

MySQL Connector/Net Developer's Guide

MySQL Connector/Net Developer's Guide

Abstract

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.

Document generated on: 2012-12-04 (revision: 33367)

Table of Contents

Preface and Legal Notices	vii
1. MySQL Connector/Net	1
2. Connector/Net Versions	3
3. Installing Connector/Net	5
3.1. Installing Connector/Net on Windows	5
3.1.1. Installing Connector/Net using the Installer	5
3.1.2. Installing Connector/Net Using the Zip Packages	10
3.2. Installing Connector/Net on Unix with Mono	11
3.3. Installing Connector/Net from the Source Code	12
4. Connector/Net Visual Studio Integration	15
4.1. Making a Connection	15
4.2. Using IntelliSense in the SQL Editor	17
4.3. Editing Tables	18
4.3.1. Column Editor	19
4.3.2. Editing Indexes	20
4.3.3. Editing Foreign Keys	20
4.3.4. Column Properties	21
4.3.5. Table Properties	21
4.4. Editing Views	23
4.5. Editing Stored Procedures and Functions	24
4.6. Editing Triggers	26
4.7. Editing User Defined Functions (UDF)	27
4.8. Debugging Stored Procedures and Functions	27
4.9. Cloning Database Objects	37
4.10. Dropping Database Objects	37
4.11. Using the ADO.NET Entity Framework	37
4.12. MySQL Website Configuration Tool	38
4.13. MySQL SQL Editor	42
4.14. DDL T4 Template Macro	43
5. Connector/Net Tutorials	47
5.1. Tutorial: An Introduction to Connector/Net Programming	47
5.1.1. The MySqlConnection Object	47
5.1.2. The MySqlCommand Object	48
5.1.3. Working with Decoupled Data	50
5.1.4. Working with Parameters	53
5.1.5. Working with Stored Procedures	54
5.2. Tutorial: MySQL Connector/Net ASP.NET Membership and Role Provider	56
5.3. Tutorial: MySQL Connector/Net ASP.NET Session State Provider	60
5.4. Tutorial: MySQL Connector/Net ASP.NET Profile Provider	62
5.5. Tutorial: Using an Entity Framework Entity as a Windows Forms Data Source	64
5.6. Tutorial: Databinding in ASP.NET using LINQ on Entities	76
5.7. Tutorial: Using SSL with MySQL Connector/Net	80
5.8. Tutorial: Using MySqlScript	82
5.8.1. Using Delimiters with MySqlScript	84
5.9. Tutorial: Generating MySQL DDL from an Entity Framework Model	86
6. Connector/Net Programming	87
6.1. Connecting to MySQL Using Connector/Net	88
6.2. Creating a Connector/Net Connection String	88
6.2.1. Opening a Connection	88
6.2.2. Handling Connection Errors	90
6.2.3. Using GetSchema on a Connection	91
6.3. Using MySqlCommand	92
6.4. Using Connector/Net with Connection Pooling	93
6.5. Using the Windows Native Authentication Plugin	94
6.6. Writing a Custom Authentication Plugin	94

6.7. Using Connector/Net with Table Caching	98
6.8. Using the Connector/Net with Prepared Statements	98
6.8.1. Preparing Statements in Connector/Net	98
6.9. Accessing Stored Procedures with Connector/Net	99
6.9.1. Using Stored Routines from Connector/Net	100
6.10. Handling BLOB Data With Connector/Net	102
6.10.1. Preparing the MySQL Server	102
6.10.2. Writing a File to the Database	103
6.10.3. Reading a BLOB from the Database to a File on Disk	104
6.11. Using the Connector/Net Interceptor Classes	105
6.12. Handling Date and Time Information in Connector/Net	107
6.12.1. Fractional Seconds	107
6.12.2. Problems when Using Invalid Dates	107
6.12.3. Restricting Invalid Dates	107
6.12.4. Handling Invalid Dates	108
6.12.5. Handling NULL Dates	108
6.13. Using the MySQLBulkLoader Class	108
6.14. Using the MySQL Connector/Net Trace Source Object	110
6.14.1. Viewing MySQL Trace Information	111
6.14.2. Building Custom Listeners	113
6.15. Binary/Nonbinary Issues	115
6.16. Character Set Considerations for Connector/Net	115
6.17. Using Connector/Net with Crystal Reports	116
6.17.1. Creating a Data Source	116
6.17.2. Creating the Report	117
6.17.3. Displaying the Report	117
6.18. ASP.NET Provider Model	120
6.19. Working with Partial Trust / Medium Trust	122
6.19.1. Evolution of Partial Trust Support Across Connector/Net Versions	122
6.19.2. Configuring Partial Trust with Connector/Net Library Installed in GAC	123
6.19.3. Configuring Partial Trust with Connector/Net Library Not Installed in GAC	124
7. Connector/Net Connection String Options Reference	127
8. Connector/Net API Reference	135
8.1. MySQL.Data.MySqlClient Namespace	135
8.1.1. MySQL.Data.MySqlClientHierarchy	136
8.1.2. BaseCommandInterceptor	136
8.1.3. BaseExceptionInterceptor	137
8.1.4. MySQLCommand Class	137
8.1.5. MySQLCommandBuilder Class	199
8.1.6. MySQLException Class	216
8.1.7. MySQLHelper Class	217
8.1.8. MySQLErrorCode Enumeration	227
8.2. MySQL.Data.Types Namespace	228
8.2.1. MySQL.Data.TypesHierarchy	228
8.2.2. MySQLConversionException Class	228
8.2.3. MySQLDateTime Class	230
9. Connector/Net Support	239
9.1. Connector/Net Community Support	239
9.2. How to Report Connector/Net Problems or Bugs	239
9.3. Connector/Net Change History	239
A. MySQL Connector/Net Change History	241
A.1. Changes in MySQL Connector/Net Version 6.6	243
A.2. Changes in MySQL Connector/Net Version 6.5	245
A.3. Changes in MySQL Connector/Net Version 6.4	247
A.3.1. Changes in MySQL Connector/Net 6.4.6 (2012-11-26)	247
A.3.2. Changes in MySQL Connector/Net 6.4.5 (2012-05-19)	249
A.3.3. Changes in MySQL Connector/Net 6.4.4 (2011-09-26)	250
A.3.4. Changes in MySQL Connector/Net 6.4.3 (2011-07-03)	251

A.3.5. Changes in MySQL Connector/Net 6.4.2 (2011-06-29)	251
A.3.6. Changes in MySQL Connector/Net 6.4.1 (2011-06-06, Alpha)	251
A.3.7. Changes in MySQL Connector/Net 6.4.0 (Unknown)	251
A.4. Changes in MySQL Connector/Net Version 6.3	251
A.4.1. Changes in MySQL Connector/Net 6.3.9 (2012-04-11)	251
A.4.2. Changes in MySQL Connector/Net 6.3.8 (2011-12-16)	253
A.4.3. Changes in MySQL Connector/Net 6.3.7 (2011-06-22)	255
A.4.4. Changes in MySQL Connector/Net 6.3.6 (2011-01-03)	255
A.4.5. Changes in MySQL Connector/Net 6.3.5 (2010-10-12)	257
A.4.6. Changes in MySQL Connector/Net 6.3.4 (2010-09-01, Generally Available)	257
A.4.7. Changes in MySQL Connector/Net 6.3.3 (2010-07-27)	258
A.4.8. Changes in MySQL Connector/Net 6.3.2 (2010-05-24, Beta)	260
A.4.9. Changes in MySQL Connector/Net 6.3.1 (2010-03-02)	261
A.4.10. Changes in MySQL Connector/Net 6.3.0 (2010-02-16, Alpha)	262
A.5. Changes in MySQL Connector/Net Version 6.2	262
A.5.1. Changes in MySQL Connector/Net 6.2.6 (Not yet released)	262
A.5.2. Changes in MySQL Connector/Net 6.2.5 (2011-07-01)	262
A.5.3. Changes in MySQL Connector/Net 6.2.4 (2010-08-30)	264
A.5.4. Changes in MySQL Connector/Net 6.2.3 (2010-04-10)	267
A.5.5. Changes in MySQL Connector/Net 6.2.2 (2009-12-22, Generally Available)	268
A.5.6. Changes in MySQL Connector/Net 6.2.1 (2009-11-16, Beta)	269
A.5.7. Changes in MySQL Connector/Net 6.2.0 (2009-10-21, Alpha)	270
A.6. Changes in MySQL Connector/Net Version 6.1	271
A.6.1. Changes in MySQL Connector/Net 6.1.7 (Not released, Generally Available)	271
A.6.2. Changes in MySQL Connector/Net 6.1.6 (Not released, Generally Available)	271
A.6.3. Changes in MySQL Connector/Net 6.1.5 (2010-08-30, Generally Available)	271
A.6.4. Changes in MySQL Connector/Net 6.1.4 (2010-04-28, Generally Available)	273
A.6.5. Changes in MySQL Connector/Net 6.1.3 (2009-11-16, Generally Available)	275
A.6.6. Changes in MySQL Connector/Net 6.1.2 (2009-09-08, Generally Available)	276
A.6.7. Changes in MySQL Connector/Net 6.1.1 (2009-08-20, Beta)	277
A.6.8. Changes in MySQL Connector/Net 6.1.0 (2009-07-15, Alpha)	279
A.7. Changes in MySQL Connector/Net Version 6.0	280
A.7.1. Changes in MySQL Connector/Net 6.0.8 (Not released)	280
A.7.2. Changes in MySQL Connector/Net 6.0.7 (2010-08-30)	280
A.7.3. Changes in MySQL Connector/Net 6.0.6 (2010-04-28)	281
A.7.4. Changes in MySQL Connector/Net 6.0.5 (2009-11-12)	283
A.7.5. Changes in MySQL Connector/Net 6.0.4 (2009-06-16)	288
A.7.6. Changes in MySQL Connector/Net 6.0.3 (2009-04-28)	289
A.7.7. Changes in MySQL Connector/Net 6.0.2 (2009-04-07, Beta)	290
A.7.8. Changes in MySQL Connector/Net 6.0.1 (2009-04-02, Beta)	290
A.7.9. Changes in MySQL Connector/Net 6.0.0 (2009-03-02, Alpha)	290
A.8. Changes in MySQL Connector/Net Version 5.3	290
A.8.1. Changes in MySQL Connector/Net 5.3.0 (Not released)	290
A.9. Changes in MySQL Connector/Net Version 5.2	290
A.9.1. Changes in MySQL Connector/Net 5.2.8 (Not released)	290
A.9.2. Changes in MySQL Connector/Net 5.2.7 (2009-07-15)	290
A.9.3. Changes in MySQL Connector/Net 5.2.6 (2009-04-28)	291
A.9.4. Changes in MySQL Connector/Net 5.2.5 (2008-11-19)	293
A.9.5. Changes in MySQL Connector/Net 5.2.4 (2008-11-13)	293
A.9.6. Changes in MySQL Connector/Net 5.2.3 (2008-08-19)	294
A.9.7. Changes in MySQL Connector/Net 5.2.2 (2008-05-12)	295
A.9.8. Changes in MySQL Connector/Net 5.2.1 (2008-02-27)	296
A.9.9. Changes in MySQL Connector/Net 5.2.0 (2008-02-11)	296
A.10. Changes in MySQL Connector/Net Version 5.1	297
A.10.1. Changes in MySQL Connector/Net 5.1.8 (Not released)	297
A.10.2. Changes in MySQL Connector/Net 5.1.7 (2008-08-21)	297
A.10.3. Changes in MySQL Connector/Net 5.1.6 (2008-05-12)	298
A.10.4. Changes in MySQL Connector/Net 5.1.5 (Not released)	299

A.10.5. Changes in MySQL Connector/Net 5.1.4 (2007-11-20)	299
A.10.6. Changes in MySQL Connector/Net 5.1.3 (2007-09-21, Beta)	300
A.10.7. Changes in MySQL Connector/Net 5.1.2 (2007-06-18)	301
A.10.8. Changes in MySQL Connector/Net 5.1.1 (2007-05-23)	301
A.10.9. Changes in MySQL Connector/Net 5.1.0 (2007-05-01)	301
A.11. Changes in MySQL Connector/Net Version 5.0	302
A.11.1. Changes in MySQL Connector/Net 5.0.10 (Not released)	302
A.11.2. Changes in MySQL Connector/Net 5.0.9 (Not released)	302
A.11.3. Changes in MySQL Connector/Net 5.0.8 (2007-08-21)	302
A.11.4. Changes in MySQL Connector/Net 5.0.7 (2007-05-18)	303
A.11.5. Changes in MySQL Connector/Net 5.0.6 (2007-03-22)	303
A.11.6. Changes in MySQL Connector/Net 5.0.5 (2007-03-07)	304
A.11.7. Changes in MySQL Connector/Net 5.0.4 (Not released)	305
A.11.8. Changes in MySQL Connector/Net 5.0.3 (2007-01-05)	305
A.11.9. Changes in MySQL Connector/Net 5.0.2 (2006-11-06)	306
A.11.10. Changes in MySQL Connector/Net 5.0.1 (2006-10-01)	307
A.11.11. Changes in MySQL Connector/Net 5.0.0 (2006-08-08)	307
A.12. Changes in MySQL Connector/Net Version 1.0	308
A.12.1. Changes in MySQL Connector/Net 1.0.11 (Not released)	308
A.12.2. Changes in MySQL Connector/Net 1.0.10 (2007-08-24)	308
A.12.3. Changes in MySQL Connector/Net 1.0.9 (2007-02-02)	308
A.12.4. Changes in MySQL Connector/Net 1.0.8 (2006-10-20)	310
A.12.5. Changes in MySQL Connector/Net 1.0.7 (2005-11-21)	311
A.12.6. Changes in MySQL Connector/Net 1.0.6 (2005-10-03)	311
A.12.7. Changes in MySQL Connector/Net 1.0.5 (2005-08-29)	312
A.12.8. Changes in MySQL Connector/Net 1.0.4 (2005-01-20)	312
A.12.9. Changes in MySQL Connector/Net 1.0.3 (2004-10-12, gamma)	313
A.12.10. Changes in MySQL Connector/Net 1.0.2 (2004-11-15, gamma)	314
A.12.11. Changes in MySQL Connector/Net 1.0.1 (2004-10-27, Beta)	314
A.12.12. Changes in MySQL Connector/Net 1.0.0 (2004-09-01)	315
A.13. Changes in MySQL Connector/Net Version 0.9.0 (30 August 2004)	316
A.14. Changes in MySQL Connector/Net Version 0.76	319
A.15. Changes in MySQL Connector/Net Version 0.75	320
A.16. Changes in MySQL Connector/Net Version 0.74	320
A.17. Changes in MySQL Connector/Net Version 0.71	322
A.18. Changes in MySQL Connector/Net Version 0.70	322
A.19. Changes in MySQL Connector/Net Version 0.68	324
A.20. Changes in MySQL Connector/Net Version 0.65	325
A.21. Changes in MySQL Connector/Net Version 0.60	325
A.22. Changes in MySQL Connector/Net Version 0.50	325
B. Licenses for Third-Party Components	327
B.1. ANTLR 3.3 License	327
B.2. RFC 3174 - US Secure Hash Algorithm 1 (SHA1) License	327
B.3. zlib License	328
B.4. ZLIB.NET License	328

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

Copyright © 1997, 2012, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. MySQL is a trademark of Oracle Corporation and/or its affiliates, and shall not be used without Oracle's express written authorization. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this material is subject to the terms and conditions of your Oracle Software License and Service Agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle or as specifically provided below. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

This documentation is NOT distributed under a GPL license. Use of this documentation is subject to the following terms:

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.

For more information on the terms of this license, or for details on how the MySQL documentation is built and produced, please visit [MySQL Contact & Questions](#).

For additional licensing information, including licenses for third-party libraries used by MySQL products, see [Preface and 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](#).

Chapter 1. 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#.

Connector/Net includes full support for:

- Features provided by MySQL Server up to and including MySQL Server version 5.5.
- Large-packet support for sending and receiving rows and BLOBs 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.
- Fully managed, does not utilize the MySQL client library.

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 [Documentation.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 [Chapter 4, Connector/Net Visual Studio Integration](#).

Note

Connector/Net 5.1.2 and later include the Visual Studio Plugin by default.

MySQL Connector/Net supports full versions of Visual Studio 2005, 2008, and 2010, although certain features are only available in Visual Studio 2010 when using MySQL Connector/Net version 6.3.2 and later. Note that MySQL Connector/Net does not currently support Express versions of Microsoft products, including Microsoft Visual Web Developer.

Key topics:

- For connection string properties when using the [MySqlConnection](#) class, see [Chapter 7, Connector/Net Connection String Options Reference](#).

Chapter 2. Connector/Net Versions

There are several versions of Connector/Net available:

- Connector/Net 6.6 includes support for MySQL Server 5.6, 5.5, 5.1, and 5.0. Important new features include stored procedure debugging in Microsoft Visual Studio, support for pluggable authentication including the ability to write your own authentication plugin, 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.
- Connector/Net 6.5 includes support for MySQL Server 5.6, 5.5, 5.1, and 5.0. Important new features include 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.
- Connector/Net 6.4 includes support for MySQL Server 5.5, 5.1, and 5.0. Important new features include 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.
- Connector/Net 6.3 includes support for MySQL Server 5.5, 5.1, and 5.0. Important new features include integration with Visual Studio 2010, such as 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.
- Connector/Net 6.2 includes support for MySQL Server 5.5, 5.1, 5.0, and 4.1. Important new features include a new logging system and client SSL certificates.
- Connector/Net 6.1 includes support for MySQL Server 5.5, 5.1, 5.0, and 4.1. Important new features include the MySQL Website Configuration Tool and a Session State Provider.
- Connector/Net 6.0 includes support for MySQL Server 5.5, 5.1, 5.0, and 4.1.

This version of Connector/Net is no longer supported.

- Connector/Net 5.2 includes support for MySQL Server 5.5, 5.1, 5.0, and 4.1 features. Connector/Net 5.2 also 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 MySQL Server 5.5, 5.1, 5.0, 4.1, and 4.0 features. Connector/Net 5.1 also 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 support for MySQL Server 5.1, 5.0, 4.1 and 4.0 features. Connector/Net 5.0 also 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 support for MySQL Server 5.0, 4.1, and 4.0 features, and full compatibility with the ADO.NET driver interface.

This version of Connector/Net is no longer supported.

The latest source code for Connector/Net can be downloaded from the MySQL public Subversion server. For further details, see [Section 3.3, “Installing Connector/Net from the Source Code”](#).

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
6.6	2.x+	2.x+, 4.x+ for VS 2010 support	5.6, 5.5, 5.1, 5.0	Yes
6.5	2.x+	2.x+, 4.x+ for VS 2010 support	5.6, 5.5, 5.1, 5.0	Yes
6.4	2.x+	2.x+, 4.x+ for VS 2010 support	5.6, 5.5, 5.1, 5.0	Yes
6.3	2.x+	2.x+, 4.x+ for VS 2010 support	5.6, 5.5, 5.1, 5.0	Yes
6.2	2.x+	2.x+	5.6, 5.5, 5.1, 5.0, 4.1	Yes
6.1	2.x+	2.x+	5.6, 5.5, 5.1, 5.0, 4.1	Yes
6.0	2.x+	2.x+	5.5, 5.1, 5.0, 4.1	Critical issues only
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, and Z>=2 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. Installing Connector/Net

Table of Contents

3.1. Installing Connector/Net on Windows	5
3.1.1. Installing Connector/Net using the Installer	5
3.1.2. Installing Connector/Net Using the Zip Packages	10
3.2. Installing Connector/Net on Unix with Mono	11
3.3. Installing Connector/Net from the Source Code	12

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 framework (see <http://www.mono-project.com>).

Connector/Net is available for download from <http://dev.mysql.com/downloads/connector/net/>.

3.1. Installing Connector/Net on Windows

On Windows, you can install either through a binary installation process or by downloading a zip file with the Connector/Net components.

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

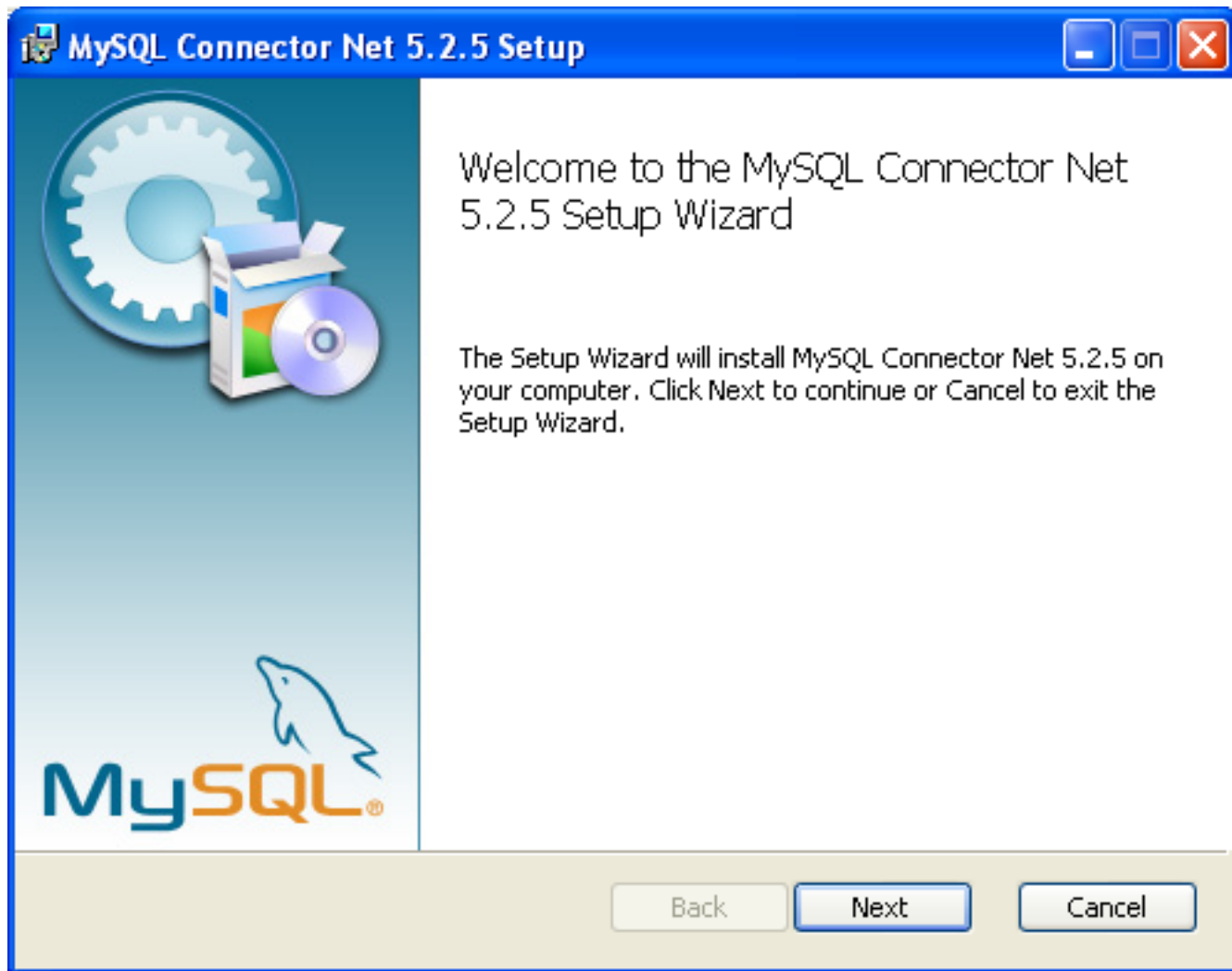
3.1.1. Installing Connector/Net using the Installer

Using the installer is the most straightforward method of installing Connector/Net on Windows and the installed components include the source code, test code and full reference documentation.

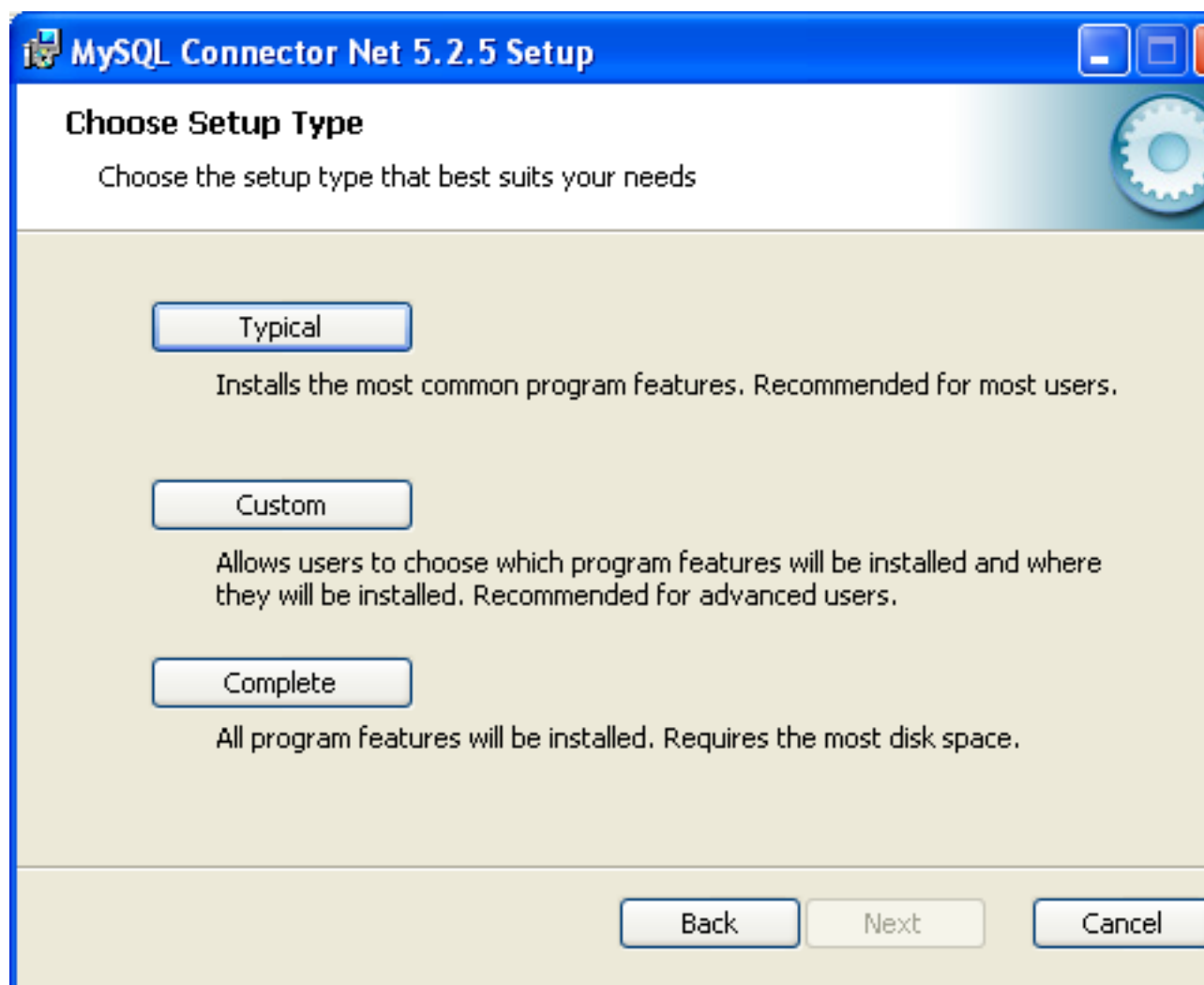
You install Connector/Net through a Windows Installer ([`.msi`](#)) installation package, which can install Connector/Net on all Windows operating systems. The MSI package is contained within a zip archive named [`mysql-connector-net-version.zip`](#), where [`version`](#) indicates the Connector/Net version.

To install Connector/Net:

1. Double-click the MSI installer file extracted from the zip you downloaded. Click **Next** to start the installation.



2. You must choose the type of installation to perform.



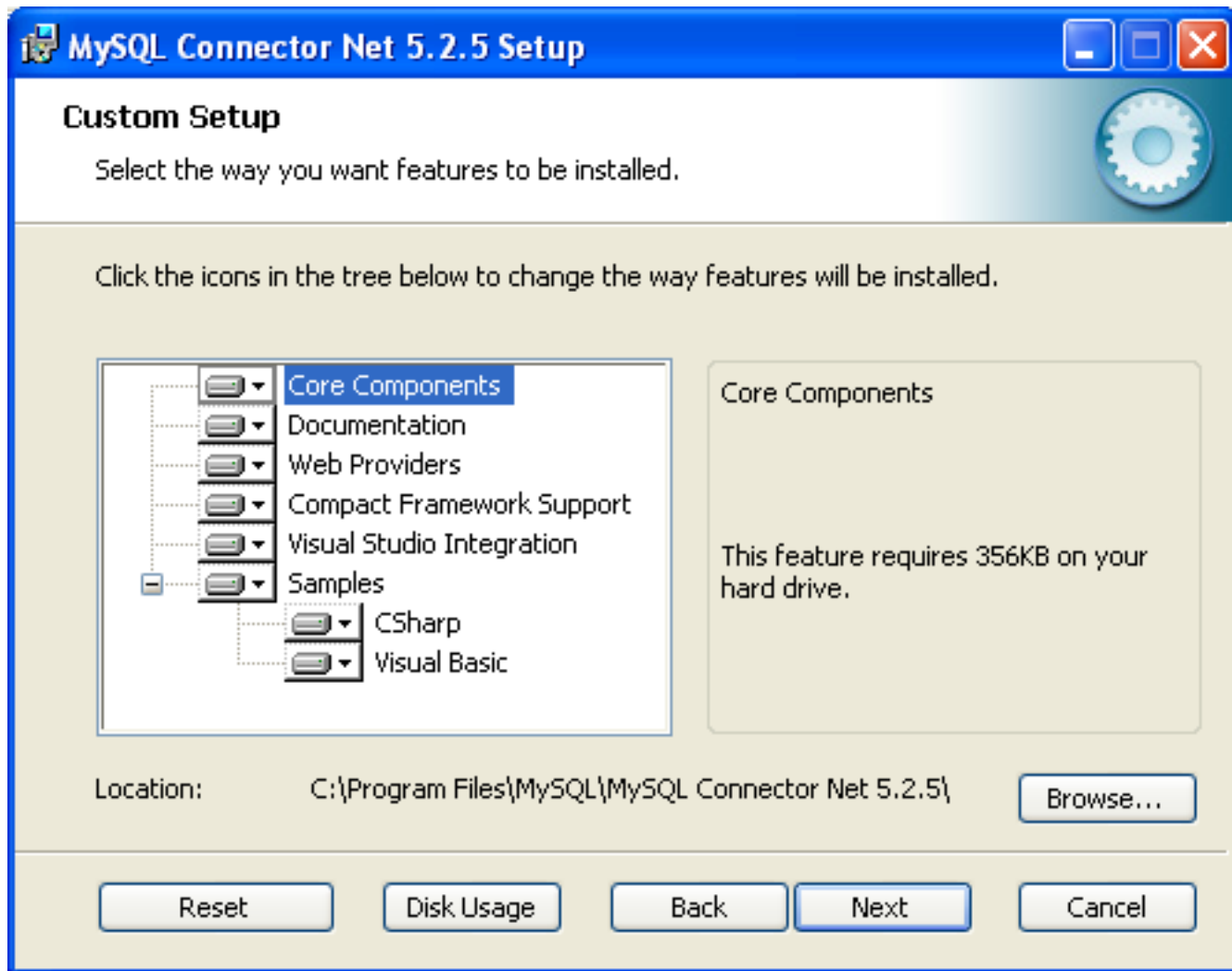
For most situations, the Typical installation is suitable. Click the **Typical** button and proceed to Step 5. A Complete installation installs all the available files. To conduct a Complete installation, click the **Complete** button and proceed to step 5. To customize your installation, including choosing 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.

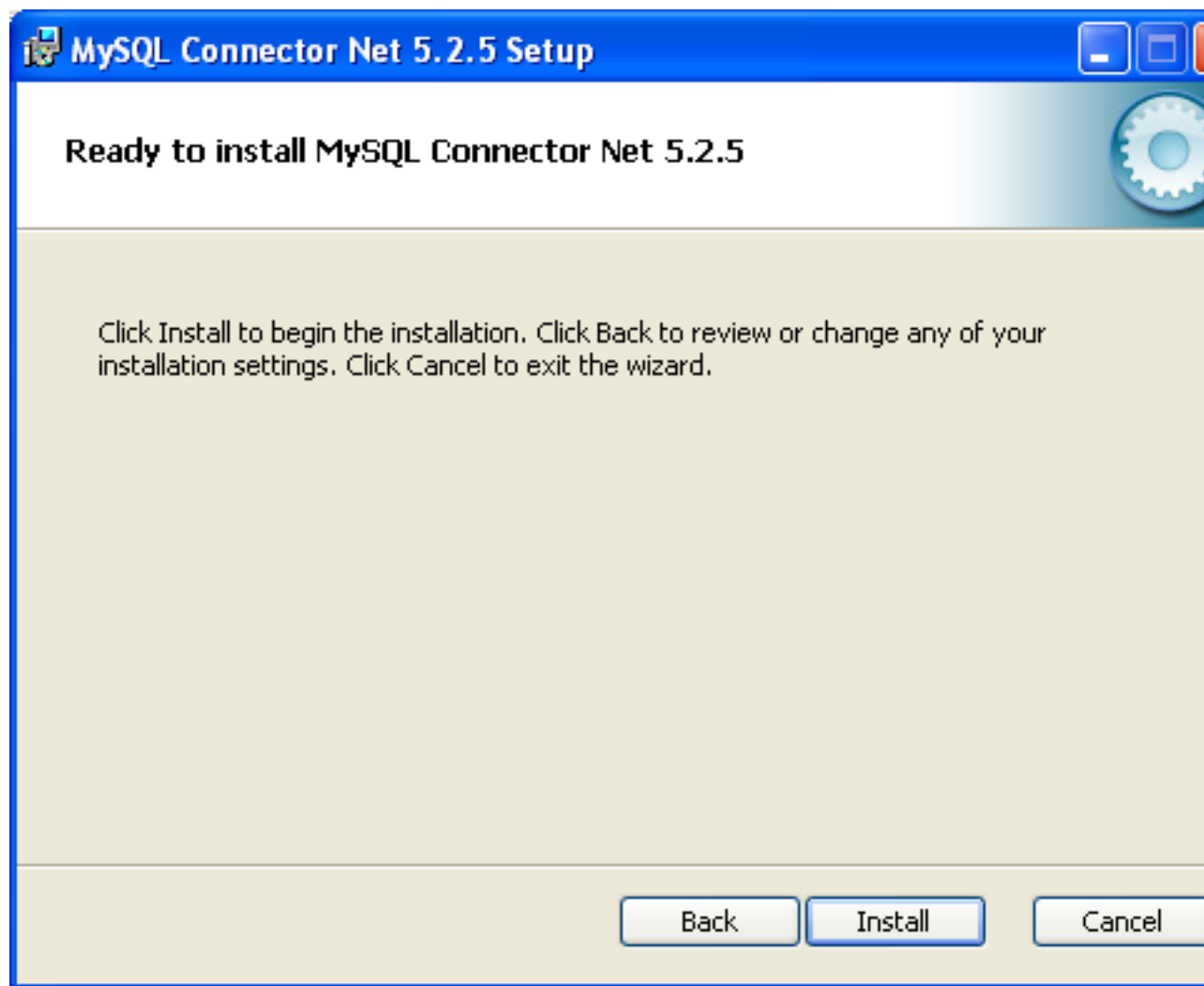
3. If you have chosen a custom installation, you can select the individual components to install, including the core interface component, supporting documentation (a CHM file) samples and examples, and the source code. Select the items, and their installation level, and then click **Next** to continue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

Note

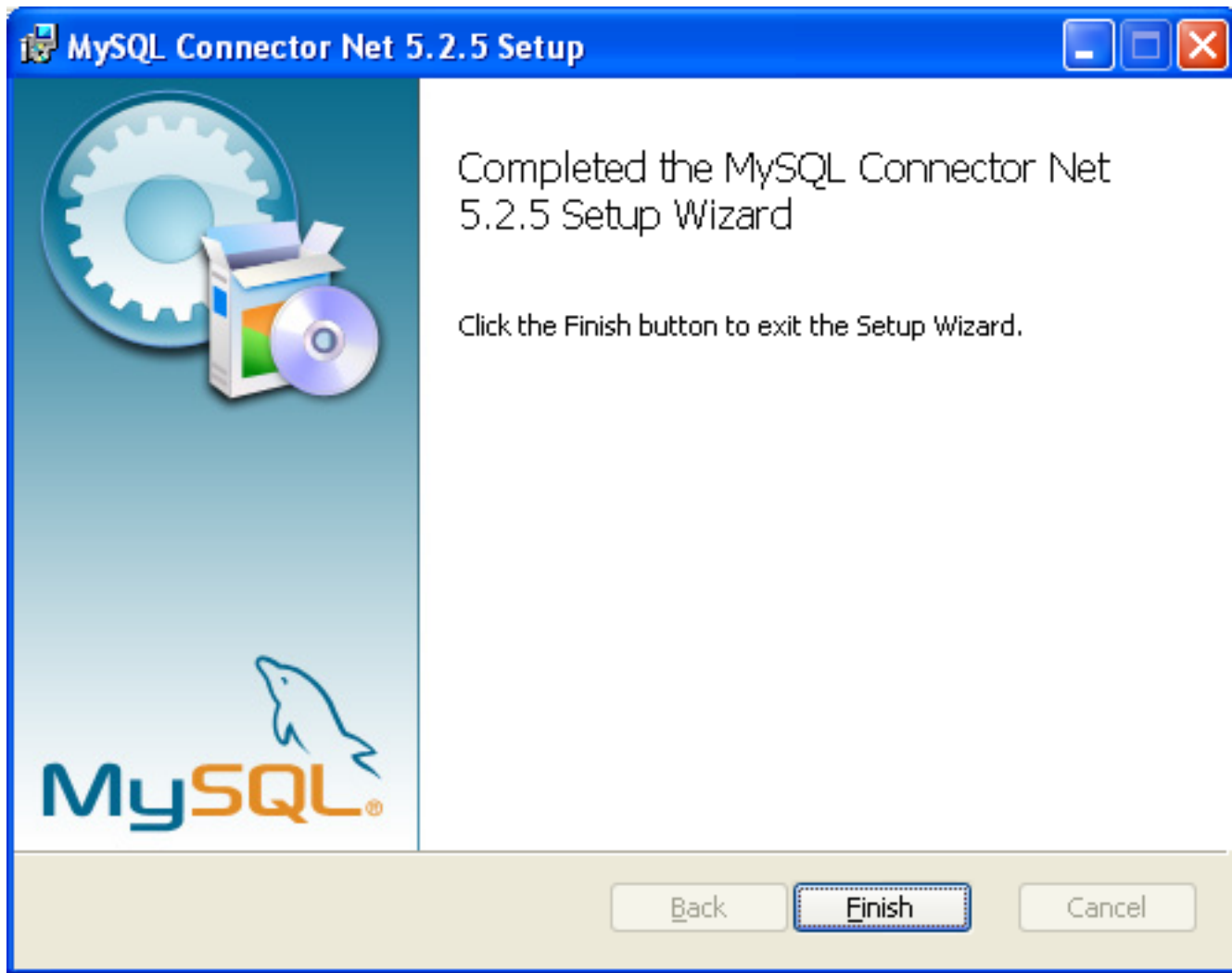
For Connector/Net 1.0.8 or lower and Connector 5.0.4 and lower the installer will attempt to install binaries for both 1.x and 2.x of the .NET Framework. If you only have one version of the framework installed, the connector installation may fail. If this happens, you can choose the framework version to be installed through the custom installation step.



4. You will be given a final opportunity to confirm the installation. Click **Install** to copy and install the files onto your machine.



5. Once the installation has been completed, click **Finish** to exit the installer.



Unless you choose otherwise, Connector/Net is installed in `C:\Program Files\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:

- `bin`: Connector/Net MySQL libraries for different versions of the .NET environment.
- `docs`: Connector/Net documentation in CHM format.
- `samples`: Sample code and applications that use the Connector/Net component.
- `src`: The source code for 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.

3.1.2. Installing Connector/Net Using the Zip Packages

If you have problems running the installer, you can download a zip file without an installer as an alternative. That file is called `mysql-connector-net-version-noinstall.zip`. After downloading the zip archive, extract the files to a location of your choice.

The file contains the following directories:

- `bin`: Connector/Net MySQL libraries for different versions of the .NET environment.
- `Docs`: Connector/Net documentation in CHM format.
- `Samples`: Sample code and applications that use the Connector/Net component.

Connector/Net 6.0.x has a different directory structure:

- `Assemblies`: A collection of DLLs that make up the connector functionality.
- `Documentation`: Connector/Net documentation in CHM format.
- `Samples`: sample code and applications that use the Connector/Net component.

Another zip file available for download contains the source code distribution. This file is named `mysql-connector-net-version-src.zip`.

The file contains the following directories:

- `Documentation`: Source files to build the documentation into the compiled HTML (CHM) format.
- `Installer`: Source files to build the Connector/Net installer program.
- `MySql.Data`: Source files for the core data provider.
- `MySql.VisualStudio`: Source files for the Microsoft Visual Studio extensions.
- `MySql.Web`: Source files for the web providers. This includes code for the membership provider, role provider and profile provider. These are used in ASP.NET web sites.
- `Samples`: Source files for several example applications.
- `Tests`: A spreadsheet listing test cases.
- `VisualStudio`: Resources used by the Visual Studio plugin.

Finally, ensure that `MySql.Data.dll` is accessible to your program at build time (and run time). If using Microsoft Visual Studio, add `MySql.Data` as a Reference to your project.

Note

If using MySQL Connector/Net 6.3.5 and above, the `MySql.Data.dll` file provided will work with both .NET Framework 2.x and 4.x.

