultset in a DataSet. The state of the MySqlConnection object remains unchanged after execution of this method.

- public static DataSet ExecuteDataset(MySqlConnection,string);

Executes a single SQL statement and returns the resultset in a DataSet. The state of the MySqlConnection object remains unchanged after execution of this method.

- public static DataSet ExecuteDataset(MySqlConnection,string,params MySqlParameter[]);

Executes a single SQL statement and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.

- public static DataSet ExecuteDataset(string,string);

Executes a single SQL statement and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.

- public static DataSet ExecuteDataset(string,string,params MySqlParameter[]);

See Also

MySqlHelper Class, MySql.Data.MySqlClient Namespace

MySqlHelper.ExecuteDataset Method

Executes a single SQL statement and returns the resultset in a DataSet. The state of the MySqlConnection object remains unchanged after execution of this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _
   ByVal connection As MySqlConnection, _
   ByVal commandText As String _
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(
   MySqlConnection connection,
   string commandText
);
```

Parameters

- connection: MySqlConnection object to use

- commandText: Command to execute

Return Value

DataSetcontaining the resultset

See Also

MySqlHelper Class, MySql.Data.MySqlClient Namespace, MySqlHelper.ExecuteDataset Overload List

MySqlHelper.ExecuteDataset Method

Executes a single SQL statement and returns the resultset in a DataSet. The state of the MySqlConnection object remains unchanged after execution of this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _
```

```
   ByVal connection As MySqlConnection, _
   ByVal commandText As String, _
   ParamArray commandParameters As MySqlParameter() _
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(
   MySqlConnection connection,
   string commandText,
   params MySqlParameter[]commandParameters
);
```

Parameters

- connection: MySqlConnection object to use

- commandText: Command to execute

- commandParameters: Parameters to use for the command

Return Value

DataSetcontaining the resultset

See Also

MySqlHelper Class, MySql.Data.MySqlClient Namespace, MySqlHelper.ExecuteDataset Overload List

MySqlHelper.ExecuteDataset Method

Executes a single SQL statement and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _
   ByVal connectionString As String, _
   ByVal commandText As String _
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(
   string connectionString,
   string commandText
);
```

Parameters

- connectionString: Settings to be used for the connection

- commandText: Command to execute

Return Value

DataSetcontaining the resultset

See Also

MySqlHelper.ExecuteDataset Method

Executes a single SQL statement and returns the resultset in a DataSet. A new MySqlConnection object is created, opened, and closed during this method.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteDataset( _
   ByVal connectionString As String, _
   ByVal commandText As String, _
   ParamArray commandParameters As MySqlParameter() _
) As DataSet
```

Syntax: C#

```
public static DataSet ExecuteDataset(
   string connectionString,
   string commandText,
   params MySqlParameter[]commandParameters
);
```

Parameters

- `connectionString`: Settings to be used for the connection

- `commandText`: Command to execute

- `commandParameters`: Parameters to use for the command

Return Value

DataSetcontaining the resultset

See Also

ExecuteNonQuery Method

Executes a single command against a MySQL database. The MySqlConnection is assumed to be open when the method is called and remains open after the method completes.

Overload List

Executes a single command against a MySQL database. The MySqlConnection is assumed to be open when the method is called and remains open after the method completes.

- public static int ExecuteNonQuery(MySqlConnection,string,params MySqlParameter[]);

Executes a single command against a MySQL database. A new MySqlConnection is created using the ConnectionString given.

- public static int ExecuteNonQuery(string,string,params MySqlParameter[]);

See Also

MySqlHelper.ExecuteNonQuery Method

Executes a single command against a MySQL database. The MySqlConnection is assumed to be open when the method is called and remains open after the method completes.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteNonQuery( _
   ByVal connection As MySqlConnection, _
   ByVal commandText As String, _
   ParamArray commandParameters As MySqlParameter() _
) As Integer
```

Syntax: C#

```
public static int ExecuteNonQuery(
   MySqlConnection connection,
   string commandText,
   params MySqlParameter[]commandParameters
);
```

Parameters

- `connection`: MySqlConnection object to use

- `commandText`: SQL statement to be executed

- `commandParameters`: Array of MySqlParameter objects to use with the command.

Return Value

See Also

MySqlHelper Class, MySql.Data.MySqlClient Namespace, MySqlHelper.ExecuteNonQuery Overload List

MySqlHelper.ExecuteNonQuery Method

Executes a single command against a MySQL database. A new MySqlConnection is created using the ConnectionString given.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteNonQuery( _
   ByVal connectionString As String, _
   ByVal commandText As String, _
   ParamArray parms As MySqlParameter() _
) As Integer
```

Syntax: C#

```
public static int ExecuteNonQuery(
   string connectionString,
   string commandText,
   params MySqlParameter[]parms
);
```

Parameters

- `connectionString`: ConnectionString to use

- `commandText`: SQL statement to be executed

- `parms`: Array of MySqlParameter objects to use with the command.

Return Value

See Also

MySqlHelper Class, MySql.Data.MySqlClient Namespace, MySqlHelper.ExecuteNonQuery Overload List

ExecuteReader Method

Executes a single command against a MySQL database.

Overload List

Executes a single command against a MySQL database.

- public static MySqlDataReader ExecuteReader(string,string);

Executes a single command against a MySQL database.

- public static MySqlDataReader ExecuteReader(string,string,params MySqlParameter[]);

See Also

MySqlHelper Class, MySql.Data.MySqlClient Namespace

MySqlHelper.ExecuteReader Method

Executes a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteReader( _
   ByVal connectionString As String, _
   ByVal commandText As String _
) As MySqlDataReader
```

Syntax: C#

```
public static MySqlDataReader ExecuteReader(
   string connectionString,
   string commandText
);
```

Parameters

- `connectionString`: Settings to use for this command

- `commandText`: Command text to use

Return Value

MySqlDataReader object ready to read the results of the command

See Also

MySqlHelper Class, MySql.Data.MySqlClient Namespace, MySqlHelper.ExecuteReader Overload List

MySqlHelper.ExecuteReader Method

Executes a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteReader( _
   ByVal connectionString As String, _
   ByVal commandText As String, _
   ParamArray commandParameters As MySqlParameter() _
) As MySqlDataReader
```

Syntax: C#

```
public static MySqlDataReader ExecuteReader(
   string connectionString,
   string commandText,
   params MySqlParameter[]commandParameters
);
```

Parameters

- `connectionString`: Settings to use for this command

- `commandText`: Command text to use

- `commandParameters`: Array of MySqlParameter objects to use with the command

Return Value

MySqlDataReader object ready to read the results of the command

See Also

MySqlHelper Class, MySql.Data.MySqlClient Namespace, MySqlHelper.ExecuteReader Overload List

ExecuteScalar Method

Execute a single command against a MySQL database.

Overload List

Execute a single command against a MySQL database.

- public static object ExecuteScalar(MySqlConnection,string);

Execute a single command against a MySQL database.

- public static object ExecuteScalar(MySqlConnection,string,params MySqlParameter[]);

Execute a single command against a MySQL database.

- public static object ExecuteScalar(string,string);

Execute a single command against a MySQL database.

- public static object ExecuteScalar(string,string,params MySqlParameter[]);

See Also

MySqlHelper Class, MySql.Data.MySqlClient Namespace

MySqlHelper.ExecuteScalar Method (MySqlConnection, string)

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _
   ByVal connection As MySqlConnection, _
   ByVal commandText As String _
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(
   MySqlConnection connection,
   string commandText
);
```

Parameters

- connection: MySqlConnection object to use

- commandText: Command text to use for the command

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

MySqlHelper Class, MySql.Data.MySqlClient Namespace, MySqlHelper.ExecuteScalar Overload List

MySqlHelper.ExecuteScalar Method (MySqlConnection,string,params MySqlParameter[])

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _
   ByVal connection As MySqlConnection, _
   ByVal commandText As String, _
   ParamArray commandParameters As MySqlParameter() _
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(
   MySqlConnection connection,
   string commandText,
   params MySqlParameter[]commandParameters
);
```

Parameters

- connection: MySqlConnection object to use

- commandText: Command text to use for the command

- `commandParameters`: Parameters to use for the command

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

MySqlHelper Class, MySql.Data.MySqlClient Namespace, MySqlHelper.ExecuteScalar Overload List

MySqlHelper.ExecuteScalar Method (string, string)

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _
   ByVal connectionString As String, _
   ByVal commandText As String _
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(
   string connectionString,
   string commandText
);
```

Parameters

- `connectionString`: Settings to use for the update

- `commandText`: Command text to use for the update

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

MySqlHelper Class, MySql.Data.MySqlClient Namespace, MySqlHelper.ExecuteScalar Overload List

MySqlHelper.ExecuteScalar Method (string, string, params MySqlParameter[])

Execute a single command against a MySQL database.

Syntax: Visual Basic

```
Overloads Public Shared Function ExecuteScalar( _
   ByVal connectionString As String, _
   ByVal commandText As String, _
   ParamArray commandParameters As MySqlParameter() _
) As Object
```

Syntax: C#

```
public static object ExecuteScalar(
   string connectionString,
```

```
    string commandText,
    params MySqlParameter[]commandParameters
);
```

Parameters

- `connectionString`: Settings to use for the command

- `commandText`: Command text to use for the command

- `commandParameters`: Parameters to use for the command

Return Value

The first column of the first row in the result set, or a null reference if the result set is empty.

See Also

MySqlHelper Class, MySql.Data.MySqlClient Namespace, MySqlHelper.ExecuteScalar Overload List

MySqlHelper.UpdateDataSet Method

Updates the given table with data from the given DataSet

Syntax: Visual Basic

```
Public Shared Sub UpdateDataSet( _
    ByVal connectionString As String, _
    ByVal commandText As String, _
    ByVal ds As DataSet, _
    ByVal tablename As String _
)
```

Syntax: C#

```
public static void UpdateDataSet(
    string connectionString,
    string commandText,
    DataSet ds,
    string tablename
);
```

Parameters

- `connectionString`: Settings to use for the update

- `commandText`: Command text to use for the update

- `ds`: DataSetcontaining the new data to use in the update

- `tablename`: Tablename in the data set to update

See Also

MySqlHelper Class, MySql.Data.MySqlClient Namespace

10.1.14 MySqlInfoMessageEventArgs Class

Provides data for the InfoMessage event. This class cannot be inherited.

For a list of all members of this type, see MySqlInfoMessageEventArgs Members .

Syntax: Visual Basic

```
Public Class MySqlInfoMessageEventArgs_
  Inherits EventArgs
```

Syntax: C#

```
public class MySqlInfoMessageEventArgs : EventArgs
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlInfoMessageEventArgs Members, MySql.Data.MySqlClient Namespace

10.1.14.1 MySqlInfoMessageEventArgs Members

MySqlInfoMessageEventArgs overview

Public Instance Constructors

MySqlInfoMessageEventArgs Constructor	Initializes a new instance of the MySqlInfoMessageEventArgs class.

Public Instance Fields

errors	

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.

MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

MySqlInfoMessageEventArgs Class, MySql.Data.MySqlClient Namespace

MySqlInfoMessageEventArgs Constructor

Initializes a new instance of the MySqlInfoMessageEventArgs class.