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 that there are two Mono C# compilers available, `mcs`, which accesses the 1.0-profile libraries, and `gmcs`, which accesses the 2.0-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
```

Note, the assemblies that are referenced depend on the requirements of the application, but applications using Connector/Net must provide `-r:MySQL.Data` as 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

Caution

Read this section only if you are interested in helping us test our new code. If you just want to get Connector/Net up and running on your system, use a standard release distribution.

Obtaining the Source Code

To obtain the most recent development source tree, first download and install Bazaar. You can obtain Bazaar from the [Bazaar VCS Website](#). Bazaar is supported by any platform that supports Python, and is therefore compatible with any Linux, Unix, Windows or Mac OS X host. Instructions for downloading and installing Bazaar on the different platforms are available on the Bazaar Web site.

The most recent development source tree is available from our public Subversion trees at <http://dev.mysql.com/tech-resources/sources.html>.

To check out the Connector/Net sources, change to the directory where you want the copy of the Connector/Net tree to be stored, then use the following command:

```
shell> bzip branch lp:connectornet/trunk
```

To download a specific version of Connector/Net, specify the version number instead of `trunk`. For example, to obtain a copy of the 6.0 version of the source tree:

```
shell> bzip branch lp:connectornet/6.0
```

Source packages are also available on the downloads page.

Building the Source Code on Windows

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

- Obtain the source code, either from the Subversion server, or through one of the prepared source code packages.
- Navigate to the root of the source code tree.
- A Microsoft Visual Studio 2005 solution file is available to build the connector, this is called [MySQL-
VS2005.sln](#). Click this file to load the solution into 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 currently not available.

Chapter 4. Connector/Net Visual Studio Integration

Table of Contents

4.1. Making a Connection	15
4.2. Using IntelliSense in the SQL Editor	17
4.3. Editing Tables	18
4.3.1. Column Editor	19
4.3.2. Editing Indexes	20
4.3.3. Editing Foreign Keys	20
4.3.4. Column Properties	21
4.3.5. Table Properties	21
4.4. Editing Views	23
4.5. Editing Stored Procedures and Functions	24
4.6. Editing Triggers	26
4.7. Editing User Defined Functions (UDF)	27
4.8. Debugging Stored Procedures and Functions	27
4.9. Cloning Database Objects	37
4.10. Dropping Database Objects	37
4.11. Using the ADO.NET Entity Framework	37
4.12. MySQL Website Configuration Tool	38
4.13. MySQL SQL Editor	42
4.14. DDL T4 Template Macro	43

When MySQL Connector/Net is installed on Microsoft Windows, Visual Studio integration components are also installed and initialized. This enables the developer to work seamlessly with MySQL Connector/Net in the familiar Visual Studio environment, as described in the following sections of the manual.

MySQL Connector/Net supports Visual Studio versions 2005, 2008, and 2010. However, only MySQL Connector/Net version 6.3 and higher fully integrate with Visual Studio 2010, although applications using earlier versions of the connector can be built with the Visual Studio 2010 environment using .NET 2.x frameworks.

Visual Studio 2010 support was introduced with MySQL Connector/Net 6.3.2. From version 6.3.2 the connector ships with both .NET 2.x and .NET 4.x versions of the entity framework support files, [mysql.data.ef.dll](#) and [mysql.visualstudio.dll](#). The .NET 4.x versions are required to enable new integration features supported in Visual Studio 2010, including:

- New DDL T4 template for the Entity Framework (EF) - This enables developers to design an EF model from scratch and use the native Visual Studio 2010 facility to generate MySQL DDL from that model. This is done by creating the model, and with the model open, choosing the SSDLToMySQL template in the properties window. The correct DDL is then generated. The developer can then save this code as a [.mysql](#) file in their project and execute it against the MySQL server.
- New SQL Editor - A new SQL editor lets you connect to a MySQL server to execute SQL. This is activated by creating a new file with a [.mysql](#) extension. A new template lets you create files with this file type using the Visual Studio 2010 main menu item **File, New**. Note that the MySQL SQL Editor is also available in Visual Studio 2005 and 2008.

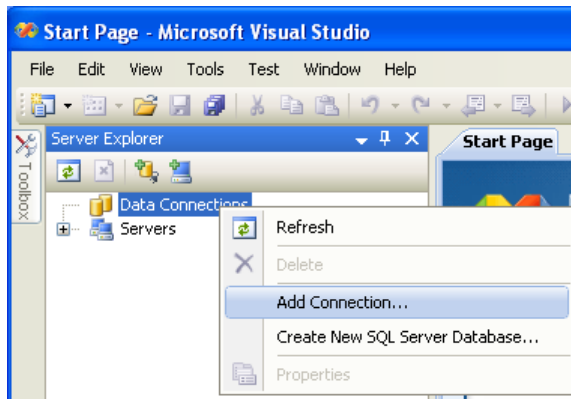
4.1. Making a Connection

Once the connector is installed, you can use it to create, modify, and delete connections to MySQL databases. To create a connection with a MySQL database, perform the following steps:

- Start Visual Studio, and open the Server Explorer window (**View, Server Explorer** option in the main Visual Studio menu, or **Control+W, L** keyboard shortcuts).

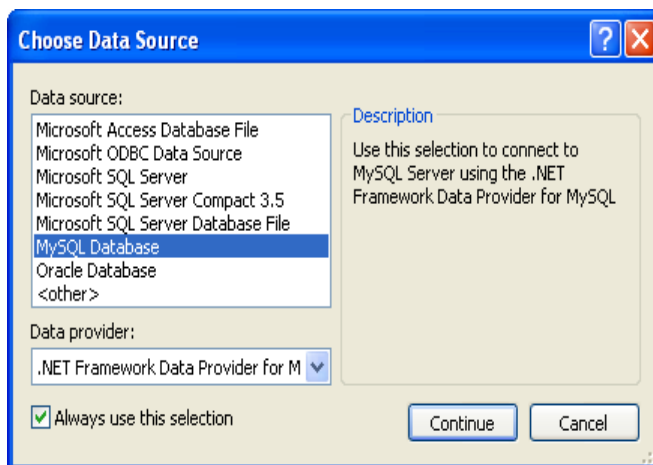
- Right-click the Data Connections node, and choose the **Add Connection...** menu item.
- Add Connection dialog opens. Press the **Change** button to choose MySQL Database as a data source.

Figure 4.1. Add Connection Context Menu

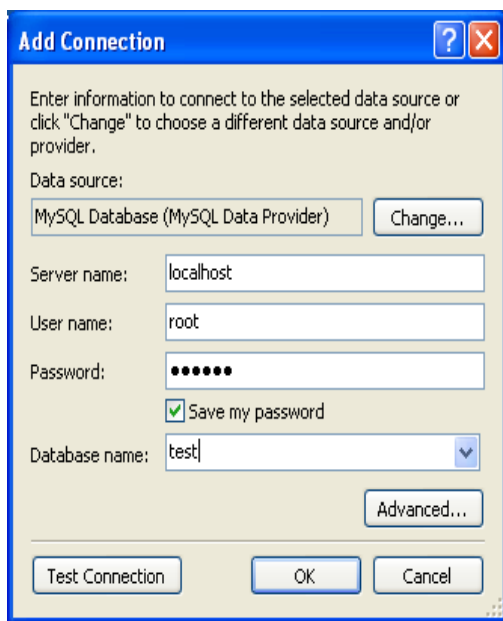


- **Change Data Source** dialog opens. Choose **MySQL Database** in the list of data sources (or the **<other>** option, if MySQL Database is absent), and then choose **.NET Framework Data Provider for MySQL** in the combo box of data providers.

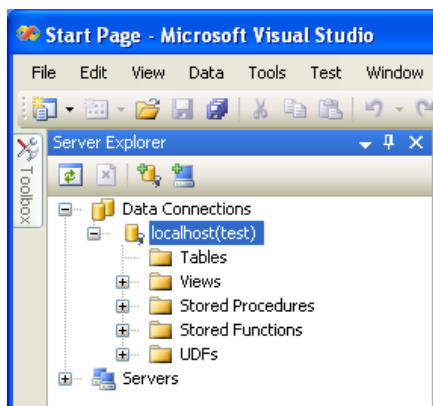
Figure 4.2. Choose Data Source



- Input the connection settings: the server host name (for example, localhost if the MySQL server is installed on the local machine), the user name, the password, and the default schema name. Note that you must specify the default schema name to open the connection.

Figure 4.3. Add Connection Dialog

- You can also set the port to connect with the MySQL server by pressing the **Advanced** button. To test connection with the MySQL server, set the server host name, the user name, and the password, and press the **Test Connection** button. If the test succeeds, the success confirmation dialog opens.
- After you set all settings and test the connection, press **OK**. The newly created connection is displayed in Server Explorer. Now you can work with the MySQL server through standard Server Explorer GUI.

Figure 4.4. New Data Connection

After the connection is successfully established, all settings are saved for future use. When you start Visual Studio for the next time, open the connection node in Server Explorer to establish a connection to the MySQL server again.

To modify and delete a connection, use the Server Explorer context menu for the corresponding node. You can modify any of the settings by overwriting the existing values with new ones. Note that the connection may be modified or deleted only if no active editor for its objects is opened: otherwise, you may lose your data.

4.2. Using IntelliSense in the SQL Editor

IntelliSense support is available starting in Connector/Net 6.5. Once you have established a connection, for example, using the **Connect to MySQL** toolbar button, you can get autocompletion as

you type, or by pressing **Control+J**. Depending on the context, the autocomplete dialog can show the list of available tables, table columns, or stored procedures (with the routine's signature as a tooltip). Typing some characters before pressing **Control+J** filters the choices to those items starting with that prefix.

4.3. Editing Tables

Connector/Net contains a table editor, which enables the visual creation and modification of tables.

The Table Designer can be accessed through a mouse action on table-type node of Server Explorer. To create a new table, right-click the **Tables** node (under the connection node) and choose the Create Table command from the context menu.

To modify an existing table, double-click the node of the table to modify, or right-click this node and choose the Design item from the context menu. Either of the commands opens the Table Designer.

The table editor is implemented in the manner of the well-known Query Browser Table Editor, but with minor differences.

Figure 4.5. Editing New Table

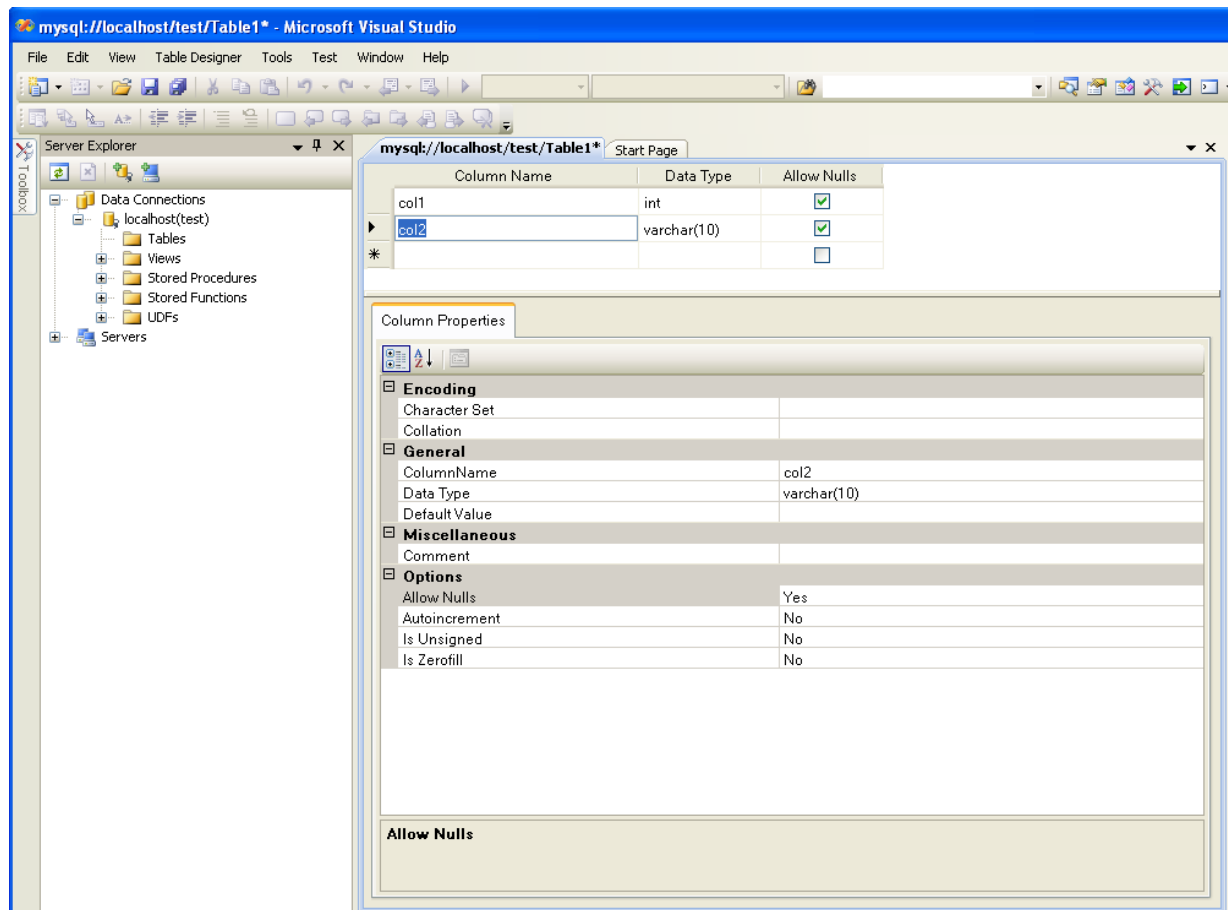


Table Designer consists of the following parts:

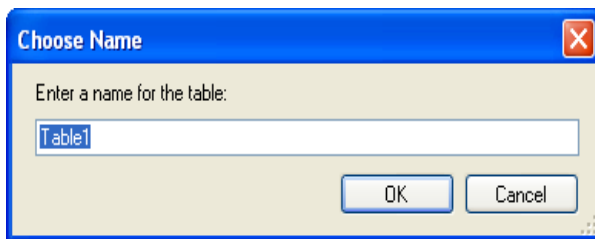
- Columns Editor - a data grid on top of the Table Designer. Use the Columns grid for column creation, modification, and deletion.
- Indexes tab - a tab on bottom of the Table Designer. Use the Indexes tab for indexes management.
- Foreign Keys tab - a tab on bottom of the Table Designer. Use the Foreign Keys tab for foreign keys management.

- Column Details tab - a tab on bottom of the Table Designer. Use the Column Details tab to set advanced column options.
- Properties window - a standard Visual Studio Properties window, where the properties of the edited table are displayed. Use the Properties window to set the table properties.

Each of these areas is discussed in more detail in subsequent sections.

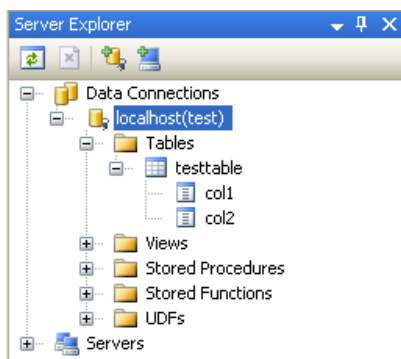
To save changes you have made in the Table Designer, use either **Save** or **Save All** button of the Visual Studio main toolbar, or press **Control+S**. If you have not already named the table, you will be prompted to do so.

Figure 4.6. Choose Table Name



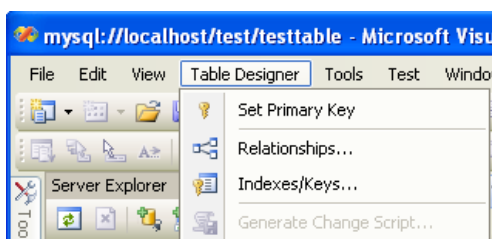
Once the table is created, you can view it in the Server Explorer.

Figure 4.7. Newly Created Table



The Table Designer main menu lets you set a [primary key](#) column, edit relationships such as [foreign keys](#), and create [indexes](#).

Figure 4.8. Table Designer Main Menu



4.3.1. Column Editor

You can use the Column Editor to set or change the name, data type, default value, and other properties of a table column. To set the focus to a needed cell of a grid, use the mouse click. Also you can move through the grid using **Tab** and **Shift+Tab** keys.

To set or change the name, data type, default value and comment of a column, activate the appropriate cell and type the desired value.

To set or unset flag-type column properties ([NOT NULL](#), auto incremented, flags), check or uncheck the corresponding check boxes. Note that the set of column flags depends on its data type.

To reorder columns, index columns or foreign key columns in the Column Editor, select the whole column to reorder by clicking the selector column on the left of the column grid. Then move the column by using **Control+Up** (to move the column up) or **Control+Down** (to move the column down) keys.

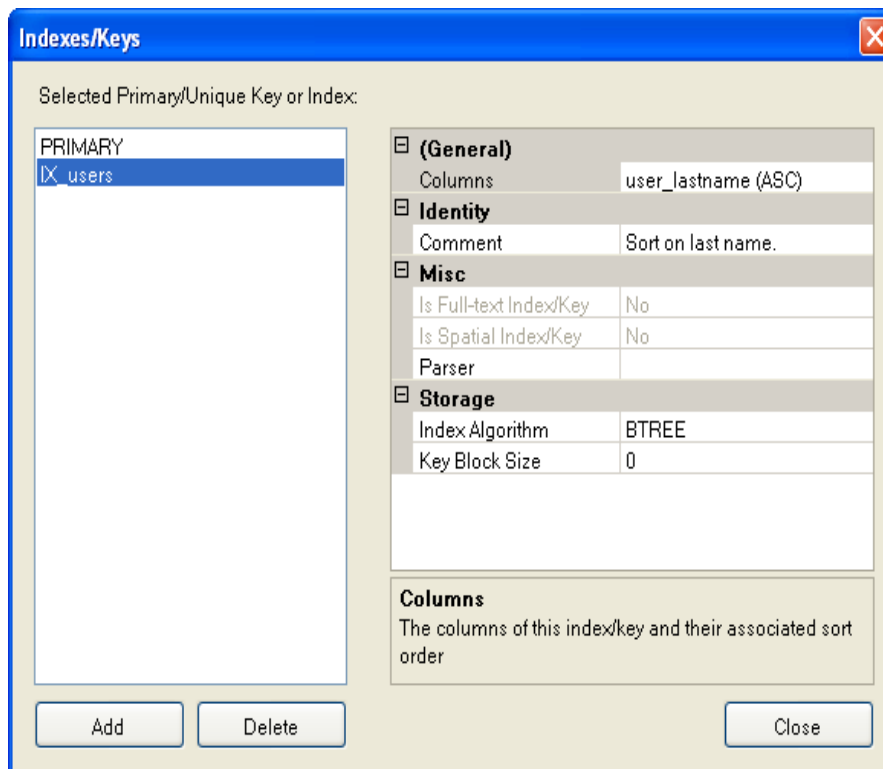
To delete a column, select it by clicking the selector column on the left of the column grid, then press the **Delete** button on a keyboard.

4.3.2. Editing Indexes

Indexes management is performed using the **Indexes/Keys** dialog.

To add an index, select **Table Designer**, **Indexes/Keys...** from the main menu, and click **Add** to add a new index. You can then set the index name, index kind, index type, and a set of index columns.

Figure 4.9. Indexes Dialog



To remove an index, select it in the list box on the left, and click the **Delete** button.

To change index settings, select the needed index in the list box on the left. The detailed information about the index is displayed in the panel on the right hand side. Change the desired values.

4.3.3. Editing Foreign Keys

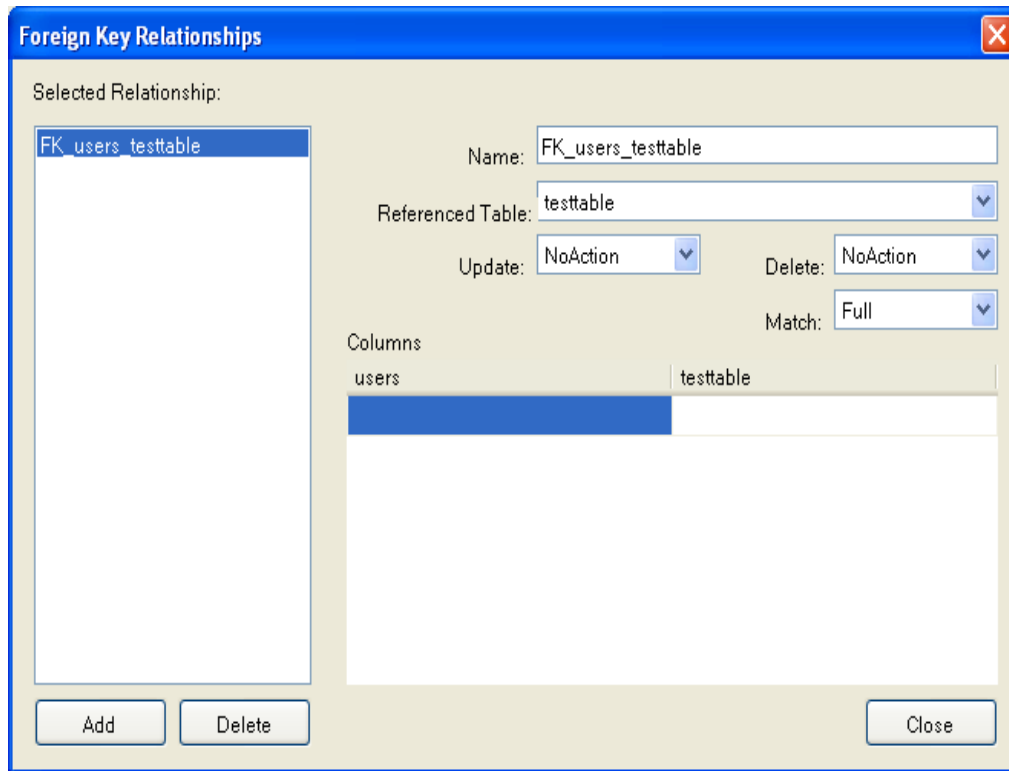
You manage [foreign keys](#) for [InnoDB](#) tables using the **Foreign Key Relationships** dialog.

To add a foreign key, select **Table Designer**, **Relationships...** from the main menu. This displays the **Foreign Key Relationship** dialog. Click **Add**. You can then set the foreign key name, referenced table name, foreign key columns, and actions upon update and delete.

To remove a foreign key, select it in the list box on the left, and click the **Delete** button.

To change foreign key settings, select the required foreign key in the list box on the left. The detailed information about the foreign key is displayed in the right hand panel. Change the desired values.

Figure 4.10. Foreign Key Relationships Dialog



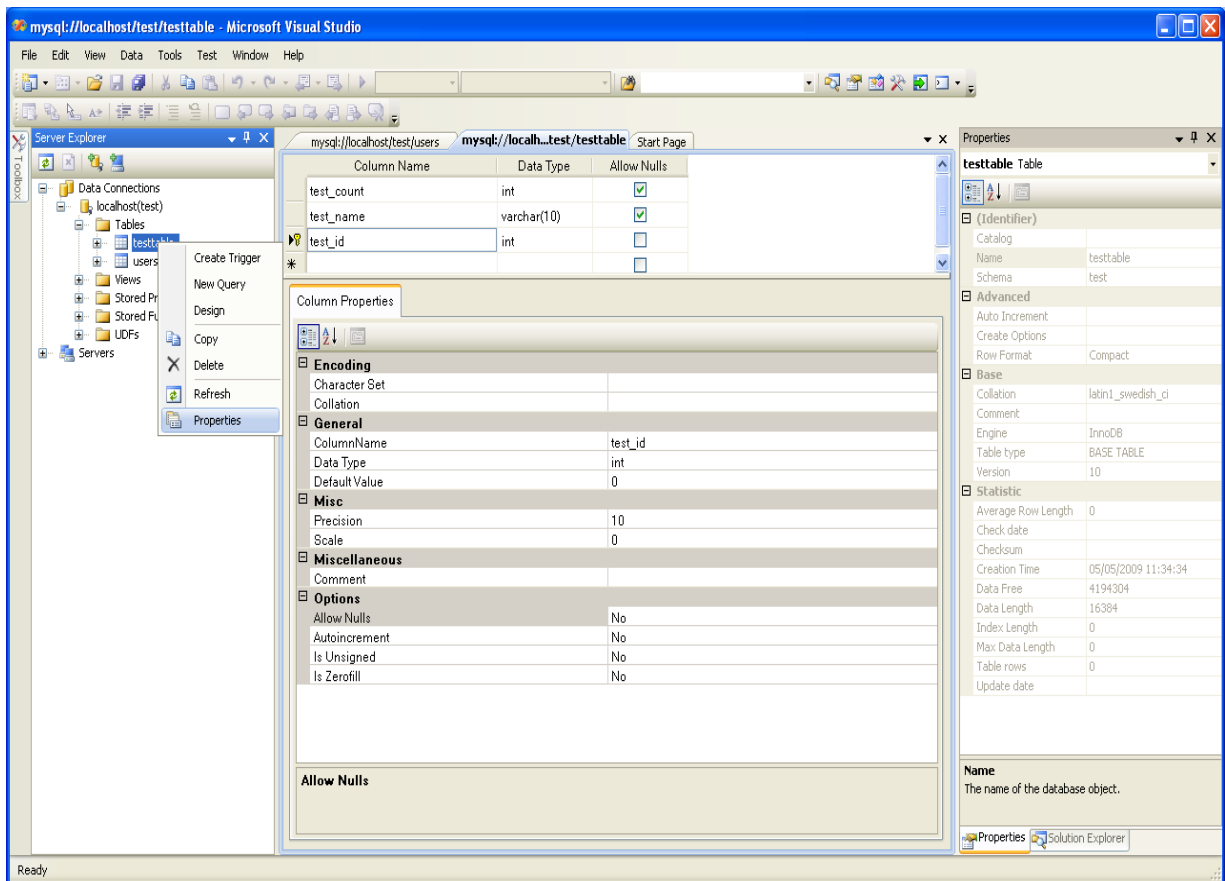
4.3.4. Column Properties

The **Column Properties** tab can be used to set column options. In addition to the general column properties presented in the Column Editor, in the **Column Properties** tab you can set additional properties such as Character Set, Collation and Precision.

4.3.5. Table Properties

To bring up Table Properties select the table and right-click to activate the context menu. Select Properties. The **Table Properties** dockable window will be displayed.

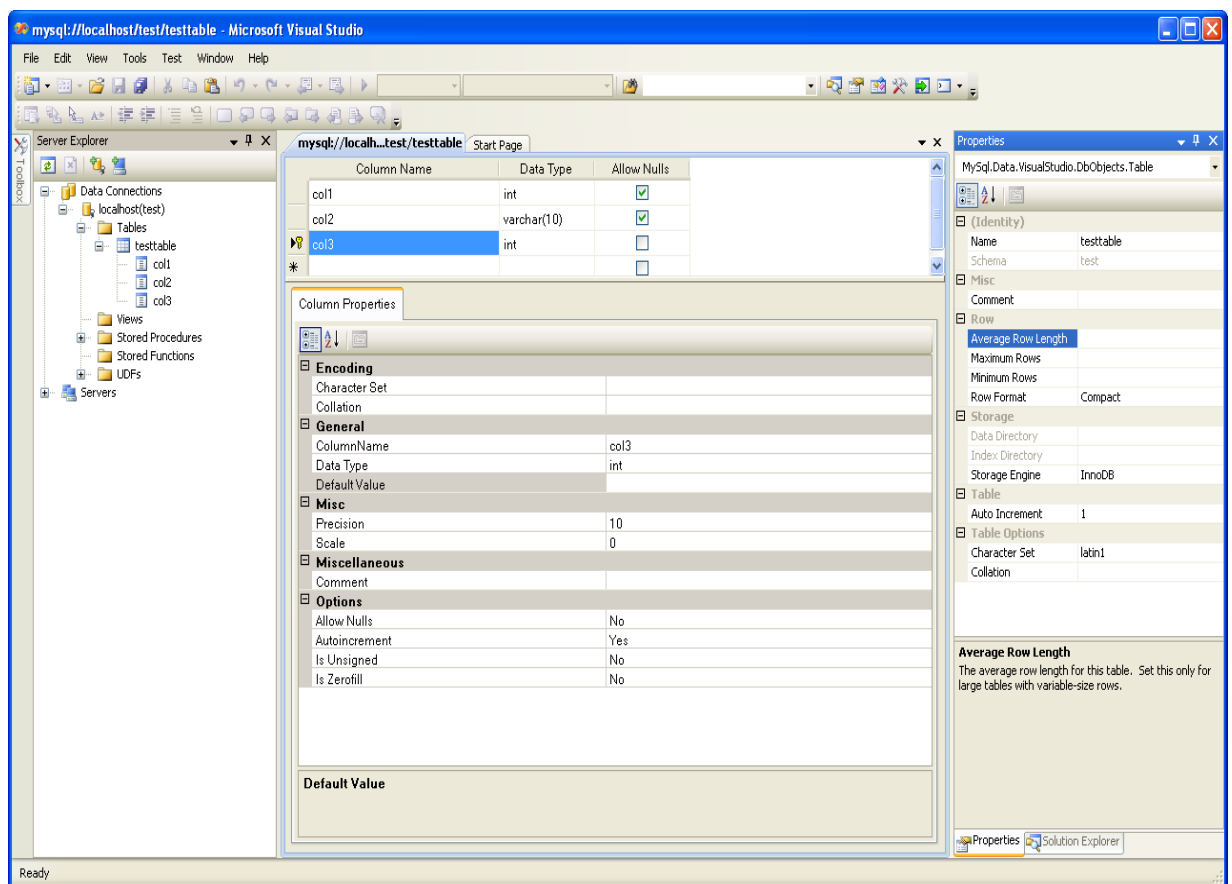
Figure 4.11. Table Properties Menu Item



The following table properties can be set:

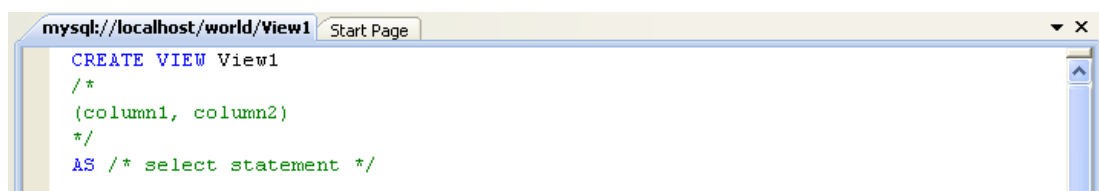
- Auto Increment.
- Average Row Length.
- Character Set.
- Collation.
- Comment.
- Data Directory.
- Index Directory.
- Maximum Rows.
- Minimum Rows.
- Name.
- Row Format.
- Schema.
- [Storage Engine](#). Note that in MySQL 5.5 and higher, the default storage engine for new tables is [InnoDB](#). See [InnoDB as the Default MySQL Storage Engine](#) for more information about the choice of storage engine, and considerations if you convert existing tables to [InnoDB](#).

The property [Schema](#) is read-only.

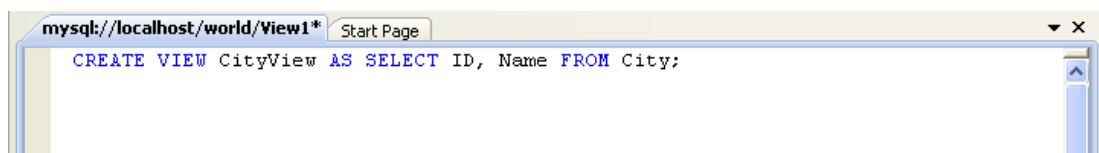
Figure 4.12. Table Properties

4.4. Editing Views

To create a new view, right-click the Views node under the connection node in Server Explorer. From the node's context menu, choose the Create View command. This command opens the SQL Editor.

Figure 4.13. Editing View SQL

You can then enter the SQL for your view.

Figure 4.14. View SQL Added

To modify an existing view, double-click a node of the view to modify, or right-click this node and choose the Alter View command from a context menu. Either of the commands opens the SQL Editor.

All other view properties can be set in the Properties window. These properties are:

- Catalog

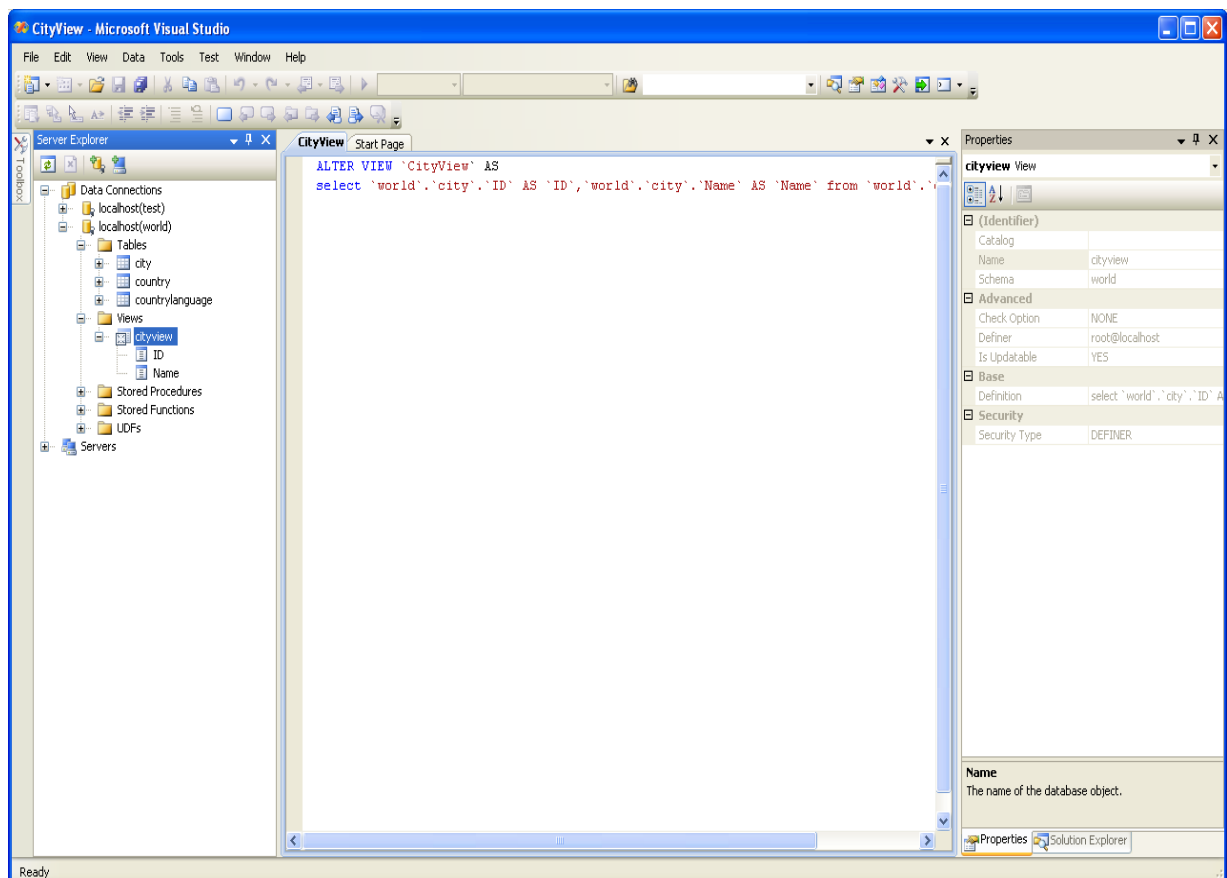
- Check Option
- Definer
- Definition
- Definer
- Is Updatable
- Name
- Schema
- Security Type

Some of these properties can have arbitrary text values, others accept values from a predefined set. In the latter case, set the desired value with an embedded combobox.

The properties `Is Updatable` and `Schema` are readonly.

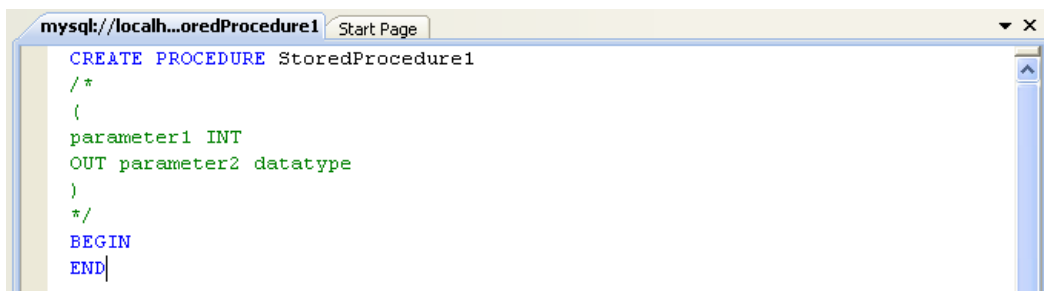
To save changes you have made, use either `Save` or `Save All` buttons of the Visual Studio main toolbar, or press **Control+S**.

Figure 4.15. View SQL Saved



4.5. Editing Stored Procedures and Functions

To create a new stored procedure, right-click the **Stored Procedures** node under the connection node in Server Explorer. From the node's context menu, choose the **Create Routine** command. This command opens the SQL Editor.

Figure 4.16. Edit Stored Procedure SQL

To create a new stored function, right-click the **Functions** node under the connection node in Server Explorer. From the node's context menu, choose the **Create Routine** command.

To modify an existing stored routine (procedure or function), double-click the node of the routine to modify, or right-click this node and choose the **Alter Routine** command from the context menu. Either of the commands opens the SQL Editor.

To create or alter the routine definition using SQL Editor, type this definition in the SQL Editor using standard SQL. All other routine properties can be set in the Properties window. These properties are:

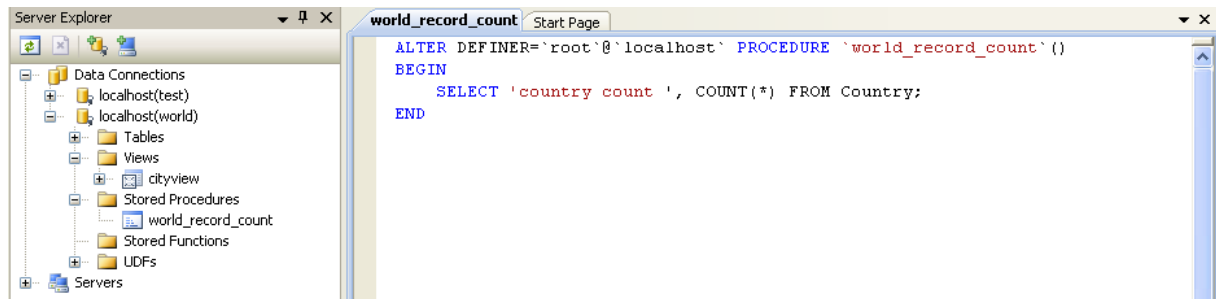
- Body
- Catalog
- Comment
- Creation Time
- Data Access
- Definer
- Definition
- External Name
- External Language
- Is Deterministic
- Last Modified
- Name
- Parameter Style
- Returns
- Schema
- Security Type
- Specific Name
- SQL Mode
- SQL Path
- Type

Some of these properties can have arbitrary text values, others accept values from a predefined set. In the latter case, set the desired value using the embedded combo box.

You can also set all the options directly in the SQL Editor, using the standard `CREATE PROCEDURE` or `CREATE FUNCTION` statement. However, it is recommended to use the Properties window instead.

To save changes you have made, use either **Save** or **Save All** buttons of the Visual Studio main toolbar, or press **Control+S**.

Figure 4.17. Stored Procedure SQL Saved



To observe the runtime behavior of a stored routine and debug any problems, use the Stored Procedure Debugger (available in Connector/Net 6.6 and higher). See [Section 4.8, “Debugging Stored Procedures and Functions”](#) for details.

4.6. Editing Triggers

To create a new trigger, right-click the node of the table in which to add the trigger. From the node's context menu, choose the **Create Trigger** command. This command opens the SQL Editor.

To modify an existing trigger, double-click the node of the trigger to modify, or right-click this node and choose the **Alter Trigger** command from the context menu. Either of the commands opens the SQL Editor.

To create or alter the trigger definition using SQL Editor, type the trigger statement in the SQL Editor using standard SQL.

Note

Enter only the trigger statement, that is, the part of the `CREATE TRIGGER` query that is placed after the `FOR EACH ROW` clause.

All other trigger properties are set in the Properties window. These properties are:

- Definer
- Event Manipulation
- Name
- Timing

Some of these properties can have arbitrary text values, others accept values from a predefined set. In the latter case, set the desired value using the embedded combo box.

The properties `Event Table`, `Schema`, and `Server` in the Properties window are read-only.

To save changes you have made, use either **Save** or **Save All** buttons of the Visual Studio main toolbar, or press **Control+S**. Before changes are saved, you will be asked to confirm the execution of the corresponding SQL query in a confirmation dialog.

To observe the runtime behavior of a stored routine and debug any problems, use the Stored Procedure Debugger (available in Connector/Net 6.6 and higher). See [Section 4.8, “Debugging Stored Procedures and Functions”](#) for details.

4.7. Editing User Defined Functions (UDF)

To create a new User Defined Function (UDF), right-click the **UDFs** node under the connection node in Server Explorer. From the node's context menu, choose the **Create UDF** command. This command opens the UDF Editor.

To modify an existing UDF, double-click the node of the UDF to modify, or right-click this node and choose the **Alter UDF** command from the context menu. Either of these commands opens the UDF Editor.

The UDF editor enables you to set the following properties:

- Name
- So-name (DLL name)
- Return type
- Is Aggregate

There are text fields for both names, a combo box for the return type, and a check box to indicate if the UDF is aggregate. All these options are also accessible using the Properties window.

The property **Server** in the Properties window is read-only.

To save changes you have made, use either **Save** or **Save All** buttons of the Visual Studio main toolbar, or press **Control+S**. Before changes are saved, you will be asked to confirm the execution of the corresponding SQL query in a confirmation dialog.

To observe the runtime behavior of a stored routine and debug any problems, use the Stored Procedure Debugger (available in Connector/Net 6.6 and higher). See [Section 4.8, "Debugging Stored Procedures and Functions"](#) for details.

4.8. Debugging Stored Procedures and Functions

The stored procedure debugger, new in Connector/Net 6.6, provides facilities for setting breakpoints, stepping into individual statements (Step Into, Step Out, Step Over), evaluating and changing local variable values, evaluating breakpoints, and other typical debugging tasks.

Installing the Debugger

To enable the stored procedure debugger, install Connector/Net 6.6 or higher and choose the **Complete** option.

Privileges

The debugger recreates at the start of each debug session a **serversidedebugger** database in your server. This database helps to track the instrumented code and implement observability logic in the debugged routine. Your current connection needs to have privileges to create that database, and its associated stored routines, functions, and tables.

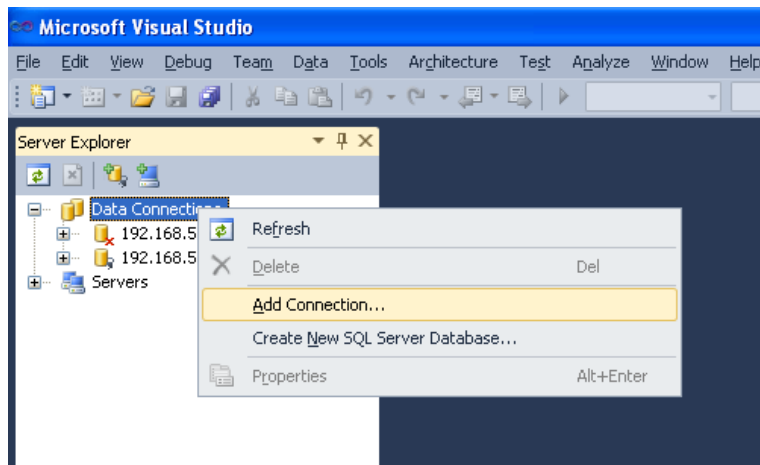
The debugger makes changes behind the scenes to temporarily add instrumentation code to the stored routines that you debug. You must have the **ALTER ROUTINE** privilege for each stored procedure, function, or trigger that you debug. (Including procedures and functions that are called, and triggers that are fired, by a procedure that you are debugging.)

Starting the Debugger

To start the debugger, follow these steps:

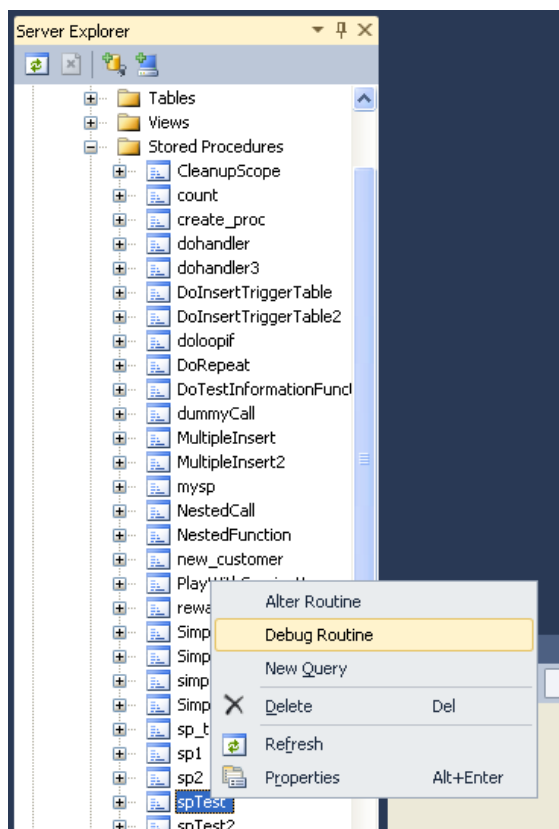
- Choose a connection in the Visual Studio Server Explorer.

Figure 4.18. Connection Dialog



- Expand the **Stored Procedures** folder. Only stored procedures can be debugged directly. To debug a user-defined function, create a stored procedure that calls the function.
- Click on a stored procedure node, then right-click and from the context menu choose **Debug Routine**.

Figure 4.19. Choose a Stored Routine to Debug



At this point, Visual Studio switches to debug mode, opening the source code of the routine being debugged in step mode, positioned on the first statement.

If the initial routine you debug has one or more arguments, a popup will show up with a grid (a row per each argument and three columns: one for the argument, one for the argument value (this is editable)

and one for nullifying that argument value (a checkbox)). After setting up all the argument values, you can press **OK** to start the debug session, or **Cancel** to cancel the debug session.

Figure 4.20. Setting Arguments (1 of 2)

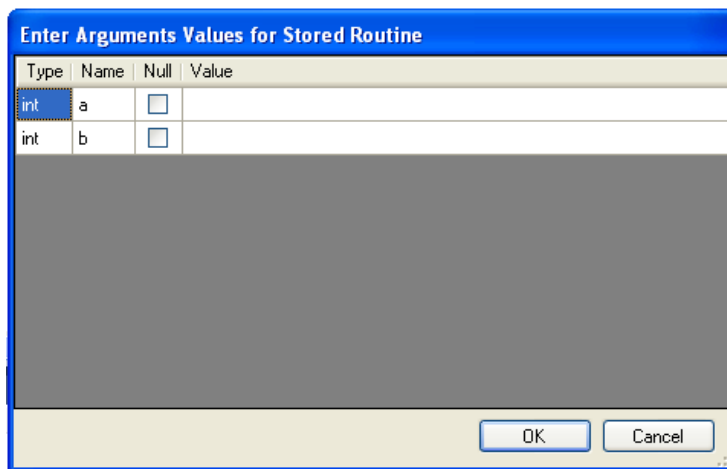
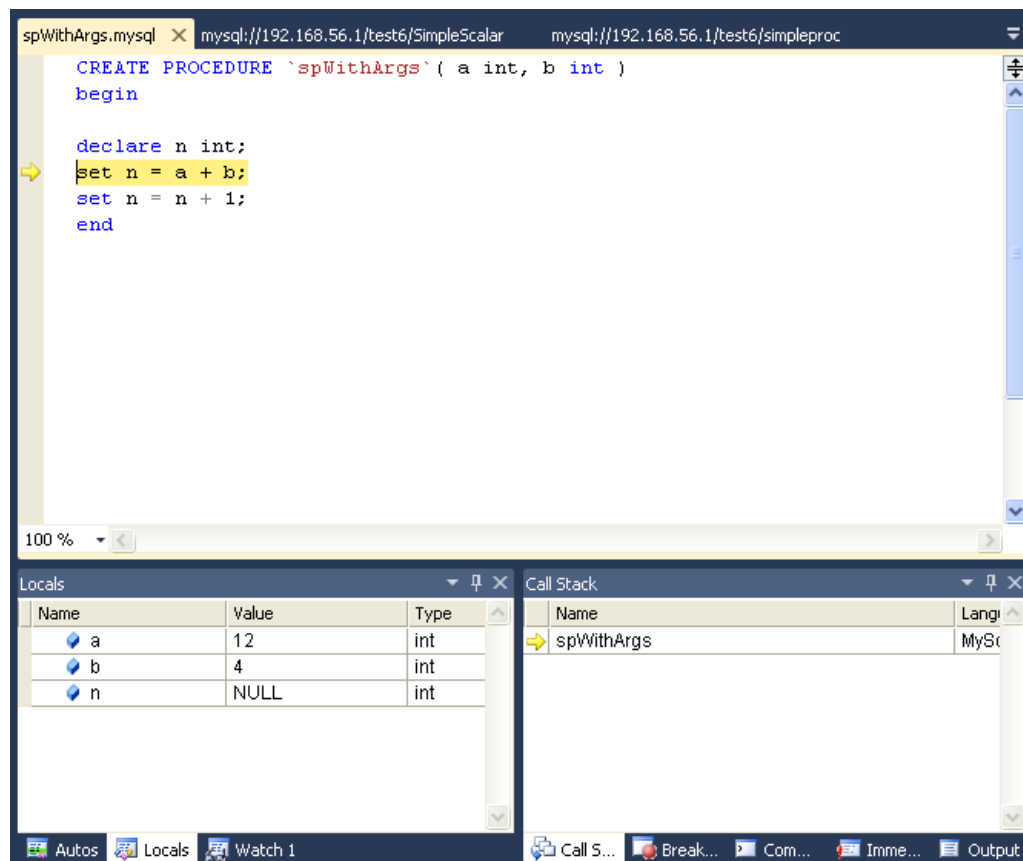


Figure 4.21. Setting Arguments (2 of 2)



How the Debugger Works

To have visibility into the internal workings of a stored routine, the debugger prepares a special version of the procedure, function, or trigger being debugged, instrumented with extra code to keep track of the current line being stepped into and the values of all the local variables. Any other stored procedures, functions, or triggers called from the routine being debugged are instrumented the same way. The debug versions of the routines are prepared for you automatically, and when the debug session ends (by either pressing **F5** or **Shift+F5**), the original versions of the routines are automatically restored.

A copy of the original version of each instrumented routine (the version without instrumentation) is stored in the `AppData\Roaming\MySQLDebuggerCache` folder for the current Windows user (the path returned by calling `System.Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData)` in .NET, plus appending `MySQLDebuggerCache`. There is one file for each instrumented routine, named `routine_name.mysql`. For example, in Windows 7, for a user named `fergs`, the path is `C:\Users\fergs\AppData\Roaming\MySQLDebuggerCache`.

Two threads are used, one for the debugger and one for the routine being debugged. The threads run in strict alternation, switching between the debugger and the routine as each statement is executed in the stored routine.

Basic Debugging Operations

The debugger has the same look and feel as the standard Visual Studio debuggers for C#, VB.NET or C++. In particular, the following are true:

Locals and Watches

- To show the Locals tab, choose the menu item **Debug -> Windows -> Locals**.

The **Locals** tab lists all the variables available in the current scope: variables defined with `DECLARE` at any point in the routine, argument parameters, and session variables that are referenced.

- If the last step operation changes the value of a local, its value will be highlighted in red (until another statement is executed or stepped).
- You can change the value of any local.
- To show the **Watch** tab, choose the menu item **Debug -> Windows -> Watch**.

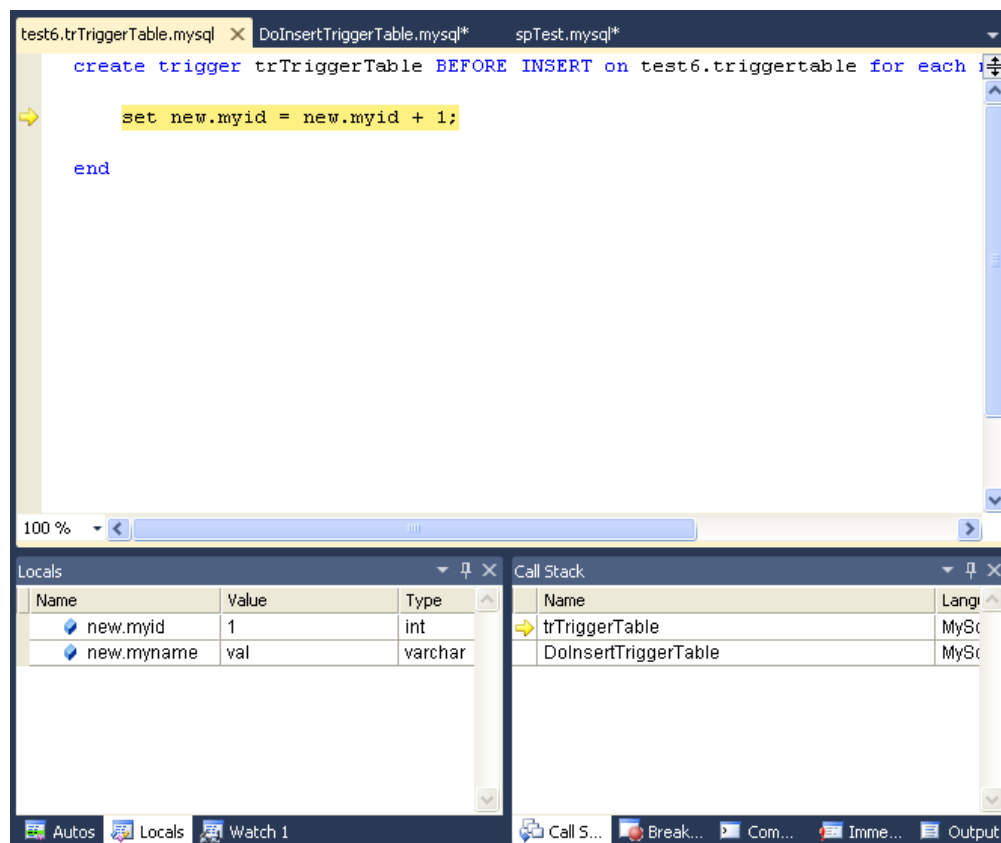
To define a watch, type any valid MySQL expression, optionally including function calls. If the watch evaluation makes sense in the current context (current stack frame), it will show its value, otherwise it will show an error message in the same row the watch was defined.

- When debugging a trigger, in addition to any locals declared or session variables referenced, the new and old object (when applicable) will be listed. For example in a trigger for `INSERT`, for a table defined like:

```
create table t1( id int, myname varchar( 50 ));
```

the locals will list the extra variables `new.id` and `new.myname`. For an `UPDATE` trigger, you will also get the extra variables `old.id` and `old.myname`. These variables from the new and old objects can be manipulated the same way as any ordinary local variable.

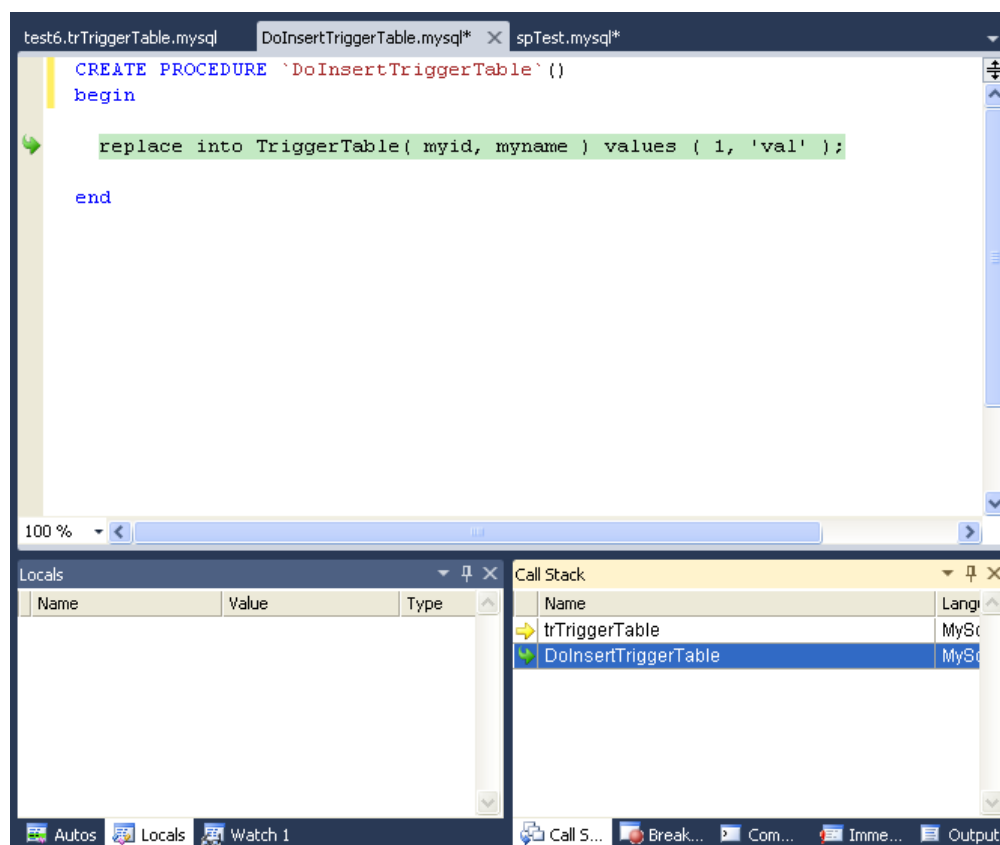
Figure 4.22. Debugging a Trigger



Call Stack

- To show the **Call Stack** tab, choose the menu item **Debug -> Windows -> Call Stack**.
- The stack trace (in the **Call Stack** tab) will list all the stack traces, one for each routine invocation. The one with a yellow mark is the current stepping point. Clicking in another will activate in the editor the tab for that routine source, highlighting in green the last statement stepped.

Figure 4.23. Call Stack



Stepping

- Stepping of a new routine starts in the first executable instruction (excluding declares, handlers, cursor declarations, and so on).

Figure 4.24. Debug Stepping



Figure 4.25. Function Stepping (1 of 2)

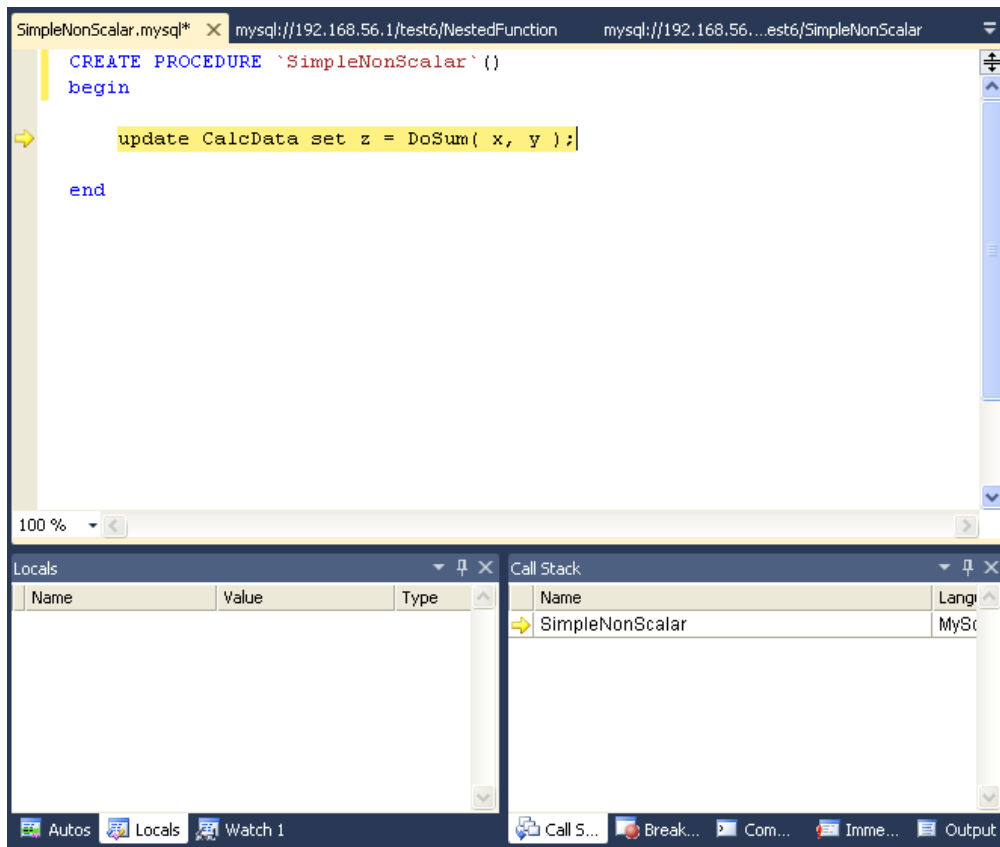
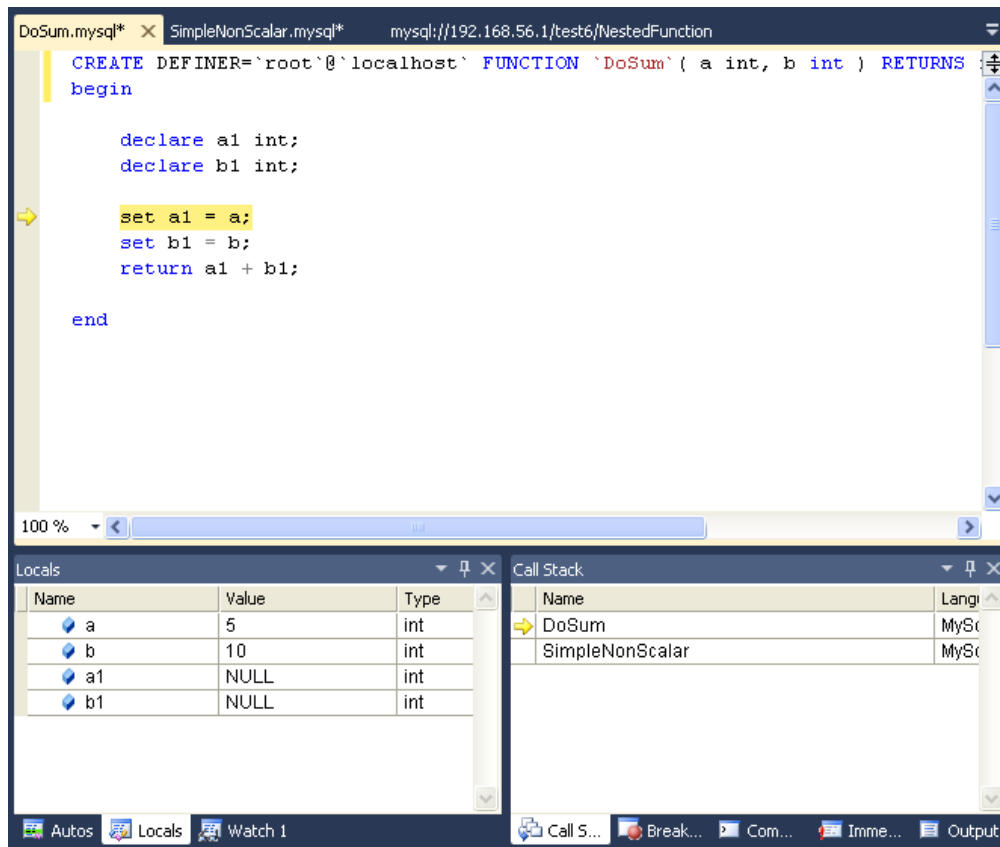


Figure 4.26. Function Stepping (2 of 2)



- To step into the code of a condition handler, the condition must be triggered in the rest of the MySQL routine.
- The next statement to be executed is highlighted in yellow.
- To continue stepping, you can choose between **Step Into** (by pressing **F11**), **Step Out** (by pressing **F10**) or **Step Over** (by pressing **Shift+F11**).
- You can step out of any of functions, triggers or stored procedures. If you step from the main routine, it will run that routine to completion and finish the debug session.
- You can step over stored procedure calls, stored functions, and triggers. (To step over a trigger, step over the statement that would cause the trigger to fire.)
- When stepping into a single statement, the debugger will step into each individual function invoked by that statement and each trigger fired by that statement. The order in which they are debugged is the same order in which the MySQL server executes them.
- You can step into triggers triggered from `INSERT`, `DELETE`, `UPDATE`, and `REPLACE` statements.
- Also, the number of times you enter into a stored function or trigger depends on how many rows are evaluated by the function or affected by the trigger. For example, if you press **F11 (Step Into)** into an `UPDATE` statement that modifies three rows (calling a function for a column in the `SET` clause, thus invoking the function for each of the three rows), you will step into that function three times in succession, once for each of the rows. You can accelerate this debug session by disabling any breakpoints defined in the given stored function and pressing **Shift+F11** to step out. In this example, the order in which the different instances of the stored function are debugged is server-specific: the same order used by the current MySQL server instance to evaluate the three function invocations.

Breakpoints

- To show the **Breakpoints** tab, choose the menu item **Debug -> Windows -> Breakpoints**.
- The **Breakpoints** tab will show all the breakpoints defined. From here, you can enable and disable breakpoints one by one or all at once (using the toolbar on top of the **Breakpoints** tab).
- You can define new breakpoints only in the middle of a debug session. Click in the left gray border of any MySQL editor, or click anywhere in a MySQL editor and press **F9**. In the familiar Visual Studio way, you press **F9** once to create a breakpoint in that line, and press it again to remove that breakpoint.
- Once a breakpoint is defined, it will appear enabled (as filled red circle left to the current row if that line is a valid statement to put a breakpoint) or disabled (as a non-filled red circle left to the current row if that row is not valid to put a breakpoint).
- To define conditional breakpoints, after creating the breakpoint, right click in the red dot and choose **Condition...**. There you can put any valid MySQL expression and state if the condition is **Is True** or **Has changed**. The former will trigger the breakpoint every time the condition is true, the latter every time the condition value has changed. (If you define a conditional breakpoint, it is not enough to step into the line with the breakpoint defined to trigger such a breakpoint.)

Figure 4.27. Conditional Breakpoints

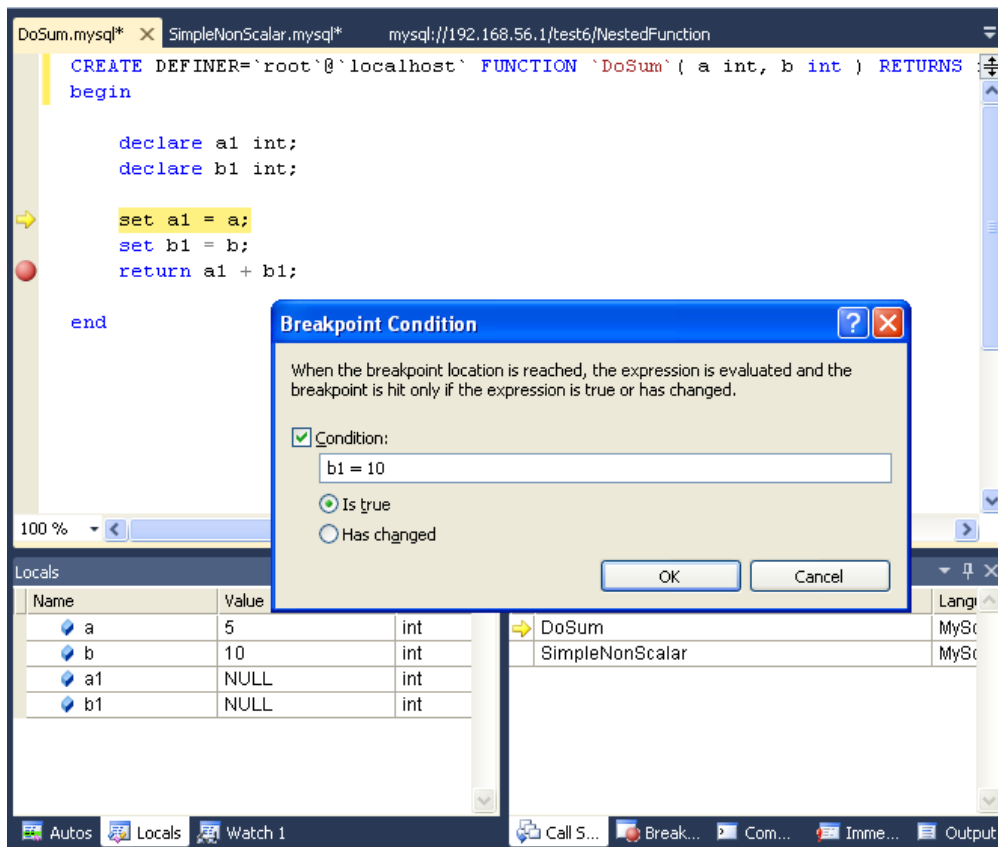
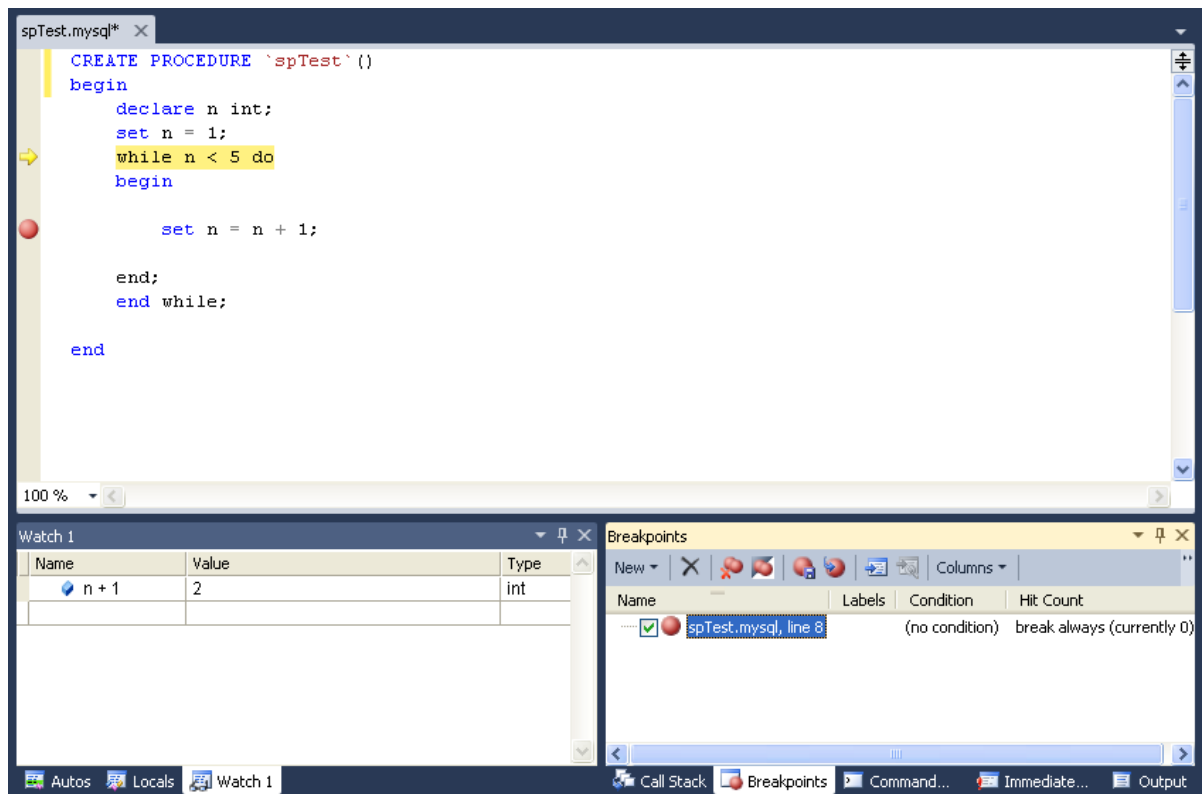


Figure 4.28. Expressions and Breakpoints



- To define pass count breakpoints, after creating the breakpoint, right click in the red dot and choose **Hit Count...**. In the popup dialog, define the specific condition to set. For example, **break when the hit count is equal to** and a value 3 will trigger the breakpoint the third time it is hit.

Other Features

- To abort the debug session (and the execution of the current call stack of routines), press Shift+F5.
- To run the routine to completion (or until next breakpoint hit), press F5.
- For all functionality you can use (in addition to the shortcuts documented), see the options in the **Debug** menu of Visual Studio.

Limitations

- Code being debugged must not use `get_lock` or `release_lock` MySQL functions, since they are used internally by the debugger infrastructure to synchronize the debugger and the debugged routine.
- Code being debugged must avoid using any transaction code (`START TRANSACTION`, `COMMIT`, `ROLLBACK`) due to the possibility of wiping out the contents of the debugger tables. (This limitation may be removed in the future).
- You cannot debug the routines in the `serversidedebugger` database.
- The MySQL server running the routine being debugged can be any version between 5.0 and 5.6, running on Windows, Linux, or any other supported platform.
- We recommend always running debug sessions on test and development servers, rather than against a MySQL production server, because debugging can cause temporary performance issues or even deadlocks. The instrumented versions of the routines being debugged use locks, that the rest of the production code may not be aware of.

Keyboard Shortcuts

The following list summarizes the keyboard shortcuts for debugging:

- **F9** Toggles breakpoints
- **F11**: Step into once
- **F10**: Step over once
- **Shift+F11**: Step out once
- **F5**: Run
- **Shift+F5**: Abort current debug session

4.9. Cloning Database Objects

Tables, views, stored procedures, and functions can be cloned using the appropriate Clone command from the context menu: Clone Table, Clone View, Clone Routine. The clone commands open the corresponding editor for a new object: the **Table Editor** for cloning a table, and the **SQL Editor** for cloning a view or a routine.

The editor is filled with values of the original object. You can modify these values in the usual manner.

To save the cloned object, use either **Save** or **Save All** buttons of the Visual Studio main toolbar, or press **Control+S**. Before changes are saved, you will be asked to confirm the execution of the corresponding SQL query in a confirmation dialog.

4.10. Dropping Database Objects

Tables, views, stored routines, triggers, and UDFs can be dropped with the appropriate Drop command selected from its context menu: Drop Table, Drop View, Drop Routine, Drop Trigger, Drop UDF.

You will be asked to confirm the execution of the corresponding drop query in a confirmation dialog.

You can only drop a single object at a time.

4.11. Using the ADO.NET Entity Framework

Connector/Net 6.0 introduced support for the ADO.NET Entity Framework. ADO.NET Entity Framework was included with .NET Framework 3.5 Service Pack 1, and Visual Studio 2008 Service Pack 1. ADO.NET Entity Framework was released on 11th August 2008.

ADO.NET Entity Framework provides an Object Relational Mapping (ORM) service, mapping the relational database schema to objects. The ADO.NET Entity Framework defines several layers, these can be summarized as:

- **Logical** - this layer defines the relational data and is defined by the Store Schema Definition Language (SSDL).
- **Conceptual** - this layer defines the .NET classes and is defined by the Conceptual Schema Definition Language (CSDL)
- **Mapping** - this layer defines the mapping from .NET classes to relational tables and associations, and is defined by Mapping Specification Language (MSL).

Connector/Net integrates with Visual Studio 2008 to provide a range of helpful tools to assist the developer.

A full treatment of ADO.NET Entity Framework is beyond the scope of this manual. If you are unfamiliar with ADO.NET, review the [Microsoft ADO.NET Entity Framework documentation](#).

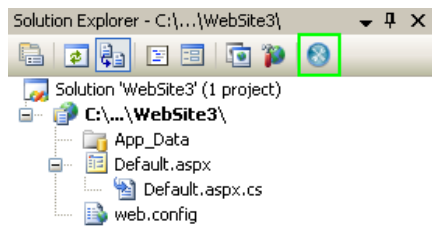
Tutorials on getting started with ADO.NET Entity Framework are available. See [Section 5.5, "Tutorial: Using an Entity Framework Entity as a Windows Forms Data Source"](#) and [Section 5.6, "Tutorial: Databinding in ASP.NET using LINQ on Entities"](#).

4.12. MySQL Website Configuration Tool

MySQL Connector/Net 6.1 introduced the MySQL Website Configuration Tool. This is a facility available in Visual Studio that enables you to configure the Membership, Role, Session State and Profile Provider, without editing configuration files. You set your configuration options within the tool, and the tool modifies your `web.config` file accordingly.

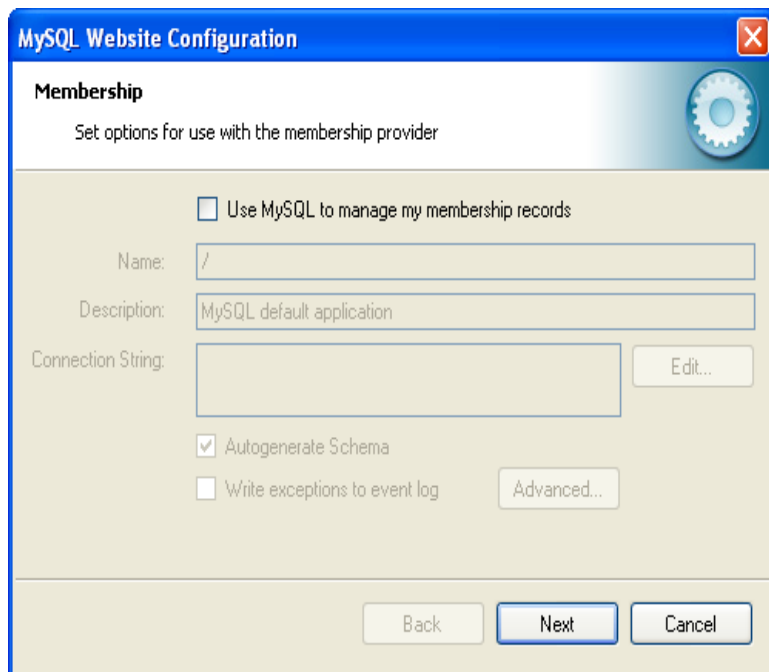
The MySQL Website Configuration Tool appears as a small icon on the Solution Explorer toolbar in Visual Studio, as show by the following screenshot:

Figure 4.29. MySQL Website Configuration Tool



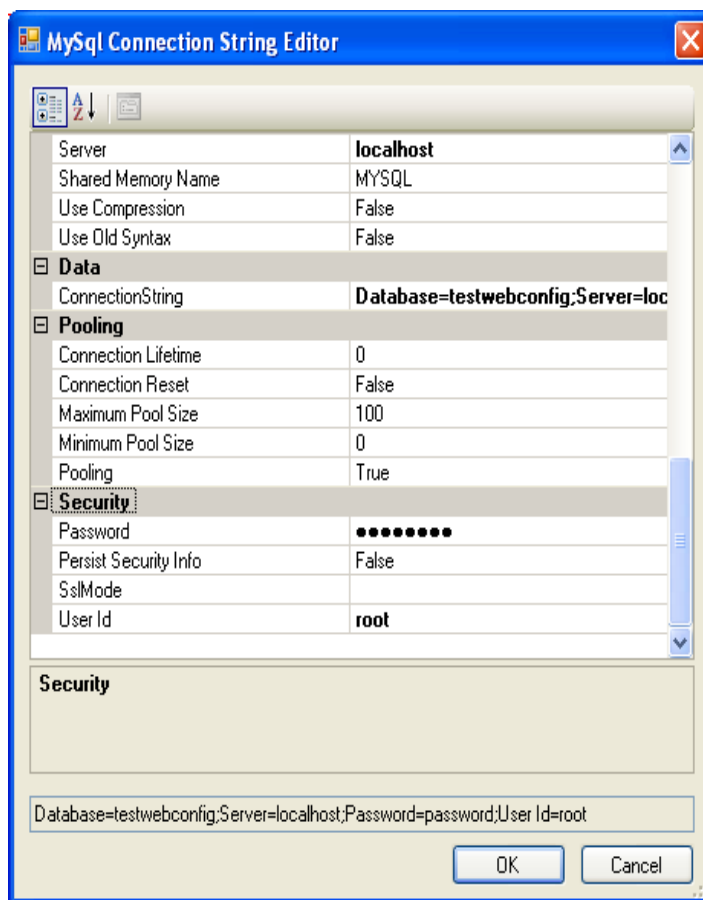
Clicking the Website Configuration Tool icon launches the wizard and displays the first screen:

Figure 4.30. MySQL Website Configuration Tool - Membership



This allows you to enable use of the MySQL Membership Provider. Click the check box to enable this. You can now enter the name of the application that you are creating the configuration for. You can also enter a description for the application.

You can then click the `Edit...` button to launch the Connection String Editor:

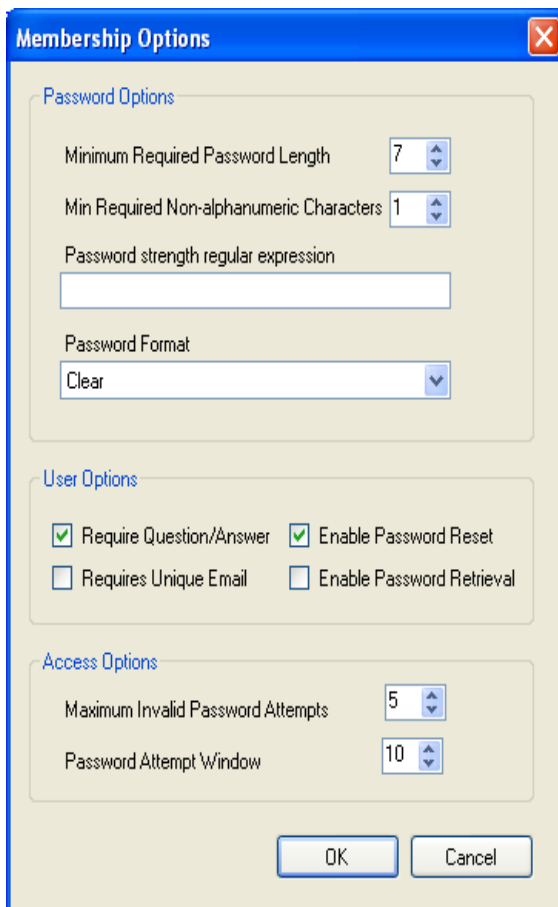
Figure 4.31. MySQL Website Configuration Tool - Connection String Editor

Note that if you have already defined a connection string for the providers manually in `web.config`, or previously using the tool, this will be automatically loaded and displayed, and can then be modified in this dialog.

You can also ensure that the necessary schemas are created automatically for you by selecting the Autogenerate Schema check box. These schemas are used to store membership information. The database used to storage is the one specified in the connection string.

You can also ensure that exceptions generated by the application will be written to the Windows event log by selecting the **Write exceptions to event log** check box.

Clicking the **Advanced...** button launches a dialog that enables you to set Membership Options. These options dictate such variables as password length required when a user signs up, whether the password is encrypted and whether the user can reset their password or not.

Figure 4.32. MySQL Website Configuration Tool - Advanced Options

The screenshot shows the 'Membership Options' dialog box. It has a blue title bar with a close button. The dialog is divided into three sections: 'Password Options', 'User Options', and 'Access Options'. In 'Password Options', 'Minimum Required Password Length' is set to 7, 'Min Required Non-alphanumeric Characters' is set to 1, 'Password strength regular expression' is empty, and 'Password Format' is set to 'Clear'. In 'User Options', 'Require Question/Answer' and 'Enable Password Reset' are checked, while 'Requires Unique Email' and 'Enable Password Retrieval' are unchecked. In 'Access Options', 'Maximum Invalid Password Attempts' is set to 5 and 'Password Attempt Window' is set to 10. At the bottom are 'OK' and 'Cancel' buttons.

Membership Options

Password Options

Minimum Required Password Length: 7

Min Required Non-alphanumeric Characters: 1

Password strength regular expression:

Password Format: Clear

User Options

☒ Require Question/Answer ☒ Enable Password Reset

☐ Requires Unique Email ☐ Enable Password Retrieval

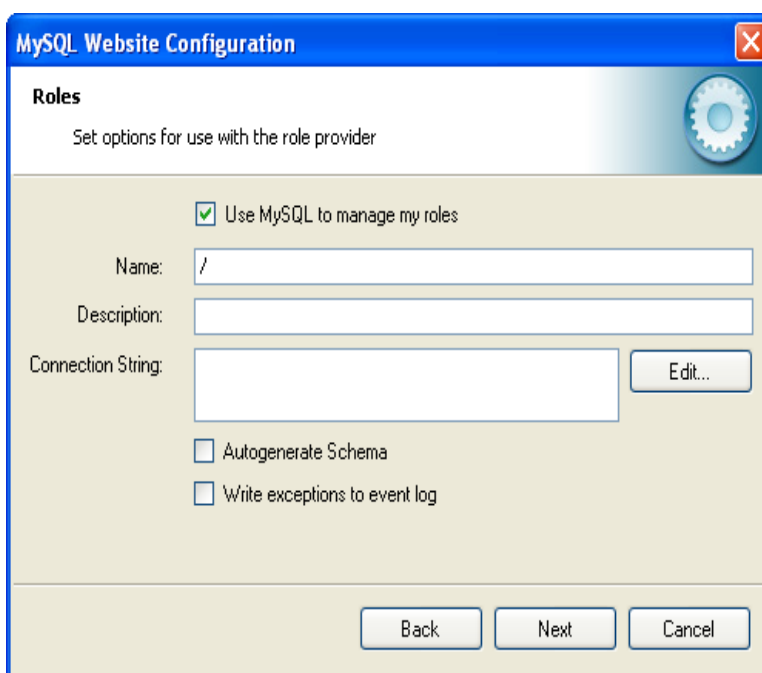
Access Options

Maximum Invalid Password Attempts: 5

Password Attempt Window: 10

OK Cancel

Once information has been set up as required for configuration of the Membership Provider, the **Next** button can be clicked to display the Roles Provider screen:

Figure 4.33. MySQL Website Configuration Tool - Roles

The screenshot shows the 'MySQL Website Configuration - Roles' dialog box. It has a blue title bar with a close button. The dialog is titled 'Roles' and has a subtitle 'Set options for use with the role provider'. There is a gear icon in the top right corner. The main area contains a checked checkbox 'Use MySQL to manage my roles'. Below this are three text boxes: 'Name:' with a slash '/', 'Description:', and 'Connection String:'. To the right of the 'Connection String' box is an 'Edit...' button. At the bottom are two unchecked checkboxes: 'Autogenerate Schema' and 'Write exceptions to event log'. At the very bottom are 'Back', 'Next', and 'Cancel' buttons.

MySQL Website Configuration

Roles

Set options for use with the role provider

☒ Use MySQL to manage my roles

Name: /

Description:

Connection String: Edit...

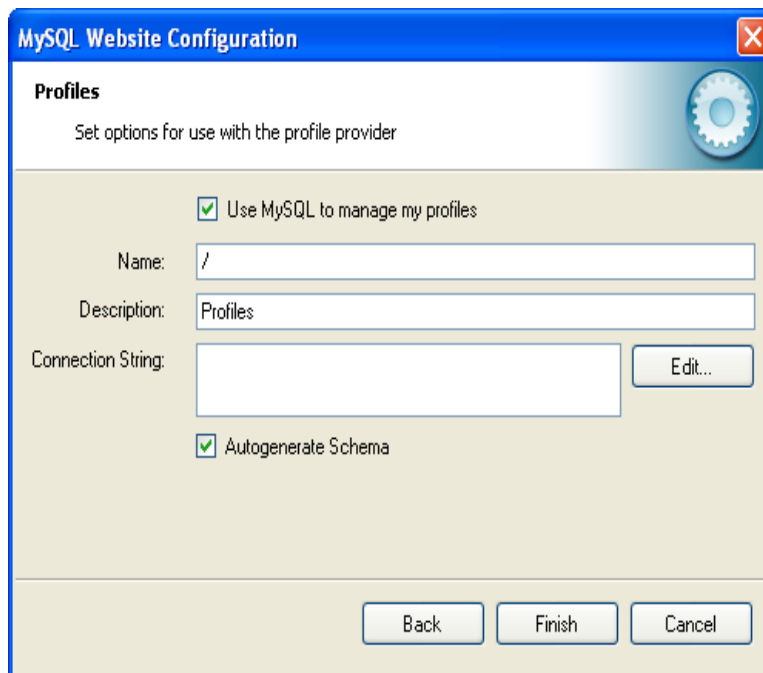
☐ Autogenerate Schema

☐ Write exceptions to event log

Back Next Cancel

Again the connection string can be edited, a description added and Autogenerate Schema can be enabled before clicking **Next** to go to the Profiles Provider screen:

Figure 4.34. MySQL Website Configuration Tool - Profiles

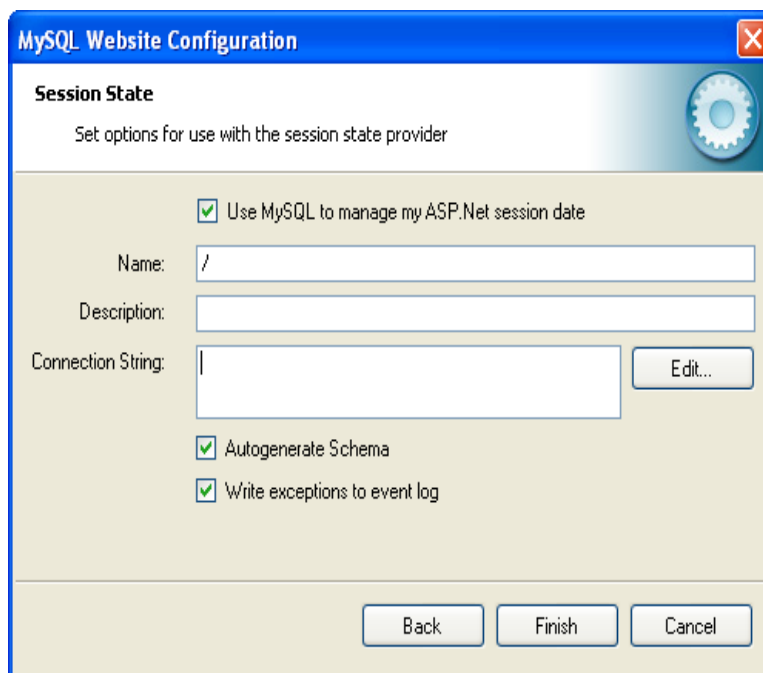


The screenshot shows the 'Profiles' configuration window. The title bar is 'MySQL Website Configuration'. The main heading is 'Profiles' with a subtitle 'Set options for use with the profile provider'. There is a gear icon in the top right. The main area contains a checkbox 'Use MySQL to manage my profiles' which is checked. Below it are three text boxes: 'Name:' with a slash '/', 'Description:' with the word 'Profiles', and 'Connection String:' which is empty. To the right of the 'Connection String' box is an 'Edit...' button. Below these is another checkbox 'Autogenerate Schema' which is also checked. At the bottom are three buttons: 'Back', 'Finish', and 'Cancel'.

This screen display similar options to the previous screens.

Click **Next** to proceed to the Session State configuration page:

Figure 4.35. MySQL Website Configuration Tool - Session State



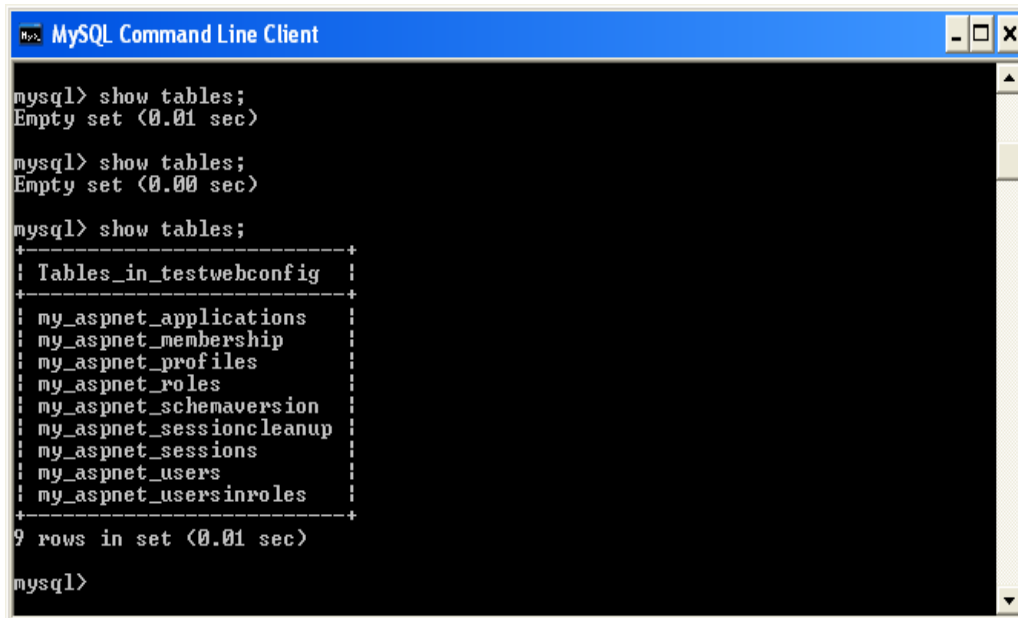
The screenshot shows the 'Session State' configuration window. The title bar is 'MySQL Website Configuration'. The main heading is 'Session State' with a subtitle 'Set options for use with the session state provider'. There is a gear icon in the top right. The main area contains a checkbox 'Use MySQL to manage my ASP.Net session date' which is checked. Below it are three text boxes: 'Name:' with a slash '/', 'Description:' which is empty, and 'Connection String:' which is empty. To the right of the 'Connection String' box is an 'Edit...' button. Below these are two checkboxes: 'Autogenerate Schema' and 'Write exceptions to event log', both of which are checked. At the bottom are three buttons: 'Back', 'Finish', and 'Cancel'.

Once you have set up the Session State Provider as required, click **Finish** to exit the wizard.

At this point, select the **Authentication Type** to **From Internet**. Launch the ASP.NET Configuration Tool and select the Security tab. Click the **Select authentication type** link and ensure that the

From the internet radio button is selected. You can now examine the database you created to store membership information. All the necessary tables will have been created for you:

Figure 4.36. MySQL Website Configuration Tool - Tables