Syntax: Visual Basic

```
Public Sub New()
```

Syntax: C#

```
public MySqlInfoMessageEventArgs();
```

See Also

MySqlInfoMessageEventArgs Class, MySql.Data.MySqlClient Namespace

MySqlInfoMessageEventArgs.errors Field

Syntax: Visual Basic

```
Public errors As MySqlError()
```

Syntax: C#

```
public MySqlError[] errors;
```

See Also

MySqlInfoMessageEventArgs Class, MySql.Data.MySqlClient Namespace

10.1.15 MySqlInfoMessageEventHandler Delegate

Represents the method that will handle the InfoMessage event of a MySqlConnection.

Syntax: Visual Basic

```
Public Delegate Sub MySqlInfoMessageEventHandler( _
   ByVal sender As Object, _
   ByVal args As MySqlInfoMessageEventArgs _
)
```

Syntax: C#

```
public delegate void MySqlInfoMessageEventHandler(
   object sender,
   MySqlInfoMessageEventArgs args
);
```

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySql.Data.MySqlClient Namespace

10.1.16 MySqlParameter Class

Represents a parameter to a MySqlCommand, and optionally, its mapping to DataSetcolumns. This class cannot be inherited.

For a list of all members of this type, see MySqlParameter Members .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlParameter_
  Inherits MarshalByRefObject_
  Implements IDataParameter, IDbDataParameter, ICloneable
```

Syntax: C#

```
public sealed class MySqlParameter : MarshalByRefObject, IDataParameter, IDbDataParameter, ICloneable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlParameter Members, MySql.Data.MySqlClient Namespace

10.1.16.1 MySqlParameter Members

MySqlParameter overview

Public Instance Constructors

MySqlParameter	Overloaded. Initializes a new instance of the MySqlParameter class.

Public Instance Properties

DbType	Gets or sets the DbType of the parameter.
Direction	Gets or sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter. As of

	MySQL version 4.1 and earlier, input-only is the only valid choice.
IsNullable	Gets or sets a value indicating whether the parameter accepts null values.
IsUnsigned	
MySqlDbType	Gets or sets the MySqlDbType of the parameter.
ParameterName	Gets or sets the name of the MySqlParameter.
Precision	Gets or sets the maximum number of digits used to represent the Value property.
Scale	Gets or sets the number of decimal places to which Value is resolved.
Size	Gets or sets the maximum size, in bytes, of the data within the column.
SourceColumn	Gets or sets the name of the source column that is mapped to the DataSetand used for loading or returning the Value.
SourceVersion	Gets or sets the DataRowVersion to use when loading Value.
Value	Gets or sets the value of the parameter.

Public Instance Methods

CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
ToString	Overridden. Gets a string containing the ParameterName.

See Also

MySqlParameter Class, MySql.Data.MySqlClient Namespace

MySqlParameter Constructor

Initializes a new instance of the MySqlParameter class.

Overload List

Initializes a new instance of the MySqlParameter class.

- public MySqlParameter();

Initializes a new instance of the MySqlParameter class with the parameter name and the data type.

- public MySqlParameter(string,MySqlDbType);

Initializes a new instance of the MySqlParameter class with the parameter name, the MySqlDbType, and the size.

- public MySqlParameter(string,MySqlDbType,int);

Initializes a new instance of the MySqlParameter class with the parameter name, the type of the parameter, the size of the parameter, a ParameterDirection, the precision of the parameter, the scale of the parameter, the source column, a DataRowVersion to use, and the value of the parameter.

- public MySqlParameter(string,MySqlDbType,int,ParameterDirection,bool,byte,byte,string,DataRowVersion,object);

Initializes a new instance of the MySqlParameter class with the parameter name, the MySqlDbType, the size, and the source column name.

- public MySqlParameter(string,MySqlDbType,int,string);

Initializes a new instance of the MySqlParameter class with the parameter name and a value of the new MySqlParameter.

- public MySqlParameter(string,object);

See Also

MySqlParameter Class, MySql.Data.MySqlClient Namespace

MySqlParameter Constructor ()

Initializes a new instance of the MySqlParameter class.

Syntax: Visual Basic

```
Overloads Public Sub New()
```

Syntax: C#

```
public MySqlParameter();
```

See Also

MySqlParameter Class, MySql.Data.MySqlClient Namespace, MySqlParameter Constructor Overload List

MySqlParameter Constructor (string, MySqlDbType)

Initializes a new instance of the MySqlParameter class with the parameter name and the data type.

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal parameterName As String, _
   ByVal dbType As MySqlDbType _
```

```
)
```

Syntax: C#

```
public MySqlParameter(
  string parameterName,
  MySqlDbType dbType
);
```

Parameters

- `parameterName`: The name of the parameter to map.

- `dbType`: One of the MySqlDbType values.

See Also

MySqlParameter Class, MySql.Data.MySqlClient Namespace, MySqlParameter Constructor Overload List

MySqlParameter Constructor (String, MySqlDbType, Int32)

Initializes a new instance of the MySqlParameter class with the parameter name, the MySqlDbType, and the size.

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal parameterName As String, _
   ByVal dbType As MySqlDbType, _
   ByVal size As Integer _
)
```

Syntax: C#

```
public MySqlParameter(
   string parameterName,
   MySqlDbType dbType,
   int size
);
```

Parameters

- `parameterName`: The name of the parameter to map.

- `dbType`: One of the MySqlDbType values.

- `size`: The length of the parameter.

See Also

MySqlParameter Class, MySql.Data.MySqlClient Namespace, MySqlParameter Constructor Overload List

MySqlParameter Constructor (string, MySqlDbType, int,ParameterDirection, bool, byte, byte, string, DataRowVersion, object)

Initializes a new instance of the MySqlParameter class with the parameter name, the type of the parameter, the size of the parameter, a ParameterDirection, the precision of the parameter, the scale of the parameter, the source column, a DataRowVersion to use, and the value of the parameter.

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal parameterName As String, _
   ByVal dbType As MySqlDbType, _
   ByVal size As Integer, _
   ByVal direction As ParameterDirection, _
   ByVal isNullable As Boolean, _
   ByVal precision As Byte, _
   ByVal scale As Byte, _
   ByVal sourceColumn As String, _
   ByVal sourceVersion As DataRowVersion, _
   ByVal value As Object _
)
```

Syntax: C#

```
public MySqlParameter(
   string parameterName,
   MySqlDbType dbType,
   int size,
   ParameterDirection direction,
   bool isNullable,
   byte precision,
   byte scale,
   string sourceColumn,
   DataRowVersion sourceVersion,
   object value
);
```

Parameters

- parameterName: The name of the parameter to map.

- dbType: One of the MySqlDbType values.

- size: The length of the parameter.

- direction: One of the ParameterDirectionvalues.

- isNullable: true if the value of the field can be null, otherwise false.

- precision: The total number of digits to the left and right of the decimal point to which Value is resolved.

- scale: The total number of decimal places to which Value is resolved.

- sourceColumn: The name of the source column.

- sourceVersion: One of the DataRowVersionvalues.

- value: An Object that is the value of the MySqlParameter.

Exceptions

Exception Type	Condition
ArgumentException	

See Also

MySqlParameter Constructor (string, MySqlDbType, int, string)

Initializes a new instance of the MySqlParameter class with the parameter name, the MySqlDbType, the size, and the source column name.

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal parameterName As String, _
   ByVal dbType As MySqlDbType, _
   ByVal size As Integer, _
   ByVal sourceColumn As String _
)
```

Syntax: C#

```
public MySqlParameter(
   string parameterName,
   MySqlDbType dbType,
   int size,
   string sourceColumn
);
```

Parameters

- parameterName: The name of the parameter to map.

- dbType: One of the MySqlDbType values.

- size: The length of the parameter.

- sourceColumn: The name of the source column.

See Also

MySqlParameter Constructor (string, object)

Initializes a new instance of the MySqlParameter class with the parameter name and a value of the new MySqlParameter.

Syntax: Visual Basic

```
Overloads Public Sub New( _
   ByVal parameterName As String, _
   ByVal value As Object _
)
```

Syntax: C#

```
public MySqlParameter(
   string parameterName,
   object value
);
```

Parameters

- `parameterName`: The name of the parameter to map.

- `value`: An Object that is the value of the MySqlParameter.

See Also

MySqlParameter Class, MySql.Data.MySqlClient Namespace, MySqlParameter Constructor Overload List

DbType Property

Gets or sets the DbType of the parameter.

Syntax: Visual Basic

```
NotOverridable Public Property DbType As DbType _
  Implements IDataParameter.DbType
```

Syntax: C#

```
public System.Data.DbType DbType {get; set;}
```

Implements

IDataParameter.DbType

See Also

MySqlParameter Class, MySql.Data.MySqlClient Namespace

Direction Property

Gets or sets a value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter. As of MySQL version 4.1 and earlier, input-only is the only valid choice.

Syntax: Visual Basic

```
NotOverridable Public Property Direction As ParameterDirection _
  Implements IDataParameter.Direction
```

Syntax: C#

```
public System.Data.ParameterDirection Direction {get; set;}
```

Implements

IDataParameter.Direction

See Also

MySqlParameter Class, MySql.Data.MySqlClient Namespace

IsNullable Property

Gets or sets a value indicating whether the parameter accepts null values.

Syntax: Visual Basic

```
NotOverridable Public Property IsNullable As Boolean _
  Implements IDataParameter.IsNullable
```

Syntax: C#

```
public bool IsNullable {get; set;}
```

Implements

IDataParameter.IsNullable

See Also

MySqlParameter Class, MySql.Data.MySqlClient Namespace

IsUnsigned Property

Syntax: Visual Basic

```
Public Property IsUnsigned As Boolean
```

Syntax: C#

```
public bool IsUnsigned {get; set;}
```

See Also

MySqlParameter Class, MySql.Data.MySqlClient Namespace

MySqlDbType Property

Gets or sets the MySqlDbType of the parameter.

Syntax: Visual Basic

```
Public Property MySqlDbType As MySqlDbType
```

Syntax: C#

```
public MySqlDbType MySqlDbType {get; set;}
```

See Also

MySqlParameter Class, MySql.Data.MySqlClient Namespace

ParameterName Property

Gets or sets the name of the MySqlParameter.

Syntax: Visual Basic

```
NotOverridable Public Property ParameterName As String _
  Implements IDataParameter.ParameterName
```

Syntax: C#

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

Implements

IDataParameter.ParameterName

See Also

MySqlParameter Class, MySql.Data.MySqlClient Namespace

Precision Property

Gets or sets the maximum number of digits used to represent the Value property.

Syntax: Visual Basic

```
NotOverridable Public Property Precision As Byte _
  Implements IDbDataParameter.Precision
```

Syntax: C#

```
public byte Precision {get; set;}
```

Implements

IDbDataParameter.Precision

See Also

MySqlParameter Class, MySql.Data.MySqlClient Namespace

Scale Property

Gets or sets the number of decimal places to which Value is resolved.

Syntax: Visual Basic

```
NotOverridable Public Property Scale As Byte _
  Implements IDbDataParameter.Scale
```

Syntax: C#

```
public byte Scale {get; set;}
```

Implements

IDbDataParameter.Scale

See Also

MySqlParameter Class, MySql.Data.MySqlClient Namespace

Size Property

Gets or sets the maximum size, in bytes, of the data within the column.