```
mysql> show tables;
Empty set (0.01 sec)

mysql> show tables;
Empty set (0.00 sec)

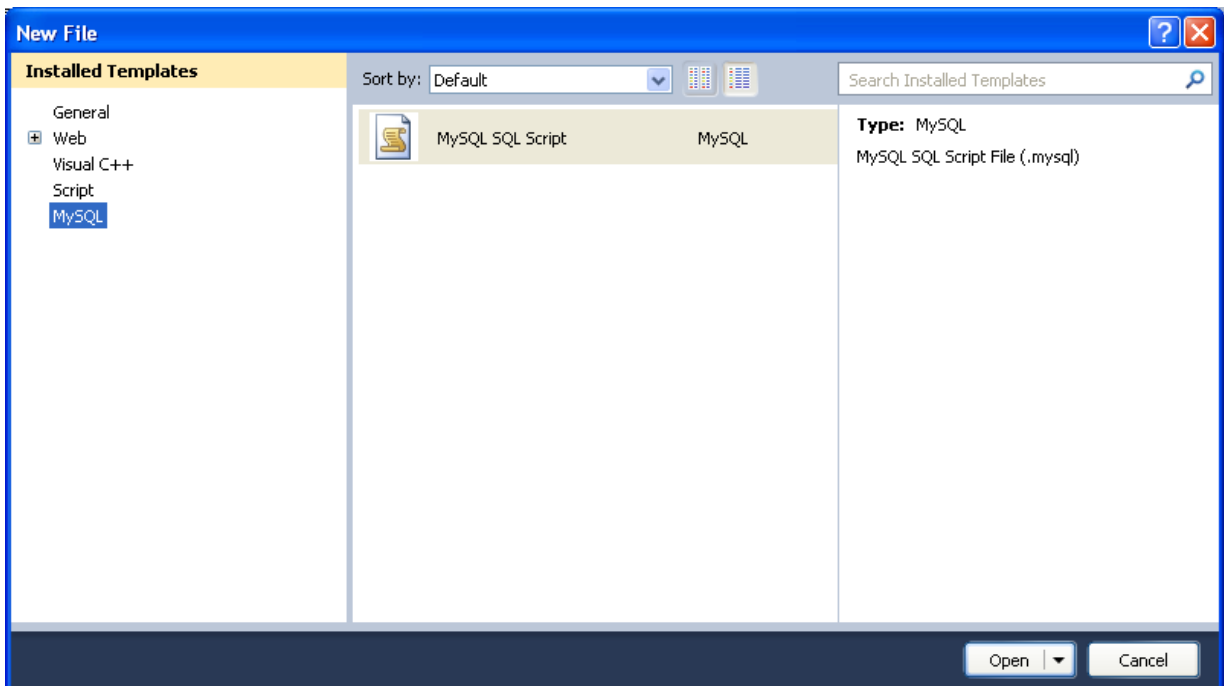
mysql> show tables;
+-----+
| Tables_in_testwebconfig |
+-----+
| my_aspnet_applications |
| my_aspnet_membership   |
| my_aspnet_profiles     |
| my_aspnet_roles        |
| my_aspnet_schemaversion |
| my_aspnet_sessioncleanup |
| my_aspnet_sessions     |
| my_aspnet_users        |
| my_aspnet_usersinroles |
+-----+
9 rows in set (0.01 sec)

mysql>
```

4.13. MySQL SQL Editor

MySQL Connector/Net 6.3.2 introduced a new MySQL SQL Editor. The easiest way to invoke the editor is by selecting the **New**, **File** menu item from the Visual Studio main menu. This displays the **New File** dialog:

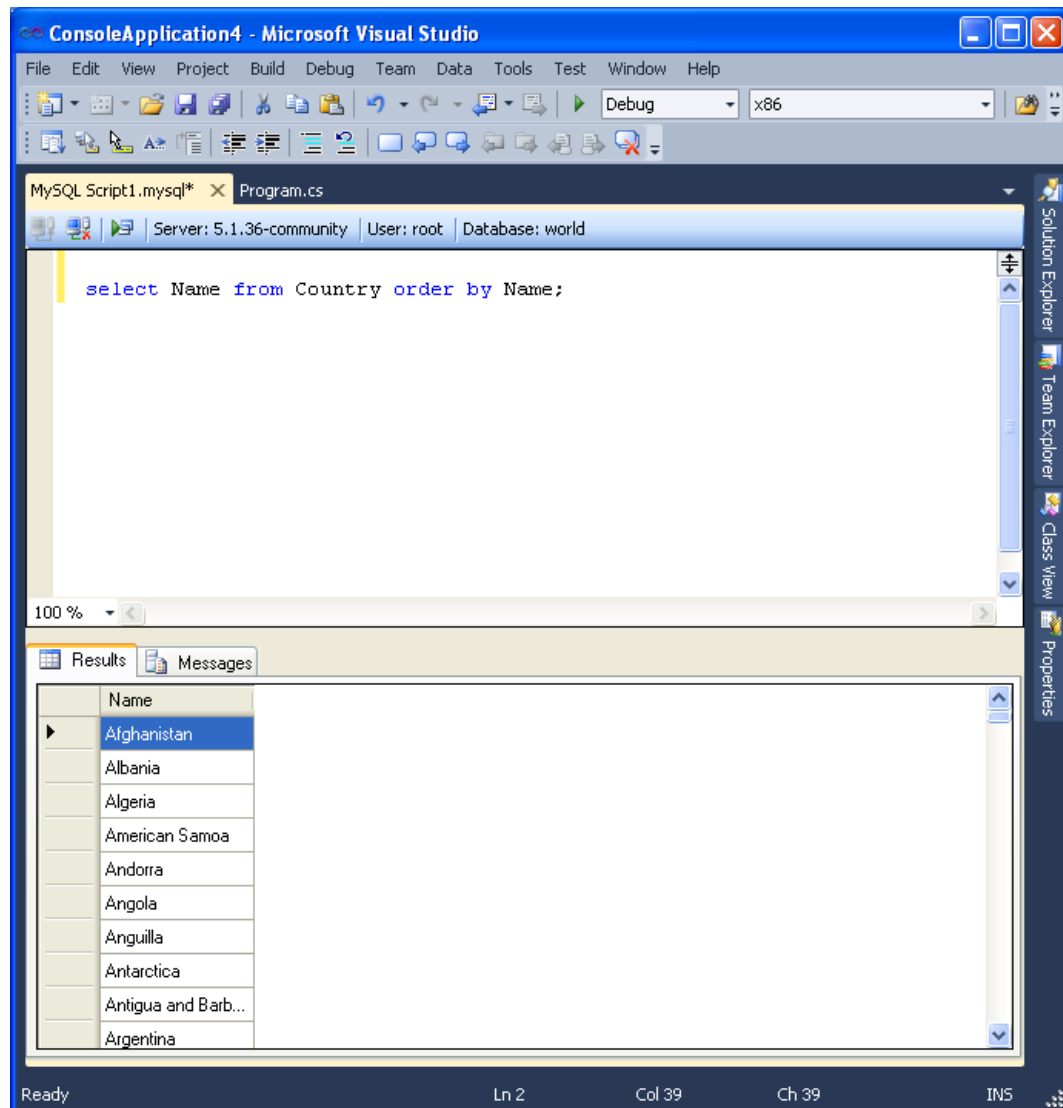
Figure 4.37. MySQL SQL Editor - New File



From the **New File** dialog, select the MySQL template, and then double-click the **MySQL SQL Script** document, or click the **Open** button.

The MySQL SQL Editor will be displayed. You can now enter SQL code as required, or connect to a MySQL server. Click the **Connect to MySQL** button in the MySQL SQL Editor toolbar. You can enter the connection details into the **Connect to MySQL** dialog that is displayed. You can enter the server name, user ID, password and database to connect to, or click the **Advanced** button to select other connection string options. Click the **Connect** button to connect to the MySQL server. To execute your SQL code against the server, click the **Run SQL** button on the toolbar.

Figure 4.38. MySQL SQL Editor - Query

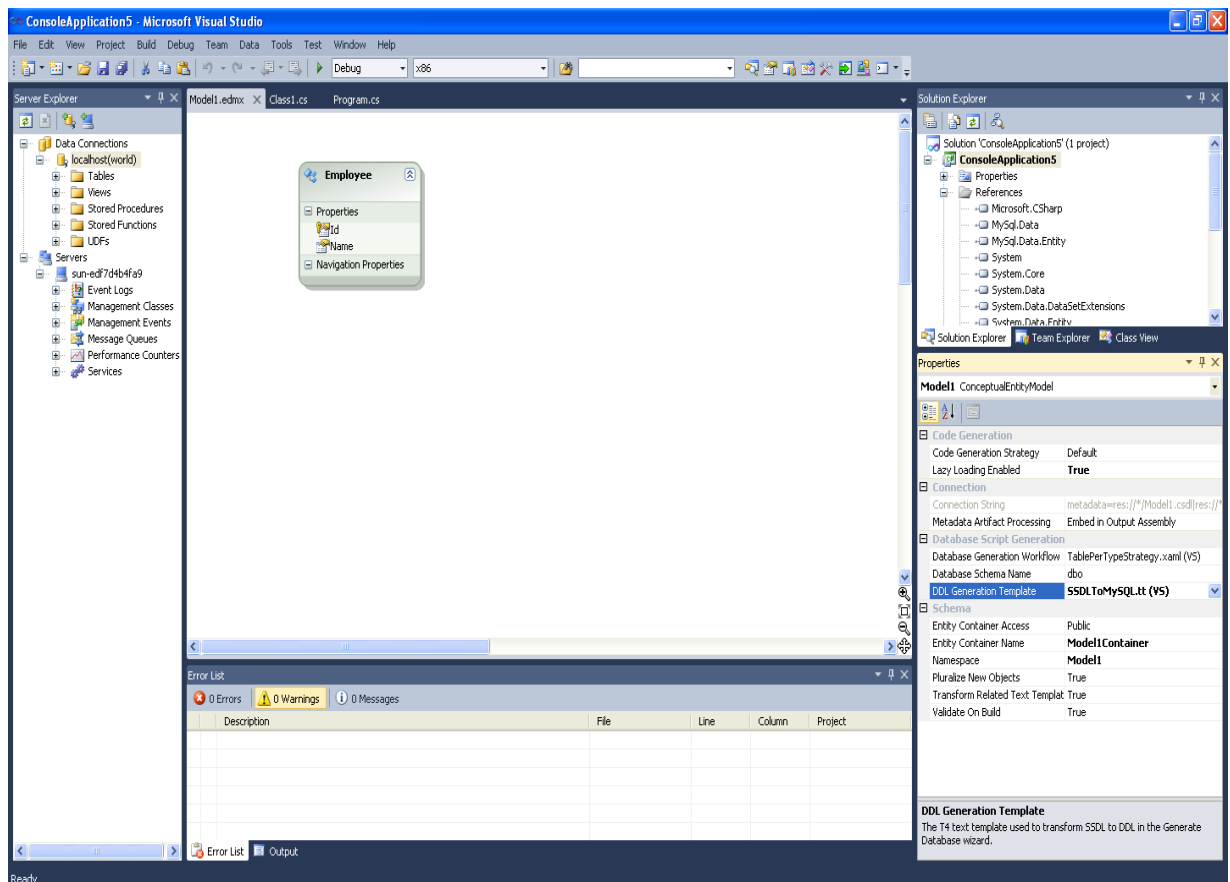


The results from any queries are displayed on the **Results** tab. Any errors are displayed on the **Messages** tab.

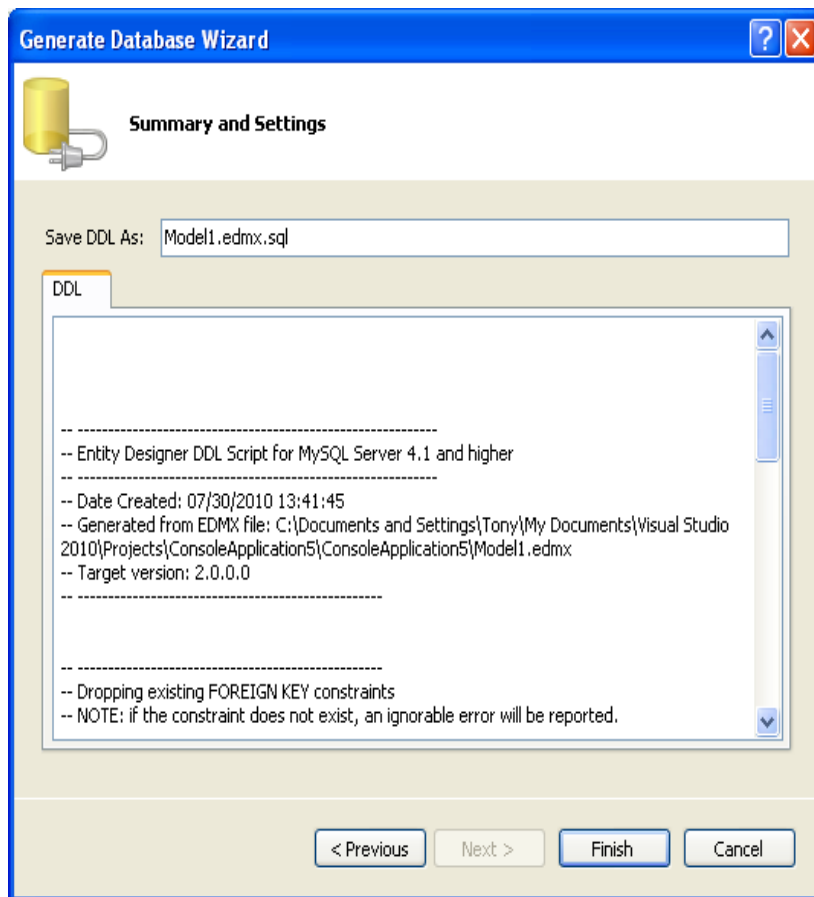
4.14. DDL T4 Template Macro

MySQL Connector/Net 6.3 introduced the ability to convert an Entity Framework model to MySQL **DDL** code. Starting with a blank model, you can develop an entity model in Visual Studio's designer. Once the model is created, you can select the model's properties, and in the Database Script Generation category of the model's properties, the property **DDL Generation** can be found. Select the value **SSDLToMySQL.tt(VS)** from the drop-down listbox.

Figure 4.39. DDL T4 Template Macro - Model Properties



Right-clicking the model design area displays a context-sensitive menu. Selecting **Generate Database from Model** from the menu displays the **Generate Database Wizard**. The wizard can then be used to generate MySQL DDL code.

Figure 4.40. DDL T4 Template Macro - Generate Database Wizard

Chapter 5. Connector/Net Tutorials

Table of Contents

5.1. Tutorial: An Introduction to Connector/Net Programming	47
5.1.1. The MySqlConnection Object	47
5.1.2. The MySqlCommand Object	48
5.1.3. Working with Decoupled Data	50
5.1.4. Working with Parameters	53
5.1.5. Working with Stored Procedures	54
5.2. Tutorial: MySQL Connector/Net ASP.NET Membership and Role Provider	56
5.3. Tutorial: MySQL Connector/Net ASP.NET Session State Provider	60
5.4. Tutorial: MySQL Connector/Net ASP.NET Profile Provider	62
5.5. Tutorial: Using an Entity Framework Entity as a Windows Forms Data Source	64
5.6. Tutorial: Databinding in ASP.NET using LINQ on Entities	76
5.7. Tutorial: Using SSL with MySQL Connector/Net	80
5.8. Tutorial: Using MySqlScript	82
5.8.1. Using Delimiters with MySqlScript	84
5.9. Tutorial: Generating MySQL DDL from an Entity Framework Model	86

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.

5.1. 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 example database, 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`.

5.1.1. 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 6.1, “Connecting to MySQL Using Connector/Net”](#). For a list of supported connection string options, see [Chapter 7, 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.

5.1.2. The MySqlCommand Object

Once a connection has been established with the MySQL database, the next step is to 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. Note that 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', 'Mick";
            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.

5.1.3. 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 that 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");
```

Note the `Fill` method is a `MySqlDataAdapter` method, the Data Adapter knows how to establish a connection with the database and retrieve the required data, and then populates 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);

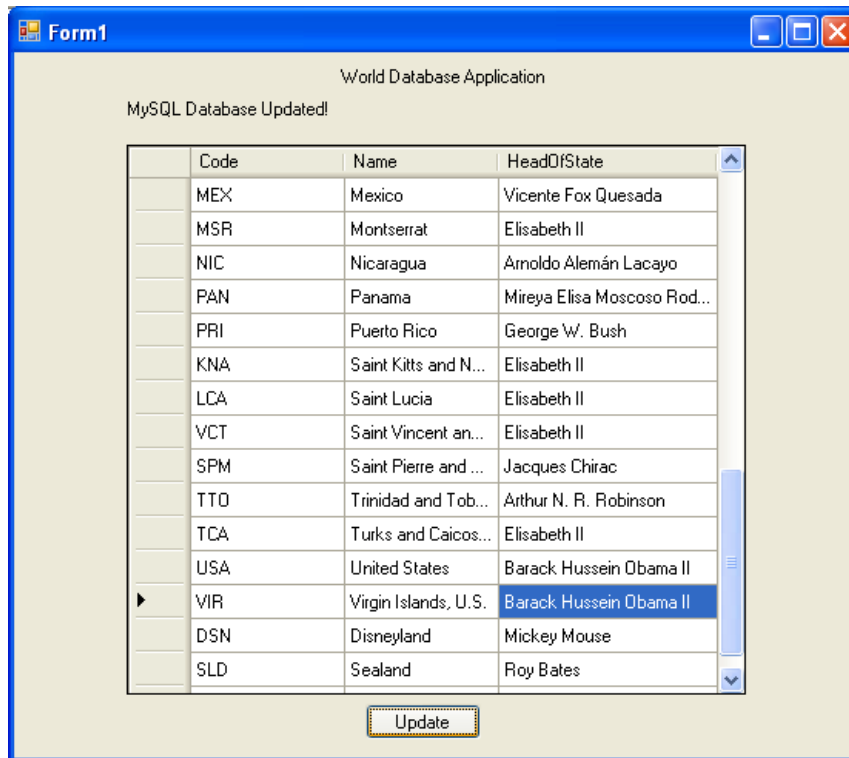
                dsCountry = new DataSet();
                daCountry.Fill(dsCountry, "Country");
                dataGridView1.DataSource = dsCountry;
                dataGridView1.DataMember = "Country";
            }
            catch (Exception ex)
            {
                label2.Text = ex.ToString();
     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

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

```
}
```

The application running is shown below:

Figure 5.1. World Database Application



5.1.4. 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 run time 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 would be:

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

Note that 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 seen how to use parameters to make your code more secure.

5.1.5. 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 that 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 5.1.4, "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.

5.2. 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 5.6, "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 has been 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, Pub  
      connectionStringName="LocalMySqlServer"  
      ... />  
  </providers>  
</membership>
```

Note that the name for the connection string to be used to connect to the server that contains the membership database is [LocalMySqlServer](#).

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>
```

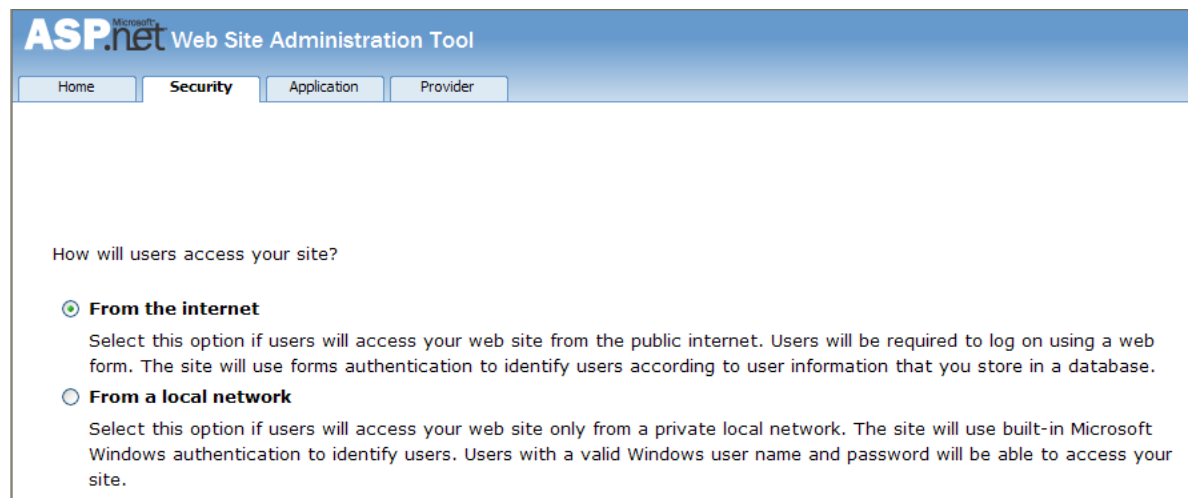
The database specified is the one created in the first step. You could alternatively have used an existing database.

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.

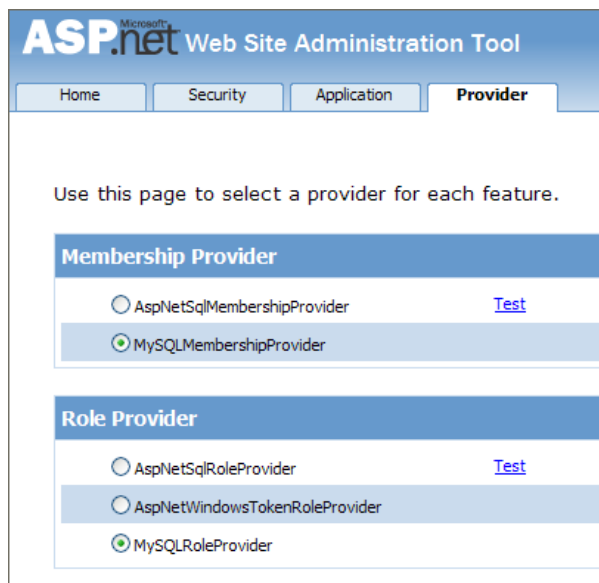
Figure 5.2. Authentication Type



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

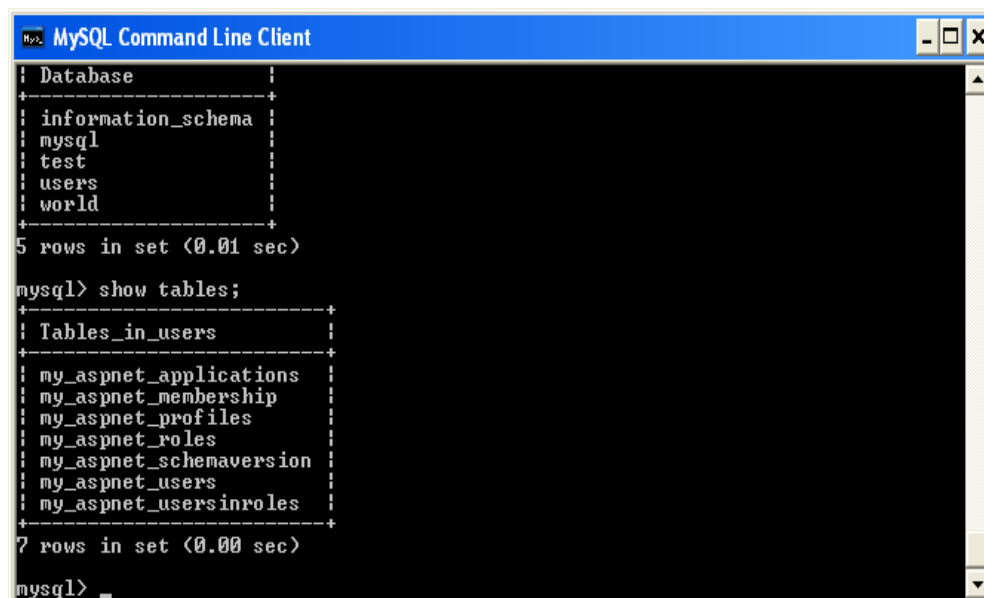
15. Click the **Select a different provider for each feature (advanced)** link.
16. Now select the **MySQLMembershipProvider** and the **MySQLRoleProvider** radio buttons.

Figure 5.3. 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 5.4. 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 5.5. Security Tab

ASP.NET Web Site Administration Tool

Home **Security** Application Provider

You can use the Web Site Administration Tool to manage all the security settings for your application. You can set up users and passwords (authentication), create roles (groups of users), and create permissions (rules for controlling access to parts of your application).

By default, user information is stored in a Microsoft SQL Server Express database in the Data folder of your Web site. If you want to store user information in a different database, use the Provider tab to select a different provider.

[Use the security Setup Wizard to configure security step by step.](#)

Click the links in the table to manage the settings for your application.

Users	Roles	Access Rules
Existing users: 1 Create user Manage users Select authentication type	Existing roles: 2 Disable Roles Create or Manage roles	Create access rules Manage access rules

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 5.6. Create User

ASP.NET Web Site Administration Tool

Home **Security** Application Provider

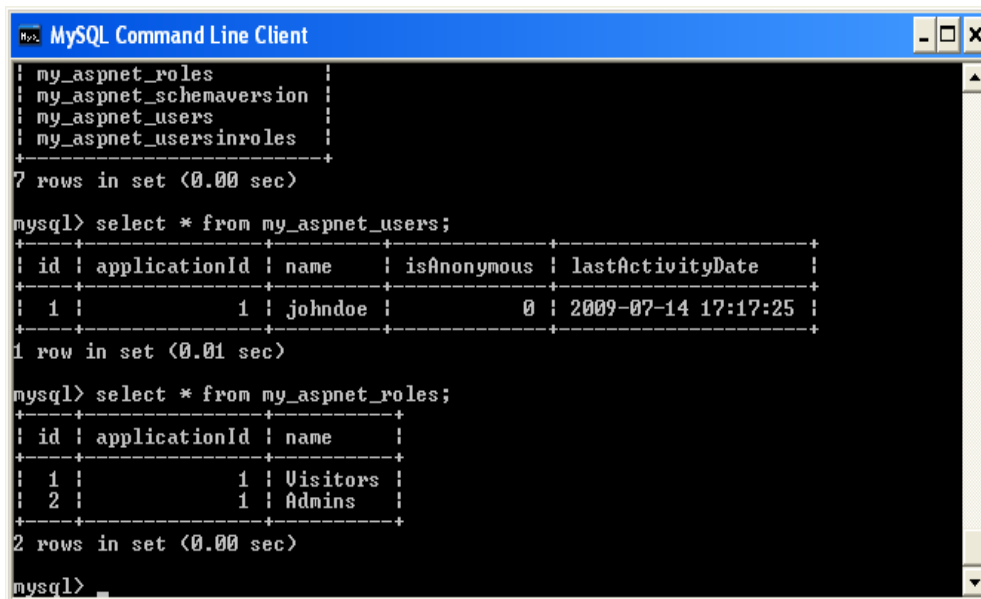
Add a user by entering the user's ID, password, and e-mail address on this page.

Create User	Roles
<p>Sign Up for Your New Account</p> <p>User Name: <input type="text" value="johndoe"/></p> <p>Password: <input type="password" value="••••••••"/></p> <p>Confirm Password: <input type="password" value="••••••••"/></p> <p>E-mail: <input type="text" value="johndoe@example.com"/></p> <p>Security Question: <input type="text" value="Statler and ..."/></p> <p>Security Answer: <input type="text" value="Waldorf"/></p> <p><input type="button" value="Create User"/></p>	<p>Select roles for this user:</p> <p><input type="checkbox"/> Visitors</p> <p><input checked="" type="checkbox"/> Admins</p>

☒ Active User

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

Figure 5.7. Membership and Roles Table Contents



```

MySQL Command Line Client
mysql> show tables;
+-----+
| my_aspnet_roles |
| my_aspnet_schemaversion |
| my_aspnet_users |
| my_aspnet_usersinroles |
+-----+
7 rows in set (0.00 sec)

mysql> select * from my_aspnet_users;
+----+-----+-----+-----+-----+
| id | applicationId | name | isAnonymous | lastActivityDate |
+----+-----+-----+-----+-----+
| 1 | 1 | johndoe | 0 | 2009-07-14 17:17:25 |
+----+-----+-----+-----+-----+
1 row in set (0.01 sec)

mysql> select * from my_aspnet_roles;
+----+-----+-----+
| id | applicationId | name |
+----+-----+-----+
| 1 | 1 | Visitors |
| 2 | 1 | Admins |
+----+-----+-----+
2 rows in set (0.00 sec)

mysql>

```

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.

5.3. Tutorial: MySQL Connector/Net ASP.NET Session State Provider

MySQL Connector/Net from version 6.1 has included a MySQL Session State Provider. This provider enables you to store session state in a MySQL database. The following tutorial shows you how to prepare to use the MySQL Session State Provider, and then store session data into the MySQL database. This tutorial uses Microsoft Visual Studio 2008 Professional Edition, MySQL Connector/Net 6.1.1 and MySQL Server 5.1. This tutorial also assumes you have created an empty database, for example `test`, where you will store session data. You could do this using the `mysql` command interpreter.

1. In Visual Studio, create a new ASP.NET web site. If you are not sure how to do this, refer to the tutorial [Section 5.6, "Tutorial: Databinding in ASP.NET using LINQ on Entities"](#), which demonstrates how to do this.
2. Launch the MySQL Website Configuration tool. Due to a bug in 6.1.1, this may not appear unless you are connected to a server in the Server Explorer. If you are unfamiliar with the MySQL Website Configuration tool, consider first working through the tutorial in [Section 4.12, "MySQL Website Configuration Tool"](#).
3. Navigate through the wizard to the Session State page. Make sure the check box **Use MySQL to manage my ASP.NET session data** is selected.
4. On the same page, configure the connection string to the database that will contain your session data. If this database is empty, MySQL Connector/Net will create the schema required to store session data.
5. Ensure that the check box **Autogenerate Schema** is selected so that MySQL Connector/Net will create the schema in your database to store the session data correctly.
6. Enter the name of your application.

- Click **Finish**. The MySQL Website Configuration tool will now update your application's `web.config` file with information about the connection string and default providers to be used. In this case, we have selected the MySQL Session State Provider.

At this point, you are ready to use the MySQL database to store session data. To test that the set up has worked, you can write a simple program that uses session variables.

- Open `Default.aspx.cs`. In the `Page_Load` method, add the following code:

```
Session["SessionVariable1"] = "Test string";
```

- Build your solution.
- Run the solution (without debugging). When the application runs, the provider will autogenerate tables required in the database you chose when setting up the application.
- Check that the schema was in fact created. Using the MySQL Command Line Client use the target database and then type `SHOW TABLES;`. You will see that MySQL Connector/Net has created the required schema automatically, as we selected this to happen in the MySQL Website Configuration tool.
- Now view the contents of these tables by typing `SELECT * FROM my_aspnet_sessions;` in the `mysql` command interpreter. This will display the session data our application used. Note that this is stored in binary format so some data may not display as expected.

At this point, you have installed the Session State Provider and carried out a preliminary test of the installation. You will now work a bit more with the Session State Provider.

In this part of the tutorial, you will set and retrieve a session variable. You can work with your existing project.

- Select the `Default.aspx` and switch to Design View. Add a text box and three buttons. Change the text property for the buttons to "Store Session Variable", "Clear Textbox", and "Show Session Variable". These will be `Button1`, `Button2` and `Button3` respectively. Build your solution to ensure that no errors have been introduced.
- Still in the Design View, double-click `Button1`. Now to the `Button1_Click` event handler add code some the handler resembles the following:

```
protected void Button1_Click(object sender, EventArgs e)
{
    Session["SessionString"] = TextBox1.Text;
}
```

You have created a new Session variable accessed using the key "SessionString". This will be set to the text that was entered into the text box when `Button1` is clicked.

- In Design View, double-click `Button2` to add its click event handler. This button needs to clear text from the text box. The code to do this is as follows:

```
protected void Button2_Click(object sender, EventArgs e)
{
    TextBox1.Text = "";
}
```

The code simply assigns an empty string to the `Text` property of the text box.

- In the Design View double-click `Button3` and modify the click handler as follows:

```
protected void Button3_Click(object sender, EventArgs e)
{
    TextBox1.Text = (String)Session["SessionString"];
}
```

This will retrieve the session string and display it in the text box.

5. Now modify the `Page_Load` method as follows:

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        TextBox1.Text = "Enter some text";
    }
}
```

This ensures that when the page loads the text box `Text` property is reset.

6. Ensure that the solution is saved and then rebuild the solution.
7. Run the solution without debugging.
8. The form will be displayed. Enter some text into the text box. Now click `Store Session Variable`. At this point you have stored the string in a session variable.
9. Now click `Clear Text` to clear the text box.
10. Now click `Show Session Variable` to retrieve and display the session variable.
11. Refresh the page to destroy the form and display a new form.
12. Click `Show Session Variable` the text box will display the stored session variable, demonstrating that the refreshing the page does not destroy the session variable.

This illustrates that the session state data is not destroyed when a page is reloaded.

5.4. 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 considerably 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:

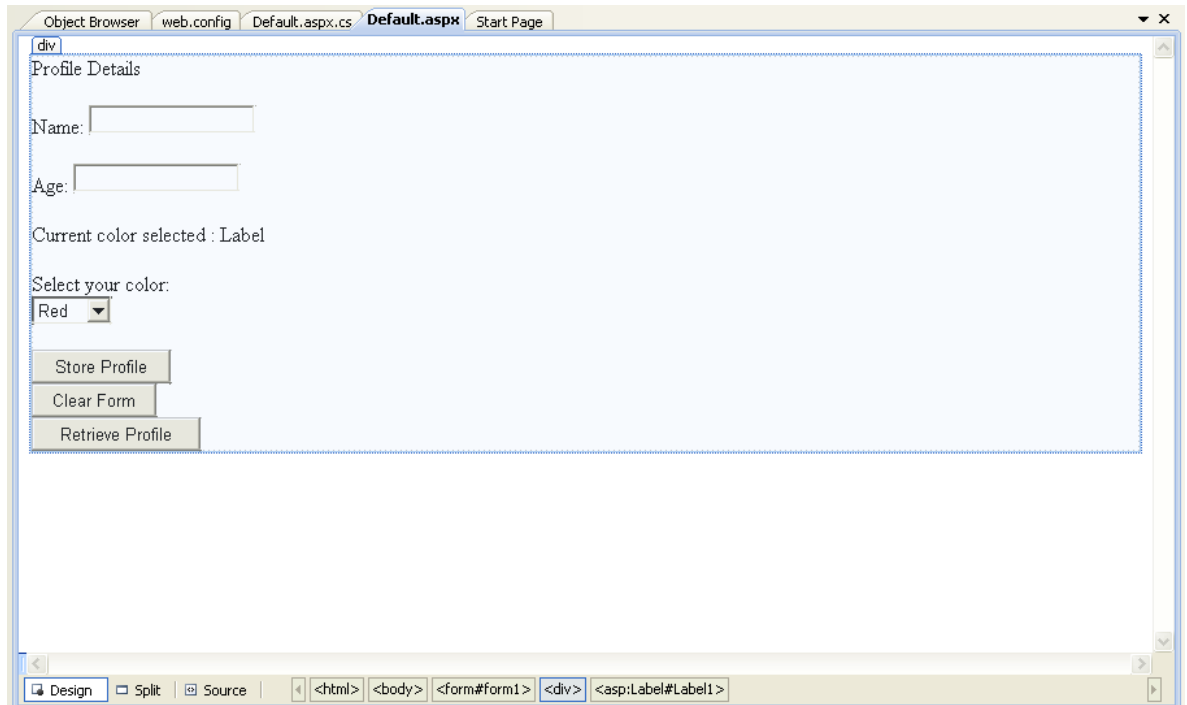
```
<system.web>
  <anonymousIdentification enabled="true" />
  <profile defaultProvider="MySQLProfileProvider">
    ...
    <properties>
      <add name="Name" allowAnonymous="true" />
      <add name="Age" allowAnonymous="true" type="System.UInt16" />
      <group name="UI">
        <add name="Color"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

Note that `anonymousIdentification` has been set to true. This enables users who have not been authenticated to use profiles. They are identified by a GUID in a cookie rather than by 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 5.8. 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**. Note that if you do not select a color from the listbox your profile will use 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. Note that when the page loads your profile information is restored from the MySQL database.

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

5.5. 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 example database, 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 [Section 4.1, "Making a Connection"](#).

Creating a new Windows Forms application

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

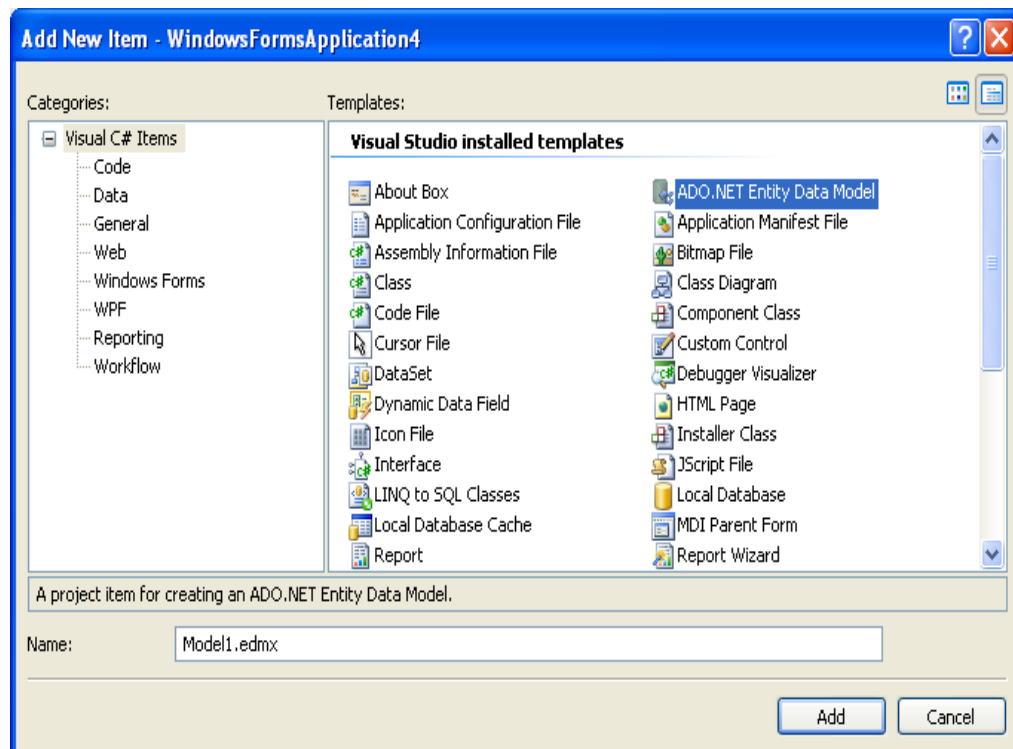
1. In Visual Studio, select **File, New, Project** from the main menu.
2. Choose the **Windows Forms Application** installed template. Click **OK**. The solution is created.

Adding an Entity Data Model

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

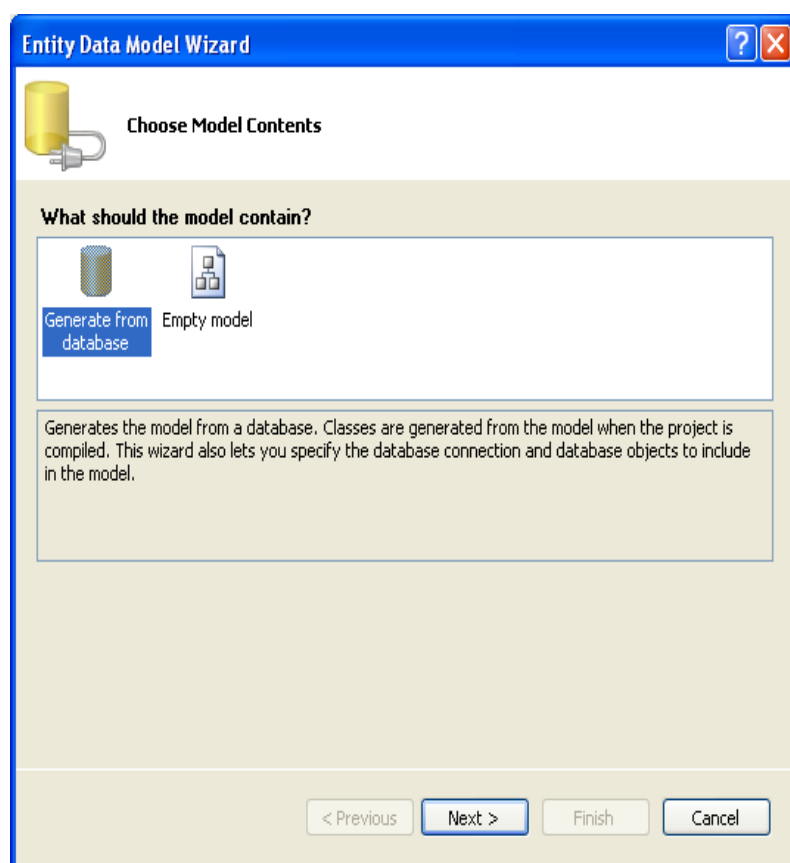
1. In the Solution Explorer, right-click your application and select **Add, New Item...**. From **Visual Studio installed templates** select **ADO.NET Entity Data Model**. Click **Add**.

Figure 5.9. Add Entity Data Model

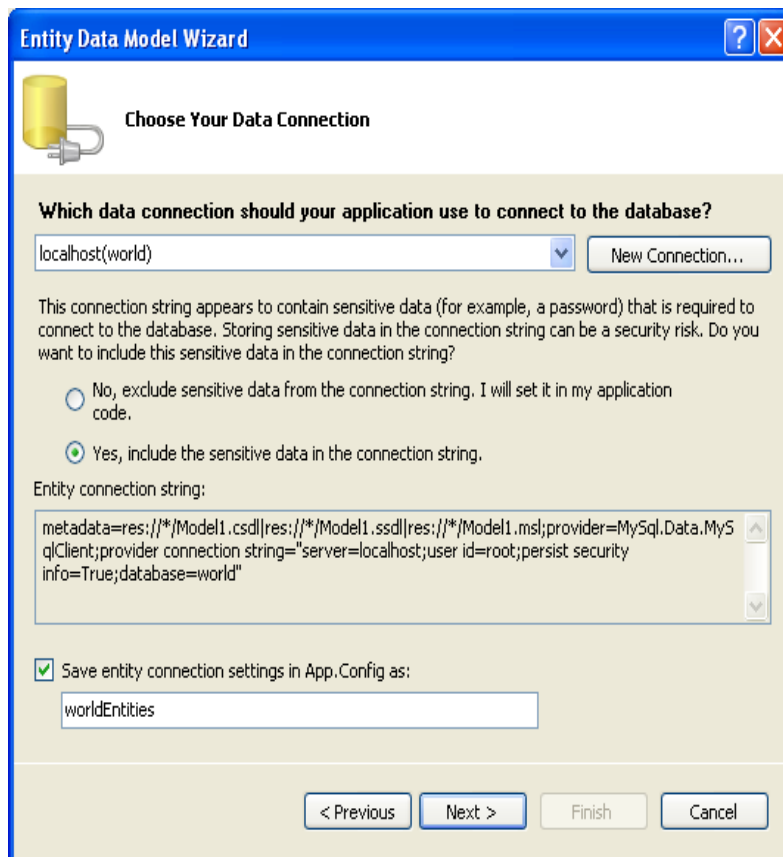


2. You will now see the Entity Data Model Wizard. You will use the wizard to generate the Entity Data Model from the world example database. Select the icon **Generate from database**. Click **Next**.

Figure 5.10. Entity Data Model Wizard Screen 1



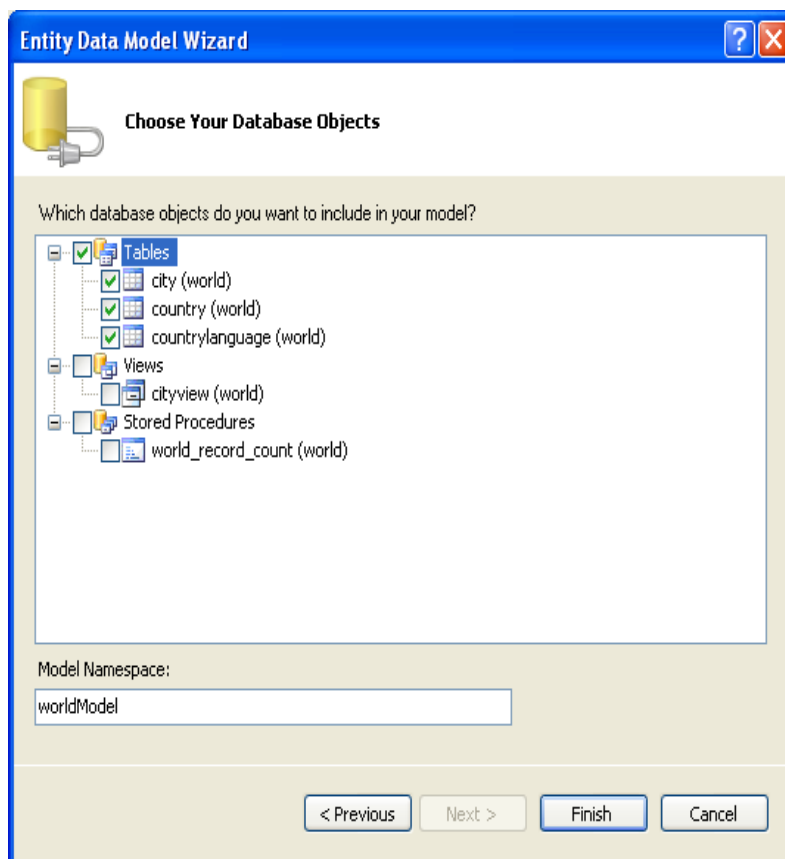
3. You can now select the connection you made earlier to the World database. If you have not already done so, you can create the new connection at this time by clicking **New Connection...** For further instructions on creating a connection to a database see [Section 4.1, "Making a Connection"](#).

Figure 5.11. Entity Data Model Wizard Screen 2

The screenshot shows the 'Entity Data Model Wizard' window, specifically the 'Choose Your Data Connection' screen. The window has a blue title bar with the text 'Entity Data Model Wizard' and standard Windows window controls. Below the title bar is a yellow cylinder icon with a plug. The main area is titled 'Choose Your Data Connection'. It asks 'Which data connection should your application use to connect to the database?'. There is a dropdown menu showing 'localhost(world)' and a 'New Connection...' button. Below this, a warning message states: 'This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?'. There are two radio buttons: 'No, exclude sensitive data from the connection string. I will set it in my application code.' (unselected) and 'Yes, include the sensitive data in the connection string.' (selected). Below the radio buttons is a text box labeled 'Entity connection string:' containing the string: 'metadata=res://*/Model1.csdl|res://*/Model1.ssdl|res://*/Model1.msl;provider=MySql.Data.MySqlClient;provider connection string="server=localhost;user id=root;persist security info=True;database=world"'. At the bottom, there is a checkbox 'Save entity connection settings in App.Config as:' which is checked, followed by a text box containing 'worldEntities'. At the very bottom are four buttons: '< Previous', 'Next >', 'Finish', and 'Cancel'.

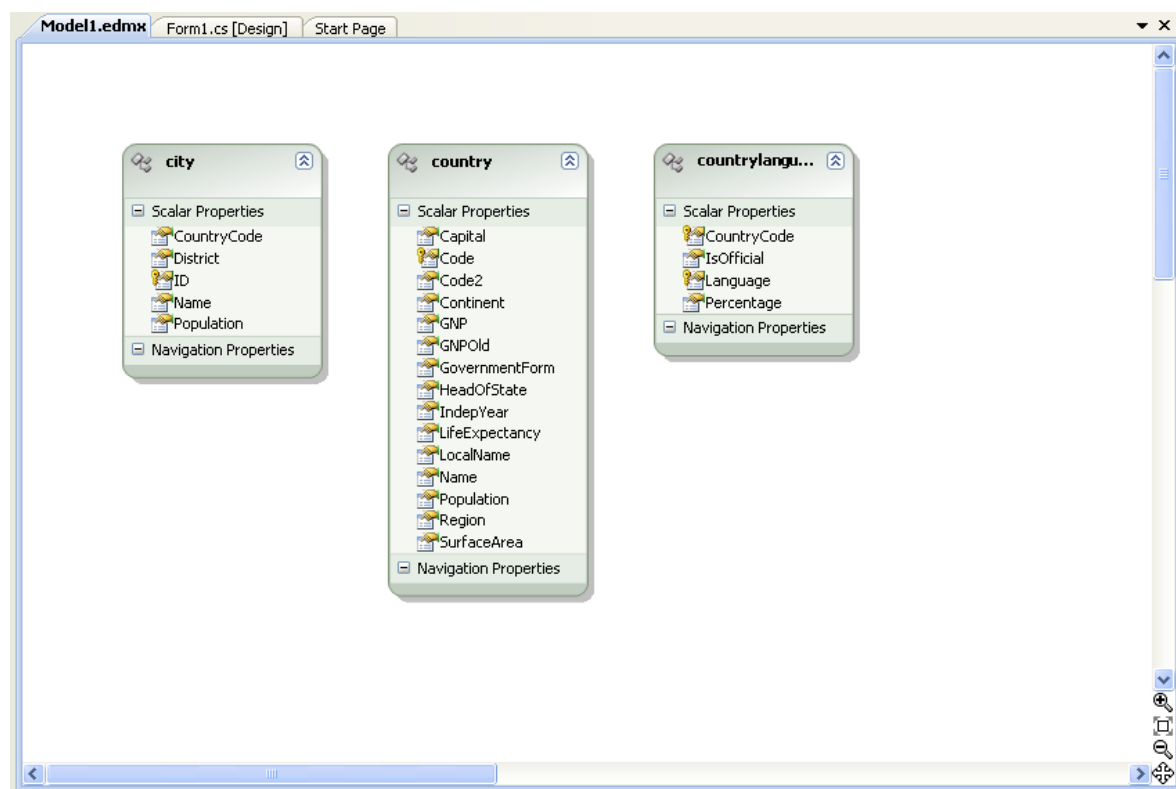
4. Make a note of the entity connection settings to be used in App.Config, as these will be used later to write the necessary control code.
5. Click **Next**.
6. The Entity Data Model Wizard connects to the database. You are then presented with a tree structure of the database. From this you can select 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 5.12. Entity Data Model Wizard Screen 3



7. Visual Studio will generate the model and then display it.

Figure 5.13. Entity Data Model Diagram



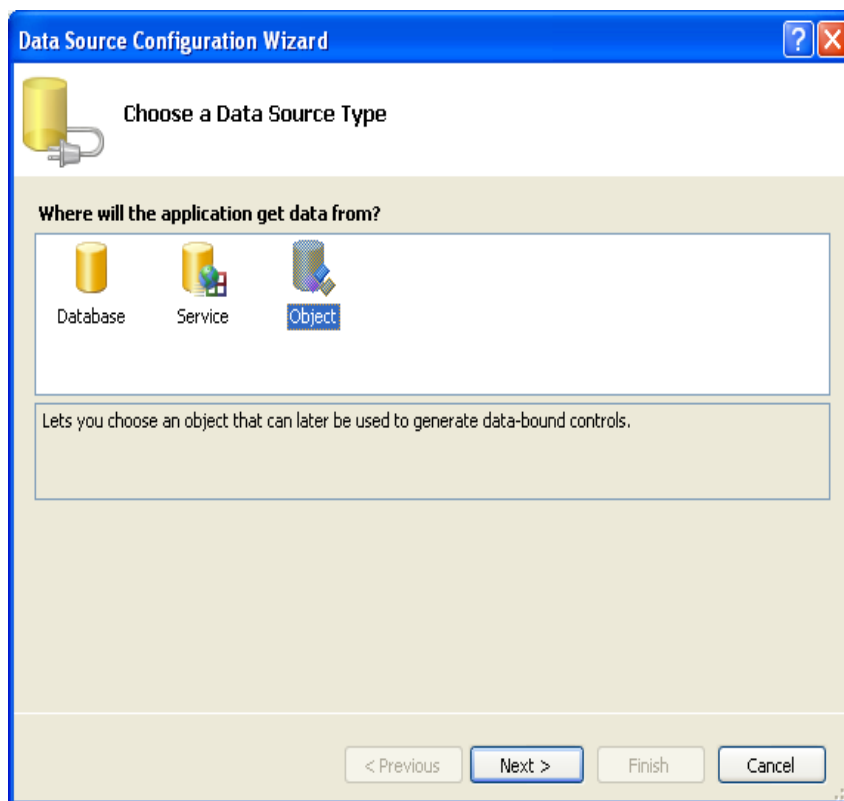
8. From the Visual Studio main menu select **Build, Build Solution**, to ensure that everything compiles correctly so far.

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.

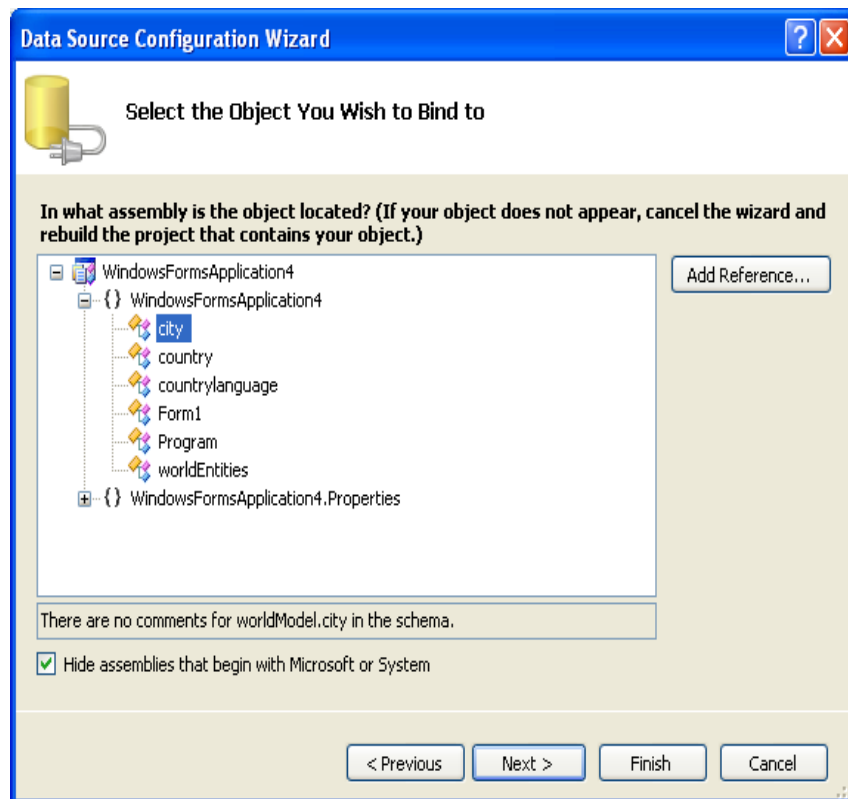
1. From the Visual Studio main menu select **Data, Add New Data Source...**. You will be presented with the Data Source Configuration Wizard.

Figure 5.14. Entity Data Source Configuration Wizard Screen 1



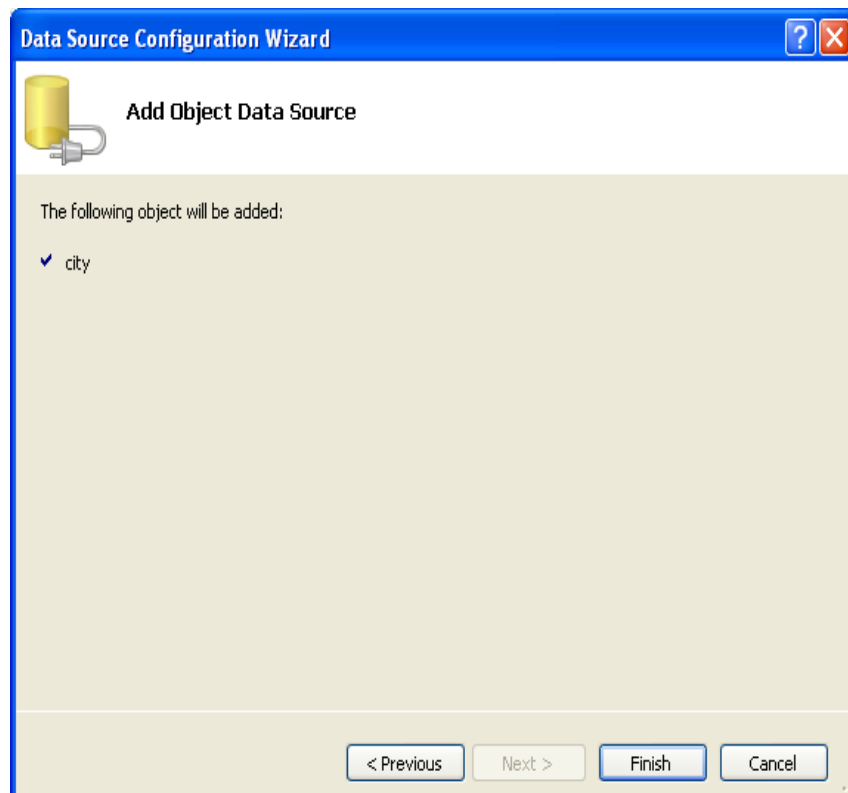
2. Select the **Object** icon. Click **Next**.
3. You will now select the Object to bind to. Expand the tree. In this tutorial, you will select the city table. Once the city table has been selected click **Next**.

Figure 5.15. Entity Data Source Configuration Wizard Screen 2



4. The wizard will confirm that the city object is to be added. Click **Finish**.

Figure 5.16. 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 5.17. Data Sources

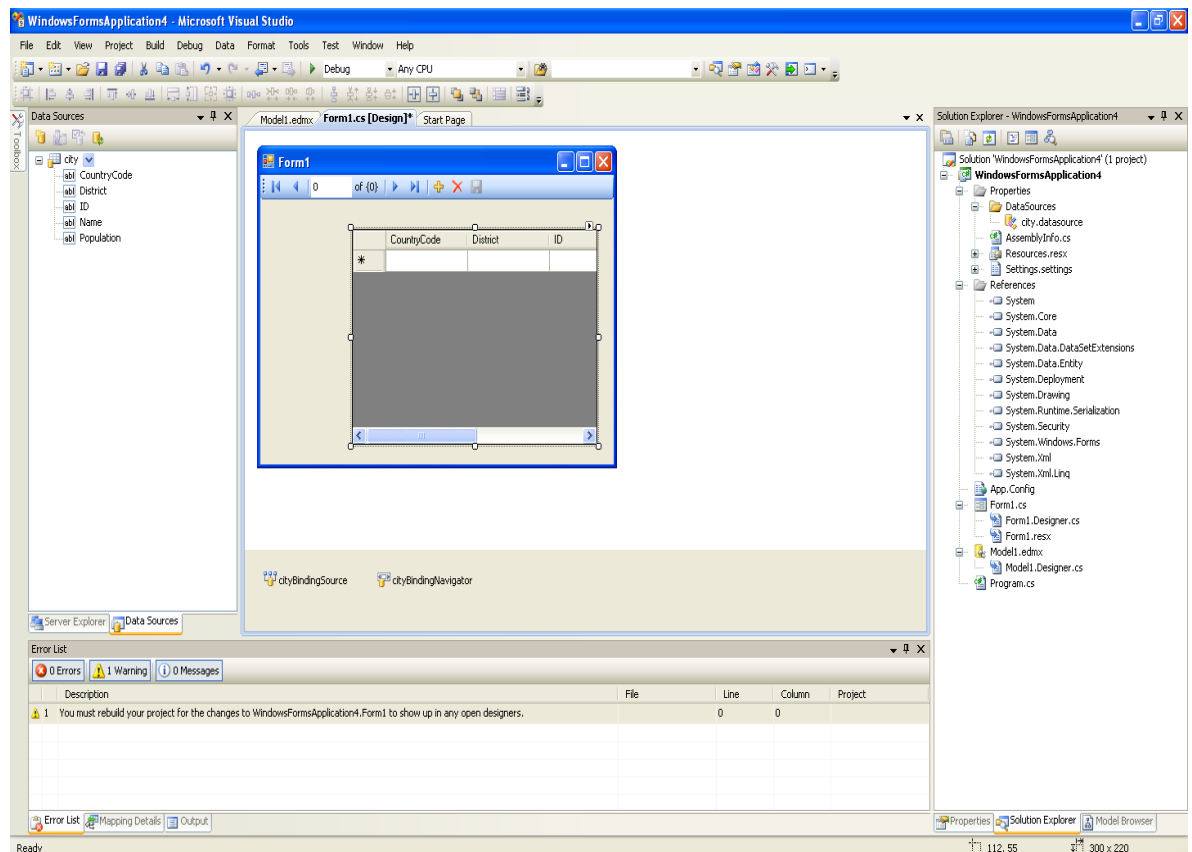


Using the Data Source in a Windows Form

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 that the Data Grid View control is bound to the `cityBindingSource` and the Navigator control is bound to `cityBindingNavigator`.

Figure 5.18. Data Form Designer



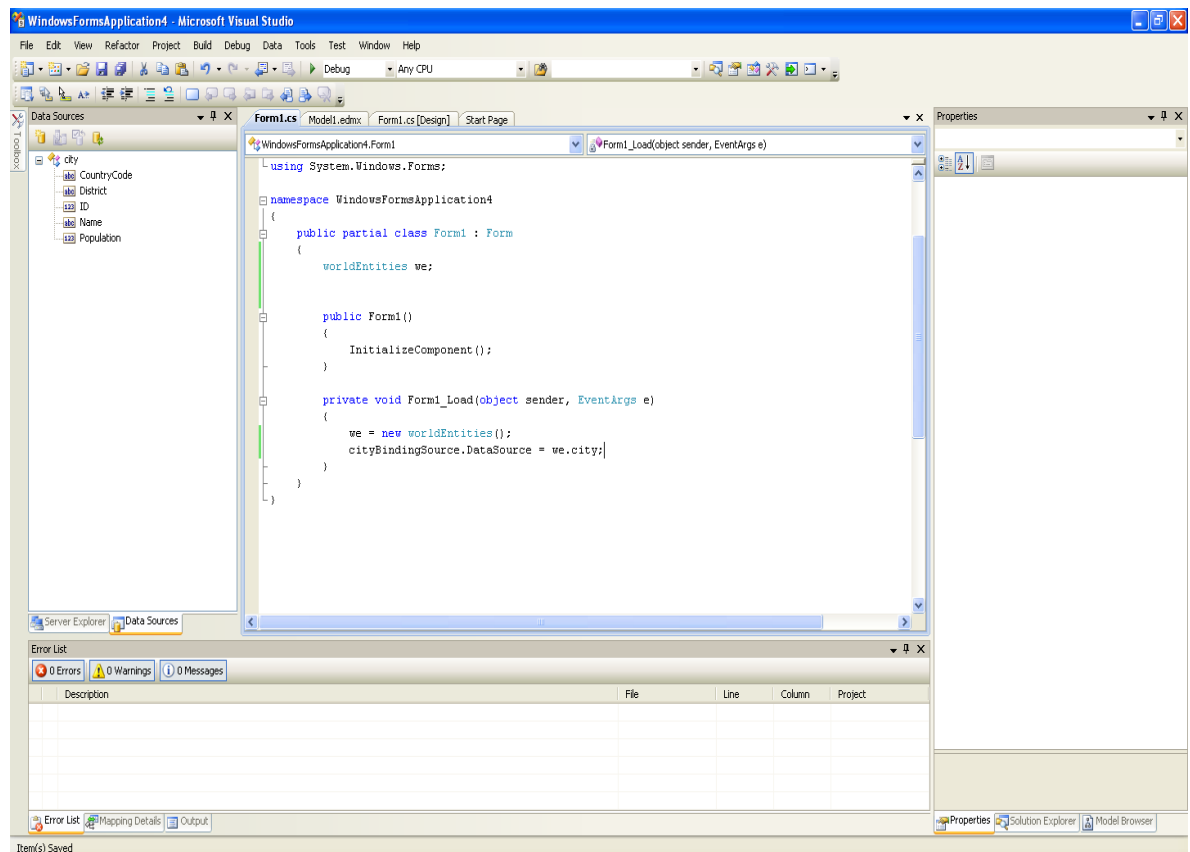
2. Save and rebuild the solution before continuing.

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.

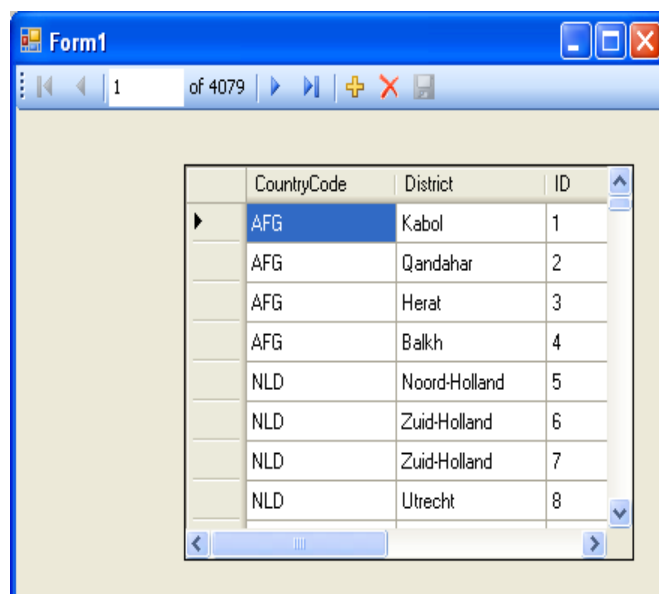
1. Double-click the form to access its code.
2. Add code to instantiate the Entity Data Model's EntityContainer object and retrieve data from the database to populate the control.

Figure 5.19. 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 5.20. The Populated Grid Control



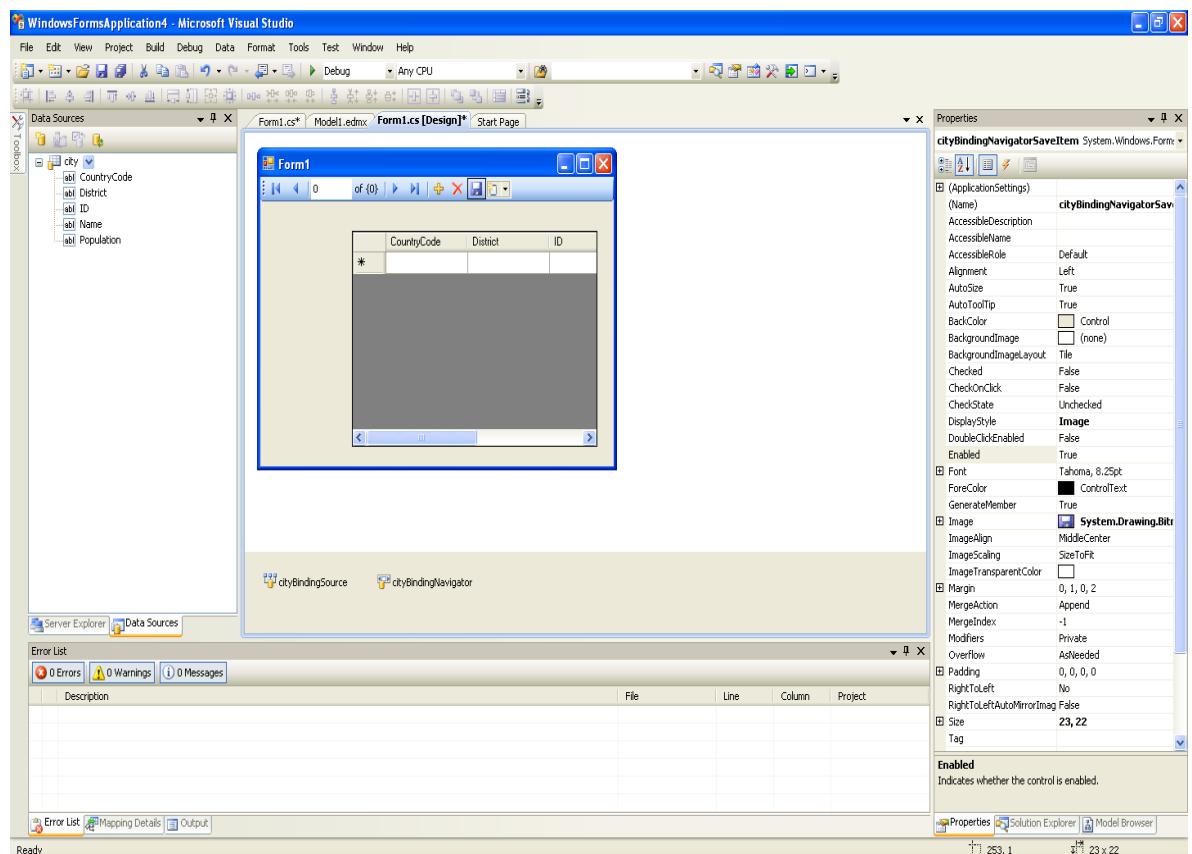
Adding Code to Save Changes to the Database

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 5.21. 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 5.22. 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.

5.6. Tutorial: Databinding in ASP.NET using LINQ on Entities

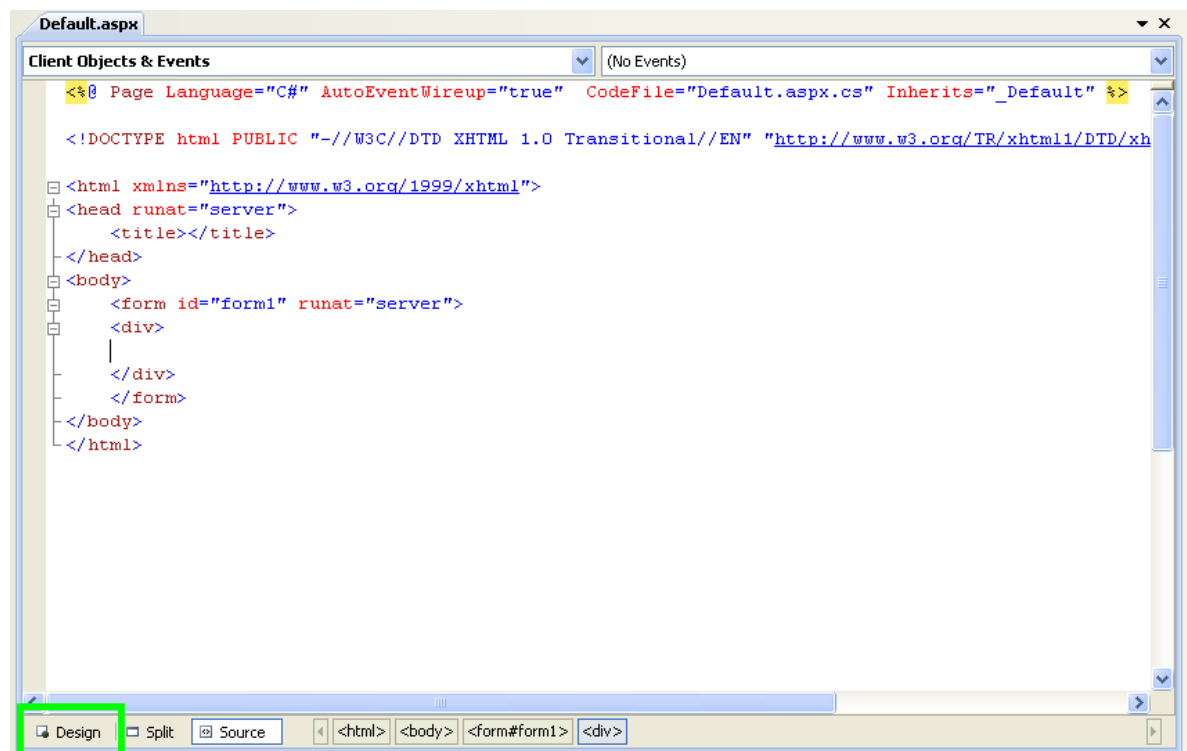
In this tutorial you create an ASP.NET web page that binds LINQ queries to entities using the Entity Framework mapping.

If you have not already done so, install the World example database prior to attempting this tutorial. See the tutorial [Section 5.5, "Tutorial: Using an Entity Framework Entity as a Windows Forms Data Source"](#) for instructions on downloading and installing this database.

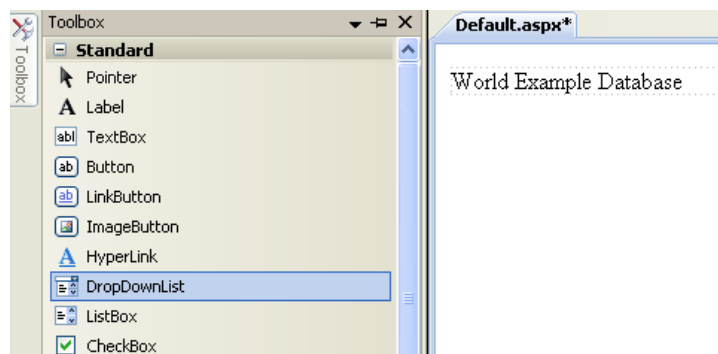
Creating an ASP.NET web site

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.

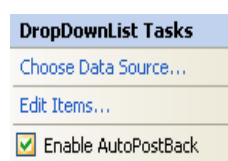
1. From the Visual Studio main menu select **File, New, Web Site...**.
2. From the Visual Studio installed templates select **ASP.NET Web Site**. Click **OK**. You will be presented with the Source view of your web page by default.
3. Click the Design view tab situated underneath the Source view panel.

Figure 5.23. The Design Tab

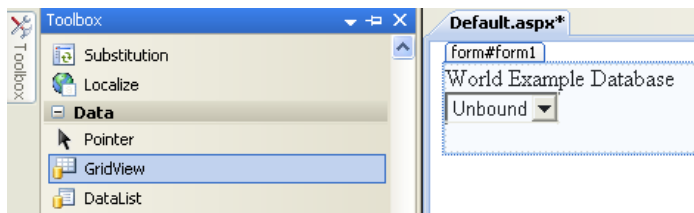
4. In the Design view panel, enter some text to decorate the blank web page.
5. Click Toolbox. From the list of controls select **DropDownList**. Drag and drop the control to a location beneath the text on your web page.

Figure 5.24. Drop Down List

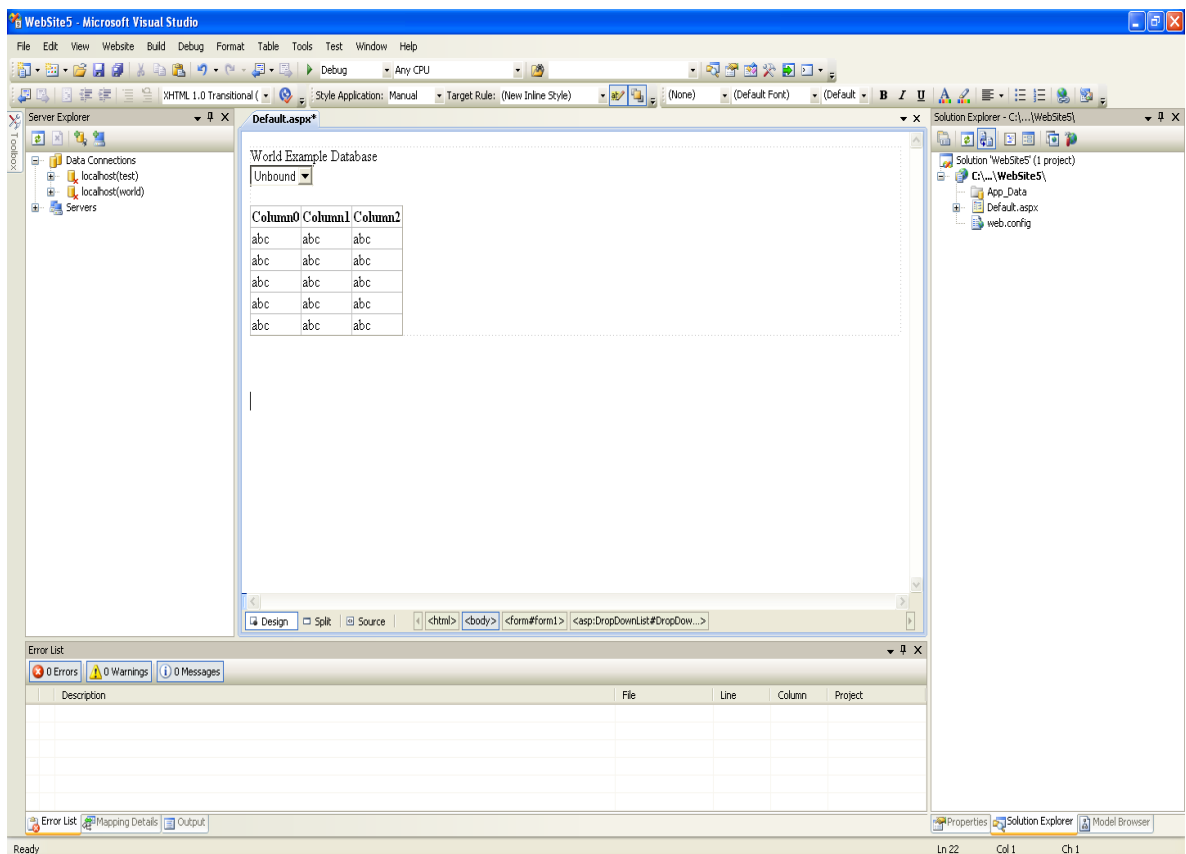
6. From the **DropDownList** control's context menu, ensure that the **Enable AutoPostBack** check box is enabled. This will ensure the control's event handler is called when an item is selected. The user's choice will in turn be used to populate the **GridView** control.

Figure 5.25. Enable AutoPostBack

7. From the Toolbox select the **GridView** control.

Figure 5.26. Grid View Control

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

Figure 5.27. Placed Grid View 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 5.5, “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();
        }
    }
    ...
}
```

Note that 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

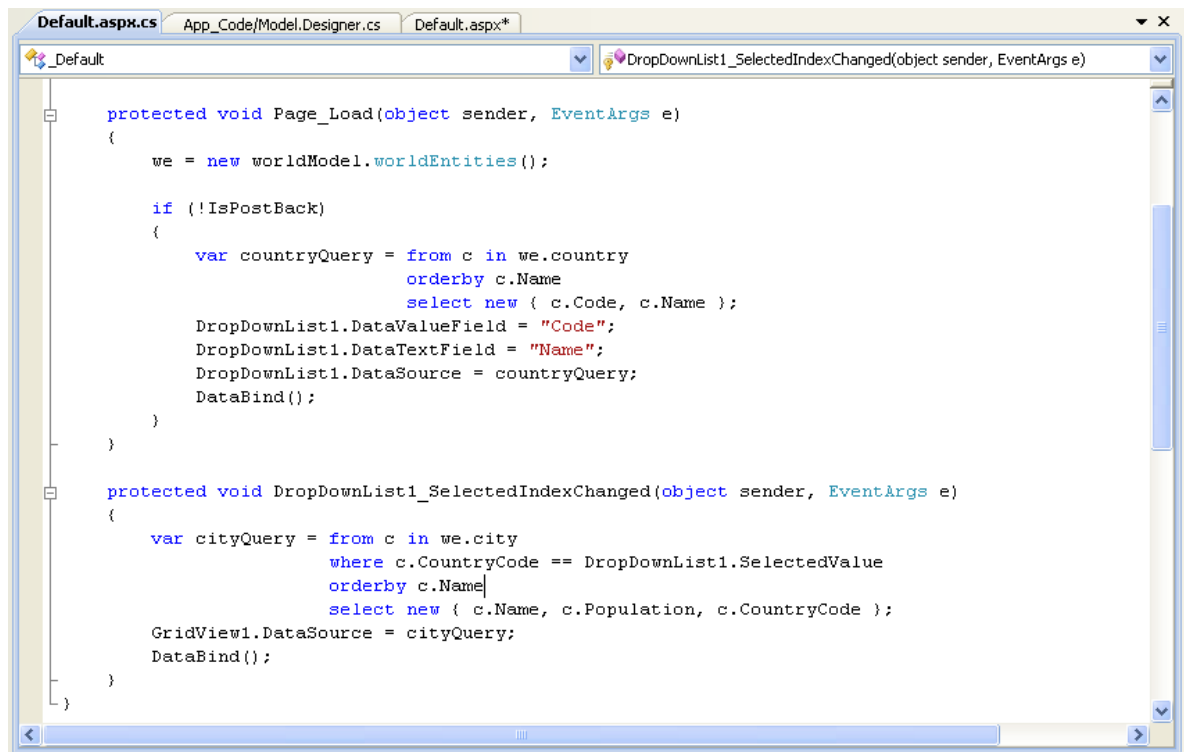
In the last part of this tutorial you will populate the Grid View Control using a LINQ query on your entity data model.

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:

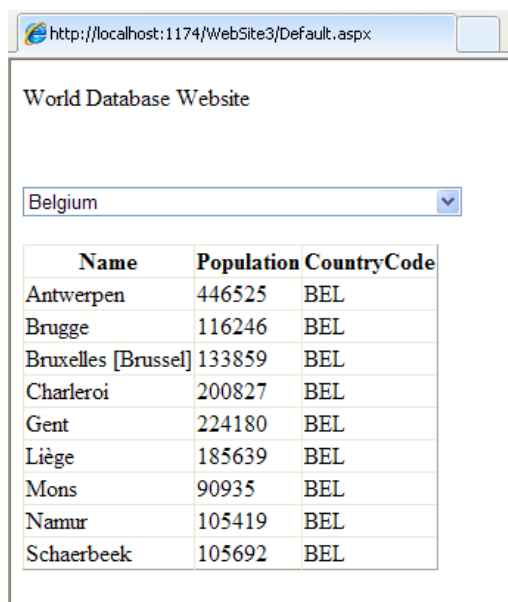
```
...
protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
{
    var cityQuery = from c in we.city
                     where c.CountryCode == DropDownList1.SelectedValue
                     orderby c.Name
                     select new { c.Name, c.Population, c.CountryCode };
    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 5.28. 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 5.29. 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.

5.7. 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 7, 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:\\bzt\\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 succinct 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=479436009a40f3017a145cf8479e7694d7aadeef0;" +
    "SSL Mode=Required"))
{
    connection.Open();
}
```

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

5.8. 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. Note the Query property can be set through the [MySqlScript](#) constructor or 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 MySqlScriptErrorHandler(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 MySqlScriptErrorHandler(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());
}
}
```

Note that 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`.

5.8.1. 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:

```
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.");
        }
    }
}
```

5.9. 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 6. Connector/Net Programming

Table of Contents

6.1. Connecting to MySQL Using Connector/Net	88
6.2. Creating a Connector/Net Connection String	88
6.2.1. Opening a Connection	88
6.2.2. Handling Connection Errors	90
6.2.3. Using GetSchema on a Connection	91
6.3. Using MySqlCommand	92
6.4. Using Connector/Net with Connection Pooling	93
6.5. Using the Windows Native Authentication Plugin	94
6.6. Writing a Custom Authentication Plugin	94
6.7. Using Connector/Net with Table Caching	98
6.8. Using the Connector/Net with Prepared Statements	98
6.8.1. Preparing Statements in Connector/Net	98
6.9. Accessing Stored Procedures with Connector/Net	99
6.9.1. Using Stored Routines from Connector/Net	100
6.10. Handling BLOB Data With Connector/Net	102
6.10.1. Preparing the MySQL Server	102
6.10.2. Writing a File to the Database	103
6.10.3. Reading a BLOB from the Database to a File on Disk	104
6.11. Using the Connector/Net Interceptor Classes	105
6.12. Handling Date and Time Information in Connector/Net	107
6.12.1. Fractional Seconds	107
6.12.2. Problems when Using Invalid Dates	107
6.12.3. Restricting Invalid Dates	107
6.12.4. Handling Invalid Dates	108
6.12.5. Handling NULL Dates	108
6.13. Using the MySqlBulkLoader Class	108
6.14. Using the MySQL Connector/Net Trace Source Object	110
6.14.1. Viewing MySQL Trace Information	111
6.14.2. Building Custom Listeners	113
6.15. Binary/Nonbinary Issues	115
6.16. Character Set Considerations for Connector/Net	115
6.17. Using Connector/Net with Crystal Reports	116
6.17.1. Creating a Data Source	116
6.17.2. Creating the Report	117
6.17.3. Displaying the Report	117
6.18. ASP.NET Provider Model	120
6.19. Working with Partial Trust / Medium Trust	122
6.19.1. Evolution of Partial Trust Support Across Connector/Net Versions	122
6.19.2. Configuring Partial Trust with Connector/Net Library Installed in GAC	123
6.19.3. Configuring Partial Trust with Connector/Net Library Not Installed in GAC	124

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.

6.1. 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.

6.2. 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 7, 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 7, Connector/Net Connection String Options Reference](#). The [Old Syntax](#) connection string option has now been deprecated.

6.2.1. 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 6.5, "Using the Windows Native Authentication Plugin"](#) for details.

You can further extend the authentication mechanism by writing your own authentication plugin. See [Section 6.6, “Writing a Custom Authentication Plugin”](#) for details.

Visual Basic Example

```
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

```
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

```
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.

6.2.2. 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");
        case 1045:
            MessageBox.Show("Invalid username/password, please try again");
    }
}
```

Important

Note that if you are using multilanguage databases 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;");
```

6.2.3. 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](#).

6.2.3.1. 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](#).

6.3. 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 - A 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 5.1.2, "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.WriteLine(reader[0], reader[1]...);
}
...
```

Examples of using the `CommandType` of `StoredProcedure` can be found in the section [Section 6.9, "Accessing Stored Procedures with Connector/Net"](#).

Commands can have a timeout associated with them. This is useful as you may not want a situation where 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 `MySqlDataReader.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](#).

6.4. 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 6.2, "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. Note, 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.

6.5. 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. 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, [The Windows Native Authentication Plugin](#).

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.

6.6. 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 MySqlConnection  
/// 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 can be an 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 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"/>
</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.

6.7. 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.

6.8. Using the Connector/Net with Prepared Statements

As of MySQL 4.1, it is possible to use prepared statements with Connector/Net. Use of prepared statements can provide significant performance improvements on queries that are executed more than once.

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 it uses a binary protocol that makes data transfer between client and server more efficient.

6.8.1. 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, 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();
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);
}

```

6.9. 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.

6.9.1. 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 `MySqlConnection.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,
    "IN fname VARCHAR(20), IN lname VARCHAR(20), IN bday DATETIME, OUT empno INT UNSIGNED)";
    cmd.ExecuteNonQuery();
    cmd.CommandText = "CREATE PROCEDURE add_emp(" +
        "IN fname VARCHAR(20), IN lname VARCHAR(20), IN bday DATETIME, OUT empno INT UNSIGNED) " +
        "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, fi
        cmd.ExecuteNonQuery()
        cmd.CommandText = "CREATE PROCEDURE add_emp(" & "IN fname VARCHAR(20), IN lname VARCHAR(20), IN
        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

```

6.10. 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.