Syntax: Visual Basic

```
NotOverridable Public Property Size As Integer _
  Implements IDbDataParameter.Size
```

Syntax: C#

```
public int Size {get; set;}
```

Implements

IDbDataParameter.Size

See Also

MySqlParameter Class, MySql.Data.MySqlClient Namespace

SourceColumn Property

Gets or sets the name of the source column that is mapped to the DataSetand used for loading or returning the Value.

Syntax: Visual Basic

```
NotOverridable Public Property SourceColumn As String _
  Implements IDataParameter.SourceColumn
```

Syntax: C#

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

Implements

IDataParameter.SourceColumn

See Also

MySqlParameter Class, MySql.Data.MySqlClient Namespace

SourceVersion Property

Gets or sets the DataRowVersion to use when loading Value.

Syntax: Visual Basic

```
NotOverridable Public Property SourceVersion As DataRowVersion _
  Implements IDataParameter.SourceVersion
```

Syntax: C#

```
public System.Data.DataRowVersion SourceVersion {get; set;}
```

Implements

IDataParameter.SourceVersion

See Also

MySqlParameter Class, MySql.Data.MySqlClient Namespace

Value Property

Gets or sets the value of the parameter.

Syntax: Visual Basic

```
NotOverridable Public Property Value As Object _
  Implements IDataParameter.Value
```

Syntax: C#

```
public object Value {get; set;}
```

Implements

IDataParameter.Value

See Also

MySqlParameter Class, MySql.Data.MySqlClient Namespace

MySqlParameter.ToString Method

Overridden. Gets a string containing the ParameterName.

Syntax: Visual Basic

```
Overrides Public Function ToString() As String
```

Syntax: C#

```
public override string ToString();
```

Return Value

See Also

MySqlParameter Class, MySql.Data.MySqlClient Namespace

10.1.17 MySqlParameterCollection Class

Represents a collection of parameters relevant to a MySqlCommand as well as their respective mappings to columns in a DataSet. This class cannot be inherited.

For a list of all members of this type, see MySqlParameterCollection Members .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlParameterCollection_
  Inherits MarshalByRefObject_
  Implements IDataParameterCollection, IList, ICollection, IEnumerable
```

Syntax: C#

```
public sealed class MySqlParameterCollection : MarshalByRefObject, IDataParameterCollection, IList, IColle
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlParameterCollection Members, MySql.Data.MySqlClient Namespace

10.1.17.1 MySqlParameterCollection Members

MySqlParameterCollection overview

Public Instance Constructors

MySqlParameterCollection Constructor	Initializes a new instance of the MySqlParameterCollection class.

Public Instance Properties

Count	Gets the number of MySqlParameter objects in the collection.
Item	Overloaded. Gets the MySqlParameter with a specified attribute. In C#, this property is the indexer for the MySqlParameterCollection class.

Public Instance Methods

Add	Overloaded. Adds the specified MySqlParameter object to the MySqlParameterCollection.
Clear	Removes all items from the collection.
Contains	Overloaded. Gets a value indicating whether a MySqlParameter exists in the collection.
CopyTo	Copies MySqlParameter objects from the MySqlParameterCollection to the specified array.
CreateObjRef (inherited from MarshalByRefObject)	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.

GetLifetimeService (inherited from MarshalByRefObject)	Retrieves the current lifetime service object that controls the lifetime policy for this instance.
GetType (inherited from Object)	Gets the Type of the current instance.
IndexOf	Overloaded. Gets the location of a MySqlParameter in the collection.
InitializeLifetimeService (inherited from MarshalByRefObject)	Obtains a lifetime service object to control the lifetime policy for this instance.
Insert	Inserts a MySqlParameter into the collection at the specified index.
Remove	Removes the specified MySqlParameter from the collection.
RemoveAt	Overloaded. Removes the specified MySqlParameter from the collection.
ToString (inherited from Object)	Returns a String that represents the current Object.

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace

MySqlParameterCollection Constructor

Initializes a new instance of the MySqlParameterCollection class.

Syntax: Visual Basic

```
Public Sub New()
```

Syntax: C#

```
public MySqlParameterCollection();
```

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace

Count Property

Gets the number of MySqlParameter objects in the collection.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property Count As Integer _
  Implements ICollection.Count
```

Syntax: C#

```
public int Count {get;}
```

Implements

ICollection.Count

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace

Item Property

Gets the MySqlParameter with a specified attribute. In C#, this property is the indexer for the MySqlParameterCollection class.

Overload List

Gets the MySqlParameter at the specified index.

- public MySqlParameter this[int] {get; set;}

Gets the MySqlParameter with the specified name.

- public MySqlParameter this[string] {get; set;}

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace

Item Property (Int32)

Gets the MySqlParameter at the specified index.

Syntax: Visual Basic

```
Overloads Public Default Property Item( _
   ByVal index As Integer _
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter this[
   int index
] {get; set;}
```

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace, MySqlParameterCollection.Item Overload List

Item Property (String)

Gets the MySqlParameter with the specified name.

Syntax: Visual Basic

```
Overloads Public Default Property Item( _
   ByVal name As String _
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter this[
```

```
    string name
] {get; set;}
```

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace, MySqlParameterCollection.Item
Overload List

Add Method

Adds the specified MySqlParameter object to the MySqlParameterCollection.

Overload List

Adds the specified MySqlParameter object to the MySqlParameterCollection.

- public MySqlParameter Add(MySqlParameter);

Adds the specified MySqlParameter object to the MySqlParameterCollection.

- public int Add(object);

Adds a MySqlParameter to the MySqlParameterCollection given the parameter name and the data type.

- public MySqlParameter Add(string,MySqlDbType);

Adds a MySqlParameter to the MySqlParameterCollection with the parameter name, the data type, and the
column length.

- public MySqlParameter Add(string,MySqlDbType,int);

Adds a MySqlParameter to the MySqlParameterCollection with the parameter name, the data type, the
column length, and the source column name.

- public MySqlParameter Add(string,MySqlDbType,int,string);

Adds a MySqlParameter to the MySqlParameterCollection given the specified parameter name and value.

- public MySqlParameter Add(string,object);

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace

MySqlParameterCollection.Add Method

Adds the specified MySqlParameter object to the MySqlParameterCollection.

Syntax: Visual Basic

```
Overloads Public Function Add( _
   ByVal value As MySqlParameter _
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter Add(
   MySqlParameter value
```

```
);
```

Parameters

- `value`: The MySqlParameter to add to the collection.

Return Value

The newly added MySqlParameter object.

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace, MySqlParameterCollection.Add Overload List

MySqlParameterCollection.Add Method

Adds the specified MySqlParameter object to the MySqlParameterCollection.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function Add( _
   ByVal value As Object _
) As Integer _
  Implements IList.Add
```

Syntax: C#

```
public int Add(
   object value
);
```

Parameters

- `value`: The MySqlParameter to add to the collection.

Return Value

The index of the new MySqlParameter object.

Implements

IList.Add

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace, MySqlParameterCollection.Add Overload List

MySqlParameterCollection.Add Method

Adds a MySqlParameter to the MySqlParameterCollection given the parameter name and the data type.

Syntax: Visual Basic

```
Overloads Public Function Add( _
   ByVal parameterName As String, _
   ByVal dbType As MySqlDbType _
```

```
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter Add(
   string parameterName,
   MySqlDbType dbType
);
```

Parameters

- parameterName: The name of the parameter.

- dbType: One of the MySqlDbType values.

Return Value

The newly added MySqlParameter object.

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace, MySqlParameterCollection.Add Overload List

MySqlParameterCollection.Add Method

Adds a MySqlParameter to the MySqlParameterCollection with the parameter name, the data type, and the column length.

Syntax: Visual Basic

```
Overloads Public Function Add( _
   ByVal parameterName As String, _
   ByVal dbType As MySqlDbType, _
   ByVal size As Integer _
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter Add(
   string parameterName,
   MySqlDbType dbType,
   int size
);
```

Parameters

- parameterName: The name of the parameter.

- dbType: One of the MySqlDbType values.

- size: The length of the column.

Return Value

The newly added MySqlParameter object.

See Also

MySqlParameterCollection.Add Method

Adds a MySqlParameter to the MySqlParameterCollection with the parameter name, the data type, the column length, and the source column name.

Syntax: Visual Basic

```
Overloads Public Function Add( _
   ByVal parameterName As String, _
   ByVal dbType As MySqlDbType, _
   ByVal size As Integer, _
   ByVal sourceColumn As String _
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter Add(
   string parameterName,
   MySqlDbType dbType,
   int size,
   string sourceColumn
);
```

Parameters

- `parameterName`: The name of the parameter.

- `dbType`: One of the MySqlDbType values.

- `size`: The length of the column.

- `sourceColumn`: The name of the source column.

Return Value

The newly added MySqlParameter object.

See Also

MySqlParameterCollection.Add Method

Adds a MySqlParameter to the MySqlParameterCollection given the specified parameter name and value.

Syntax: Visual Basic

```
Overloads Public Function Add( _
   ByVal parameterName As String, _
   ByVal value As Object _
) As MySqlParameter
```

Syntax: C#

```
public MySqlParameter Add(
   string parameterName,
   object value
);
```

Parameters

- `parameterName`: The name of the parameter.

- `value`: The Value of the MySqlParameter to add to the collection.

Return Value

The newly added MySqlParameter object.

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace, MySqlParameterCollection.Add Overload List

MySqlParameterCollection.Clear Method

Removes all items from the collection.

Syntax: Visual Basic

```
NotOverridable Public Sub Clear() _
  Implements IList.Clear
```

Syntax: C#

```
public void Clear();
```

Implements

IList.Clear

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace

Contains Method

Gets a value indicating whether a MySqlParameter exists in the collection.

Overload List

Gets a value indicating whether a MySqlParameter exists in the collection.

- public bool Contains(object);

Gets a value indicating whether a MySqlParameter with the specified parameter name exists in the collection.

- public bool Contains(string);

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace

MySqlParameterCollection.Contains Method

Gets a value indicating whether a MySqlParameter exists in the collection.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function Contains( _
   ByVal value As Object _
) As Boolean _
   Implements IList.Contains
```

Syntax: C#

```
public bool Contains(
   object value
);
```

Parameters

- value: The value of the MySqlParameter object to find.

Return Value

true if the collection contains the MySqlParameter object; otherwise, false.

Implements

IList.Contains

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace,
MySqlParameterCollection.Contains Overload List

MySqlParameterCollection.Contains Method

Gets a value indicating whether a MySqlParameter with the specified parameter name exists in the collection.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function Contains( _
   ByVal name As String _
) As Boolean _
   Implements IDataParameterCollection.Contains
```

Syntax: C#

```
public bool Contains(
   string name
);
```

Parameters

- name: The name of the MySqlParameter object to find.

Return Value

true if the collection contains the parameter; otherwise, false.

Implements

IDataParameterCollection.Contains

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace,
MySqlParameterCollection.Contains Overload List

MySqlParameterCollection.CopyTo Method

Copies MySqlParameter objects from the MySqlParameterCollection to the specified array.