6.10.1. 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.

6.10.2. 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

```
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.

6.10.3. 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.Information)
    myData.Close()
    conn.Close()

```



```

Catch ex As Exception
    MessageBox.Show("There was an error: " & ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.
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.

6.11. 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 a 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.MyExceptionHandler,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");
```

6.12. 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.

6.12.1. 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

6.12.2. 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.

6.12.3. 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](#).

6.12.4. 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
```

Please note that the use of 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.

6.12.5. 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.

6.13. 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
```

Note that with this test file the first three lines will need to be ignored, 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:

```
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.

6.14. 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 a [TraceSource](#) object first needs to be created. To create a [TraceSource](#) object in MySQL Connector/Net you would use code similar to the following:

```
TraceSource ts = new TraceSource("mysql");
```

To enable trace messages, configure a trace switch. There are three main switch classes, [BooleanSwitch](#), [SourceSwitch](#), and [TraceSwitch](#). Trace switches also have associated with them a trace level enumeration, these are [Off](#), [Error](#), [Warning](#), [Info](#), and [Verbose](#). The following code snippet illustrates creating a switch:

```
ts.Switch = new SourceSwitch("MySwitch", "Verbose");
```

This creates a [SourceSwitch](#), called [MySwitch](#), and 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 run time, 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.

The [TraceInformation\(\)](#) method simply prints a string passed as a parameter. The [TraceEvent\(\)](#) method, as well as the optional informational string, requires a [TraceEventType](#) value to be passed to indicate the trace message type, and also an application specific ID. The [TraceEventType](#) can have a value of [Verbose](#), [Information](#), [Warning](#), [Error](#), and [Critical](#). Using the [TraceData\(\)](#) method you can pass any object, for example an exception object, instead of a message.

To ensure that these generated trace messages get flushed from the trace source buffers to listeners, invoke the [Flush\(\)](#) method. When you are finished using a trace source, call the [Close\(\)](#) method. The [Close\(\)](#) method first calls [Flush\(\)](#), to ensure any remaining data is written out. It then frees up resources, and closes the listeners associated with the trace source.

```
ts.TraceInformation("Informational message");
ts.TraceEvent(TraceEventType.Error, 3, "Optional error message");
ts.TraceData(TraceEventType.Error, 3, ex); // pass exception object
ts.Flush();
...
ts.Close();
```

6.14.1. 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 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>
      <add name="SourceSwitch" value="Verbose" />
    </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 run time. 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 run time. 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.

6.14.2. 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
ConnectionClosed	No additional parameters
QueryOpened	mysql server thread id, query text
ResultOpened	field count, affected rows (-1 if select), inserted id (-1 if select)

MySQL-specific event type	Arguments (params Object[] args)
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.

6.15. 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`.

6.16. Character Set Considerations for Connector/Net

Treating Binary Blobs As UTF8

MySQL doesn't currently support 4-byte UTF8 sequences. This makes it difficult to represent some multi-byte languages such as Japanese. To try and alleviate this, Connector/Net now 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.

Currently, 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.

6.17. 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.

6.17.1. 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.

6.17.2. 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.

6.17.3. 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 6.17.1, "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 is 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 then 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`.`Populati
FROM `world`.`country` `country` LEFT OUTER JOIN `world`.`city` `city` ON `country`.`Code`=`city`.`Coun
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
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

```
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.

6.18. ASP.NET Provider Model

MySQL Connector/Net provides 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, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
    </providers>
  </membership>
</system.web>
```



```

    <add name="MySQLMembershipProvider" type="MySQL.Web.Security.MySQLMembershipProvider, MySQL.Web, Version=6.1.0.0, Culture=neutral, PublicKeyToken=8659616e1e949dab" />
  </providers>
</membership>
<profile>
  <providers>
    <add name="AspNetSqlProfileProvider" connectionStringName="LocalSqlServer" applicationName="/" type="System.Web.Profile.SqlProfileProvider, System.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f567f5a491343" />
    <add name="MySQLProfileProvider" type="MySQL.Web.Profile.MySQLProfileProvider, MySQL.Web, Version=6.1.0.0, Culture=neutral, PublicKeyToken=8659616e1e949dab" />
  </providers>
</profile>
<roleManager>
  <providers>
    <add name="AspNetSqlRoleProvider" connectionStringName="LocalSqlServer" applicationName="/" type="System.Web.Security.RoleProvider, System.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f567f5a491343" />
    <add name="AspNetWindowsTokenRoleProvider" applicationName="/" type="System.Web.Security.WindowsTokenRoleProvider, System.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f567f5a491343" />
    <add name="MySQLRoleProvider" type="MySQL.Web.Security.MySQLRoleProvider, MySQL.Web, Version=6.1.0.0, Culture=neutral, PublicKeyToken=8659616e1e949dab" />
  </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=6.1.0.0, Culture=neutral, PublicKeyToken=8659616e1e949dab" />
  </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">
    ...
  </compilation>
</system.web>

```

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 the database to connect to must be specified.

Rather than manually editing configuration files, consider using the MySQL Website Configuration tool to configure your desired provider setup. From MySQL Connector/Net 6.1.1 onwards, all providers can be selected and configured from this wizard. The tool modifies your `website.config` file to the desired configuration. A tutorial on doing this is available in the following section [Section 4.12, “MySQL Website Configuration Tool”](#).

A tutorial demonstrating how to use the Membership and Role Providers can be found in the following section [Section 5.2, “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.

6.19. 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.

6.19.1. 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 6.19.2, “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 6.19.3, “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.

6.19.2. 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.MySQLClientPermission" />
          </SecurityClasses>
        </PolicyLevel>
      </security>
    </mscorlib>
  </configuration>
```

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.

6.19.3. 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 7. Connector/Net Connection String Options Reference

For usage information about connection strings, see [Section 6.2, “Creating a Connector/Net Connection String”](#). The first table lists 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 7.1. Connector/Net Connection String Options - General

Name	Default	Description
<code>Allow Batch, AllowBatch</code>	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 ';'.
<code>Allow User Variables, AllowUserVariables</code>	false	Setting this to <code>true</code> indicates that the provider expects user variables in the SQL. This option was added in Connector/Net version 5.2.2.
<code>Allow Zero Datetime, AllowZeroDateTime</code>	false	If set to <code>True</code> , <code>MySqlDataReader.GetValue()</code> returns a <code>MySqlDateTime</code> object for date or datetime columns that have disallowed values, such as zero datetime values, and a <code>System.DateTime</code> object for valid values. If set to <code>False</code> (the default setting) it causes a <code>System.DateTime</code> object to be returned for all valid values and an exception to be thrown for disallowed values, such as zero datetime values.
<code>Auto Enlist, AutoEnlist</code>	true	If <code>AutoEnlist</code> is set to <code>true</code> , which is the default, a connection opened using <code>TransactionScope</code> participates in this scope, it commits when the scope commits and rolls back if <code>TransactionScope</code> does not commit. However, this feature is considered security sensitive and therefore cannot be used in a medium trust environment.
<code>BlobAsUTF8ExcludePattern</code>	null	A POSIX-style regular expression that matches the names of BLOB columns that do not contain UTF-8 character data. See Section 6.16, “Character Set Considerations for Connector/Net” for usage details.
<code>BlobAsUTF8IncludePattern</code>	null	A POSIX-style regular expression that matches the names of BLOB columns containing UTF-8 character data. See Section 6.16, “Character Set Considerations for Connector/Net” for usage details.
<code>Certificate File, CertificateFile</code>	null	This option specifies the path to a certificate file in PKCS #12 format (<code>.pfx</code>). For an example of usage, see Section 5.7, “Tutorial: Using SSL with MySQL Connector/Net” . Was introduced with 6.2.1.
<code>Certificate Password, CertificatePassword</code>	null	Specifies a password that is used in conjunction with a certificate specified using the option <code>CertificateFile</code> . For an example of usage, see Section 5.7, “Tutorial: Using SSL with MySQL Connector/Net” . Was introduced with 6.2.1.

Name	Default	Description
<code>Certificate Store Location</code> , <code>CertificateStoreLocation</code>	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 5.7 , “ Tutorial: Using SSL with MySQL Connector/Net ”. Was introduced with 6.2.1.
<code>Certificate Thumbprint</code> , <code>CertificateThumbprint</code>	null	Specifies a certificate thumbprint to ensure correct identification of a certificate contained within a personal store. For an example of usage, see Section 5.7 , “ Tutorial: Using SSL with MySQL Connector/Net ”. Was introduced with 6.2.1.
<code>CharSet</code> , <code>Character Set</code> , <code>CharacterSet</code>		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.
<code>Check Parameters</code> , <code>CheckParameters</code>	true	Indicates if stored routine parameters should be checked against the server.
<code>Command Interceptors</code> , <code>CommandInterceptors</code>		The list of interceptors that can intercept SQL command operations.
<code>Connect Timeout</code> , <code>Connection Timeout</code> , <code>ConnectionTimeout</code>	15	The length of time (in seconds) to wait for a connection to the server before terminating the attempt and generating an error.
<code>Convert Zero Datetime</code> , <code>ConvertZeroDateTime</code>	false	True to have <code>MySqlDataReader.GetValue()</code> and <code>MySqlDataReader.GetDateTime()</code> return <code>DateTime.MinValue</code> for date or datetime columns that have disallowed values.
<code>Default Command Timeout</code> , <code>DefaultCommandTimeout</code>	30	Sets the default value of the command timeout to be used. This does not supercede 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
<code>Default Table Cache Age</code> , <code>DefaultTableCacheAge</code>	60	Specifies how long a <code>TableDirect</code> result should be cached, in seconds. For usage information about table caching, see Section 6.7 , “ Using Connector/Net with Table Caching ”. This option was added in Connector/Net 6.4.
<code>enableSessionExpireCallback</code>	false	When set to <code>true</code> , causes the session-expiry scanner to raise the <code>session_end</code> event before deleting the session data stored in the <code>my_aspnet_sessions</code> table, when a session times out. Enable this option to write additional application-specific cleanup code to handle the <code>session_end</code> event of the <code>global.asax</code> class, before the stored data of the session gets deleted. Within the <code>session_end</code> 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.
<code>Encrypt</code> , <code>UseSSL</code>	false	For Connector/Net 5.0.3 and later, when <code>true</code> , SSL encryption is used for all data sent between the client and server if the server has a certificate installed. Recognized values are <code>true</code> , <code>false</code> , <code>yes</code> , and <code>no</code> . In versions before 5.0.3, this option had no effect. From version 6.2.1, this option is deprecated and is replaced by <code>SSL Mode</code> . The option still works if used. If this

Name	Default	Description
		option is set to true, it is equivalent to <code>SSL Mode = Preferred</code> .
<code>Exception Interceptors</code> , <code>ExceptionInterceptors</code>		The list of interceptors that can triage thrown <code>MySQLException</code> exceptions.
<code>Functions Return String</code> , <code>FunctionsReturnString</code>	false	Causes the connector to return binary/varbinary values as strings, if they do not have a tablename in the metadata.
<code>Host</code> , <code>Server</code> , <code>Data Source</code> , <code>DataSource</code> , <code>Address</code> , <code>Addr</code> , <code>Network Address</code>	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.
<code>Ignore Prepare</code> , <code>IgnorePrepare</code>	true	When true, instructs the provider to ignore any calls to <code>MySQLCommand.Prepare()</code> . 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.
<code>includesecurityasserts</code> , <code>include security asserts</code>	false	Must be set to <code>true</code> when using the <code>MySQLClientPermissions</code> 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 6.19, "Working with Partial Trust / Medium Trust" for details.
<code>Initial Catalog</code> , <code>Database</code>	mysql	The case-sensitive name of the database to use initially.
<code>Interactive</code> , <code>Interactive Session</code> , <code>InteractiveSession</code>	false	If set to true, the client is interactive. An interactive client is one where the server variable <code>CLIENT_INTERACTIVE</code> is set. If an interactive client is set, the <code>wait_timeout</code> variable is set to the value of <code>interactive_timeout</code> . The client will then time out after this period of inactivity. For more details, see Server System Variables in the MySQL Reference Manual.
<code>Integrated Security</code> , <code>IntegratedSecurity</code>	no	Use Windows authentication when connecting to server. By default, it is turned off. To enable, specify a value of <code>yes</code> . (You can also use the value <code>sspi</code> as an alternative to <code>yes</code> .) For details, see Section 6.5, "Using the Windows Native Authentication Plugin" . This option was introduced in Connector/Net 6.4.4.

Name	Default	Description
<code>Keep Alive</code> , <code>Keepalive</code>	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.
<code>Logging</code>	false	When true, various pieces of information is output to any configured TraceListeners. See Section 6.14, "Using the MySQL Connector/Net Trace Source Object" for further details.
<code>Old Guids</code> , <code>OldGuids</code>	false	This option was introduced in Connector/Net 6.1.1. The backend representation of a GUID type was changed from <code>BINARY(16)</code> to <code>CHAR(36)</code> . This was done to allow developers to use the server function <code>UUID()</code> to populate a GUID table - <code>UUID()</code> generates a 36-character string. Developers of older applications can add ' <code>Old Guids=true</code> ' to the connection string to use a GUID of data type <code>BINARY(16)</code> .
<code>Old Syntax</code> , <code>OldSyntax</code> , <code>Use Old Syntax</code> , <code>UseOldSyntax</code>	false	This option was deprecated in Connector/Net 5.2.2. All code should now be written using the '@' symbol as the parameter marker.
<code>Password</code> , <code>pwd</code>		The password for the MySQL account being used.
<code>Persist Security Info</code> , <code>PersistSecurityInfo</code>	false	When set to <code>false</code> or <code>no</code> (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 <code>true</code> , <code>false</code> , <code>yes</code> , and <code>no</code> .
<code>Pipe Name</code> , <code>Pipe</code> , <code>PipeName</code>	mysql	When set to the name of a named pipe, the <code>MySQLConnection</code> attempts to connect to MySQL on that named pipe. This setting only applies to the Windows platform.
<code>Port</code>	3306	The port MySQL is using to listen for connections. This value is ignored if Unix socket is used.
<code>Procedure Cache Size</code> , <code>ProcedureCacheSize</code> , <code>procedure cache</code> , <code>procedurecache</code>	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 (0). This option was added in Connector/Net 5.0.2 and Connector/Net 1.0.9.
<code>Protocol</code> , <code>Connection Protocol</code> , <code>ConnectionProtocol</code>	socket	Specifies the type of connection to make to the server. Values can be: <code>socket</code> or <code>tcp</code> for a socket connection, <code>pipe</code> for a named pipe connection, <code>unix</code> for a Unix socket connection, <code>memory</code> to use MySQL shared memory.
<code>Replication</code>	false	Indicates if this connection is to use replicated servers.
<code>Respect Binary Flags</code> , <code>RespectBinaryFlags</code>	true	Setting this option to <code>false</code> 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.
<code>Shared Memory Name</code> , <code>SharedMemoryName</code>	MYSQL	The name of the shared memory object to use for communication if the connection protocol is set to <code>memory</code> .

Name	Default	Description
Sql Server Mode, sqlservermode	false	Allow SQL Server syntax. When set to <code>true</code> , enables Connector/Net to support square brackets around symbols instead of backticks. This enables Visual Studio wizards that bracket symbols with <code>[]</code> 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	<p>This option has the following values:</p> <ul style="list-style-type: none"> • 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. <p>This option was introduced in MySQL Connector/Net 6.2.1.</p>
Table Cache, tablecache, tablecaching	false	Enables or disables caching of <code>TableDirect</code> commands. A value of <code>true</code> enables the cache while <code>false</code> disables it. For usage information about table caching, see Section 6.7, "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 <code>false</code> causes <code>TINYINT(1)</code> to be treated as an <code>INT</code> . See Numeric Type Overview for a further explanation of the <code>TINYINT</code> and <code>BOOL</code> data types.
Use Affected Rows, UseAffectedRows	false	When <code>true</code> , 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 <code>true</code> , 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 <code>false</code> when the user connecting to the database does not have the <code>SELECT</code> privileges for the <code>mysql.proc</code> (stored procedures) table, or cannot view <code>INFORMATION_SCHEMA.ROUTINES</code> . 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 <code>false</code> . When set to <code>false</code> , 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

Name	Default	Description
		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	<p>Setting this option to <code>true</code> enables compression of packets exchanged between the client and the server. This exchange is defined by the MySQL client/server protocol.</p> <p>Compression is used if both client and server support ZLIB compression, and the client has requested compression using this option.</p> <p>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.</p>
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 6.4, "Using Connector/Net with Connection Pooling"](#).

Table 7.2. Connector/Net Connection String Options - Connection Pooling

Name	Default	Description
Cache Server Properties, CacheServerProperties	false	Specifies whether server variable settings are updated by a <code>SHOW VARIABLES</code> 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 <code>Connection Lifetime</code> . 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

Name	Default	Description
		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	100	The maximum number of connections allowed in the pool.
Minimum Pool Size, Min Pool Size, MinimumPoolSize	0	The minimum number of connections allowed in the pool.
Pooling	true	When <code>true</code> , the <code>MySQLConnection</code> object is drawn from the appropriate pool, or if necessary, is created and added to the appropriate pool. Recognized values are <code>true</code> , <code>false</code> , <code>yes</code> , and <code>no</code> .

Chapter 8. Connector/Net API Reference

Table of Contents

8.1. MySql.Data.MySqlClient Namespace	135
8.1.1. MySql.Data.MySqlClientHierarchy	136
8.1.2. BaseCommandInterceptor	136
8.1.3. BaseExceptionInterceptor	137
8.1.4. MySqlCommand Class	137
8.1.5. MySqlCommandBuilder Class	199
8.1.6. MySqlException Class	216
8.1.7. MySqlHelper Class	217
8.1.8. MySqlErrorCode Enumeration	227
8.2. MySql.Data.Types Namespace	228
8.2.1. MySql.Data.TypesHierarchy	228
8.2.2. MySqlConversionException Class	228
8.2.3. MySqlDateTime Class	230

This section of the manual contains a complete reference to the Connector/Net ADO.NET component, automatically generated from the embedded documentation.

8.1. [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 6.19, “Working with Partial Trust / Medium Trust” .
MySqlCommand	
MySqlCommandBuilder	
MySqlConnection	
MySqlDataAdapter	
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.

Class	Description
MySqlInfoMessageEventArgs	Provides data for the InfoMessage event. This class cannot be inherited.
MySqlParameter	Represents a parameter to a MySqlCommand , and optionally, its mapping to DataSet columns. 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	

Delegates

Delegate	Description
MySqlInfoMessageEventHandler	Represents the method that will handle the InfoMessage event of a MySqlConnection .
MySqlRowUpdatedEventHandler	Represents the method that will handle the RowUpdated event of a MySqlDataAdapter .
MySqlRowUpdatingEventHandler	Represents the method that will handle the RowUpdating event of a MySqlDataAdapter .

Enumerations

Enumeration	Description
MySqlDbType	Specifies MySQL-specific data type of a field, property, for use in a MySqlParameter .
MySqlErrorCode	

8.1.1. [MySql.Data.MySqlClientHierarchy](#)

See Also

[MySql.Data.MySqlClient Namespace](#)

8.1.2. [BaseCommandInterceptor](#)

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 6.11, "Using the Connector/Net Interceptor Classes"](#).

8.1.3. BaseExceptionInterceptor

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 6.11, "Using the Connector/Net Interceptor Classes"](#).

8.1.4. 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](#)

8.1.4.1. 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](#)

8.1.4.1.1. [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](#)

8.1.4.1.1.1. [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](#)

8.1.4.1.1.2. [MySQLCommand](#) Constructor (String)

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal cmdText As String _  
)
```

Syntax: C#

```
public MySQLCommand(  
stringcmdText  
);
```

See Also

[MySQLCommand Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLCommand Constructor Overload List](#)

8.1.4.1.1.3. [MySQLCommand](#) Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal cmdText As String, _  
    ByVal connection As MySqlConnection _  
)
```

Syntax: C#

```
public MySQLCommand(  
stringcmdText,  
MySqlConnectionconnection  
);
```

See Also

[MySQLCommand Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLCommand Constructor Overload List](#)

8.1.4.1.1.3.1. [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](#)

8.1.4.1.1.3.1.1. [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](#)

8.1.4.1.1.3.1.1.1. [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](#)

8.1.4.1.1.3.1.1.1.1. [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](#)

8.1.4.1.1.3.1.1.2. MySQLConnection Constructor

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](#)

8.1.4.1.1.3.1.1.2. 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](#)

8.1.4.1.1.3.1.1.3. 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](#)

8.1.4.1.1.3.1.1.4. 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](#)

8.1.4.1.1.3.1.1.5. 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](#)

8.1.4.1.1.3.1.1.6. ServerThread Property

Returns the ID of the server thread this connection is executing on

Syntax: Visual Basic

```
Public ReadOnly Property ServerThread As Integer
```

Syntax: C#

```
public int ServerThread {get;}
```

See Also

[MySqlConnection Class](#), [MySql.Data.MySqlClient Namespace](#)

8.1.4.1.1.3.1.1.7. 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](#)

8.1.4.1.1.3.1.1.8. 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](#)

8.1.4.1.1.3.1.1.9. 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](#)

8.1.4.1.1.3.1.1.10. BeginTransaction Method

Overload List

- [public MySqlTransaction BeginTransaction\(\);](#)
- [public MySqlTransaction BeginTransaction\(IsolationLevel\);](#)

See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.1.3.1.1.10.1. 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](#)

8.1.4.1.1.3.1.1.10.1.1. 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.MySqlCommand](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySQLTransaction Members](#), [MySQL.Data.MySqlCommand Namespace](#)

8.1.4.1.1.3.1.1.10.1.1.1. 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

[MySQLTransaction Class](#), [MySQL.Data.MySqlCommand Namespace](#)

8.1.4.1.1.3.1.1.10.1.1.1.1. 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

[MySQLTransaction Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.1.3.1.1.10.1.1.1.2. 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 Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.1.3.1.1.10.1.1.1.3. MySQLTransaction.Commit Method

Syntax: Visual Basic

```
NotOverridable Public Sub Commit() _  
- Implements IDbTransaction.Commit
```

Syntax: C#

```
public void Commit();
```

Implements

IDbTransaction.Commit

See Also

[MySQLTransaction Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.1.3.1.1.10.1.1.4. [MySQLTransaction.Rollback](#) Method

Syntax: Visual Basic

```
NotOverridable Public Sub Rollback() _
-
    Implements IDbTransaction.Rollback
```

Syntax: C#

```
public void Rollback();
```

Implements

IDbTransaction.Rollback

See Also

[MySQLTransaction Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.1.3.1.1.10.2. [MySQLConnection.BeginTransaction](#) Method

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](#)

8.1.4.1.1.3.1.1.11. [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](#)

8.1.4.1.1.3.1.1.12. [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](#)

8.1.4.1.1.3.1.1.13. [MySQLConnection.CreateCommand](#) Method

Syntax: Visual Basic

```
Public Function CreateCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand CreateCommand();
```

See Also

[MySQLConnection Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.1.3.1.1.14. [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](#)

8.1.4.1.1.3.1.1.15. [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](#)

8.1.4.1.1.3.1.1.16. [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](#)

8.1.4.1.1.3.1.1.16.1. [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](#)

8.1.4.1.1.3.1.1.16.1.1. [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](#)

8.1.4.1.1.3.1.1.16.1.1.1.1. [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](#)

8.1.4.1.1.3.1.1.16.1.1.1.1.1. [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](#)

8.1.4.1.1.3.1.1.16.1.1.1.2. [MySQLInfoMessageEventArgs.errors](#) Field

Syntax: Visual Basic

```
Public errors As MySQLError()
```

Syntax: C#

```
public MySQLError[] errors;
```

See Also

[MySQLInfoMessageEventArgs Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.1.3.1.1.16.1.1.1.2.1. [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](#)

8.1.4.1.1.3.1.1.16.1.1.1.2.1.1. [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 Class](#), [MySql.Data.MySqlClient Namespace](#)

8.1.4.1.1.3.1.1.16.1.1.1.2.1.1.1. MySqlCommand Constructor

Syntax: Visual Basic

```
Public Sub New( _
    ByVal level As String, _
    ByVal code As Integer, _
    ByVal message As String _
)
```

Syntax: C#

```
public MySqlCommand(
    string level,
    int code,
    string message
);
```

Parameters

- **level:**
- **code:**
- **message:**

See Also

[MySqlError Class](#), [MySql.Data.MySqlClient Namespace](#)

8.1.4.1.1.3.1.1.16.1.1.1.2.1.1.2. Code Property

Error code

Syntax: Visual Basic

```
Public ReadOnly Property Code As Integer
```

Syntax: C#

```
public int Code {get;}
```


See Also

[MySqlError Class](#), [MySql.Data.MySqlClient Namespace](#)

8.1.4.1.1.3.1.1.16.1.1.1.2.1.1.3. 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](#)

8.1.4.1.1.3.1.1.16.1.1.1.2.1.1.4. 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](#)

8.1.4.1.1.3.1.1.17. [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](#)

8.1.4.1.1.4. [MySQLCommand](#) Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal cmdText As String, _  
    ByVal connection As MySqlConnection, _  
    ByVal transaction As MySqlTransaction _  
)
```

Syntax: C#

```
public MySqlCommand(  
    stringcmdText,  
    MySqlConnectionconnection,  
    MySqlTransactiontransaction
```

```
);
```

See Also

[MySQLCommand Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLCommand Constructor Overload List](#)

8.1.4.1.2. 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](#)

8.1.4.1.3. 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](#)

8.1.4.1.4. 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](#)

8.1.4.1.5. 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](#)

8.1.4.1.6. 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](#)

8.1.4.1.7. 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](#)

8.1.4.1.7.1. [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, IC
```

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](#)

8.1.4.1.7.1.1. [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](#)

8.1.4.1.7.1.1.1. [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](#)

8.1.4.1.7.1.1.2. 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](#)

8.1.4.1.7.1.1.3. 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

8.1.4.1.7.1.1.3.1. 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](#)

8.1.4.1.7.1.1.3.1.1. 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 DataSet and 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](#)

8.1.4.1.7.1.1.3.1.1.1. [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](#)

8.1.4.1.7.1.1.3.1.1.1.1. [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](#)

8.1.4.1.7.1.1.3.1.1.1.2. [MySQLParameter](#) Constructor

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(  
stringparameterName,  
MySQLDbTypedbType  
);
```

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](#)

8.1.4.1.7.1.1.3.1.1.1.2.1. [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](#)

8.1.4.1.7.1.1.3.1.1.3. [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(
    stringparameterName,
    MySqlDbTypeDbType,
    intsize
);
```

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](#)

8.1.4.1.7.1.1.3.1.1.4. [MySqlParameter](#) Constructor

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(
    stringparameterName,
    MySqlDbTypeDbType,
    intsize,
    ParameterDirectiondirection,
    boolisNullable,
    byteprecision,
    bytescale,
    stringsourceColumn,
    DataRowVersionsourceVersion,
    objectvalue
);
```

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 ParameterDirection values.
- **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 DataRowVersion values.
- **value**: An Object that is the value of the **MySQLParameter**.

Exceptions

Exception Type	Condition
ArgumentException	

See Also

[MySQLParameter Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLParameter Constructor Overload List](#)

8.1.4.1.7.1.1.3.1.1.4.1. 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](#)

8.1.4.1.7.1.1.3.1.1.5. MySQLParameter Constructor

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 Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySqlParameter Constructor Overload List](#)

8.1.4.1.7.1.1.3.1.1.6. [MySqlParameter](#) Constructor

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](#)

8.1.4.1.7.1.1.3.1.1.2. 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](#)

8.1.4.1.7.1.1.3.1.1.3. 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](#)

8.1.4.1.7.1.1.3.1.1.4. 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](#)

8.1.4.1.7.1.1.3.1.1.5. 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](#)

8.1.4.1.7.1.1.3.1.1.6. 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](#)

8.1.4.1.7.1.1.3.1.1.7. 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](#)

8.1.4.1.7.1.1.3.1.1.8. 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](#)

8.1.4.1.7.1.1.3.1.1.9. 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](#)

8.1.4.1.7.1.1.3.1.1.10. 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](#)

8.1.4.1.7.1.1.3.1.1.11. SourceColumn Property

Gets or sets the name of the source column that is mapped to the DataSet and 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](#)

8.1.4.1.7.1.1.3.1.1.12. 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](#)

8.1.4.1.7.1.1.3.1.13. [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](#)

8.1.4.1.7.1.1.3.2. 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](#)

8.1.4.1.7.1.1.3.3. 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[
    stringname
] {get; set;}
```

See Also

[MySqlParameterCollection Class](#), [MySql.Data.MySqlClient Namespace](#),
[MySqlParameterCollection.Item Overload List](#)

8.1.4.1.7.1.1.4. 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](#)

8.1.4.1.7.1.1.4.1. [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(
    MySqlParametervalue
);
```

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](#)

8.1.4.1.7.1.1.4.2. [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(
    objectvalue
);
```

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](#)

8.1.4.1.7.1.1.4.3. [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(
    stringparameterName,
    MySqlDbTypedbType
);
```

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](#)

8.1.4.1.7.1.1.4.4. [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 Class](#), [MySQL.Data.MySqlClient Namespace](#),
[MySQLParameterCollection.Add Overload List](#)

8.1.4.1.7.1.1.4.5. [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 Class](#), [MySql.Data.MySqlClient Namespace](#),
[MySqlParameterCollection.Add Overload List](#)

8.1.4.1.7.1.1.4.6. [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](#)

8.1.4.1.7.1.1.5. [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](#)

8.1.4.1.7.1.1.6. 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](#)

8.1.4.1.7.1.1.6.1. 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(
    objectvalue
);
```

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](#)

8.1.4.1.7.1.1.6.2. [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(
    stringname
);
```

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](#)

8.1.4.1.7.1.1.7. [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(
    Arrayarray,
    intindex
);
```

Parameters

- [array](#):
- [index](#):

Implements

ICollection.CopyTo

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.7.1.1.8. 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](#)

8.1.4.1.7.1.1.8.1. [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(
    objectvalue
);
```

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](#)

8.1.4.1.7.1.1.8.2. [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](#)

8.1.4.1.7.1.1.9. [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](#)

8.1.4.1.7.1.1.10. [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(
    objectvalue
);
```

Parameters

- **value:**

Implements

IList.Remove

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.7.1.1.11. 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](#)

8.1.4.1.7.1.1.11.1. 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(
    intindex
);
```

Parameters

- **index:** The zero-based index of the parameter.

Implements

IList.RemoveAt

See Also

[MySQLParameterCollection Class](#), [MySQL.Data.MySqlClient Namespace](#),
[MySQLParameterCollection.RemoveAt Overload List](#)

8.1.4.1.7.1.1.11.2. [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(
    stringname
);
```

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](#)

8.1.4.1.8. 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](#)

8.1.4.1.9. 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](#)

8.1.4.1.10. [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](#)

8.1.4.1.11. [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](#)

8.1.4.1.12. [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](#)

8.1.4.1.13. ExecuteReader Method

Overload List

- [public MySqlDataReader ExecuteReader\(\);](#)
- [public MySqlDataReader ExecuteReader\(CommandBehavior\);](#)

See Also

[MySQLCommand Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.13.1. 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](#)

8.1.4.1.13.1.1. 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

[MySQLDataReader Members](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.13.1.1.1. [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](#)

8.1.4.1.13.1.1.1. 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](#)

8.1.4.1.13.1.1.1.2. 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](#)

8.1.4.1.13.1.1.1.3. 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](#)

8.1.4.1.13.1.1.1.4. 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](#)

8.1.4.1.13.1.1.1.5. 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](#)

8.1.4.1.13.1.1.1.5.1. 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[
    inti
] {get;}
```

Implements

IDataRecord.Item

See Also

[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLDataReader.Item Overload List](#)

8.1.4.1.13.1.1.1.5.2. 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[
    stringname
] {get;}
```

Implements

IDataRecord.Item

See Also

[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLDataReader.Item Overload List](#)

8.1.4.1.13.1.1.1.6. 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](#)

8.1.4.1.13.1.1.1.7. [MySQLDataReader.Close](#) Method

Closes the MySQLDataReader object.

Syntax: Visual Basic

```
NotOverridable Public Sub Close() _
- Implements IDataReader.Close
```

Syntax: C#

```
public void Close();
```

Implements

IDataReader.Close

See Also

[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.13.1.1.1.8. [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(
    inti
);
```

Parameters

- **i:**

Return Value

Implements

IDataRecord.GetBoolean

See Also

[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.13.1.1.1.9. [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(
    inti
);
```

Parameters

- **i:**

Return Value

Implements

IDataRecord.GetByte

See Also

[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.13.1.1.1.10. [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(
    inti,
    longdataIndex,
    byte[]buffer,
    intbufferIndex,
    intlength
);
```

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 Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.13.1.1.11. [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(
    inti
);
```

Parameters

- **i**:

Return Value

Implements

IDataRecord.GetChar

See Also

[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.13.1.1.1.12. [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(
    inti,
    longfieldOffset,
    char[]buffer,
    intbufferoffset,
    intlength
);
```

Parameters

- **i**:
- **fieldOffset**:
- **buffer**:
- **bufferoffset**:
- **length**:

Return Value

Implements

IDataRecord.GetChars

See Also

[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.13.1.1.1.13. [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](#)

8.1.4.1.13.1.1.14. [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](#)

8.1.4.1.13.1.1.15. [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](#)

8.1.4.1.13.1.1.1.16. [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](#)

8.1.4.1.13.1.1.1.17. [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](#)

8.1.4.1.13.1.1.1.18. [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](#)

8.1.4.1.13.1.1.19. [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](#)

8.1.4.1.13.1.1.20. [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 Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.13.1.1.21. [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 Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.13.1.1.1.22. [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 Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.13.1.1.1.23. [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](#)

8.1.4.1.13.1.1.1.24. [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(
    inti
);
```

Parameters

- `i`:

Return Value

Implements

IDataRecord.GetName

See Also

[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.13.1.1.1.25. [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(
    stringname
);
```

Parameters

- `name`:

Return Value

Implements

IDataRecord.GetOrdinal

See Also

[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.13.1.1.1.26. [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](#)

8.1.4.1.13.1.1.1.27. [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](#)

8.1.4.1.13.1.1.1.28. [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](#)

8.1.4.1.13.1.1.1.29. [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](#)

8.1.4.1.13.1.1.1.30. [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](#)

8.1.4.1.13.1.1.1.31. [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](#)

8.1.4.1.13.1.1.1.32. [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](#)

8.1.4.1.13.1.1.1.33. [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](#)

8.1.4.1.13.1.1.1.34. [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(
    inti
);
```

Parameters

- **i**:

Return Value

Implements

IDataRecord.IsDBNull

See Also

[MySQLDataReader Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.4.1.13.1.1.1.35. [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](#)

8.1.4.1.13.1.1.1.36. [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](#)

8.1.4.1.13.2. [MySQLCommand.ExecuteReader](#) Method

Syntax: Visual Basic

```
Overloads Public Function ExecuteReader( _
    ByVal behavior As CommandBehavior _
) As MySQLDataReader
```

Syntax: C#

```
public MySQLDataReader ExecuteReader(
    CommandBehaviorbehavior
);
```

See Also

[MySQLCommand Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLCommand.ExecuteReader Overload List](#)

8.1.4.1.14. [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](#)

8.1.4.1.15. [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](#)

8.1.5. [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.MySqlCommand](#)

Assembly: MySQL.Data (in MySQL.Data.dll)

See Also

[MySQLCommandBuilder Members](#), [MySQL.Data.MySqlCommand Namespace](#)

8.1.5.1. 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](#)

8.1.5.1.1. 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](#)

8.1.5.1.1.1. 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](#)

8.1.5.1.1.2. 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](#)

8.1.5.1.2. 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](#)

8.1.5.1.2.1. 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](#)

8.1.5.1.2.2. MySqlCommandBuilder Constructor

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](#)

8.1.5.1.2.2.1. 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](#)

8.1.5.1.2.2.1.1. 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 AcceptChanges is called on a DataRow after it is added to the DataTable during any of the Fill operations.
--	--

AcceptChangesDuringUpdate (inherited from DataAdapter)	Gets or sets whether AcceptChanges is 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 DataTable from 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 DataSet schema does not match incoming data.
ReturnProviderSpecificTypes (inherited from DataAdapter)	Gets or sets whether the Fill method 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 DataSet to match those in the data source using the DataSet name, and creates a DataTable named "Table."
FillSchema (inherited from DbDataAdapter)	Overloaded. Configures the schema of the specified DataTable based on the specified SchemaType.
GetFillParameters (inherited from DbDataAdapter)	Gets the parameters set by the user when executing an SQL <code>SELECT</code> 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 FillLoadOption to its default state and causes Fill to honor AcceptChangesDuringFill.
ShouldSerializeAcceptChangesDuringFill (inherited from DataAdapter)	Determines whether the AcceptChangesDuringFill property should be persisted.
ShouldSerializeFillLoadOption (inherited from DataAdapter)	Determines whether the FillLoadOption property 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](#)

8.1.5.1.2.2.1.1.1. [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](#)

8.1.5.1.2.2.1.1.1.1. [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](#)

8.1.5.1.2.2.1.1.1.2. [MySQLDataAdapter](#) Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal selectCommand As MySqlCommand _  
)
```

Syntax: C#

```
public MySQLDataAdapter(  
    MySqlCommandselectCommand  
) ;
```

See Also

[MySQLDataAdapter Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLDataAdapter Constructor Overload List](#)

8.1.5.1.2.2.1.1.1.3. [MySQLDataAdapter](#) Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal selectCommandText As String, _  
    ByVal connection As MySqlConnection _  
)
```

Syntax: C#

```
public MySQLDataAdapter(  
    stringselectCommandText,  
    MySqlConnectionconnection  
) ;
```

See Also

[MySQLDataAdapter Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLDataAdapter Constructor Overload List](#)

8.1.5.1.2.2.1.1.1.4. [MySQLDataAdapter](#) Constructor

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](#)

8.1.5.1.2.2.1.1.2. 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](#)

8.1.5.1.2.2.1.1.3. 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](#)

8.1.5.1.2.2.1.1.4. 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](#)

8.1.5.1.2.2.1.1.5. UpdateCommand Property

Syntax: Visual Basic

```
Overloads Public Property UpdateCommand As MySqlCommand
```

Syntax: C#

```
new public MySqlCommand UpdateCommand {get; set;}
```

See Also

[MySQLDataAdapter Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.5.1.2.2.1.1.6. [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 DataRow sent 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 DataTableMapping sent through an Update.

See Also

[MySQLDataAdapter Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.5.1.2.2.1.1.6.1. [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](#)

8.1.5.1.2.2.1.1.6.1.1. [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](#)

8.1.5.1.2.2.1.1.6.1.1.1. [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 DataRow sent 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](#)

8.1.5.1.2.2.1.1.6.1.1.1.1. 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 DataRow sent through an Update.
- **command**: The IDbCommand executed when Update is called.
- **statementType**: One of the StatementType values that specifies the type of query executed.
- **tableMapping**: The DataTableMapping sent through an Update.

See Also

[MySQLRowUpdatedEventArgs Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.5.1.2.2.1.1.6.1.1.2. 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](#)

8.1.5.1.2.2.1.1.7. MySqlCommandAdapter.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](#)

8.1.5.1.2.2.1.1.7.1. 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(  
objectsender,  
MySqlRowUpdatingEventArgs  
);
```

Requirements

Namespace: [MySql.Data.MySqlClient](#)

Assembly: MySql.Data (in MySql.Data.dll)

See Also

[MySql.Data.MySqlClient Namespace](#)

8.1.5.1.2.2.1.1.7.1.1. [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](#)

8.1.5.1.2.2.1.1.7.1.1.1. [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 Commandproperty.
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](#)

8.1.5.1.2.2.1.1.7.1.1.1.1.1. MySqlCommandBuilder Constructor

Initializes a new instance of the MySqlCommandBuilder 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 MySqlCommandBuilder(
    DataRow row,
    IDbCommand command,
    StatementType statementType,
    DataTableMapping tableMapping
);
```

Parameters

- **row**: The DataRow to Update.
- **command**: The IDbCommand to execute during Update.
- **statementType**: One of the StatementType values that specifies the type of query executed.
- **tableMapping**: The DataTableMapping sent through an Update.

See Also

[MySQLRowUpdatingEventArgs Class, MySql.Data.MySqlClient Namespace](#)

8.1.5.1.2.2.1.7.1.1.1.2. 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](#)

8.1.5.1.2.3. MySqlCommandBuilder Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal adapter As MySqlDataAdapter, _  
    ByVal lastOneWins As Boolean _  
)
```

Syntax: C#

```
public MySqlCommandBuilder(  
    MySqlDataAdapteradapter,  
    boollastOneWins  
);
```

See Also

[MySqlCommandBuilder Class, MySql.Data.MySqlClient Namespace](#), [MySqlCommandBuilder Constructor Overload List](#)

8.1.5.1.2.4. MySqlCommandBuilder Constructor

Syntax: Visual Basic

```
Overloads Public Sub New( _  
    ByVal lastOneWins As Boolean _  
)
```

Syntax: C#

```
public MySqlCommandBuilder(  
    boollastOneWins  
);
```

See Also

[MySqlCommandBuilder Class, MySql.Data.MySqlClient Namespace](#), [MySqlCommandBuilder Constructor Overload List](#)

8.1.5.1.3. 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](#)

8.1.5.1.4. QuotePrefix Property

Syntax: Visual Basic

```
Public Property QuotePrefix As String
```

Syntax: C#

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

See Also

[MySQLCommandBuilder Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.5.1.5. QuoteSuffix Property

Syntax: Visual Basic

```
Public Property QuoteSuffix As String
```

Syntax: C#

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

See Also

[MySQLCommandBuilder Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.5.1.6. [MySQLCommandBuilder.GetDeleteCommand](#) Method

Syntax: Visual Basic

```
Public Function GetDeleteCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand GetDeleteCommand();
```

See Also

[MySQLCommandBuilder Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.5.1.7. [MySQLCommandBuilder.GetInsertCommand](#) Method

Syntax: Visual Basic

```
Public Function GetInsertCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand GetInsertCommand();
```

See Also

[MySQLCommandBuilder Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.5.1.8. [MySQLCommandBuilder.GetUpdateCommand](#) Method

Syntax: Visual Basic

```
Public Function GetUpdateCommand() As MySqlCommand
```

Syntax: C#

```
public MySqlCommand GetUpdateCommand();
```

See Also

[MySqlCommandBuilder Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.5.1.9. MySqlCommandBuilder.RefreshSchema Method
Syntax: Visual Basic

```
Public Sub RefreshSchema()
```

Syntax: C#

```
public void RefreshSchema();
```

See Also

[MySqlCommandBuilder Class](#), [MySQL.Data.MySqlClient Namespace](#)

8.1.6. 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](#)

8.1.6.1. 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 Exception instance 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](#)

8.1.6.1.1. 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](#)

8.1.7. 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](#)

8.1.7.1. 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](#)

8.1.7.1.1. 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](#)

8.1.7.1.2. 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](#)

8.1.7.1.2.1. [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(
    MySqlConnectionconnection,
    stringcommandText
);
```

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](#)

8.1.7.1.2.2. [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(
    MySqlConnectionconnection,
    stringcommandText,
    params MySqlParameter[]commandParameters
);
```

Parameters

- `connection`: [MySQLConnection](#) object to use
- `commandText`: Command to execute
- `commandParameters`: Parameters to use for the command

Return Value

DataSet containing the resultset

See Also

[MySQLHelper Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLHelper.ExecuteDataset Overload List](#)

8.1.7.1.2.3. [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

DataSet containing the resultset

See Also

[MySQLHelper Class](#), [MySQL.Data.MySqlClient Namespace](#), [MySQLHelper.ExecuteDataset Overload List](#)

8.1.7.1.2.4. [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

DataSet containing the resultset

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#), [MySqlHelper.ExecuteDataset Overload List](#)

8.1.7.1.3. 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 Class](#), [MySql.Data.MySqlClient Namespace](#)

8.1.7.1.3.1. 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](#)

8.1.7.1.3.2. 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](#)

8.1.7.1.4. 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](#)

8.1.7.1.4.1. 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](#)

8.1.7.1.4.2. [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](#)

8.1.7.1.5. [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](#)

8.1.7.1.5.1. [MySQLHelper.ExecuteScalar](#) Method

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(
    MySqlConnectionconnection,
    stringcommandText
);
```

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](#)

8.1.7.1.5.2. [MySQLHelper.ExecuteScalar](#) Method

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](#)

8.1.7.1.5.3. MySQLHelper.ExecuteScalar Method

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](#)

8.1.7.1.5.4. MySQLHelper.ExecuteScalar Method

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](#)

8.1.7.1.6. **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**: DataSet containing the new data to use in the update
- **tablename**: Tablename in the data set to update

See Also

[MySqlHelper Class](#), [MySql.Data.MySqlClient Namespace](#)

8.1.8. **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)

See Also

[MySQL.Data.MySqlClient Namespace](#)

8.2. MySQL.Data.Types Namespace

[Namespace hierarchy](#)

Classes

Class	Description
MySQLConversionException	Summary description for MySQLConversionException.
MySQLDateTime	Summary description for MySQLDateTime.
MySQLValue	

8.2.1. MySQL.Data.TypesHierarchy

See Also

[MySQL.Data.Types Namespace](#)

8.2.2. 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](#)

8.2.2.1. 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](#)

8.2.2.1.1. MySqlConversionException Constructor

Syntax: Visual Basic

```
Public Sub New( _
    ByVal msg As String _
)
```

Syntax: C#

```
public MySqlConversionException(
    stringmsg
);
```

See Also

[MySqlConversionException Class](#), [MySql.Data.Types Namespace](#)

8.2.3. MySqlDateTime Class

Summary description for MySqlDateTime.