Syntax: Visual Basic

```
NotOverridable Public Sub CopyTo( _
   ByVal array As Array, _
   ByVal index As Integer _
) _
   Implements ICollection.CopyTo
```

Syntax: C#

```
public void CopyTo(
   Array array,
   int index
);
```

Parameters

- array:

- index:

Implements

ICollection.CopyTo

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace

IndexOf Method

Gets the location of a MySqlParameter in the collection.

Overload List

Gets the location of a MySqlParameter in the collection.

- public int IndexOf(object);

Gets the location of the MySqlParameter in the collection with a specific parameter name.

- public int IndexOf(string);

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace

MySqlParameterCollection.IndexOf Method

Gets the location of a MySqlParameter in the collection.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function IndexOf( _
   ByVal value As Object _
) As Integer _
  Implements IList.IndexOf
```

Syntax: C#

```
public int IndexOf(
   object value
);
```

Parameters

- value: The MySqlParameter object to locate.

Return Value

The zero-based location of the MySqlParameter in the collection.

Implements

IList.IndexOf

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace,
MySqlParameterCollection.IndexOf Overload List

MySqlParameterCollection.IndexOf Method

Gets the location of the MySqlParameter in the collection with a specific parameter name.

Syntax: Visual Basic

```
NotOverridable Overloads Public Function IndexOf( _
   ByVal parameterName As String _
) As Integer _
  Implements IDataParameterCollection.IndexOf
```

Syntax: C#

```
public int IndexOf(
   string parameterName
);
```

Parameters

- parameterName: The name of the MySqlParameter object to retrieve.

Return Value

The zero-based location of the MySqlParameter in the collection.

Implements

IDataParameterCollection.IndexOf

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace, MySqlParameterCollection.IndexOf Overload List

MySqlParameterCollection.Insert Method

Inserts a MySqlParameter into the collection at the specified index.

Syntax: Visual Basic

```
NotOverridable Public Sub Insert( _
   ByVal index As Integer, _
   ByVal value As Object _
) _
  Implements IList.Insert
```

Syntax: C#

```
public void Insert(
   int index,
   object value
);
```

Parameters

- index:

- value:

Implements

IList.Insert

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace

MySqlParameterCollection.Remove Method

Removes the specified MySqlParameter from the collection.

Syntax: Visual Basic

```
NotOverridable Public Sub Remove( _
   ByVal value As Object _
) _
  Implements IList.Remove
```

Syntax: C#

```
public void Remove(
   object value
);
```

Parameters

- value:

Implements

IList.Remove

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace

RemoveAt Method

Removes the specified MySqlParameter from the collection.

Overload List

Removes the specified MySqlParameter from the collection using a specific index.

- public void RemoveAt(int);

Removes the specified MySqlParameter from the collection using the parameter name.

- public void RemoveAt(string);

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace

MySqlParameterCollection.RemoveAt Method

Removes the specified MySqlParameter from the collection using a specific index.

Syntax: Visual Basic

```
NotOverridable Overloads Public Sub RemoveAt( _
   ByVal index As Integer _
) _
   Implements IList.RemoveAt
```

Syntax: C#

```
public void RemoveAt(
   int index
);
```

Parameters

- index: The zero-based index of the parameter.

Implements

IList.RemoveAt

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace,
MySqlParameterCollection.RemoveAt Overload List

MySqlParameterCollection.RemoveAt Method

Removes the specified MySqlParameter from the collection using the parameter name.

Syntax: Visual Basic

```
NotOverridable Overloads Public Sub RemoveAt( _
   ByVal name As String _
) _
   Implements IDataParameterCollection.RemoveAt
```

Syntax: C#

```
public void RemoveAt(
   string name
);
```

Parameters

• name: The name of the MySqlParameter object to retrieve.

Implements

IDataParameterCollection.RemoveAt

See Also

MySqlParameterCollection Class, MySql.Data.MySqlClient Namespace,
MySqlParameterCollection.RemoveAt Overload List

10.1.18 MySqlRowUpdatedEventArgs Class

Provides data for the RowUpdated event. This class cannot be inherited.

For a list of all members of this type, see MySqlRowUpdatedEventArgs Members .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlRowUpdatedEventArgs_
   Inherits RowUpdatedEventArgs
```

Syntax: C#

```
public sealed class MySqlRowUpdatedEventArgs : RowUpdatedEventArgs
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlRowUpdatedEventArgs Members, MySql.Data.MySqlClient Namespace

10.1.18.1 MySqlRowUpdatedEventArgs Members

MySqlRowUpdatedEventArgs overview

Public Instance Constructors

MySqlRowUpdatedEventArgs Constructor	Initializes a new instance of the MySqlRowUpdatedEventArgs class.

Public Instance Properties

Command	Overloaded. Gets or sets the MySqlCommand executed when Update is called.
Errors (inherited from RowUpdatedEventArgs)	Gets any errors generated by the .NET Framework data provider when the Commandwas executed.
RecordsAffected (inherited from RowUpdatedEventArgs)	Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.
Row (inherited from RowUpdatedEventArgs)	Gets the DataRowsent through an Update.
RowCount (inherited from RowUpdatedEventArgs)	Gets the number of rows processed in a batch of updated records.
StatementType (inherited from RowUpdatedEventArgs)	Gets the type of SQL statement executed.
Status (inherited from RowUpdatedEventArgs)	Gets the UpdateStatus of the Commandproperty.
TableMapping (inherited from RowUpdatedEventArgs)	Gets the DataTableMappingsent through an Update.

Public Instance Methods

CopyToRows (inherited from RowUpdatedEventArgs)	Overloaded. Copies references to the modified rows into the provided array.
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

See Also

MySqlRowUpdatedEventArgs Class, MySql.Data.MySqlClient Namespace

MySqlRowUpdatedEventArgs Constructor

Initializes a new instance of the MySqlRowUpdatedEventArgs class.

Syntax: Visual Basic

```
Public Sub New( _
   ByVal row As DataRow, _
   ByVal command As IDbCommand, _
   ByVal statementType As StatementType, _
   ByVal tableMapping As DataTableMapping _
)
```

Syntax: C#

```
public MySqlRowUpdatedEventArgs(
   DataRow row,
   IDbCommand command,
   StatementType statementType,
   DataTableMapping tableMapping
);
```

Parameters

- `row`: The DataRowsent through an Update.

- `command`: The IDbCommand executed when Update is called.

- `statementType`: One of the StatementTypevalues that specifies the type of query executed.

- `tableMapping`: The DataTableMapping sent through an Update.

See Also

MySqlRowUpdatedEventArgs Class, MySql.Data.MySqlClient Namespace

Command Property

Gets or sets the MySqlCommand executed when Update is called.

Syntax: Visual Basic

```
Overloads Public ReadOnly Property Command As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand Command {get;}
```

See Also

MySqlRowUpdatedEventArgs Class, MySql.Data.MySqlClient Namespace

10.1.19 MySqlRowUpdatedEventHandler Delegate

Represents the method that will handle the RowUpdatedevent of a MySqlDataAdapter.

Syntax: Visual Basic

```
Public Delegate Sub MySqlRowUpdatedEventHandler( _
   ByVal sender As Object, _
   ByVal e As MySqlRowUpdatedEventArgs _
)
```

Syntax: C#

```
public delegate void MySqlRowUpdatedEventHandler(
   object sender,
   MySqlRowUpdatedEventArgs
);
```

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySql.Data.MySqlClient Namespace

10.1.20 MySqlRowUpdatingEventArgs Class

Provides data for the RowUpdating event. This class cannot be inherited.

For a list of all members of this type, see MySqlRowUpdatingEventArgs Members .

Syntax: Visual Basic

```
NotInheritable Public Class MySqlRowUpdatingEventArgs_
  Inherits RowUpdatingEventArgs
```

Syntax: C#

```
public sealed class MySqlRowUpdatingEventArgs : RowUpdatingEventArgs
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlRowUpdatingEventArgs Members, MySql.Data.MySqlClient Namespace

10.1.20.1 MySqlRowUpdatingEventArgs Members

MySqlRowUpdatingEventArgs overview

Public Instance Constructors

MySqlRowUpdatingEventArgs Constructor	Initializes a new instance of the MySqlRowUpdatingEventArgs class.

Public Instance Properties

Command	Overloaded. Gets or sets the MySqlCommand to execute when performing the Update.
Errors (inherited from RowUpdatingEventArgs)	Gets any errors generated by the .NET Framework data provider when the Command executes.
Row (inherited from RowUpdatingEventArgs)	Gets the DataRow that will be sent to the server as part of an insert, update, or delete operation.
StatementType (inherited from RowUpdatingEventArgs)	Gets the type of SQL statement to execute.
Status (inherited from RowUpdatingEventArgs)	Gets or sets the UpdateStatus of the Command property.
TableMapping (inherited from RowUpdatingEventArgs)	Gets the DataTableMapping to send through the Update.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString (inherited from Object)	Returns a String that represents the current Object.

See Also

MySqlRowUpdatingEventArgs Class, MySql.Data.MySqlClient Namespace

MySqlRowUpdatingEventArgs Constructor

Initializes a new instance of the MySqlRowUpdatingEventArgs class.

Syntax: Visual Basic

```
Public Sub New( _
   ByVal row As DataRow, _
   ByVal command As IDbCommand, _
   ByVal statementType As StatementType, _
   ByVal tableMapping As DataTableMapping _
)
```

Syntax: C#

```
public MySqlRowUpdatingEventArgs(
   DataRow row,
   IDbCommand command,
   StatementType statementType,
   DataTableMapping tableMapping
);
```

Parameters

- `row`: The DataRowto Update.

- `command`: The IDbCommandto execute during Update.

- `statementType`: One of the StatementTypevalues that specifies the type of query executed.

- `tableMapping`: The DataTableMappingsent through an Update.

See Also

MySqlRowUpdatingEventArgs Class, MySql.Data.MySqlClient Namespace

Command Property

Gets or sets the MySqlCommand to execute when performing the Update.

Syntax: Visual Basic

```
Overloads Public Property Command As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand Command {get; set;}
```

See Also

MySqlRowUpdatingEventArgs Class, MySql.Data.MySqlClient Namespace

10.1.21 MySqlRowUpdatingEventHandler Delegate

Represents the method that will handle the RowUpdatingevent of a MySqlDataAdapter.

Syntax: Visual Basic

```
Public Delegate Sub MySqlRowUpdatingEventHandler( _
   ByVal sender As Object, _
   ByVal e As MySqlRowUpdatingEventArgs _
)
```

Syntax: C#

```
public delegate void MySqlRowUpdatingEventHandler(
   object sender,
   MySqlRowUpdatingEventArgs
);
```

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySql.Data.MySqlClient Namespace

10.1.22 MySqlTransaction Class

For a list of all members of this type, see MySqlTransaction Members.

Syntax: Visual Basic

```
NotInheritable Public Class MySqlTransaction_
  Implements IDbTransaction, IDisposable
```

Syntax: C#

```
public sealed class MySqlTransaction : IDbTransaction, IDisposable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlTransaction Members, MySql.Data.MySqlClient Namespace

10.1.22.1 MySqlTransaction Members

MySqlTransaction overview

Public Instance Properties

Connection	Gets the MySqlConnection object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.
IsolationLevel	Specifies the IsolationLevel for this transaction.

Public Instance Methods

Commit	
Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
Rollback	
ToString (inherited from Object)	Returns a String that represents the current Object.

See Also

Connection Property

Gets the MySqlConnection object associated with the transaction, or a null reference (Nothing in Visual Basic) if the transaction is no longer valid.

Syntax: Visual Basic

```
Public ReadOnly Property Connection As MySqlConnection
```

Syntax: C#

```
public MySqlConnection Connection {get;}
```

Property Value

The MySqlConnection object associated with this transaction.

Remarks

A single application may have multiple database connections, each with zero or more transactions. This property enables you to determine the connection object associated with a particular transaction created by BeginTransaction.

See Also

IsolationLevel Property

Specifies the IsolationLevel for this transaction.

Syntax: Visual Basic

```
NotOverridable Public ReadOnly Property IsolationLevel As IsolationLevel _
  Implements IDbTransaction.IsolationLevel
```

Syntax: C#

```
public System.Data.IsolationLevel IsolationLevel {get;}
```

Property Value

The IsolationLevel for this transaction. The default is ReadCommitted.

Implements

IDbTransaction.IsolationLevel

Remarks

Parallel transactions are not supported. Therefore, the IsolationLevel applies to the entire transaction.

See Also

MySqlTransaction.Commit Method

Syntax: Visual Basic

```
NotOverridable Public Sub Commit() _
  Implements IDbTransaction.Commit
```

Syntax: C#

```
public void Commit();
```

Implements

IDbTransaction.Commit

See Also

MySqlTransaction.Rollback Method

Syntax: Visual Basic

```
NotOverridable Public Sub Rollback() _
  Implements IDbTransaction.Rollback
```

Syntax: C#

```
public void Rollback();
```

Implements

IDbTransaction.Rollback

See Also

10.2 MySql.Data.Types Namespace

Namespace hierarchy

Classes

Class	Description
MySqlConversionException	Summary description for MySqlConversionException.

Structures

Structure	Description
MySqlDateTime	Summary description for MySqlDateTime.

10.2.1 MySql.Data.TypesHierarchy

See Also

MySql.Data.Types Namespace

10.2.2 MySqlConversionException Class

Summary description for MySqlConversionException.

For a list of all members of this type, see MySqlConversionException Members.

Syntax: Visual Basic

```
Public Class MySqlConversionException _
  Inherits ApplicationException
```

Syntax: C#

```
public class MySqlConversionException : ApplicationException
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.Types

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlConversionException Members, MySql.Data.Types Namespace

10.2.2.1 MySqlConversionException Members

MySqlConversionException overview

Public Instance Constructors

MySqlConversionException Constructor	Ctor

Public Instance Properties

Data (inherited from Exception)	Gets a collection of key/value pairs that provide additional, user-defined information about the exception.
HelpLink (inherited from Exception)	Gets or sets a link to the help file associated with this exception.
InnerException (inherited from Exception)	Gets the `Exceptioninstance` that caused the current exception.

Message (inherited from Exception)	Gets a message that describes the current exception.
Source (inherited from Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace (inherited from Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite (inherited from Exception)	Gets the method that throws the current exception.

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetBaseException (inherited from Exception)	When overridden in a derived class, returns the Exception that is the root cause of one or more subsequent exceptions.
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetObjectData (inherited from Exception)	When overridden in a derived class, sets the SerializationInfo with information about the exception.
GetType (inherited from Exception)	Gets the runtime type of the current instance.
ToString (inherited from Exception)	Creates and returns a string representation of the current exception.

Protected Instance Properties

HResult (inherited from Exception)	Gets or sets `HRESULT`, a coded numeric value that is assigned to a specific exception.

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

MySqlConversionException Class, MySql.Data.Types Namespace

MySqlConversionException Constructor

Syntax: Visual Basic

```
Public Sub New( _
   ByVal msg As String _
)
```

Syntax: C#

```
public MySqlConversionException(
   string msg
);
```

See Also

MySqlConversionException Class, MySql.Data.Types Namespace

10.2.3 MySqlDateTime Structure

Summary description for MySqlDateTime.

For a list of all members of this type, see MySqlDateTime Members.

Syntax: Visual Basic

```
Public Class MySqlDateTime_
  Implements IConvertible, IComparable
```

Syntax: C#

```
public class MySqlDateTime : IConvertible, IComparable
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.Types

Assembly: MySql.Data (in MySql.Data.dll)

See Also

MySqlDateTime Members, MySql.Data.Types Namespace

10.2.3.1 MySqlDateTime Members

MySqlDateTime overview

Public Static (Shared) Type Conversions

Explicit MySqlDateTime to DateTime Conversion	

Public Instance Properties

Day	Returns the day portion of this datetime
Hour	Returns the hour portion of this datetime
IsNull	Returns true if this datetime object has a null value
IsValidDateTime	Indicates if this object contains a value that can be represented as a DateTime
Millisecond	Returns the millisecond portion of this datetime

Minute	Returns the minute portion of this datetime
Month	Returns the month portion of this datetime
Second	Returns the second portion of this datetime
ValueAsObject	Returns the value of this field as an object
Year	Returns the year portion of this datetime

Public Instance Methods

Equals (inherited from Object)	Determines whether the specified Object is equal to the current Object.
GetDateTime	Returns this value as a DateTime
GetHashCode (inherited from Object)	Serves as a hash function for a particular type. GetHashCode is suitable for use in hashing algorithms and data structures like a hash table.
GetType (inherited from Object)	Gets the Type of the current instance.
ToString	Returns a MySQL-specific string representation of this value.

Protected Instance Fields

classType	The system type represented by this value.
dbType	The generic dbtype of this value.
isNull	Is this value null.
mySqlDbType	The specific MySQL db type.
mySqlTypeName	The MySQL-specific type name of this value.
objectValue	

Protected Instance Methods

Finalize (inherited from Object)	Allows an Object to attempt to free resources and perform other cleanup operations before the Object is reclaimed by garbage collection.
MemberwiseClone (inherited from Object)	Creates a shallow copy of the current Object.

See Also

MySqlDateTime Structure, MySql.Data.Types Namespace

MySqlDateTime Explicit MySqlDateTime to DateTime Conversion

Syntax: Visual Basic

```
MySqlDateTime.op_Explicit(val)
```

Syntax: C#

```
public static explicit operator DateTime(
    MySqlDateTime val
```

```
);
```

Parameters

- `val`:

Return Value

See Also

MySqlDateTime Structure, MySql.Data.Types Namespace

Day Property

Returns the day portion of this `DateTime` type.

Syntax: Visual Basic

```
Public Property Day As Integer
```

Syntax: C#

```
public int Day {get; set;}
```

See Also

MySqlDateTime Structure, MySql.Data.Types Namespace

Hour Property

Returns the hour portion of this `DateTime` type.

Syntax: Visual Basic

```
Public Property Hour As Integer
```

Syntax: C#

```
public int Hour {get; set;}
```

See Also

MySqlDateTime Structure, MySql.Data.Types Namespace

IsNull Property

Syntax: Visual Basic

```
Public Property IsNull As Boolean
```

Syntax: C#

```
public bool IsNull {get; set;}
```

See Also

IsValidDateTime Property

Indicates if this object contains a value that can be represented as a `DateTime` type.

Syntax: Visual Basic

```
Public ReadOnly Property IsValidDateTime As Boolean
```

Syntax: C#

```
public bool IsValidDateTime {get;}
```

See Also

MySqlDateTime Structure, MySql.Data.Types Namespace

Millisecond Property

Returns the millisecond portion of this `DateTime` type.

Syntax: Visual Basic

```
Public Property Millisecond As Integer
```

Syntax: C#

```
public int Millisecond {get; set;}
```

See Also

MySqlDateTime Structure, MySql.Data.Types Namespace

Minute Property

Returns the minute portion of this `DateTime` type.

Syntax: Visual Basic

```
Public Property Minute As Integer
```

Syntax: C#

```
public int Minute {get; set;}
```

See Also

MySqlDateTime Structure, MySql.Data.Types Namespace

Month Property

Returns the month portion of this `DateTime` type.

Syntax: Visual Basic

```
Public Property Month As Integer
```

Syntax: C#

```
public int Month {get; set;}
```

See Also

MySqlDateTime Structure, MySql.Data.Types Namespace

Second Property

Returns the second portion of this `DateTime` type.

Syntax: Visual Basic

```
Public Property Second As Integer
```

Syntax: C#

```
public int Second {get; set;}
```

See Also

MySqlDateTime Structure, MySql.Data.Types Namespace

Year Property

Returns the year portion of this `DateTime` type.

Syntax: Visual Basic

```
Public Property Year As Integer
```

Syntax: C#

```
public int Year {get; set;}
```

See Also

MySqlDateTime Structure, MySql.Data.Types Namespace

MySqlDateTime.GetDateTime Method

Returns this value as a `DateTime` type.

Syntax: Visual Basic

```
Public Function GetDateTime() As Date
```

Syntax: C#

```
public DateTime GetDateTime();
```

See Also

[MySqlDateTime Structure](#), [MySql.Data.Types Namespace](#)

MySqlDateTime.ToString Method

Returns a MySQL-specific string representation of this value.

Syntax: Visual Basic

```
Overrides Public Function ToString() As String
```

Syntax: C#

```
public override string ToString();
```

See Also

[MySqlDateTime Structure](#), [MySql.Data.Types Namespace](#)

10.3 MySql.Data.MySqlClient.Memcached Namespace

Classes

Class	Description
[BinaryClient](#)	Implementation of Memcached binary client protocol.
[Client](#)	An interface of the client Memcached protocol.
[TextClient](#)	Implementation of the Memcached text client protocol

10.3.1 BinaryClient Class

For a list of all members of this type, see [BinaryClient Members](#) .

Syntax: Visual Basic

```
Public Class BinaryClient
 Inherits Client
```

Syntax: C#

```
public class BinaryClient : Client
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: [MySql.Data.MySqlClient.Memcached](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

BinaryClient Members, MySql.Data.MySqlClient.Memcached Namespace

10.3.1.1 BinaryClient Members

BinaryClient overview

Public Static Methods

GetInstance	Inherited from Client

Protected Instance Properties

server	Inherited from Client.
port	Inherited from Client.
stream	Inherited from Client.

Public Instance Methods

Open	Inherited from Client.
Close	Inherited from Client.
Add	Inherited from Client.
Append	Inherited from Client.
Cas	Inherited from Client.
Decrement>	Inherited from Client.
Delete	Inherited from Client.
FlushAll	Inherited from Client.
Get	Inherited from Client.
Increment	Inherited from Client.
Prepend	Inherited from Client.
Replace	Inherited from Client.
Set	Inherited from Client.

See Also

Client Class, MySql.Data.MySqlClient.Memcached Namespace

10.3.2 Client Class

For a list of all members of this type, see Client Members .

Syntax: Visual Basic

```
Public MustInherit Class Client
```

Syntax: C#

```
public abstract class Client
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient.Memcached

Assembly: MySql.Data (in MySql.Data.dll)

See Also

Client Members, MySql.Data.MySqlClient.Memcached Namespace

10.3.2.1 Client Members

Client overview

Public Static Methods

GetInstance	Factory method for creating instances of Client that implement a connection with the requested features. The connection object returned must be explicitely opened see method Client.Open

Protected Instance Properties

server	The server DNS or IP address used by the connection.
port	The port used by the connection.
stream	The network stream used by the connecition.

Public Instance Methods

Open	Opens the client connection.
Close	Closes the client connection.
Add	Adds a new key/value pair with the given TimeSpan expiration.
Append	Appends the data to the existing data for the associated key..
Cas	Executes the Check-and-set Memcached operation.
Decrement>	Decrements the value associated with a key by the given amount.
Delete	Removes they pair key/value given the specified key.
FlushAll	Removes all entries from the storage, effectively invalidating the whole cache.
Get	Get the key/value pair associated with a given key.
Increment	Increments the value associated with a key by the given amount.

Prepend	Prepends the data to the existing data for the associated key.
Replace	Replaces the value associated with the given key with another value.
Set	Set the value of a given key.

Public Static Methods

GetInstance	Factory method for creating instances of Client that implement a connection with the requested features. The connection object returned must be explicitly opened see method Open.

See Also

Client Class, MySql.Data.MySqlClient.Memcached Namespace

server Property

The server DNS or IP address used by the connection.

Syntax: Visual Basic

```
Protected server As String
```

Syntax: C#

```
protected string server
```

Property Value

The server DNS or IP address used by the connection.

Remarks

None

See Also

Client Class, MySql.Data.MySqlClient.Memcached Namespace

Port Property

The port used by the connection.

Syntax: Visual Basic

```
Protected port As UInteger
```

Syntax: C#

```
protected uint port
```

Property Value

The TCP port where the Innodb Memcached plugin will be listening for connections.

Remarks

None.

See Also

Client Class, MySql.Data.MySqlClient.Memcached Namespace

stream

The network stream used by the connection.

Syntax: Visual Basic

```
Protected stream As Stream
```

Syntax: C#

```
protected Stream stream;
```

See Also

Client Class, MySql.Data.MySqlClient.Memcached Namespace

Client.Open Method

Syntax: Visual Basic

```
Public Overridable Sub Open
```

Syntax: C#

```
public virtual void Open();
```

Opens the client connection.

Remarks

A client object can be opened and closed many times during its life cycle.

See Also

Client Class, MySql.Data.MySqlClient.Memcached Namespace

Client.Close Method

Syntax: Visual Basic

```
Public Overridable Sub Close
```

Syntax: C#

```
public virtual void Close();
```

Closes the client connection.

Remarks

A client object can be opened and closed many times during its life cycle.

See Also

Client Class, MySql.Data.MySqlClient.Memcached Namespace

Client.Add Method

Syntax: Visual Basic

```
Public MustOverride Sub Add (
 key As String,
 data As Object,
 expiration As TimeSpan
)
```

Syntax: C#

```
public abstract void Add(
 string key,
 Object data,
 TimeSpan expiration
);
```

Adds a new key/value pair with the given TimeSpan expiration.

Remarks

None.

See Also

Client Class, MySql.Data.MySqlClient.Memcached Namespace

Parameters

- `key`, The key for identifying the entry.

- `data`, The data to associate with the key.

- `expiration`, The interval of timespan, use `TimeSpan.Zero` for no expiration.

Client.Append Method

Syntax: Visual Basic

```
Public MustOverride Sub Append (
 key As String,
 data As Object
)
```

Syntax: C#

```
public abstract void Append(
 string key,
 Object data
);
```

Appends the data to the existing data for the associated key.

Remarks

None.

See Also

Client Class, MySql.Data.MySqlClient.Memcached Namespace

Parameters

- `key`, The key for identifying the entry.

- `data`, The data to append with the data associated with the key.

Client.Cas Method

Syntax: Visual Basic

```
Public MustOverride Sub Cas (
 key As String,
 data As Object,
 expiration As TimeSpan,
 casUnique As ULong
)
```

Syntax: C#

```
public abstract void Cas(
 string key,
 Object data,
 TimeSpan expiration,
 ulong casUnique
);
```

Executes the Check-and-set Memcached operation.

Remarks

None.

See Also

Client Class, MySql.Data.MySqlClient.Memcached Namespace

Parameters

- `key`, The key for identifying the entry.

- `data`, The data to use in the CAS.

- `expiration`, The interval of timespan, use `TimeSpan.Zero` for no expiration.

• `casUnique`, The CAS unique value to use.

Client.Decrement Method

Syntax: Visual Basic

```
Public MustOverride Sub Decrement (
 key As String,
 amount As Integer
)
```

Syntax: C#

```
public abstract void Decrement(
 string key,
 int amount
);
```

Decrements the value associated with a key by the given amount.

Remarks

None.

See Also

Client Class, MySql.Data.MySqlClient.Memcached Namespace

Parameters

• `key`, The key associated with the value to decrement.

• `amount`, The amount to decrement the value.

Client.Delete Method

Syntax: Visual Basic

```
Public MustOverride Sub Delete (
 key As String
)
```

Syntax: C#

```
public abstract void Delete(
 string key
);
```

Removes they pair key/value given the specified key.

Remarks

None.

See Also

Client Class, MySql.Data.MySqlClient.Memcached Namespace

Parameters

- `key`, The key associated with the value to delete.

Client.FlushAll Method

Syntax: Visual Basic

```
Public MustOverride Sub FlushAll (
 delay As TimeSpan
)
```

Syntax: C#

```
public abstract void FlushAll(
 TimeSpan delay
);
```

Removes all entries from the storage, effectively invalidating the whole cache.

Remarks

None.

See Also

Client Class, MySql.Data.MySqlClient.Memcached Namespace

Parameters

- `TimeSpan`, The interval after which the cache will be cleaned. Can be TimeSpan.Zero for immediately.

Client.Get Method

Syntax: Visual Basic

```
Public MustOverride Function Get (
 key As String
) As KeyValuePair(Of String, Object)
```

Syntax: C#

```
public abstract KeyValuePair string, Object> Get(
 string key
);
```

Get the key/value pair associated with a given key.

Remarks

None.

See Also

Client Class, MySql.Data.MySqlClient.Memcached Namespace

Parameters

- `key`, The key for which to return the key/value.

Return Value

The key/value associated with the key or a MemcachedException if it does not exists.

Client.Increment Method

Syntax: Visual Basic

```
Public MustOverride Sub Increment (
 key As String,
 amount As Integer
)
```

Syntax: C#

```
public abstract void Increment(
 string key,
 int amount
);
```

Increments the value associated with a key by the given amount.

Remarks

None.

See Also

Client Class, MySql.Data.MySqlClient.Memcached Namespace

Parameters

- `key`, The key associated with the value to increment.

- `amount`, The amount to increment the value.

Client.Prepend Method

Syntax: Visual Basic

```
Public MustOverride Sub Prepend (
 key As String,
 data As Object
)
```

Syntax: C#

```
public abstract void Prepend(
 string key,
 Object data
);
```

Prepends the data to the existing data for the associated key.

Remarks

None.

See Also

Client Class, MySql.Data.MySqlClient.Memcached Namespace

Parameters

- `key`, The key for identifying the entry.

- `data`, The data to append with the data associated with the key.

Client.Replace Method

Syntax: Visual Basic

```
Public MustOverride Sub Replace (
 key As String,
 data As Object,
 expiration As TimeSpan
)
```

Syntax: C#

```
public abstract void Replace(
 string key,
 Object data,
 TimeSpan expiration
);
```

Replaces the value associated with the given key with another value.

Remarks

None.

See Also

Client Class, MySql.Data.MySqlClient.Memcached Namespace

Parameters

- `key`, The key for identifying the entry.

- `data`, The data to replace the value associated with the key.

- `expiration`, The interval of timespan, use `TimeSpan.Zero` for no expiration.

Client.Set Method

Syntax: Visual Basic

```
Public MustOverride Sub Set (
 key As String,
 data As Object,
 expiration As TimeSpan
)
```

Syntax: C#

```
public abstract void Set(
 string key,
 Object data,
 TimeSpan expiration
);
```

Set the value of a given key.

Remarks

None.

See Also

Client Class, MySql.Data.MySqlClient.Memcached Namespace

Parameters

- `key`, The key for identifying the entry.

- `data`, The data to associate with the given key.

- `expiration`, The interval of timespan, use `TimeSpan.Zero` for no expiration.

Client.GetInstance Method

Syntax: Visual Basic

```
Public Shared Function GetInstance (
 server As String,
 port As UInteger,
 flags As MemcachedFlags
) As Client
```

Syntax: C#

```
public static Client GetInstance(
 string server,
 uint port,
 MemcachedFlags flags
);
```

Factory method for creating instances of Client that implement a connection with the requested features. The connection object returned must be explicitly opened see method Open().

Remarks

None.

See Also

Client Class, MySql.Data.MySqlClient.Memcached Namespace

Parameters

- `server`, The Memcached server DNS or IP address.

- `port`, The port for the Memcached server.

- `flags`, A set of flags indicating characteristics requested.

Return Value

An instance of a client connection ready to be used.

10.3.3 TextClient Class

For a list of all members of this type, see TextClient Members .

Syntax: Visual Basic

```
Public Class TextClient
 Inherits Client
```

Syntax: C#

```
public class TextClient : Client
```

Thread Safety

Public static (Shared in Visual Basic) members of this type are safe for multithreaded operations. Instance members are not guaranteed to be thread-safe.

Requirements

Namespace: MySql.Data.MySqlClient.Memcached

Assembly: MySql.Data (in MySql.Data.dll)

See Also

TextClient Members, MySql.Data.MySqlClient.Memcached Namespace

10.3.3.1 TextClient Members

TextClient overview

Public Static Methods

GetInstance	Inherited from Client

Protected Instance Properties

server	Inherited from Client.
port	Inherited from Client.
stream	Inherited from Client.

Public Instance Methods

Open	Inherited from Client.
Close	Inherited from Client.
Add	Inherited from Client.
Append	Inherited from Client.

Cas	Inherited from Client.
Decrement>	Inherited from Client.
Delete	Inherited from Client.
FlushAll	Inherited from Client.
Get	Inherited from Client.
Increment	Inherited from Client.
Prepend	Inherited from Client.
Replace	Inherited from Client.
Set	Inherited from Client.

See Also

Client Class, MySql.Data.MySqlClient.Memcached Namespace

10.4 MySql.Data.MySqlClient.Replication Namespace

This section of the manual contains the API reference for Replication and Load balancing components.

Classes

Class	Description
ReplicationManager	Manager for Replication and Load Balancing features.
ReplicationRoundRobinServerGroup	Class that implements Round Robing Load Balancing technique.
ReplicationServer	Represents a server in Replication environment that contains information about
ReplicationServerGroup	Abstract class used to implement a custom load balancing plugin

10.4.1 ReplicationManager Class

Members

Namespace: MySql.Data.MySqlClient.Replication

Assembly: MySql.Data (in `MySql.Data.dll`)

Syntax C#

```
public static class ReplicationManager
```

Syntax Visual Basic

```
Public NotInheritable Class ReplicationManager
```

Syntax Visual C++

```
public ref class ReplicationManager abstract sealed
```

See Also

ReplicationManager Members MySql.Data.MySqlClient.Replication Namespace

10.4.1.1 ReplicationManager Members

The ReplicationManager type exposes the following members.

Public Static Methods

Name	Description
AddGroup(String, Int32)	Adds a Default Server Group to the list
AddGroup(String, String, Int32)	Adds a Server Group to the list
GetGroup	Gets a Server Group by name
GetNewConnection	Assigns a new server driver to the connection object
GetServer	Gets the next server from a replication group
IsReplicationGroup	Validates if the replication group name exists

Public Static Properties

Name	Description
Groups	Returns Replication Server Group List

See Also

ReplicationManager Class MySql.Data.MySqlClient.Replication Namespace

AddGroup Method

Overload List

- AddGroup(String, Int32)

- AddGroup(String, String, Int32)

See Also

ReplicationManager Class ReplicationManager Members MySql.Data.MySqlClient.Replication Namespace

ReplicationManager.AddGroup Method (String, Int32)

Adds a Default Server Group to the list

Syntax C#

```
public static ReplicationServerGroup AddGroup(
  string name,
  int retryTime
)
```

Syntax Visual Basic

```
Public Shared Function AddGroup ( _
  name As String, _
  retryTime As Integer _
) As ReplicationServerGroup
```

Syntax Visual C++

```
public:
static ReplicationServerGroup^ AddGroup(
  String^ name,
  int retryTime
)
```

Parameters

name

Type: System.String

Group name

retryTime

Type: System.Int32

Time between reconnections for failed servers

Return Value

A ReplicationServerGroup object

See Also

ReplicationManager Class AddGroup Overload MySql.Data.MySqlClient.Replication Namespace

ReplicationManager.AddGroup Method (String, String, Int32)

Adds a Server Group to the list

Syntax C#

```
public static ReplicationServerGroup AddGroup(
  string name,
  string groupType,
  int retryTime
)
```

Syntax Visual Basic

```
Public Shared Function AddGroup ( _
  name As String, _
  groupType As String, _
  retryTime As Integer _
```

```
) As ReplicationServerGroup
```

Syntax Visual C++

```
public:
static ReplicationServerGroup^ AddGroup(
  String^ name,
  String^ groupType,
  int retryTime
)
```

Parameters

name

Type: System.String

Group name

groupType

Type: System.String

ServerGroup type reference

retryTime

Type: System.Int32

Time between reconnections for failed servers

Return Value

A ReplicationServerGroup object

See Also

ReplicationManager Class AddGroup Overload MySql.Data.MySqlClient.Replication Namespace

ReplicationManager.GetGroup Method

Gets a Server Group by name

Syntax C#

```
public static ReplicationServerGroup GetGroup(
string groupName
)
```

Syntax Visual Basic

```
Public Shared Function GetGroup ( _
groupName As String _
) As ReplicationServerGroup
```

Syntax Visual C++

```
public:
static ReplicationServerGroup^ GetGroup(
String^ groupName
)
```

Parameters

groupName

Type: System.String

Group name

Return Value

A ReplicationServerGroup object

See Also

ReplicationManager Class MySql.Data.MySqlClient.Replication Namespace

ReplicationManager.GetNewConnection Method

Assigns a new server driver to the connection object

Syntax C#

```
public static void GetNewConnection(
  string groupName,
  bool master,
  MySqlConnection connection
)
```

Syntax Visual Basic

```
Public Shared Sub GetNewConnection ( _
  groupName As String, _
  master As Boolean, _
  connection As MySqlConnection _
)
```

Syntax Visual C++

```
public:
static void GetNewConnection(
  String^ groupName,
  bool master,
  MySqlConnection^ connection
)
```

Parameters

groupName

Type: System.String

Group name

master

Type: System.Boolean

True if the server connection to assign must be a master

connection

Type: MySql.Data.MySqlClient.MySqlConnection

MySqlConnection object where the new driver will be assigned

See Also

ReplicationManager Class MySql.Data.MySqlClient.Replication Namespace

ReplicationManager.GetServer Method

Gets the next server from a replication group

Syntax C#

```
public static ReplicationServer GetServer(
  string groupName,
  bool isMaster
)
```

Syntax Visual Basic

```
Public Shared Function GetServer ( _
  groupName As String, _
  isMaster As Boolean _
) As ReplicationServer
```

Syntax Visual C++

```
public:
static ReplicationServer^ GetServer(
  String^ groupName,
  bool isMaster
)
```

Parameters

groupName

Type: System.String

Group name

isMaster

Type: System.Boolean

True if the server to return must be a master

Return Value

A ReplicationServer object

See Also

ReplicationManager Class MySql.Data.MySqlClient.Replication Namespace

ReplicationManager.IsReplicationGroup Method

Validates if the replication group name exists.

Syntax C#

```
public static bool IsReplicationGroup(
  string groupName
)
```

Syntax Visual Basic

```
Public Shared Function IsReplicationGroup ( _
  groupName As String _
) As Boolean
```

Syntax Visual C++

```
public:
static bool IsReplicationGroup(
  String^ groupName
)
```

Parameters

groupName

Type: System.String

Group name

Return Value

True if replication group name is found, otherwise false

See Also

ReplicationManager Class MySql.Data.MySqlClient.Replication Namespace

ReplicationManager.Groups Property

Returns Replication Server Group List

Syntax C#

```
public static IList<ReplicationServerGroup> Groups { get; private set; }
```

Syntax Visual Basic

```
Public Shared Property Groups As IList(Of ReplicationServerGroup)
  Get
  Private Set
```

Syntax Visual C++

```
public:
static property IList<ReplicationServerGroup^>^ Groups {
  IList<ReplicationServerGroup^>^ get ();
  private: void set (IList<ReplicationServerGroup^>^ value);
}
```

See Also

ReplicationManager Class MySql.Data.MySqlClient.Replication Namespace

10.4.2 ReplicationRoundRobinServerGroup Class

Members

Namespace: MySql.Data.MySqlClient.Replication

Assembly: MySql.Data (in MySql.Data.dll)

Syntax C#

```
public class ReplicationRoundRobinServerGroup : ReplicationServerGroup
```

Syntax Visual Basic

```
Public Class ReplicationRoundRobinServerGroup _
  Inherits ReplicationServerGroup
```

Syntax Visual C++

```
public ref class ReplicationRoundRobinServerGroup : public ReplicationServerGroup
```

See Also

ReplicationRoundRobinServerGroup Members MySql.Data.MySqlClient.Replication Namespace

10.4.2.1 ReplicationRoundRobinServerGroup Members

The ReplicationRoundRobinServerGroup type exposes the following members.

Public Instance Constructors

Name	Description
ReplicationRoundRobinServerGroup	Initializes a new instance of the ReplicationRoundRobinServerGroup class

Public Instance Methods

Name	Description
AddServer	Adds a server into the group (Inherited from ReplicationServerGroup)
Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
GetServer(Boolean)	Gets an available server based on Round Robin load balancing (Overrides ReplicationServerGroup.GetServer(Boolean).)
GetServer(String)	Gets a server by name (Inherited from ReplicationServerGroup.)
GetType	Gets the Type of the current instance. (Inherited from Object.)
RemoveServer	Removes a server from group (Inherited from ReplicationServerGroup.)
ToString	Returns a string that represents the current object. (Inherited from Object.)

Public Instance Properties

Name	Description
Name	(Inherited from ReplicationServerGroup.)
RetryTime	(Inherited from ReplicationServerGroup.)
Servers	(Inherited from ReplicationServerGroup.)

See Also

ReplicationRoundRobinServerGroup Class MySql.Data.MySqlClient.Replication Namespace

ReplicationRoundRobinServerGroup Constructor

Initializes a new instance of the ReplicationRoundRobinServerGroup class

Syntax C#

```
public ReplicationRoundRobinServerGroup(
  string name,
```

```
  int retryTime
)
```

Syntax Visual Basic

```
Public Sub New ( _
  name As String, _
  retryTime As Integer _
)
```

Syntax Visual C++

```
public:
ReplicationRoundRobinServerGroup(
  String^ name,
  int retryTime
)
```

Parameters

name

Type: System.String

Server name

retryTime

Type: System.Int32

Retry time between connections to failed connections

See Also

ReplicationRoundRobinServerGroup Class MySql.Data.MySqlClient.Replication Namespace

GetServer Method

Overload List

- GetServer(Boolean)

- GetServer(String)

See Also

ReplicationRoundRobinServerGroup Class ReplicationRoundRobinServerGroup Members
MySql.Data.MySqlClient.Replication Namespace

ReplicationRoundRobinServerGroup.GetServer Method (Boolean)

Gets an available server based on Round Robin load balancing (Overrides
ReplicationServerGroup.GetServer(Boolean).)

Syntax C#

```
public override ReplicationServer GetServer(
  bool isMaster
)
```

Syntax Visual Basic

```
Public Overrides Function GetServer ( _
  isMaster As Boolean _
) As ReplicationServer
```

Syntax Visual C++

```
public:
virtual ReplicationServer^ GetServer(
  bool isMaster
) override
```

Parameters

isMaster

Type: System.Boolean

True if the server to return must be a master

Return Value

A ReplicationServer object

See Also

ReplicationRoundRobinServerGroup Class GetServer Overload MySql.Data.MySqlClient.Replication Namespace

10.4.3 ReplicationServer Class

Members

Namespace: MySql.Data.MySqlClient.Replication

Assembly: MySql.Data (in `MySql.Data.dll`)

Syntax C#

```
public class ReplicationServer
```

Syntax Visual Basic

```
Public Class ReplicationServer
```

Syntax Visual C++

```
public ref class ReplicationServer
```

See Also

ReplicationServer Members MySql.Data.MySqlClient.Replication Namespace

10.4.3.1 ReplicationServer Members

The ReplicationServer type exposes the following members.

Public Instance Constructors

Name	Description
ReplicationServer	Initializes a new instance of the ReplicationServer class

Public Instance Methods

Name	Description
Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
GetType	Gets the Type of the current instance. (Inherited from Object.)
ToString	Returns a string that represents the current object. (Inherited from Object.)

Public Instance Properties

Name	Description
ConnectionString	Connection string used to connect to the server
IsAvailable	Defines if the server is available to be considered in load balancing
IsMaster	Defines if the server is master (True) or slave
Name	Server name

See Also

ReplicationServer Class MySql.Data.MySqlClient.Replication Namespace

ReplicationServer Constructor

Initializes a new instance of the ReplicationServer class

Syntax C#

```
public ReplicationServer(
  string name,
  bool isMaster,
```

```
  string connectionString
)
```

Syntax Visual Basic

```
Public Sub New ( _
  name As String, _
  isMaster As Boolean, _
  connectionString As String _
)
```

Syntax Visual C++

```
public:
ReplicationServer(
  String^ name,
  bool isMaster,
  String^ connectionString
)
```

Parameters

name

Type: System.String

Server name

isMaster

Type: System.Boolean

Defines if the server is master (True) or slave

connectionString

Type: System.String

Connection string used to connect to the server

See Also

ReplicationServer Class MySql.Data.MySqlClient.Replication Namespace

ReplicationServer.ConnectionString Property

Connection string used to connect to the server

Syntax C#

```
public string ConnectionString { get; private set; }
```

Syntax Visual Basic

```
Public Property ConnectionString As String
  Get
  Private Set
```

Syntax Visual C++

```
public:
property String^ ConnectionString {
  String^ get ();
  private: void set (String^ value);
}
```

See Also

ReplicationServer Class MySql.Data.MySqlClient.Replication Namespace

ReplicationServer.IsAvailable Property

Defines if the server is available to be considered in load balancing

Syntax C#

```
public bool IsAvailable { get; set; }
```

Syntax Visual Basic

```
Public Property IsAvailable As Boolean
  Get
  Set
```

Syntax Visual C++

```
public:
property bool IsAvailable {
  bool get ();
  void set (bool value);
}
```

See Also

ReplicationServer Class MySql.Data.MySqlClient.Replication Namespace

ReplicationServer.IsMaster Property

Defines if the server is master (True) or slave

Syntax C#

```
public bool IsMaster { get; private set; }
```

Syntax Visual Basic

```
Public Property IsMaster As Boolean
  Get
  Private Set
```

Syntax Visual C++

```
public:
property bool IsMaster {
  bool get ();
  private: void set (bool value);
}
```

See Also

ReplicationServer Class MySql.Data.MySqlClient.Replication Namespace

ReplicationServer.Name Property

Server name

Syntax C#

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

Syntax Visual Basic

```
Public Property Name As String
  Get
  Private Set
```

Syntax Visual C++

```
public:
property String^ Name {
  String^ get ();
  private: void set (String^ value);
}
```

See Also

ReplicationServer Class MySql.Data.MySqlClient.Replication Namespace

10.4.4 ReplicationServerGroup Class

Members

Base class used to implement load balancing features

Namespace: MySql.Data.MySqlClient.Replication

Assembly: MySql.Data (in MySql.Data.dll)

Syntax C#

```
public abstract class ReplicationServerGroup
```

Syntax Visual Basic

```
Public MustInherit Class ReplicationServerGroup
```

Syntax Visual C++

```
public ref class ReplicationServerGroup abstract
```

See Also

ReplicationServerGroup Members MySql.Data.MySqlClient.Replication Namespace

10.4.4.1 ReplicationServerGroup Members

The ReplicationServerGroup type exposes the following members.

Public Instance Constructors

Name	Description
ReplicationServerGroup	Initializes a new instance of the ReplicationServerGroup class

Public Instance Methods

Name	Description
AddServer	Adds a server into the group
Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
GetServer(Boolean)	Must be implemented. Defines the next server for a custom load balancing implementation.
GetServer(String)	Gets a server by name.
GetType	Gets the Type of the current instance. (Inherited from Object.)
RemoveServer	Removes a server from group.
ToString	Returns a string that represents the current object. (Inherited from Object.)

Public Instance Properties

Name	Description
Name	Group name
RetryTime	Retry time between connections to failed servers

Name	Description
Servers	Servers list in the group

See Also

ReplicationServerGroup Class MySql.Data.MySqlClient.Replication Namespace

ReplicationServerGroup Constructor

Initializes a new instance of the ReplicationServerGroup class

Syntax C#

```
public ReplicationServerGroup(
  string name,
  int retryTime
)
```

Syntax Visual Basic

```
Public Sub New ( _
  name As String, _
  retryTime As Integer _
)
```

Syntax Visual C++

```
public:
ReplicationServerGroup(
  String^ name,
  int retryTime
)
```

Parameters

name

Type: System.String

Group name

retryTime

Type: System.Int32

Retry time between connections to failed servers

See Also

ReplicationServerGroup Class MySql.Data.MySqlClient.Replication Namespace

ReplicationServerGroup.AddServer Method

Adds a server into the group.

Syntax C#

```
public ReplicationServer AddServer(
  string name,
  bool isMaster,
  string connectionString
)
```

Syntax Visual Basic

```
Public Function AddServer ( _
  name As String, _
  isMaster As Boolean, _
  connectionString As String _
) As ReplicationServer
```

Syntax Visual C++

```
public:
ReplicationServer^ AddServer(
  String^ name,
  bool isMaster,
  String^ connectionString
)
```

Parameters

name

Type: System.String

Server name

isMaster

Type: System.Boolean

True if the server to add is master, False for slave server

connectionString

Type: System.String

Connection string used by this server

See Also

ReplicationServerGroup Class MySql.Data.MySqlClient.Replication Namespace

GetServer Method

Overload List

- GetServer(Boolean)

- GetServer(String)

See Also

ReplicationServerGroup Class ReplicationServerGroup Members MySql.Data.MySqlClient.Replication Namespace

ReplicationServerGroup.GetServer Method (Boolean)

Must be implemented. Defines the next server for a custom load balancing implementation.

Syntax C#

```
public abstract ReplicationServer GetServer(
  bool isMaster
)
```

Syntax Visual Basic

```
Public MustOverride Function GetServer ( _
  isMaster As Boolean _
) As ReplicationServer
```

Syntax Visual C++

```
public:
virtual ReplicationServer^ GetServer(
  bool isMaster
) abstract
```

Parameters

isMaster

Type: System.Boolean

Defines if the server to return is a master or any

Return Value

A ReplicationServer object

See Also

ReplicationServerGroup Class GetServer Overload MySql.Data.MySqlClient.Replication Namespace

ReplicationServerGroup.GetServer Method (String)

Gets a server by name.

Syntax C#

```
public ReplicationServer GetServer(
  string name
```

```
)
```

Syntax Visual Basic

```
Public Function GetServer ( _
  name As String _
) As ReplicationServer
```

Syntax Visual C++

```
public:
ReplicationServer^ GetServer(
  String^ name
)
```

Parameters

name

Type: System.String

Server name

Return Value

A ReplicationServer object

See Also

ReplicationServerGroup Class GetServer Overload MySql.Data.MySqlClient.Replication Namespace

ReplicationServerGroup.RemoveServer Method

Removes a server from group.

Syntax C#

```
public void RemoveServer(
  string name
)
```

Syntax Visual Basic

```
Public Sub RemoveServer ( _
  name As String _
)
```

Syntax Visual C++

```
public:
void RemoveServer(
  String^ name
)
```

Parameters

name

Type: System.String

Server name

See Also

[ReplicationServerGroup Class](#) [MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationServerGroup.Name Property

Group name.

Syntax C#

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

Syntax Visual Basic

```
Public Property Name As String
  Get
  Private Set
```

Syntax Visual C++

```
public:
property String^ Name {
  String^ get ();
  private: void set (String^ value);
}
```

See Also

[ReplicationServerGroup Class](#) [MySql.Data.MySqlClient.Replication Namespace](#)

ReplicationServerGroup.RetryTime Property

Retry time between connections to failed servers.

Syntax C#

```
public int RetryTime { get; private set; }
```

Syntax Visual Basic

```
Public Property RetryTime As Integer
  Get
  Private Set
```

Syntax Visual C++

```
public:
property int RetryTime {
  int get ();
  private: void set (int value);
}
```

See Also

ReplicationServerGroup Class MySql.Data.MySqlClient.Replication Namespace

ReplicationServerGroup.Servers Property

Servers list in the group.

Syntax C#

```
public IList<ReplicationServer> Servers { get; private set; }
```

Syntax Visual Basic

```
Public Property Servers As IList(Of ReplicationServer)
  Get
  Private Set
```

Syntax Visual C++

```
public:
property IList<ReplicationServer^>^ Servers {
  IList<ReplicationServer^>^ get ();
  private: void set (IList<ReplicationServer^>^ value);
}
```

See Also

ReplicationServerGroup Class MySql.Data.MySqlClient.Replication Namespace

Chapter 11 Connector/Net Support

Table of Contents

The developers of Connector/Net greatly value the input of our users in the software development process. If you find Connector/Net lacking some feature important to you, or if you discover a bug and need to file a bug report, please use the instructions in How to Report Bugs or Problems.

11.1 Connector/Net Community Support

- Community support for Connector/Net can be found through the forums at http://forums.mysql.com.

- Community support for Connector/Net can also be found through the mailing lists at http://lists.mysql.com.

- Paid support is available from Oracle. Additional information is available at http://dev.mysql.com/support/.

11.2 How to Report Connector/Net Problems or Bugs

If you encounter difficulties or problems with Connector/Net, contact the Connector/Net community, as explained in Section 11.1, "Connector/Net Community Support".

First try to execute the same SQL statements and commands from the `mysql` client program or from `admndemo`. This helps you determine whether the error is in Connector/Net or MySQL.

If reporting a problem, ideally include the following information with the email:

- Operating system and version.

- Connector/Net version.

- MySQL server version.

- Copies of error messages or other unexpected output.

- Simple reproducible sample.

Remember that the more information you can supply to us, the more likely it is that we can fix the problem.

If you believe the problem to be a bug, then you must report the bug through http://bugs.mysql.com/.

Chapter 12 Connector/Net FAQ

Questions

- 12.1: How do I obtain the value of an auto-incremented column?

Questions and Answers

12.1: How do I obtain the value of an auto-incremented column?

When using `CommandBuilder`, setting `ReturnGeneratedIdentifiers` property to `true` no longer works, as `CommandBuilder` does not add `last_insert_id()` by default.

CommandBuilder hooks up to the `DataAdapter.RowUpdating` event handler, which means it will get called for every row. It examines the command object and, if it is the same referenced object, it essentially rebuilds the object, thereby destroying your command text changes.

One approach to solving this problem is to clone the command object so you have a different actual reference:

```
dataAdapter.InsertCommand = cb.GetInsertCommand().Clone()
```

This will work, but since the CommandBuilder is still connected to the DataAdapter, the RowUpdating event will still fire and performance will be hit. To stop that, once all your commands have been added you need to disconnect the CommandBuilder from the DataAdapter:

```
cb.DataAdapter = null;
```

The last requirement is to make sure the `id` that is returned by `last_insert_id()` has the correct name. For example:

```
SELECT last_insert_id() AS id
```

A complete working example is shown here:

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

using System.Data;
using MySql.Data;
using MySql.Data.MySqlClient;

namespace GetAutoIncId
{
    class Program
    {
        static void Main(string[] args)
        {
            string connStr = "server=localhost;user=root;database=TestDB;port=3306;password=******;";
            MySqlConnection conn = new MySqlConnection(connStr);

            try
            {
                Console.WriteLine("Connecting to MySQL...");
                conn.Open();
```

```csharp
            string sql = "SELECT * FROM TestTable";

            MySqlDataAdapter da = new MySqlDataAdapter(sql, conn);
            MySqlCommandBuilder cb = new MySqlCommandBuilder(da);

            MySqlCommand cmd = new MySqlCommand();
            cmd.Connection = conn;
            cmd.CommandText = sql;
            // use Cloned object to avoid .NET rebuilding the object, and
            // thereby throwing away our command text additions.
            MySqlCommand insertCmd = cb.GetInsertCommand().Clone();
            insertCmd.CommandText = insertCmd.CommandText + ";SELECT last_insert_id() AS id";
            insertCmd.UpdatedRowSource = UpdateRowSource.FirstReturnedRecord;
            da.InsertCommand = insertCmd;
            cb.DataAdapter = null; // Unhook RowUpdating event handler

            DataTable dt = new DataTable();
            da.Fill(dt);

            DataRow row = dt.NewRow();
            row["name"] = "Joe Smith";

            dt.Rows.Add(row);
            da.Update(dt);

            System.Console.WriteLine("ID after update: " + row["id"]);
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }

        conn.Close();
        Console.WriteLine("Done.");
    }
  }
}
```