For a list of all members of this type, see [MySqlDateTime Members](#) .

Syntax: Visual Basic

```
Public Class MySqlDateTime_
    Inherits MySqlValue_
    Implements IConvertible, IComparable
```

Syntax: C#

```
public class MySqlDateTime : MySqlValue, 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](#)

8.2.3.1. 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 (inherited from MySQLValue)	
IsValidDateTime	Indicates if this object contains a value that can be represented as a 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
Millisecond	Returns the millisecond portion of this datetime
ValueAsObject (inherited from MySQLValue)	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 (inherited from MySQLValue)	The system type represented by this value
dbType (inherited from MySQLValue)	The generic dbtype of this value
isNull (inherited from MySQLValue)	Is this value null
mysqlDbType (inherited from MySQLValue)	The specific MySQL db type
mysqlTypeName (inherited from MySQLValue)	The MySQL-specific typename of this value
objectValue (inherited from MySQLValue)	

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 Class](#), [MySQL.Data.Types Namespace](#)

8.2.3.1.1. 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 Class](#), [MySQL.Data.Types Namespace](#)

8.2.3.1.2. Day Property

Returns the day portion of this datetime

Syntax: Visual Basic

```
Public Property Day As Integer
```

Syntax: C#

```
public int Day {get; set;}
```

See Also

[MySQLDateTime Class](#), [MySQL.Data.Types Namespace](#)

8.2.3.1.3. Hour Property

Returns the hour portion of this datetime

Syntax: Visual Basic

```
Public Property Hour As Integer
```

Syntax: C#

```
public int Hour {get; set;}
```

See Also

[MySQLDateTime Class](#), [MySQL.Data.Types Namespace](#)

8.2.3.1.4. IsNull Property

Syntax: Visual Basic

```
Public Property IsNull As Boolean
```

Syntax: C#

```
public bool IsNull {get; set;}
```

See Also

[MySQLValue Class](#), [MySQL.Data.Types Namespace](#)

8.2.3.1.4.1. MySQLValue Class

For a list of all members of this type, see [MySQLValue Members](#) .

Syntax: Visual Basic

```
MustInherit Public Class MySQLValue
```

Syntax: C#

```
public abstract class MySQLValue
```

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

[MySQLValue Members](#), [MySQL.Data.Types Namespace](#)

8.2.3.1.4.1.1. MySQLValue Members

[MySQLValue overview](#)

Protected Static (Shared) Fields

numberFormat	
------------------------------	--

Public Instance Constructors

MySQLValue Constructor	Initializes a new instance of the MySQLValue class.
--	---

Public Instance Properties

IsNull	
ValueAsObject	Returns the value of this field as an object

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	Returns a 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 typename 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

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

8.2.3.1.4.1.1.1. [MySqlValue.numberFormat](#) Field

Syntax: Visual Basic

```
Protected Shared numberFormat As NumberFormatInfo
```

Syntax: C#

```
protected static NumberFormatInfo numberFormat;
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

8.2.3.1.4.1.1.2. [MySqlValue](#) Constructor

Initializes a new instance of the [MySqlValue](#) class.

Syntax: Visual Basic

```
Public Sub New()
```

Syntax: C#

```
public MySqlValue();
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

8.2.3.1.4.1.1.3. ValueAsObject Property

Returns the value of this field as an object

Syntax: Visual Basic

```
Public ReadOnly Property ValueAsObject As Object
```

Syntax: C#

```
public object ValueAsObject {get;}
```

See Also

[MySQLValue Class](#), [MySQL.Data.Types Namespace](#)

8.2.3.1.4.1.1.4. [MySQLValue.ToString](#) Method

Returns a string representation of this value

Syntax: Visual Basic

```
Overrides Public Function ToString() As String
```

Syntax: C#

```
public override string ToString();
```

See Also

[MySQLValue Class](#), [MySQL.Data.Types Namespace](#)

8.2.3.1.4.1.1.5. [MySQLValue.classType](#) Field

The system type represented by this value

Syntax: Visual Basic

```
Protected classType As Type
```

Syntax: C#

```
protected Type classType;
```

See Also

[MySQLValue Class](#), [MySQL.Data.Types Namespace](#)

8.2.3.1.4.1.1.6. [MySQLValue.dbType](#) Field

The generic dbtype of this value

Syntax: Visual Basic

```
Protected dbType As DbType
```

Syntax: C#

```
protected DbType dbType;
```

See Also

[MySQLValue Class](#), [MySQL.Data.Types Namespace](#)

8.2.3.1.4.1.1.7. [MySQLValue.mysqlDbType](#) Field

The specific MySQL db type

Syntax: Visual Basic

```
Protected mySqlDbType As MySqlDbType
```

Syntax: C#

```
protected MySqlDbType mySqlDbType;
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

8.2.3.1.4.1.1.8. [MySqlValue.mySqlTypeName](#) Field

The MySQL-specific typename of this value

Syntax: Visual Basic

```
Protected mySqlTypeName As String
```

Syntax: C#

```
protected string mySqlTypeName;
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

8.2.3.1.4.1.1.9. [MySqlValue.objectValue](#) Field

Syntax: Visual Basic

```
Protected objectValue As Object
```

Syntax: C#

```
protected object objectValue;
```

See Also

[MySqlValue Class](#), [MySql.Data.Types Namespace](#)

8.2.3.1.5. IsValidDateTime Property

Indicates if this object contains a value that can be represented as a DateTime

Syntax: Visual Basic

```
Public ReadOnly Property IsValidDateTime As Boolean
```

Syntax: C#

```
public bool IsValidDateTime {get;}
```

See Also

[MySqlDateTime Class](#), [MySql.Data.Types Namespace](#)

8.2.3.1.6. Millisecond Property

Returns the millisecond portion of this datetime

Syntax: Visual Basic

```
Public Property Millisecond As Integer
```

Syntax: C#

```
public int Millisecond {get; set;}
```

See Also

[MySQLDateTime Class](#), [MySQL.Data.Types Namespace](#)

8.2.3.1.7. Minute Property

Returns the minute portion of this datetime

Syntax: Visual Basic

```
Public Property Minute As Integer
```

Syntax: C#

```
public int Minute {get; set;}
```

See Also

[MySQLDateTime Class](#), [MySQL.Data.Types Namespace](#)

8.2.3.1.8. Month Property

Returns the month portion of this datetime

Syntax: Visual Basic

```
Public Property Month As Integer
```

Syntax: C#

```
public int Month {get; set;}
```

See Also

[MySQLDateTime Class](#), [MySQL.Data.Types Namespace](#)

8.2.3.1.9. Second Property

Returns the second portion of this datetime

Syntax: Visual Basic

```
Public Property Second As Integer
```

Syntax: C#

```
public int Second {get; set;}
```

See Also

[MySQLDateTime Class](#), [MySQL.Data.Types Namespace](#)

8.2.3.1.10. Year Property

Returns the year portion of this datetime

Syntax: Visual Basic

```
Public Property Year As Integer
```

Syntax: C#

```
public int Year {get; set;}
```

See Also

[MySQLDateTime Class](#), [MySQL.Data.Types Namespace](#)

8.2.3.1.11. **MySQLDateTime.GetDateTime** Method

Returns this value as a DateTime

Syntax: Visual Basic

```
Public Function GetDateTime() As Date
```

Syntax: C#

```
public DateTime GetDateTime();
```

See Also

[MySQLDateTime Class](#), [MySQL.Data.Types Namespace](#)

8.2.3.1.12. **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 Class](#), [MySQL.Data.Types Namespace](#)

Chapter 9. Connector/Net Support

Table of Contents

9.1. Connector/Net Community Support	239
9.2. How to Report Connector/Net Problems or Bugs	239
9.3. Connector/Net Change History	239

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](#).

9.1. 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/>.

9.2. 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 9.1, “Connector/Net Community Support”](#).

First try to execute the same SQL statements and commands from the `mysql` client program or from `admindemo`. 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/>.

9.3. Connector/Net Change History

The Connector/Net Change History (Changelog) is located with the main Changelog for MySQL. See [Appendix A, MySQL Connector/Net Change History](#).

Appendix A. MySQL Connector/Net Change History

Table of Contents

A.1. Changes in MySQL Connector/Net Version 6.6	243
A.2. Changes in MySQL Connector/Net Version 6.5	245
A.3. Changes in MySQL Connector/Net Version 6.4	247
A.3.1. Changes in MySQL Connector/Net 6.4.6 (2012-11-26)	247
A.3.2. Changes in MySQL Connector/Net 6.4.5 (2012-05-19)	249
A.3.3. Changes in MySQL Connector/Net 6.4.4 (2011-09-26)	250
A.3.4. Changes in MySQL Connector/Net 6.4.3 (2011-07-03)	251
A.3.5. Changes in MySQL Connector/Net 6.4.2 (2011-06-29)	251
A.3.6. Changes in MySQL Connector/Net 6.4.1 (2011-06-06, Alpha)	251
A.3.7. Changes in MySQL Connector/Net 6.4.0 (Unknown)	251
A.4. Changes in MySQL Connector/Net Version 6.3	251
A.4.1. Changes in MySQL Connector/Net 6.3.9 (2012-04-11)	251
A.4.2. Changes in MySQL Connector/Net 6.3.8 (2011-12-16)	253
A.4.3. Changes in MySQL Connector/Net 6.3.7 (2011-06-22)	255
A.4.4. Changes in MySQL Connector/Net 6.3.6 (2011-01-03)	255
A.4.5. Changes in MySQL Connector/Net 6.3.5 (2010-10-12)	257
A.4.6. Changes in MySQL Connector/Net 6.3.4 (2010-09-01, Generally Available)	257
A.4.7. Changes in MySQL Connector/Net 6.3.3 (2010-07-27)	258
A.4.8. Changes in MySQL Connector/Net 6.3.2 (2010-05-24, Beta)	260
A.4.9. Changes in MySQL Connector/Net 6.3.1 (2010-03-02)	261
A.4.10. Changes in MySQL Connector/Net 6.3.0 (2010-02-16, Alpha)	262
A.5. Changes in MySQL Connector/Net Version 6.2	262
A.5.1. Changes in MySQL Connector/Net 6.2.6 (Not yet released)	262
A.5.2. Changes in MySQL Connector/Net 6.2.5 (2011-07-01)	262
A.5.3. Changes in MySQL Connector/Net 6.2.4 (2010-08-30)	264
A.5.4. Changes in MySQL Connector/Net 6.2.3 (2010-04-10)	267
A.5.5. Changes in MySQL Connector/Net 6.2.2 (2009-12-22, Generally Available)	268
A.5.6. Changes in MySQL Connector/Net 6.2.1 (2009-11-16, Beta)	269
A.5.7. Changes in MySQL Connector/Net 6.2.0 (2009-10-21, Alpha)	270
A.6. Changes in MySQL Connector/Net Version 6.1	271
A.6.1. Changes in MySQL Connector/Net 6.1.7 (Not released, Generally Available)	271
A.6.2. Changes in MySQL Connector/Net 6.1.6 (Not released, Generally Available)	271
A.6.3. Changes in MySQL Connector/Net 6.1.5 (2010-08-30, Generally Available)	271
A.6.4. Changes in MySQL Connector/Net 6.1.4 (2010-04-28, Generally Available)	273
A.6.5. Changes in MySQL Connector/Net 6.1.3 (2009-11-16, Generally Available)	275
A.6.6. Changes in MySQL Connector/Net 6.1.2 (2009-09-08, Generally Available)	276
A.6.7. Changes in MySQL Connector/Net 6.1.1 (2009-08-20, Beta)	277
A.6.8. Changes in MySQL Connector/Net 6.1.0 (2009-07-15, Alpha)	279
A.7. Changes in MySQL Connector/Net Version 6.0	280
A.7.1. Changes in MySQL Connector/Net 6.0.8 (Not released)	280
A.7.2. Changes in MySQL Connector/Net 6.0.7 (2010-08-30)	280
A.7.3. Changes in MySQL Connector/Net 6.0.6 (2010-04-28)	281
A.7.4. Changes in MySQL Connector/Net 6.0.5 (2009-11-12)	283
A.7.5. Changes in MySQL Connector/Net 6.0.4 (2009-06-16)	288
A.7.6. Changes in MySQL Connector/Net 6.0.3 (2009-04-28)	289
A.7.7. Changes in MySQL Connector/Net 6.0.2 (2009-04-07, Beta)	290
A.7.8. Changes in MySQL Connector/Net 6.0.1 (2009-04-02, Beta)	290
A.7.9. Changes in MySQL Connector/Net 6.0.0 (2009-03-02, Alpha)	290
A.8. Changes in MySQL Connector/Net Version 5.3	290
A.8.1. Changes in MySQL Connector/Net 5.3.0 (Not released)	290
A.9. Changes in MySQL Connector/Net Version 5.2	290
A.9.1. Changes in MySQL Connector/Net 5.2.8 (Not released)	290

A.9.2. Changes in MySQL Connector/Net 5.2.7 (2009-07-15)	290
A.9.3. Changes in MySQL Connector/Net 5.2.6 (2009-04-28)	291
A.9.4. Changes in MySQL Connector/Net 5.2.5 (2008-11-19)	293
A.9.5. Changes in MySQL Connector/Net 5.2.4 (2008-11-13)	293
A.9.6. Changes in MySQL Connector/Net 5.2.3 (2008-08-19)	294
A.9.7. Changes in MySQL Connector/Net 5.2.2 (2008-05-12)	295
A.9.8. Changes in MySQL Connector/Net 5.2.1 (2008-02-27)	296
A.9.9. Changes in MySQL Connector/Net 5.2.0 (2008-02-11)	296
A.10. Changes in MySQL Connector/Net Version 5.1	297
A.10.1. Changes in MySQL Connector/Net 5.1.8 (Not released)	297
A.10.2. Changes in MySQL Connector/Net 5.1.7 (2008-08-21)	297
A.10.3. Changes in MySQL Connector/Net 5.1.6 (2008-05-12)	298
A.10.4. Changes in MySQL Connector/Net 5.1.5 (Not released)	299
A.10.5. Changes in MySQL Connector/Net 5.1.4 (2007-11-20)	299
A.10.6. Changes in MySQL Connector/Net 5.1.3 (2007-09-21, Beta)	300
A.10.7. Changes in MySQL Connector/Net 5.1.2 (2007-06-18)	301
A.10.8. Changes in MySQL Connector/Net 5.1.1 (2007-05-23)	301
A.10.9. Changes in MySQL Connector/Net 5.1.0 (2007-05-01)	301
A.11. Changes in MySQL Connector/Net Version 5.0	302
A.11.1. Changes in MySQL Connector/Net 5.0.10 (Not released)	302
A.11.2. Changes in MySQL Connector/Net 5.0.9 (Not released)	302
A.11.3. Changes in MySQL Connector/Net 5.0.8 (2007-08-21)	302
A.11.4. Changes in MySQL Connector/Net 5.0.7 (2007-05-18)	303
A.11.5. Changes in MySQL Connector/Net 5.0.6 (2007-03-22)	303
A.11.6. Changes in MySQL Connector/Net 5.0.5 (2007-03-07)	304
A.11.7. Changes in MySQL Connector/Net 5.0.4 (Not released)	305
A.11.8. Changes in MySQL Connector/Net 5.0.3 (2007-01-05)	305
A.11.9. Changes in MySQL Connector/Net 5.0.2 (2006-11-06)	306
A.11.10. Changes in MySQL Connector/Net 5.0.1 (2006-10-01)	307
A.11.11. Changes in MySQL Connector/Net 5.0.0 (2006-08-08)	307
A.12. Changes in MySQL Connector/Net Version 1.0	308
A.12.1. Changes in MySQL Connector/Net 1.0.11 (Not released)	308
A.12.2. Changes in MySQL Connector/Net 1.0.10 (2007-08-24)	308
A.12.3. Changes in MySQL Connector/Net 1.0.9 (2007-02-02)	308
A.12.4. Changes in MySQL Connector/Net 1.0.8 (2006-10-20)	310
A.12.5. Changes in MySQL Connector/Net 1.0.7 (2005-11-21)	311
A.12.6. Changes in MySQL Connector/Net 1.0.6 (2005-10-03)	311
A.12.7. Changes in MySQL Connector/Net 1.0.5 (2005-08-29)	312
A.12.8. Changes in MySQL Connector/Net 1.0.4 (2005-01-20)	312
A.12.9. Changes in MySQL Connector/Net 1.0.3 (2004-10-12, gamma)	313
A.12.10. Changes in MySQL Connector/Net 1.0.2 (2004-11-15, gamma)	314
A.12.11. Changes in MySQL Connector/Net 1.0.1 (2004-10-27, Beta)	314
A.12.12. Changes in MySQL Connector/Net 1.0.0 (2004-09-01)	315
A.13. Changes in MySQL Connector/Net Version 0.9.0 (30 August 2004)	316
A.14. Changes in MySQL Connector/Net Version 0.76	319
A.15. Changes in MySQL Connector/Net Version 0.75	320
A.16. Changes in MySQL Connector/Net Version 0.74	320
A.17. Changes in MySQL Connector/Net Version 0.71	322
A.18. Changes in MySQL Connector/Net Version 0.70	322
A.19. Changes in MySQL Connector/Net Version 0.68	324
A.20. Changes in MySQL Connector/Net Version 0.65	325
A.21. Changes in MySQL Connector/Net Version 0.60	325
A.22. Changes in MySQL Connector/Net Version 0.50	325

A.1. Changes in MySQL Connector/Net Version 6.6

Changes in MySQL Connector/Net 6.6.4 (2012-10-19, GA)

Changes in MySQL Connector/Net 6.6.3 (2012-09-28, Beta)

Changes in MySQL Connector/Net 6.6.2 (August 2012, Beta)

Continued improvements and fixes to the 6.6 feature set.

Stored Procedure Debugging

- This beta release removes several of the earlier limitations on stored procedure debugging:
 - Functions and triggers can now be debugged.
 - Intellisense is enabled in the debugger window.
 - The debugger supports the SQL grammar for all MySQL versions from 5.0 to 5.6.
 - When a debugging session is finished, stored routines that were instrumented are now restored to their original form.
 - You can now evaluate and change session variables, in addition to local variables in the procedure.
 - Conditional breakpoints are supported now.
 - When debugging a routine that has parameters, the debugger prompts for values to use for the parameters. These prompted values no longer overwrite session variables that have the same names.
 - You do not need to check the **Save password** checkbox when creating a new connection in Server Explorer.

These limitations remain:

- Some MySQL functions cannot be debugged currently (`get_lock`, `release_lock`, `begin`, `commit`, `rollback`, `set transaction level`).
- Only one debug session may be active on a given server.

Changes in MySQL Connector/Net 6.6.1 (2012-08-08, Alpha)

Continued improvements and fixes to the 6.6 feature set.

Version 6.6.1 has no changelog entries.

Changes in MySQL Connector/Net 6.6.0 (2012-07-17, Alpha)

First alpha release for Connector/NET 6.6. Major features of Connector/Net 6.6:

- Stored procedure debugging in Microsoft Visual Studio.
- Entity Framework 4.3 Code First support.
- Pluggable authentication (not available in this alpha).

Entity Framework Code First Support

- To support the Entity Framework 4.3.1, Connector/Net can now use the Code First approach when developing against a model, and keep track of the changes in the Entity Model and in the Database.

This new Entity Framework 4.3.1 feature is focused in allowing your database to be updated along with your Code First Model changes.

Stored Procedure Debugging

- The Connector/Net integration with Visual Studio now includes stored procedure debugging. It works in a very intuitive manner, by simply clicking **Debug Routine** from Server Explorer. The limitations in this preliminary alpha release include:
 - Can only debug stored procedures. Functions and triggers cannot be debugged yet.
 - Intellisense is currently not enabled in the debugger window.
 - Some MySQL functions cannot be debugged currently (`get_lock`, `release_lock`, `begin`, `commit`, `rollback`, `set transaction level`).
 - Only 5.1 grammar is currently supported.
 - Only one debug session may be active on a given server.
 - The debugger instruments your procedures automatically. The original procedure might not be restored correctly.
 - Evaluating and changing session variables are not supported. Local variables in the procedure are supported.
 - Conditional breakpoints are not supported.
 - When debugging a routine that has parameters, the debugger will prompt for those values. It will create session variables out of them, so be careful to not use your own session variables that have the same name.
 - When creating a new connection in Server Explorer, please check the **Save password** checkbox.

Bugs Fixed

- When using Entity Framework with Connector/Net, the association property `onDelete` was not taken into account in the `CreateDatabaseScript` function of the `ObjectContext`, leading to an error message `System.Data.UpdateException was unhandled`. The SQL generated by the `CreateDatabaseScript` function was missing `ON DELETE` and `ON UPDATE` clauses. These clauses were filled in correctly by the DDL generation wizard. (Bug #14008752, Bug #64779)
- A call to a stored procedure or function in an application using the Code First entity framework could result in an error:

```
Unhandled Exception: MySql.Data.MySqlClient.MySqlException: You have an error
in your SQL syntax; ...
```

The code change allows syntax such as the following to invoke a stored procedure, without using the `CALL` statement and without using `CommandType.StoredProcedure`.

```
int count = myContext.Database.SqlQuery<int>("GetCount").First();
```

(Bug #14008699, Bug #64999)

- When using the Entity Framework Code First approach, the generated code could be incorrect:
 - Missing length specifier for data types, such as `VARBINARY` instead of `VARBINARY(n)`.
 - `ALTER TABLE` statements referring to nonexistent tables, when private members were specified inside the main class.

(Bug #13900091, Bug #64216)

A.2. Changes in MySQL Connector/Net Version 6.5

Changes in MySQL Connector/Net 6.5.5 (Not yet released.)

Version 6.5.5 has no changelog entries.

Changes in MySQL Connector/Net 6.5.4 (2012-03-08)

First GA release for Connector/NET 6.5.

Bugs Fixed

- In Visual Studio Table Designer, if you tried to save a new table using an existing table name, subsequently you would not be prompted to choose a new name, preventing you from saving the table. (Bug #13785918)
- When creating a Visual Studio Web Application Project, using the ADO.NET Entity Data Model and generating the model from a database, the Entity Framework Model was not created. This operation gave an error:

```
Access denied for user 'root'@'localhost' (using password: NO)
```

(Bug #13610452)

- When creating a project in VisualStudio using a .NET framework such as 3.0 or 3.5 (anything less than 4.0), the Connector/Net library ([MySQL.Data.dll](#)) was not listed in the **Add References** dialog box. The workaround was to browse to the library and add it manually. (Bug #13491678, Bug #60462)

Changes in MySQL Connector/Net 6.5.3 (2012-02-27)

Second Release Candidate (RC) release.

Bugs Fixed

- The performance when setting the [CommandText](#) property on the [MySQLCommand](#) class was improved by enhancing the efficiency of a string comparison operation. (Bug #13739383, Bug #64012)
- Fixed [MySQLTime](#) parsing to avoid throwing an exception when handling milliseconds (as result of a [timediff](#) operation). (Bug #13708884, Bug #64268)
- In Visual Studio Table Designer, when adding a second [foreign key](#), the new name was incorrectly assigned to the first foreign key in the list. (Bug #13613824)
- In Visual Studio Table Designer, changes to a field were sometimes not detected until you switched focus away from that field. (Bug #13613755)

Changes in MySQL Connector/Net 6.5.2 (2012-02-15)

First Release Candidate (RC) release.

Bugs Fixed

- In Visual Studio Table Designer, when editing a [foreign key](#) relationship, choosing a column name on the left side made that column name unavailable on the right side. (Bug #13615258)
- In Visual Studio Table Designer, an error could occur if you added and deleted column information for [foreign keys](#) in a particular sequence. (Bug #13610235)
- In Visual Studio Table Designer, deleting a [foreign key](#) relationship in the **Relationship** dialog required clicking twice. (Bug #13610283)

- In Visual Studio Table Designer, modifying the **Columns** field in the **Indexes/Keys** dialog multiple times could cause an error. (Bug #13613765)
- In Visual Studio Table Designer, it was possible to save a new **foreign key** relationship without filling in the fields of the **Foreign Key Relationship** dialog. (Bug #13613839)
- In Visual Studio Table Designer, changing the length of a **VARCHAR** field could cause an error. (Bug #13611677)
- If `MySqlCommand.CommandText` was equal to `null`, then `MySqlCommand.ExecuteReader()` would throw the wrong exception: `NullReferenceException` instead of `InvalidOperationException`. (Bug #13624659, Bug #64092)
- When using connection pooling, the connections in the pool were not automatically closed upon application exit. With the setting `log-warnings=2`, you could encounter `Aborted connection` errors in the MySQL error log. The workaround was to explicitly call `MySQL.Data.MySqlClient.MySqlConnection.ClearAllPools()` upon exiting the application. (Bug #13629471, Bug #63942)
- The MySQL script generated by using the function `CreateDatabaseScript` used names with incorrect singular/plural forms. (Bug #13582837, Bug #62150)
- In Visual Studio Table Designer, the **Add -> Function Import...** dialog could close prematurely when you pressed the **Get Column Information** button. (Bug #13511736)
- When designating a primary key for a table in Table Designer, the key icon could fail to appear until the Table Designer was restarted. (Bug #13481246)

Changes in MySQL Connector/Net 6.5.1 (2012-01-23)

Second beta release.

Bugs Fixed

- In Table Designer for Visual Studio, trying to delete **foreign keys** from an **InnoDB** table showed an error, and the change was not saved. (Bug #13481362)
- After an **UPDATE** statement, Connector/Net would generate incorrect **SELECT** SQL statements if a value in the **WHERE** clause was not also present in the **SET** clause of the **UPDATE**. (Bug #13491689, Bug #62134)
- IntelliSense would emit an error when the "-" (minus) character was typed. (Bug #13522344)

Changes in MySQL Connector/Net 6.5.0 (2011-12-22)

First beta release.

Functionality Added or Changed

- Added better IntelliSense support, including auto-completion when editing stored procedures or `.mysql` files. For more information, see [Section 4.2, "Using IntelliSense in the SQL Editor"](#).
- Adds a `MySqlClientPermission` class to help users define the security policies for the database connections within any application using a MySQL database.
- Added better partial-trust support, thus allowing Connector/NET to run in a partial trust scenario. It will work correctly in a medium-trust level environment when the library is installed in the GAC. For more information, see [Section 6.19, "Working with Partial Trust / Medium Trust"](#).
- Added fractional seconds support, as per MySQL Server 5.6 and above. For more information, see [Fractional Seconds in Time Values](#)
- Added "interceptor" classes for exceptions and commands. For more information, see [Section 6.11, "Using the Connector/Net Interceptor Classes"](#).

Bugs Fixed

- The `MySqlDataReader.GetDateTime()` method was not recognizing that `TIMESTAMP` values had already been converted to the local time zone of the MySQL server, which could cause incorrect results if the value was later processed through the `ToLocalTime()` method. The fix causes the `Kind` property to be correctly set to `Local` rather than `Unspecified`. (Bug #13591554, Bug #63812)
- Visual Studio 2010 Table Designer could give an error “Object reference not set to an instance of an object” for schemas with certain combinations of column names and foreign key references. The SQL syntax was incorrect for the `ALTER TABLE` statement generated by the Table Designer. (Bug #13591545, Bug #63714)

A.3. Changes in MySQL Connector/Net Version 6.4

A.3.1. Changes in MySQL Connector/Net 6.4.6 (2012-11-26)

This release fixes bugs since 6.4.5.

Bugs Fixed

- Under some circumstances, setting `CacheServerProperties=true` in the connection string could cause a `Packet too large` error. With connection pooling enabled and `CacheServerProperties=true`, the first connection worked as expected, but the second, third, and so on connections would fail if the query exceeded 1024 bytes. (Bug #14593547, Bug #66578)
- Connector/Net did not support creating an entity with a key of type string. During database creation, a `MySqlException` was thrown saying `BLOB/TEXT column 'Name' used in key specification without a key length`. The DDL produced by the provider specified a `MEDIUMTEXT` column for the primary key without specifying a length for the key. This fix is particularly important when working with Entity Framework versions 4.3 and later, since the `__MigrationsHistory` table (which replaces the `EdmMetadata` table) uses a string property as its key. (Bug #14540202, Bug #65289, Bug #64288)
- The `ExecuteNonQuery()` could return an error `Parameter '?' must be defined`, when attempting to execute a statement such as:

```
insert into table_name (Field1, Field1) VALUES(?,?)
```

That is, when referencing the same field twice with two different `?` placeholders. (Bug #14499549, Bug #66060)

- Customizing precision by calling the `HasPrecision()` method within the `OnModelCreating()` method in a Code First project would always produce precision settings (10,2) rather than the specified precision. (Bug #14469048, Bug #65001)
- The MySQL Connector/Net EntityFramework provider would throw `NullReferenceException` when trying to insert a new record with an empty `VALUES` clause. Such an `INSERT` should work when the only required (`NOT NULL`) column in the table is a primary key auto-increment column. (Bug #14479715, Bug #66066)
- When building commands through the `MySql.Data.MySqlClient.MySqlCommand()` class, memory could be leaked because some `IO.MemoryStream` instances were not being freed efficiently. The memory leak could be an issue in SQL-heavy applications, for example a logging application processing large numbers of `INSERT` statements. (Bug #14468204, Bug #65696)
- Using the Entity Data Model Designer `decimal` type and `CreateDatabase` function, the values were stored with 0 digits at the right of the decimal point. With this fix, the default is 2 digits to the right of the decimal point, and any precision specified through the Entity Data Model Designer is applied correctly. (Bug #14474342, Bug #65127)

- When using a MySQL database set up as `UTF32` as an ASP.net membership database, web applications could give a “key too long” error, and the Website Administration Tool would not connect to providers. The cause was that the column `my_aspnet_sessions.SessionId`, when converted from `Latin1` character set to `UTF32` with 4 bytes per character, exceeded the length limit for a [primary key](#):

```
Specified key was too long; max key length is 767 bytes
```

(Bug #14495292, Bug #65144)

- When using the ASP.net web security functionality with a MySQL database, using features that access the `my_aspnet_usersinroles` table caused an exception:

```
MySql.Data.MySqlClient.MySqlException: Table 'testdb.my_aspnet_usersinrole' doesn't exist.
```

For example, this error could occur when trying to remove the user from a role or find users in a role. The fix corrects the spelling of the table name to `my_aspnet_usersinroles`. (Bug #14405338, Bug #65805)

- Although the member variable `MySqlCommand.LastInsertedId` was a 64-bit `long`, its value was effectively capped at the maximum value of `Int32` (2,147,483,647). If a primary key exceeded this value, the value of `LastInsertedId` was wrong. This mismatch could be an issue for tables with large numbers of rows. (Bug #14171960, Bug #65452)
- When using Entity Framework with Connector/Net, the association property `OnDelete` was not taken into account in the `CreateDatabaseScript` function of the `ObjectContext`, leading to an error message `System.Data.UpdateException was unhandled`. The SQL generated by the `CreateDatabaseScript` function was missing `ON DELETE` and `ON UPDATE` clauses. These clauses were filled in correctly by the DDL generation wizard. (Bug #14008752, Bug #64779)
- A call to a stored procedure or function in an application using the Code First entity framework could result in an error:

```
Unhandled Exception: MySql.Data.MySqlClient.MySqlException: You have an error in your SQL syntax; ...
```

The code change allows syntax such as the following to invoke a stored procedure, without using the `CALL` statement and without using `CommandType.StoredProcedure`.

```
int count = myContext.Database.SqlQuery<int>("GetCount").First();
```

(Bug #14008699, Bug #64999)

- When using the Entity Framework Code First approach, the generated code could be incorrect:
 - Missing length specifier for data types, such as `VARBINARY` instead of `VARBINARY(n)`.
 - `ALTER TABLE` statements referring to nonexistent tables, when private members were specified inside the main class.
- (Bug #13900091, Bug #64216)
- When using the `MySqlProfileProvider`, calling the function `ProfileManager.DeleteProfiles` could throw an `InvalidCastException` exception. (Bug #13790123, Bug #64470)
- In Visual Studio Table Designer, the name of a new index was always derived from the name of the table and could not be changed. (Bug #13613801)
- When using the Entity Framework Code First approach, the generated code could be use the `MEDIUMTEXT` data type in contexts where other types such as `VARCHAR` were more appropriate, leading to errors such as:

```
error 0064: Facet 'MaxLength' must not be specified for type 'mediumtext'.
```


(Bug #13582335, Bug #63920)

A.3.2. Changes in MySQL Connector/Net 6.4.5 (2012-05-19)

This release fixes bugs since 6.4.4.

Bugs Fixed

- When the length of a `VARCHAR` column was edited in Table Designer, the data type could be saved incorrectly as `BIT`. (Bug #13916560)
- Any sequence of `Take(n)` method calls followed by `Count` or `LongCount` would cause a `System.Data.EntityCommandCompilationException` error. (Bug #13913047, Bug #64749)
- The performance when setting the `CommandText` property on the `MySqlCommand` class was improved by enhancing the efficiency of a string comparison operation. (Bug #13739383, Bug #64012)
- In Visual Studio Table Designer, if you tried to save a new table using an existing table name, subsequently you would not be prompted to choose a new name, preventing you from saving the table. (Bug #13785918)
- The MySQL script generated by using the function `CreateDatabaseScript` used names with incorrect singular/plural forms. (Bug #13582837, Bug #62150)
- Visual Studio 2010 Table Designer could give an error "Object reference not set to an instance of an object" for schemas with certain combinations of column names and foreign key references. The SQL syntax was incorrect for the `ALTER TABLE` statement generated by the Table Designer. (Bug #13591545, Bug #63714)
- In Table Designer for Visual Studio, trying to create a table could fail if you saved changes immediately after entering the data type for a column. The workaround was to click somewhere else in the grid before saving changes. (Bug #13477805)
- Creating a table through the Server Explorer Window on Visual Studio 2010 could fail with a MySQL syntax error. The properties in the `CREATE TABLE` statement could be listed in incorrect order. (Bug #13475830)
- In Table Designer for Visual Studio, trying to delete `foreign keys` from an `InnoDB` table showed an error, and the change was not saved. (Bug #13481362)
- In "LINQ to Entity" queries, including a child entity (1-n) and its entities (n-n) returned the wrong results. For example:

```
db.Authors.Include("Books.Editions").AsEnumerable().First();
```

(Bug #13491698, Bug #62801)

- When creating a project in VisualStudio using a .NET framework such as 3.0 or 3.5 (anything less than 4.0), the Connector/Net library (`MySql.Data.dll`) was not listed in the **Add References** dialog box. The workaround was to browse to the library and add it manually. (Bug #13491678, Bug #60462)
- After an `UPDATE` statement, Connector/Net would generate incorrect `SELECT` SQL statements if a value in the `WHERE` clause was not also present in the `SET` clause of the `UPDATE`. (Bug #13491689, Bug #62134)
- Formerly, cleanup operations for expired sessions were fully automatic, with no ability to catch the timeout event and do application-specific cleanup. This fix adds a `enableSessionExpireCallback` connection option to let developers catch the event when a session expires. When `enableSessionExpireCallback` is enabled,

the `global.asax.session_end` event is raised before data is deleted from the `my_aspnet_sessions` table. When `enableSessionExpireCallback` is disabled, the data is deleted from the `my_aspnet_sessions` table without raising the event first. The timeout period for session expiry is specified in the `web.config` file, in the `timeout` option of the `sessionState` section. (Bug #13354935, Bug #62266)

- Connector/NET experienced poor performance when adding parameters to the `MySQLCommand`. (Bug #62653, Bug #13331475)
- The Unicode quotation mark character `U+0022` was not escaped by the `MySQLHelper` class. (Bug #62585, Bug #13092886)
- Using a return parameter without a name resulted in an `IndexOutOfRangeException` exception. (Bug #62416, Bug #13006969)
- The `Mono` runtime did not support hashed passwords. (Bug #62203, Bug #13041618)
- Connector/NET incorrectly maps `PrimitiveTypeKind.Byte` to `tinyint`, instead of `utinyint`. And `PrimitiveTypeKind.SByte` mapping was added, to `tinyint`. (Bug #62135, Bug #13061713)
- Connector/NET would incorrectly map decimal values to ANSI strings. (Bug #62246, Bug #13050570)

A.3.3. Changes in MySQL Connector/Net 6.4.4 (2011-09-26)

This release fixes bugs since 6.4.3.

Functionality Added or Changed

- Connector/Net now enables clients to connect to the server through accounts that use Windows native authentication. For more information, see [Section 6.5, “Using the Windows Native Authentication Plugin”](#) and [The Windows Native Authentication Plugin](#).

Bugs Fixed

- (Bug #12897149)
- When creating a tableadapter through a Dataset form in Visual Studio, the `MaxLength` of the field for a `CHAR` column could be set to 3 times the length of the table column. Although this many bytes could be needed to hold a UTF-8 character value, the length value was not appropriate for restricting the length of a `TextBox`. (Bug #12860224, Bug #62094)
- When adding a `MEDIUMTEXT` or `LONGTEXT` column Visual Studio, the facet `Fixed length` had to be set to `false`, even though these types allow arbitrary lengths. (Bug #12848277, Bug #54915)
- An error `out of sync with server` could occur when connecting in the Visual Studio Entity Framework. The issue occurred only on some MySQL servers, all with versions earlier than MySQL 5.5. (Bug #12853286, Bug #61806)
- Using a combination of `ListView`, `EntityDataSource` with `TypeFilter` and `Include EntityCollection Navigation Property`, and `DataPager` caused a `NullReferenceException` error in the `System.Web.UI.WebControls.EntityDataSourceView.ExecuteSelect` function. (Bug #12776517, Bug #61714)
- Executing a “LINQ to Entity” query could result in a `NullReferenceException` error. (Bug #12776598, Bug #61729)
- On Model First changed column types generated in SQL scripts to produce more suitable `MySQL` types. (Bug #59244, Bug #12707104)

- `MySqlScript` was modified to enable correct processing of the `DELIMITER` command when not followed by a new line. (Bug #61680, Bug #12732279)
- `MySqlDataReader.Close` was modified to use Default behavior when clearing remaining result sets. (Bug #61690, Bug #12723423)
- The ASP.NET membership provider was modified to create and query all related tables using lowercase names. (Bug #61108, Bug #12702009)

A.3.4. Changes in MySQL Connector/Net 6.4.3 (2011-07-03)

This release fixes bugs since 6.4.2.

Bugs Fixed

- `SchemaDefinition-5.5.ssd1` was modified to treat `CHAR(36)` columns as a GUID. (Bug #61657, Bug #12708208)
- `SqlFragment.QuoteIdentifier` was modified to add MySQL quotes around identifiers. (Bug #61635, Bug #12707285)

A.3.5. Changes in MySQL Connector/Net 6.4.2 (2011-06-29)

This release fixes bugs since 6.4.1.

Bugs Fixed

- Modified `ProviderManifest.xml` to map `TIMESTAMP` database columns to the `DateTime` .NET type. (Bug #55351, Bug #12652602)
- Modified `MySqlConnection.BeginTransaction` to throw a `NotSupportedException` for `Snapshot` isolation level. (Bug #61589, Bug #12698020)
- Fixed Entity Framework provider `GROUP BY` clause generation by adding all group-by keys to the `SELECT` statement. (Bug #46742, Bug #12622129)

A.3.6. Changes in MySQL Connector/Net 6.4.1 (2011-06-06, Alpha)

First alpha release.

Functionality Added or Changed

- Calling a stored procedure with output parameters caused a marked performance decrease. (Bug #60366, Bug #12425959)
- Changed how the procedure schema collection is retrieved. If the connection string contains "`use procedure bodies=true`" then a `SELECT` is performed on the `mysql.proc` table directly, as this is up to 50 times faster than the current Information Schema implementation. If the connection string contains "`use procedure bodies=false`", then the Information Schema collection is queried. (Bug #36694)

A.3.7. Changes in MySQL Connector/Net 6.4.0 (Unknown)

Version 6.4.0 has no changelog entries.

A.4. Changes in MySQL Connector/Net Version 6.3

A.4.1. Changes in MySQL Connector/Net 6.3.9 (2012-04-11)

This release fixes bugs since 6.3.8.

Bugs Fixed

- When the length of a `VARCHAR` column was edited in Table Designer, the data type could be saved incorrectly as `BIT`. (Bug #13916560)
- Any sequence of `Take(n)` method calls followed by `Count` or `LongCount` would cause a `System.Data.EntityCommandCompilationException` error. (Bug #13913047, Bug #64749)
- When adding a ADO.NET Entity Data Model and generating the model from a database containing `foreign keys`, the foreign keys were not included in the generated model. (Bug #13800109)
- The performance when setting the `CommandText` property on the `MySqlCommand` class was improved by enhancing the efficiency of a string comparison operation. (Bug #13739383, Bug #64012)
- Fixed `MySqlTime` parsing to avoid throwing an exception when handling milliseconds (as result of a `timediff` operation). (Bug #13708884, Bug #64268)
- In Visual Studio Table Designer, when adding a second `foreign key`, the new name was incorrectly assigned to the first foreign key in the list. (Bug #13613824)
- In Visual Studio Table Designer, when editing a `foreign key` relationship, choosing a column name on the left side made that column name unavailable on the right side. (Bug #13615258)
- In Visual Studio Table Designer, an error could occur if you added and deleted column information for `foreign keys` in a particular sequence. (Bug #13610235)
- In Visual Studio Table Designer, deleting a `foreign key` relationship in the **Relationship** dialog required clicking twice. (Bug #13610283)
- In Visual Studio Table Designer, modifying the **Columns** field in the **Indexes/Keys** dialog multiple times could cause an error. (Bug #13613765)
- In Visual Studio Table Designer, it was possible to save a new `foreign key` relationship without filling in the fields of the **Foreign Key Relationship** dialog. (Bug #13613839)
- In Visual Studio Table Designer, changes to a field were sometimes not detected until you switched focus away from that field. (Bug #13613755)
- In Visual Studio Table Designer, changing the length of a `VARCHAR` field could cause an error. (Bug #13611677)
- When creating a Visual Studio Web Application Project, using the ADO.NET Entity Data Model and generating the model from a database, the Entity Framework Model was not created. This operation gave an error:

```
Access denied for user 'root'@'localhost' (using password: NO)
```

(Bug #13610452)

- If `MySqlCommand.CommandText` was equal to `null`, then `MySqlCommand.ExecuteReader()` would throw the wrong exception: `NullReferenceException` instead of `InvalidOperationException`. (Bug #13624659, Bug #64092)
- When using connection pooling, the connections in the pool were not automatically closed upon application exit. With the setting `log-warnings=2`, you could encounter `Aborted connection` errors in the MySQL error log. The workaround was to explicitly call `MySql.Data.MySqlClient.MySqlConnection.ClearAllPools()`; upon exiting the application. (Bug #13629471, Bug #63942)
- The MySQL script generated by using the function `CreateDatabaseScript` used names with incorrect singular/plural forms. (Bug #13582837, Bug #62150)
- In Visual Studio Table Designer, the **Add -> Function Import...** dialog could close prematurely when you pressed the **Get Column Information** button. (Bug #13511736)

- In “LINQ to Entity” queries, including a child entity (1-n) and its entities (n-n) returned the wrong results. For example:

```
db.Authors.Include("Books.Editions").AsEnumerable().First();
```

(Bug #13491698, Bug #62801)

- After an `UPDATE` statement, Connector/Net would generate incorrect `SELECT` SQL statements if a value in the `WHERE` clause was not also present in the `SET` clause of the `UPDATE`. (Bug #13491689, Bug #62134)
- In Visual Studio's Server Explorer, right-clicking on a table in the Tables tree and selecting Create Trigger could cause an error, `Object reference not set to an instance of an object`. (Bug #13511801)
- IntelliSense would emit an error when the "-" (minus) character was typed. (Bug #13522344)
- The class `MySQL.Data.Types.MySqlDateTime` was not serializable. (Bug #11750161, Bug #40555)
- Connector/NET incorrectly maps `PrimitiveTypeKind.Byte` to `tinyint`, instead of `utinyint`. And `PrimitiveTypeKind.SByte` mapping was added, to `tinyint`. (Bug #62135, Bug #13061713)
- Connector/NET would incorrectly map decimal values to ANSI strings. (Bug #62246, Bug #13050570)
- On Model First changed column types generated in SQL scripts to produce more suitable `MySQL` types. (Bug #59244, Bug #12707104)

A.4.2. Changes in MySQL Connector/Net 6.3.8 (2011-12-16)

This release fixes bugs since 6.3.7.

Bugs Fixed

- In Visual Studio Table Designer, if you tried to save a new table using an existing table name, subsequently you would not be prompted to choose a new name, preventing you from saving the table. (Bug #13785918)
- Visual Studio 2010 Table Designer could give an error “Object reference not set to an instance of an object” for schemas with certain combinations of column names and foreign key references. The SQL syntax was incorrect for the `ALTER TABLE` statement generated by the Table Designer. (Bug #13591545, Bug #63714)
- In table designer for Visual Studio, you could not create a **foreign key** that referenced the same table as source and destination. When adding a new relationship, the **Referenced table** list did not offer the original table as one of the choices. (Bug #13481340)
- Creating a table through the Server Explorer Window on Visual Studio 2010 could fail with a MySQL syntax error. The properties in the `CREATE TABLE` statement could be listed in incorrect order. (Bug #13475830)
- In Table Designer for Visual Studio, trying to delete **foreign keys** from an `InnoDB` table showed an error, and the change was not saved. (Bug #13481362)
- When creating a project in VisualStudio using a .NET framework such as 3.0 or 3.5 (anything less than 4.0), the Connector/Net library (`MySQL.Data.dll`) was not listed in the **Add References** dialog box. The workaround was to browse to the library and add it manually. (Bug #13491678, Bug #60462)
- When a new column was added in Table Designer without selecting an associated data type, an error would occur trying to save the column definition. (Bug #13481298)

- When creating a foreign key relationship in Table Designer, changes to the `ON UPDATE` and `ON CASCADE` settings were not reflected in the actual table definition, as displayed by `SHOW CREATE TABLE`. (Bug #13481348)
- Removing a method was not affecting the indexes list of the table object, as defined within the Table Designer. (Bug #13481313)
- The columns added in descending sort order were not included in the index, as defined within the Server Explorer. (Bug #13481709)
- The comment property and index type were not added in the definition of the index, as defined within the Server Explorer. (Bug #13481314)
- The Connector/Net installed for all users, and thus was not available in the `Add/Remove Programs` dialog for users other than the one who installed it. (Bug #13447941)
- The default value for `VARCHAR` and `CHAR` field types would contain single quotation marks. (Bug #13442506)
- When creating a tableadapter through a Dataset form in Visual Studio, the `MaxLength` of the field for a `CHAR` column could be set to 3 times the length of the table column. Although this many bytes could be needed to hold a UTF-8 character value, the length value was not appropriate for restricting the length of a `TextBox`. (Bug #12860224, Bug #62094)
- When adding a `MEDIUMTEXT` or `LONGTEXT` column Visual Studio, the facet `Fixed length` had to be set to `false`, even though these types allow arbitrary lengths. (Bug #12848277, Bug #54915)
- Using a combination of `ListView`, `EntityDataSource` with `TypeFilter` and `Include EntityCollection Navigation Property`, and `DataPager` caused a `NullReferenceException` error in the `System.Web.UI.WebControls.EntityDataSourceView.ExecuteSelect` function. (Bug #12776517, Bug #61714)
- Executing a “LINQ to Entity” query could result in a `NullReferenceException` error. (Bug #12776598, Bug #61729)
- Connector/NET experienced poor performance when adding parameters to the `MySQLCommand`. (Bug #62653, Bug #13331475)
- The Unicode quotation mark character `U+0022` was not escaped by the `MySQLHelper` class. (Bug #62585, Bug #13092886)
- Using a return parameter without a name resulted in an `IndexOutOfRangeException` exception. (Bug #62416, Bug #13006969)
- The `Mono` runtime did not support hashed passwords. (Bug #62203, Bug #13041618)
- Modified `ProviderManifest.xml` to map `TIMESTAMP` database columns to the `DateTime` .NET type. (Bug #55351, Bug #12652602)
- Modified `MySQLConnection.BeginTransaction` to throw a `NotSupportedException` for `Snapshot` isolation level. (Bug #61589, Bug #12698020)
- `SchemaDefinition-5.5.ssd1` was modified to treat `CHAR(36)` columns as a GUID. (Bug #61657, Bug #12708208)
- `SqlFragment.QuoteIdentifier` was modified to add MySQL quotes around identifiers. (Bug #61635, Bug #12707285)
- `MySQLScript` was modified to enable correct processing of the `DELIMITER` command when not followed by a new line. (Bug #61680, Bug #12732279)
- `MySQLDataReader.Close` was modified to use Default behavior when clearing remaining result sets. (Bug #61690, Bug #12723423)

- The ASP.NET membership provider was modified to create and query all related tables using lowercase names. (Bug #61108, Bug #12702009)

A.4.3. Changes in MySQL Connector/Net 6.3.7 (2011-06-22)

This release fixes bugs since 6.3.6.

Functionality Added or Changed

- Calling a stored procedure with output parameters caused a marked performance decrease. (Bug #60366, Bug #12425959)

Bugs Fixed

- `MySQLConnectionStringBuilder.ContainsKey()` incorrectly returned `false` when testing whether a keyword was part of the connection string. (Bug #11766671, Bug #59835)
- Fixed Entity Framework provider `GROUP BY` clause generation by adding all group-by keys to the `SELECT` statement. (Bug #46742, Bug #12622129)
- A `NullReferenceException` was thrown on disposal of a `TransactionScope` object. (Bug #59346, Bug #11766272)
- MySQL Connector/Net generated an exception when executing a query consisting of ';', for example:

```
mycmd( " ; ", mycon)
mycmd.ExecuteNonQuery()
```

The exception generated was:

```
System.IndexOutOfRangeException: Index was outside the bounds of the array.
   at MySql.Data.MySqlClient.MySqlCommand.TrimSemicolons(String sql)
   at MySql.Data.MySqlClient.MySqlCommand.ExecuteReader(CommandBehavior behavior)
   at MySql.Data.MySqlClient.MySqlCommand.ExecuteReader()
   at MySql.Data.MySqlClient.MySqlCommand.ExecuteNonQuery()
```

(Bug #59537, Bug #11766433)

- MySQL Connector/Net 6.3.6 did not work with Visual Studio 2010. (Bug #60723, Bug #12394470)
- `MysqlDataReader.GetSchemaTable` returned incorrect values and types. (Bug #59989, Bug #11776346)
- Setting `Membership.ApplicationName` had no effect. (Bug #59438, Bug #11770465)
- All queries other than `INSERT` were executed individually instead of as a batch even though batching was enabled for the connection. (Bug #59616, Bug #11850286)
- The setup wizard failed with the error "Setup Wizard ended prematurely because of an error". This was because it assumed .NET Framework version 4.0 was located on the C: drive, when it was actually located on the E: drive. (Bug #59301)

A.4.4. Changes in MySQL Connector/Net 6.3.6 (2011-01-03)

This release fixes bugs since 6.3.5.

Functionality Added or Changed

- Changed how the procedure schema collection is retrieved. If the connection string contains "`use procedure bodies=true`" then a `SELECT` is performed on the `mysql.proc` table directly, as this is up to 50 times faster than the current Information Schema implementation. If the connection string contains "`use procedure bodies=false`", then the Information Schema collection is queried. (Bug #36694)

Bugs Fixed

- In the ADO.NET Entity Data Model Wizard, the time to update a model scaled abnormally as the number of entities increased. (Bug #48791, Bug #12596237)
- The `MySQLTokenizer` contained unnecessary `Substring` and `Trim` calls:

```
string token = sql.Substring(startIndex, stopIndex - startIndex).Trim();
```

The variable `token` was not used anywhere in the code. (Bug #58757)

- `MembershipProvider` did not generate hashes correctly if the algorithm was keyed. The Key of the algorithm should have been set if the `HashAlgorithm` was `KeyedHashAlgorithm`. (Bug #58906)
- When attempting to create an ADO.NET Entity Data Model, MySQL connections were not available. (Bug #58278)
- `MySQLCommand.ExecuteReader(CommandBehavior)` threw a `NullReferenceException` when being called with `CommandBehavior.CloseConnection`, if the SQL statement contained a syntax error, or contained invalid data such as an invalid column name. (Bug #58652)
- The ADO.NET Entity Data Model could not add stored procedures from MySQL Server 5.0.45 but worked fine using MySQL Server 5.1. (Bug #55349)
- Code introduced to fix bug #54863 proved problematic on .NET version 3.5 and above. (Bug #58853)
- `ReadFieldLength()` returned incorrect value for `BIGINT` autoincrement columns. (Bug #58373)
- Calling the `Read()` method on a `DataReader` obtained from `MySQLHelper.ExecuteReader` generated the following exception:

```
Unhandled Exception: MySql.Data.MySqlClient.MySqlException: Invalid attempt to R
ead when reader is closed.
   at MySql.Data.MySqlClient.MySqlDataReader.Read()
   at MySqlTest.MainClass.Main(String[] args)
```

(Bug #57501)

- MySQL Connector/Net did not support the `utf8mb4` character set. When attempting to connect to `utf8mb4` tables or columns, an exception `KeyNotFoundException` was generated. (Bug #58244)
- Default values returned for text columns were not quoted. This meant that the `COLUMN_DEFAULT` field of the `GetSchema` columns collection did not return a valid SQL expression. (Bug #56509)
- When using MySQL Connector/Net on Mono 2.8 using .NET 4.0, attempting to connect to a MySQL database generated the following exception:

```
Unhandled Exception: System.MissingMethodException: Method not found:
'System.Data.Common.DbConnection.EnlistTransaction'.
   at (wrapper remoting-invoke-with-check)
   MySql.Data.MySqlClient.MySqlConnection.Open ()
```

(Bug #56509)

- When the tracing driver was used and an SQL statement was longer than 300 characters, an `ArgumentOutOfRangeException` occurred if the statement also contained a quoted character, and the 300th character was in the middle of a quoted token. (Bug #57641)
- MySQL Connector/Net for .NET/Mono attempted to dynamically load the assembly `Mono.Posix.dll` when a Unix socket was used to connect to the server. This failed and the connector was not able to use a Unix socket unless the `Mono.Posix.dll` assembly was previously loaded by the program. (Bug #56410)

- Installation of MySQL Connector/Net 6.3.5 failed. The error reported was:

```
MySQL Connector Net 6.3.5 Setup Wizard ended  
prematurely because of an error. Your system has not been modified.
```

(Bug #57654)

A.4.5. Changes in MySQL Connector/Net 6.3.5 (2010-10-12)

This release fixes bugs since 6.3.4.

Bugs Fixed

- A typed dataset did not get the table name. (Bug #57894, Bug #11764989)
- Setting `MySQLCommand.CommandTimeout` to 0 had no effect. It should have resulted in an infinite timeout. (Bug #57265)
- When an output parameter was declared as type `MySQLDbType.Bit`, it failed to return with the correct value. (Bug #56756)
- When performing a row-by-row update, only the first row was updated and all other rows were ignored. (Bug #57092)
- Setting the `Default Command Timeout` connection string option had no effect. (Bug #56806)
- MySQL Connector/Net experienced two problems as follows:
 1. A call to `System.Data.Objects.ObjectContext.DatabaseExists()` returned false, even if the database existed.
 2. A call to `System.Data.Objects.ObjectContext.CreateDatabase()` created a database but with a name other than the one specified in the connection string. It then failed to use it when EDM objects were processed.(Bug #56859)
- `MySQLHelper.ExecuteReader` did not include an overload accepting `MySQLParameter` objects when using a `MySQLConnection`. However, `MySQLHelper` did include an overload for `MySQLParameter` objects when using a string object containing the connection string to the database. (Bug #56755)
- MySQL Connector/Net 6.1.3 (GA) would not install on a Windows Server 2008 (Web Edition) clean installation. There were two problems:
 - If .NET framework version 4.0 was not installed, installation failed because `c:\windows\microsoft.net\v4.0.*` did not exist.
 - If .NET 4.0 was subsequently installed, then the following error was generated:

```
InstallFiles: File: MySql.Data.Entity.dll, Directory: , Size: 229888  
MSI (s) (E0:AC) [15:20:26:196]: Assembly Error:The assembly is built by a runtime newer  
than the currently loaded runtime, and cannot be loaded.  
MSI (s) (E0:AC) [15:20:26:196]: Note: 1: 1935 2: 3: 0x8013101B 4: IStream 5: Commit 6:  
MSI (s) (E0:A0) [15:20:26:196]: Note: 1: 1304 2: MySql.Data.Entity.dll  
Error 1304. Error writing to file: MySql.Data.Entity.dll. Verify that you have access to  
that directory.
```

(Bug #56580)

A.4.6. Changes in MySQL Connector/Net 6.3.4 (2010-09-01, Generally Available)

First GA release. This release fixes bugs since 6.3.3.

Bugs Fixed

- Opening a connection in the Visual Studio Server Explorer and choosing to alter an existing routine required another authentication at the server. (Bug #44715)
- Stored procedure enumeration code generated an error if a procedure was used in a dataset that did not return any resultsets. (Bug #50671)
- EventLog was not disposed in the SessionState provider. (Bug #52550)
- When attempting to carry out an operation such as:

```
from p in db.Products where p.PostedDate>=DateTime.Now select p;
```

Where `p.PostedDate` is a `DateTimeOffset`, and the underlying column type is a `TIMESTAMP`, the following exception was generated:

```
MySqlException occurred
Unable to serialize date/time value
```

MySQL Connector/Net has now been changed so that all `TIMESTAMP` columns will be surfaced as `DateTime`. (Bug #52550)

- Attempting to read `Double.MinValue` from a `DOUBLE` column in MySQL table generated the following exception:

```
System.OverflowException : Value was either too large or too small for a Double.

--OverflowException
at System.Number.ParseDouble(String value, NumberStyles options, NumberFormatInfo numfmt)
at MySql.Data.Types.MySqlDouble.MySql.Data.Types.IMySqlValue.ReadValue(MySqlPacket packet, Int64 length, Boolean nullVal)
at MySql.Data.MySqlClient.NativeDriver.ReadColumnValue(Int32 index, MySqlField field, IMySqlValue valObject)
at MySql.Data.MySqlClient.ResultSet.ReadColumnData(Boolean outputParms)
at MySql.Data.MySqlClient.ResultSet.NextRow(CommandBehavior behavior)
at MySql.Data.MySqlClient.MySqlDataReader.Read()
```

(Bug #55644)

- The calculation of `lockAge` in the Session Provider sometimes generated a `System.Data.SqlTypes.SqlNullValueException`. (Bug #55701)
- The `INSERT` command was significantly slower with MySQL Connector/Net 6.x compared to 5.x, when compression was enabled. (Bug #48243)
- If using MySQL Server 5.0.x it was not possible to alter stored routines in Visual Studio. If the stored routine was clicked, and the context sensitive menu option, Alter Routine, selected, the following error was generated:

```
Unable to load object with error: Object reference not
set to an instance of an object
```

(Bug #55170)

- Calling `MySqlDataAdapter.Update(DataTable)` resulted in an unacceptable performance hit when updating large amounts of data. (Bug #55609)

A.4.7. Changes in MySQL Connector/Net 6.3.3 (2010-07-27)

This release fixes bugs since 6.3.2.

Bugs Fixed

- Membership schema creation failed if the default schema collation was not Latin1. (Bug #53174)

- `CHAR(36)` columns were not recognized as GUIDs when used in views with entity models. (Bug #52085)
- MySQL Connector/Net 6.3.2 failed to install on Windows Vista. (Bug #53975)
- When batching was used in `MySqlDataAdapter`, a connection was not opened automatically in `MySqlDataAdapter.Update()`. This resulted in an `InvalidOperationException` exception being generated, with the message text "connection must be valid and open".

MySQL Connector/Net has been changed to behave more like SQL Server: if the connection is closed, it is opened for the duration of update operation. (Bug #38411)

- MySQL Connector/Net did not process `Thread.Abort()` correctly, and failed to cancel queries currently running on the server. (Bug #54012)
- MySQL Connector/Net did not throw an `EndOfStreamException` exception when `net_write_timeout` was exceeded. (Bug #53439)
- The icon for the MySQL Web Configuration Tool was not displayed in Visual Studio for Web Application Projects. (Bug #54571)
- Database name was emitted into typed datasets. This prevented users using the configured default database. (Bug #33870)
- `MySqlDataAdapter.Update()` generated concurrency violations for custom stored procedure driven update commands that used `UpdateRowSource.FirstReturnedRecord`. (Bug #54895)
- Several calls to `DataAdapter.Update()` with intervening changes to `DataTable` resulted in `ConcurrencyException` exceptions being generated. (Bug #54863)
- MySQL Connector/Net generated a null reference exception when `TransactionScope` was used by multiple threads. (Bug #54681)
- The MySQL Connector/Net installation failed due to `machine.config` files not being present in configuration folders.

MySQL Connector/Net has been changed to skip over configuration folders that do not contain a `machine.config` file. (Bug #52352)

- If `MySqlDataAdapter` was used with an `INSERT` command where the `VALUES` clause contained an expression with parentheses in it, and set the `adapter.UpdateBatchSize` parameter to be greater than one, then the call to `adapter.Update` either generated an exception or failed to batch the commands, executing each insert individually. (Bug #54386)
- The `MySqlHelper` object did not have an overloaded version of the `ExecuteReader` method that accepted a `MySqlConnection` object. (Bug #54570)
- Garbage Collector disposal of a `MySqlConnection` object caused the following exception:

```
System.IO.EndOfStreamException: Attempted to read past the end of the stream.
MySQL.Data.MySqlClient.MySqlStream.ReadFully(Stream stream, Byte[] buffer, Int32 offset,
Int32 count)
MySQL.Data.MySqlClient.MySqlStream.LoadPacket()
Outer Exception Reading from the stream has failed.
...
```

(Bug #53457)

- After a timeout exception, if an attempt was made to reuse a connection returned to the connection pool the following exception was generated:

```
[MySqlException (0x80004005): There is already an open DataReader associated with this
Connection which must be closed first.]
MySQL.Data.MySqlClient.MySqlCommand.CheckState() +278
MySQL.Data.MySqlClient.MySqlCommand.ExecuteReader(CommandBehavior behavior) +43
```

```

MySQL.Data.MySqlClient.MySqlCommand.ExecuteReader() +6
Controls.SimpleCommand.ExecuteReader(String SQL) in ...:323
Albums.GetImagesByAlbum(SimpleCommand Cmd, Int32 iAlbum, String Order, String Limit)
in ...:13
Forecast.Page_Load(Object sender, EventArgs e) in ...:70
System.Web.UI.Control.OnLoad(EventArgs e) +99
System.Web.UI.Control.LoadRecursive() +50
System.Web.UI.Page.ProcessRequestMain(Boolean includeStagesBeforeAsyncPoint, Boolean
includeStagesAfterAsyncPoint) +627

```

(Bug #53357)

- The method `MySQL.Data.Common.QueryNormalizer.CollapseValueList` generated an `ArgumentOutOfRangeException`. (Bug #54152, Bug #53865)
- When an application was subjected to increased concurrent load, MySQL Connector/Net generated the following error when calling stored procedures:

```

A DataTable named \'Procedure Parameters\'
already belongs to this DataSet.

```

(Bug #49118)

- When the connection string option “Connection Reset = True” was used, a connection reset used the previously used encoding for the subsequent authentication operation. This failed, for example, if UCS2 was used to read the last column before the reset. (Bug #47153)

A.4.8. Changes in MySQL Connector/Net 6.3.2 (2010-05-24, Beta)

First Beta release. This release fixes bugs since 6.3.1.

Functionality Added or Changed

- Procedure caching had a problem whereby if you created a procedure, dropped it, and recreated it with a different number of parameters an exception was generated.

MySQL Connector/Net has been changed so that if the procedure is recreated with a different number of parameters, it will still be recognized. (Bug #52562)

- MySQL Connector/Net has been changed to include `MySQLDataReader.GetFieldType(string columnname)`. Further, `MySQLDataReader.GetOrdinal()` now includes the name of the column in the exception if the column is not found. (Bug #47467)

Bugs Fixed

- After an exception, the internal datareader, `MySQLCommand.Connection.Reader`, was not properly closed (it was not set to null). If another query was subsequently executed on that command object an exception was generated with the message “There is already an open DataReader associated with this Connection which must be closed first.” (Bug #55558)
- MySQL Connector/Net generated an exception when used to read a `TEXT` column containing more than 32767 bytes. (Bug #54040)
- In MySQL Connector/Net, the `MySQLConnection.Abort()` method contained a `try...catch` construct, with an empty `catch` block. This meant that any exception generated at this point would not be caught. (Bug #52769)
- The procedure cache affected the MySQL Connector/Net performance, reducing it by around 65%. This was due to unnecessary calls of `String.Format()`, related to debug logging. Even though the logging was disabled the string was still being formatted, resulting in impaired performance. (Bug #52475)
- If `FunctionsReturnString=true` was used in the connection string, the decimal separator (according to locale) was not interpreted. (Bug #52187)

- In MySQL Connector/Net, the `LoadCharsetMap()` function of the `CharSetMap` class set the following incorrect mapping:

```
mapping.Add("latin1", new CharacterSet("latin1", 1));
```

This meant that, for example, the Euro sign was not handled correctly.

The correct mapping should have been:

```
mapping.Add("latin1", new CharacterSet("windows-1252", 1));
```

This is because MySQL's `latin1` character set is the same as the `windows-cp1252` character set and it extends the official ISO 8859-1 or IANA latin1. (Bug #51927)

- A non-terminated string in SQL threw a CLR exception rather than a syntax exception. (Bug #51788)
- When using table per type inheritance and listing the contents of the parent table, the result of the query was a list of child objects, even though there was no related child record with the same parent Id. (Bug #49850)
- MySQL Connector/Net 6.3.1 failed to install. (Bug #51407, Bug #51604)
- When calling `ExecuteNonQuery` on a command object, the following exception occurred:

```
Index and length must refer to a location within the string.  
Parameter name: length
```

(Bug #51610)

A.4.9. Changes in MySQL Connector/Net 6.3.1 (2010-03-02)

This release fixes bugs since 6.3.0.

Functionality Added or Changed

- Connector/Net was not compatible with Visual Studio wizards that used square brackets to delimit symbols.

Connector/Net has been changed to include a new connection string option `Sql Server mode` that supports use of square brackets to delimit symbols. (Bug #35852)

Bugs Fixed

- Binary Columns were not displayed in the Query Builder of Visual Studio. (Bug #50171)
- `MySqlCommand.Parameters.Clear()` did not work. (Bug #50444)
- When the `UpdateBatchSize` property was set to a value greater than 1, only the first row was applied to the database. (Bug #50123)
- The method `Command.TrimSemicolons` used `StringBuilder`, and therefore allocated memory for the query even if it did not need to be trimmed. This led to excessive memory consumption when executing a number of large queries. (Bug #51149)
- Specifying a connection string where an option had no value generated an error, rather than the value being set to the default. For example, a connection string such as the following would result in an error:

```
server=localhost;user=root;compress=;database=test;port=3306;password=123456;
```

(Bug #51209)

- A connection string set in `web.config` could not be reused after Visual Studio 2008 Professional was shut down. It continued working for the existing controls, but did not work for new controls added. (Bug #41629)

- When trying to create stored procedures from an SQL script, a `MySqlException` was thrown when attempting to redefine the `DELIMITER`:

```
MySql.Data.MySqlClient.MySqlException was unhandled
Message="You have an error in your SQL syntax; check the manual that corresponds to your
MySQL server version for the right syntax to use near 'DELIMITER' at line 1"
Source="MySql.Data"
ErrorCode=-2147467259
Number=1064
StackTrace:
à MySql.Data.MySqlClient.MySqlStream.ReadPacket()
à MySql.Data.MySqlClient.NativeDriver.ReadResult(UInt64& affectedRows, Int64&
lastInsertId)
à MySql.Data.MySqlClient.MySqlDataReader.GetResultSet()
à MySql.Data.MySqlClient.MySqlDataReader.NextResult()
à MySql.Data.MySqlClient.MySqlCommand.ExecuteReader(CommandBehavior behavior)
à MySql.Data.MySqlClient.MySqlCommand.ExecuteReader()
à MySql.Data.MySqlClient.MySqlCommand.ExecuteNonQuery()
à MySql.Data.MySqlClient.MySqlScript.Execute()
```

Note: The `MySqlScript` class has been fixed to support the delimiter statement as it is found in SQL scripts. (Bug #46429)

A.4.10. Changes in MySQL Connector/Net 6.3.0 (2010-02-16, Alpha)

First alpha release of 6.3.

Functionality Added or Changed

- Nested transaction scopes were not supported. MySQL Connector/Net now implements nested transaction scopes. A per-thread stack of scopes is maintained, which is necessary to handle nested scopes with the `RequiresNew` or `Suppress` options. (Bug #45098)
- Support for MySQL Server 4.1 has been removed from MySQL Connector/Net starting with version 6.3.0. The connector will now throw an exception if you try to connect to a server of version less than 5.0.

Bugs Fixed

- The method `MySqlDataReader.GetSchemaTable()` returned 0 in the `NumericPrecision` field for decimal and newdecimal columns. (Bug #48171)
- Attempting to execute a load data local infile on a file where the user did not have write permissions, or the file was open in an editor gave an access denied error. (Bug #48944)
- When adding a data set in Visual Studio 2008, the following error was generated:

```
Relations couldn't be added. Column 'REFERENCED_TABLE_CATALOG' does not belong to table.
```

This was due to a 'REFERENCED_TABLE_CATALOG' column not being included in the foreign keys collection. (Bug #48974)

A.5. Changes in MySQL Connector/Net Version 6.2

A.5.1. Changes in MySQL Connector/Net 6.2.6 (Not yet released)

This release fixes bugs since 6.2.5.

Version 6.2.6 has no changelog entries.

A.5.2. Changes in MySQL Connector/Net 6.2.5 (2011-07-01)

This release fixes bugs since 6.2.4.

Bugs Fixed

- `MySQLConnectionStringBuilder.ContainsKey()` incorrectly returned `false` when testing whether a keyword was part of the connection string. (Bug #11766671, Bug #59835)
- Modified `ProviderManifest.xml` to map `TIMESTAMP` database columns to the `DateTime` .NET type. (Bug #55351, Bug #12652602)
- Modified `MySQLConnection.BeginTransaction` to throw a `NotSupportedException` for `Snapshot` isolation level. (Bug #61589, Bug #12698020)
- Fixed Entity Framework provider `GROUP BY` clause generation by adding all group-by keys to the `SELECT` statement. (Bug #46742, Bug #12622129)
- `SchemaDefinition-5.5.ssd1` was modified to treat `CHAR(36)` columns as a GUID. (Bug #61657, Bug #12708208)
- `SqlFragment.QuoteIdentifier` was modified to add MySQL quotes around identifiers. (Bug #61635, Bug #12707285)
- A `NullReferenceException` was thrown on disposal of a `TransactionScope` object. (Bug #59346, Bug #11766272)
- MySQL Connector/Net generated an exception when executing a query consisting of ';', for example:

```
mycmd( ";", mycon)
mycmd.ExecuteNonQuery()
```

The exception generated was:

```
System.IndexOutOfRangeException: Index was outside the bounds of the array.
   at MySql.Data.MySqlClient.MySqlCommand.TrimSemicolons(String sql)
   at MySql.Data.MySqlClient.MySqlCommand.ExecuteReader(CommandBehavior behavior)
   at MySql.Data.MySqlClient.MySqlCommand.ExecuteReader()
   at MySql.Data.MySqlClient.MySqlCommand.ExecuteNonQuery()
```

(Bug #59537, Bug #11766433)

- A typed dataset did not get the table name. (Bug #57894, Bug #11764989)
- `MySqlDataReader.GetSchemaTable` returned incorrect values and types. (Bug #59989, Bug #11776346)
- Setting `Membership.ApplicationName` had no effect. (Bug #59438, Bug #11770465)
- All queries other than `INSERT` were executed individually instead of as a batch even though batching was enabled for the connection. (Bug #59616, Bug #11850286)
- The `MySQLTokenizer` contained unnecessary `Substring` and `Trim` calls:

```
string token = sql.Substring(startIndex, stopIndex - startIndex).Trim();
```

The variable `token` was not used anywhere in the code. (Bug #58757)

- `MembershipProvider` did not generate hashes correctly if the algorithm was keyed. The Key of the algorithm should have been set if the `HashAlgorithm` was `KeyedHashAlgorithm`. (Bug #58906)
- `MySQLCommand.ExecuteReader(CommandBehavior)` threw a `NullReferenceException` when being called with `CommandBehavior.CloseConnection`, if the SQL statement contained a syntax error, or contained invalid data such as an invalid column name. (Bug #58652)
- The ADO.NET Entity Data Model could not add stored procedures from MySQL Server 5.0.45 but worked fine using MySQL Server 5.1. (Bug #55349)
- Code introduced to fix bug #54863 proved problematic on .NET version 3.5 and above. (Bug #58853)

- `ReadFieldLength()` returned incorrect value for `BIGINT` autoincrement columns. (Bug #58373)
- MySQL Connector/Net did not support the `utf8mb4` character set. When attempting to connect to `utf8mb4` tables or columns, an exception `KeyNotFoundException` was generated. (Bug #58244)
- Default values returned for text columns were not quoted. This meant that the `COLUMN_DEFAULT` field of the `GetSchema` columns collection did not return a valid SQL expression. (Bug #56509)
- MySQL Connector/Net for .NET/Mono attempted to dynamically load the assembly `Mono.Posix.dll` when a Unix socket was used to connect to the server. This failed and the connector was not able to use a Unix socket unless the `Mono.Posix.dll` assembly was previously loaded by the program. (Bug #56410)
- Setting `MySqlCommand.CommandTimeout` to 0 had no effect. It should have resulted in an infinite timeout. (Bug #57265)
- When an output parameter was declared as type `MySqlDbType.Bit`, it failed to return with the correct value. (Bug #56756)
- When performing a row-by-row update, only the first row was updated and all other rows were ignored. (Bug #57092)
- Setting the `Default Command Timeout` connection string option had no effect. (Bug #56806)
- `MySqlHelper.ExecuteReader` did not include an overload accepting `MySqlParameter` objects when using a `MySqlConnection`. However, `MySqlHelper` did include an overload for `MySqlParameter` objects when using a string object containing the connection string to the database. (Bug #56755)

A.5.3. Changes in MySQL Connector/Net 6.2.4 (2010-08-30)

This release fixes bugs since 6.2.3.

Functionality Added or Changed

- Procedure caching had a problem whereby if you created a procedure, dropped it, and recreated it with a different number of parameters an exception was generated.

MySQL Connector/Net has been changed so that if the procedure is recreated with a different number of parameters, it will still be recognized. (Bug #52562)

Bugs Fixed

- In the ADO.NET Entity Data Model Wizard, the time to update a model scaled abnormally as the number of entities increased. (Bug #48791, Bug #12596237)
- Opening a connection in the Visual Studio Server Explorer and choosing to alter an existing routine required another authentication at the server. (Bug #44715)
- Stored procedure enumeration code generated an error if a procedure was used in a dataset that did not return any resultsets. (Bug #50671)
- EventLog was not disposed in the SessionState provider. (Bug #52550)
- Attempting to read `Double.MinValue` from a `DOUBLE` column in MySQL table generated the following exception:

```
System.OverflowException : Value was either too large or too small for a Double.
--OverflowException
at System.Number.ParseDouble(String value, NumberStyles options, NumberFormatInfo
numfmt)
at MySql.Data.Types.MySqlDouble.MySql.Data.Types.IMySqlValue.ReadValue(MySqlPacket
```



```
packet, Int64 length, Boolean nullVal)
at MySql.Data.MySqlClient.NativeDriver.ReadColumnValue(Int32 index, MySqlField field,
IMySqlValue valObject)
at MySql.Data.MySqlClient.ResultSet.ReadColumnData(Boolean outputParms)
at MySql.Data.MySqlClient.ResultSet.NextRow(CommandBehavior behavior)
at MySql.Data.MySqlClient.MySqlDataReader.Read()
```

(Bug #55644)

- The calculation of `lockAge` in the Session Provider sometimes generated a `System.Data.SqlTypes.SqlNullValueException`. (Bug #55701)
- The `INSERT` command was significantly slower with MySQL Connector/Net 6.x compared to 5.x, when compression was enabled. (Bug #48243)
- If using MySQL Server 5.0.x it was not possible to alter stored routines in Visual Studio. If the stored routine was clicked, and the context sensitive menu option, Alter Routine, selected, the following error was generated:

```
Unable to load object with error: Object reference not
set to an instance of an object
```

(Bug #55170)

- After an exception, the internal datareader, `MySqlCommand.Connection.Reader`, was not properly closed (it was not set to null). If another query was subsequently executed on that command object an exception was generated with the message "There is already an open DataReader associated with this Connection which must be closed first." (Bug #55558)
- Membership schema creation failed if the default schema collation was not Latin1. (Bug #53174)
- `CHAR(36)` columns were not recognized as GUIDs when used in views with entity models. (Bug #52085)
- When batching was used in `MySqlDataAdapter`, a connection was not opened automatically in `MySqlDataAdapter.Update()`. This resulted in an `InvalidOperationException` exception being generated, with the message text "connection must be valid and open".

MySQL Connector/Net has been changed to behave more like SQL Server: if the connection is closed, it is opened for the duration of update operation. (Bug #38411)

- MySQL Connector/Net did not process `Thread.Abort()` correctly, and failed to cancel queries currently running on the server. (Bug #54012)
- MySQL Connector/Net did not throw an `EndOfStreamException` exception when `net_write_timeout` was exceeded. (Bug #53439)
- The icon for the MySQL Web Configuration Tool was not displayed in Visual Studio for Web Application Projects. (Bug #54571)
- Database name was emitted into typed datasets. This prevented users using the configured default database. (Bug #33870)
- `MySqlDataAdapter.Update()` generated concurrency violations for custom stored procedure driven update commands that used `UpdateRowSource.FirstReturnedRecord`. (Bug #54895)
- Several calls to `DataAdapter.Update()` with intervening changes to `DataTable` resulted in `ConcurrencyException` exceptions being generated. (Bug #54863)
- If `MySqlDataAdapter` was used with an `INSERT` command where the `VALUES` clause contained an expression with parentheses in it, and set the `adapter.UpdateBatchSize` parameter to be greater than one, then the call to `adapter.Update` either generated an exception or failed to batch the commands, executing each insert individually. (Bug #54386)

- The `MySqlHelper` object did not have an overloaded version of the `ExecuteReader` method that accepted a `MySqlConnection` object. (Bug #54570)
- Garbage Collector disposal of a `MySqlConnection` object caused the following exception:

```
System.IO.EndOfStreamException: Attempted to read past the end of the stream.
MySql.Data.MySqlClient.MySqlStream.ReadFully(Stream stream, Byte[] buffer, Int32 offset,
Int32 count)
MySql.Data.MySqlClient.MySqlStream.LoadPacket()
Outer Exception Reading from the stream has failed.
...
```

(Bug #53457)

- After a timeout exception, if an attempt was made to reuse a connection returned to the connection pool the following exception was generated:

```
[MySqlException (0x80004005): There is already an open DataReader associated with this
Connection which must be closed first.]
  MySql.Data.MySqlClient.MySqlCommand.CheckState() +278
  MySql.Data.MySqlClient.MySqlCommand.ExecuteReader(CommandBehavior behavior) +43
  MySql.Data.MySqlClient.MySqlCommand.ExecuteReader() +6
  Controls.SimpleCommand.ExecuteReader(String SQL) in ...:323
  Albums.GetImagesByAlbum(SimpleCommand Cmd, Int32 iAlbum, String Order, String Limit)
in ...:13
  Forecast.Page_Load(Object sender, EventArgs e) in ...:70
  System.Web.UI.Control.OnLoad(EventArgs e) +99
  System.Web.UI.Control.LoadRecursive() +50
  System.Web.UI.Page.ProcessRequestMain(Boolean includeStagesBeforeAsyncPoint, Boolean
includeStagesAfterAsyncPoint) +627
```

(Bug #53357)

- The method `MySql.Data.Common.QueryNormalizer.CollapseValueList` generated an `ArgumentOutOfRangeException`. (Bug #54152, Bug #53865)
- When an application was subjected to increased concurrent load, MySQL Connector/Net generated the following error when calling stored procedures:

```
A DataTable named \'Procedure Parameters\'
already belongs to this DataSet.
```

(Bug #49118)

- In MySQL Connector/Net, the `MySqlConnection.Abort()` method contained a `try...catch` construct, with an empty `catch` block. This meant that any exception generated at this point would not be caught. (Bug #52769)
- The procedure cache affected the MySQL Connector/Net performance, reducing it by around 65%. This was due to unnecessary calls of `String.Format()`, related to debug logging. Even though the logging was disabled the string was still being formatted, resulting in impaired performance. (Bug #52475)
- If `FunctionsReturnString=true` was used in the connection string, the decimal separator (according to locale) was not interpreted. (Bug #52187)
- In MySQL Connector/Net, the `LoadCharsetMap()` function of the `CharsetMap` class set the following incorrect mapping:

```
mapping.Add("latin1", new CharacterSet("latin1", 1));
```

This meant that, for example, the Euro sign was not handled correctly.

The correct mapping should have been:

```
mapping.Add("latin1", new CharacterSet("windows-1252", 1));
```

This is because MySQL's `latin1` character set is the same as the `windows-cp1252` character set and it extends the official ISO 8859-1 or IANA latin1. (Bug #51927)

- Periodically the session provider threw an `SqlNullValueException` exception. When this happened, the row within the `my_aspnet_Sessions` table had `locked` always set to '1'. The locked status never changed back to '0' and the user experienced the exception on every page, until their browser was closed and reopened (recreating a new sessionID), or the `locked` value was manually changed to '0'. (Bug #52175)
- When the connection string option "Connection Reset = True" was used, a connection reset used the previously used encoding for the subsequent authentication operation. This failed, for example, if UCS2 was used to read the last column before the reset. (Bug #47153)

A.5.4. Changes in MySQL Connector/Net 6.2.3 (2010-04-10)

This release fixes bugs since 6.2.2.

Functionality Added or Changed

- MySQL Connector/Net has been changed to include `MySqlDataReader.GetFieldType(string columnname)`. Further, `MySqlDataReader.GetOrdinal()` now includes the name of the column in the exception if the column is not found. (Bug #47467)

Bugs Fixed

- A non-terminated string in SQL threw a CLR exception rather than a syntax exception. (Bug #51788)
- When using table per type inheritance and listing the contents of the parent table, the result of the query was a list of child objects, even though there was no related child record with the same parent Id. (Bug #49850)
- When calling `ExecuteNonQuery` on a command object, the following exception occurred:

```
Index and length must refer to a location within the string.  
Parameter name: length
```

(Bug #51610)

- Binary Columns were not displayed in the Query Builder of Visual Studio. (Bug #50171)
- `MySqlCommand.Parameters.Clear()` did not work. (Bug #50444)
- When the `UpdateBatchSize` property was set to a value greater than 1, only the first row was applied to the database. (Bug #50123)
- The method `Command.TrimSemicolons` used `StringBuilder`, and therefore allocated memory for the query even if it did not need to be trimmed. This led to excessive memory consumption when executing a number of large queries. (Bug #51149)
- Specifying a connection string where an option had no value generated an error, rather than the value being set to the default. For example, a connection string such as the following would result in an error:

```
server=localhost;user=root;compress=:database=test;port=3306;password=123456;
```

(Bug #51209)

- A connection string set in `web.config` could not be reused after Visual Studio 2008 Professional was shut down. It continued working for the existing controls, but did not work for new controls added. (Bug #41629)
- When the `MySqlCommand.execute()` method was called, the following exception was generated:

```
InvalidOperationException : The CommandText property has not been properly initialized.
```

(Bug #50344)

- When using the Compact Framework the following exception occurred when attempting to connect to a MySQL Server:

```
System.InvalidOperationException was unhandled  
Message="Timeouts are not supported on this stream."
```

(Bug #50321)

- A [FormatException](#) was generated when an empty string was returned from a stored function. (Bug #49642)
- [MySqlDataReader.GetUInt64](#) returned an incorrect value when reading a [BIGINT UNSIGNED](#) column containing a value greater than 2147483647. (Bug #49794)
- Calling a User Defined Function using Entity SQL in the Entity Framework caused a [NullReferenceException](#). (Bug #45277)
- When trying to create stored procedures from an SQL script, a [MySqlException](#) was thrown when attempting to redefine the [DELIMITER](#):

```
MySql.Data.MySqlClient.MySqlException was unhandled  
Message="You have an error in your SQL syntax; check the manual that corresponds to your  
MySQL server version for the right syntax to use near 'DELIMITER' at line 1"  
Source="MySql.Data"  
ErrorCode=-2147467259  
Number=1064  
StackTrace:  
à MySql.Data.MySqlClient.MySqlStream.ReadPacket()  
à MySql.Data.MySqlClient.NativeDriver.ReadResult(UInt64& affectedRows, Int64&  
lastInsertId)  
à MySql.Data.MySqlClient.MySqlDataReader.GetResultSet()  
à MySql.Data.MySqlClient.MySqlDataReader.NextResult()  
à MySql.Data.MySqlClient.MySqlCommand.ExecuteReader(CommandBehavior behavior)  
à MySql.Data.MySqlClient.MySqlCommand.ExecuteReader()  
à MySql.Data.MySqlClient.MySqlCommand.ExecuteNonQuery()  
à MySql.Data.MySqlClient.MySqlScript.Execute()
```

Note: The [MySqlScript](#) class has been fixed to support the delimiter statement as it is found in SQL scripts. (Bug #46429)

A.5.5. Changes in MySQL Connector/Net 6.2.2 (2009-12-22, Generally Available)

First GA release of 6.2. This release fixes bugs since 6.2.1.

Bugs Fixed

- MySQL Connector/Net generated an invalid operation exception during a transaction rollback:

```
System.InvalidOperationException: Connection must be valid and open to rollback  
transaction  
at MySql.Data.MySqlClient.MySqlTransaction.Rollback()  
at MySql.Data.MySqlClient.MySqlConnection.CloseFully()  
at  
MySql.Data.MySqlClient.MySqlPromotableTransaction.System.Transactions.IPromotableSinglePhaseNotification.  
SinglePhaseEnlistment)  
...
```

(Bug #35330)

- The method [MySqlDataReader.GetSchemaTable\(\)](#) returned 0 in the [NumericPrecision](#) field for decimal and newdecimal columns. (Bug #48171)

- Attempting to execute a load data local infile on a file where the user did not have write permissions, or the file was open in an editor gave an access denied error. (Bug #48944)
- When adding a data set in Visual Studio 2008, the following error was generated:

```
Relations couldn't be added. Column 'REFERENCED_TABLE_CATALOG' does not belong to table.
```

This was due to a 'REFERENCED_TABLE_CATALOG' column not being included in the foreign keys collection. (Bug #48974)

- Connection objects were not garbage collected when not in use. (Bug #31996)

A.5.6. Changes in MySQL Connector/Net 6.2.1 (2009-11-16, Beta)

This release fixes bugs since 6.2.0.

Functionality Added or Changed

- The `MySqlParameter` class now has a property named `PossibleValues`. This property is NULL unless the parameter is created by `MySqlCommandBuilder.DeriveParameters`. Further, it will be NULL unless the parameter is of type enum or set - in this case it will be a list of strings that are the possible values for the column. This feature is designed as an aid to the developer. (Bug #48586)
- MySQL Connector/Net now supports the processing of certificates when connecting to an SSL-enabled MySQL Server. For further information see the connection string option SSL Mode in the section [Chapter 7, Connector/Net Connection String Options Reference](#) and the tutorial [Section 5.7, "Tutorial: Using SSL with MySQL Connector/Net"](#).
- 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.

- 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.

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.

Further details on this can be found in the relevant [Microsoft documentation](#).

Bugs Fixed

- When building the `MySql.Data` project with .NET Framework 3.5 installed, the following build output was displayed:

```
Project file contains ToolsVersion="4.0", which is not supported by this version of MSBuild. Treating the project as if it had ToolsVersion="3.5".
```

The project had been created using the .NET Framework 4.0, which was beta, instead of using the 3.5 framework. (Bug #48271)

- When used, the `Encrypt` connection string option caused a "Keyword not supported" exception to be generated.

This option is in fact obsolete, and the option `SSL Mode` should be used instead. Although the `Encrypt` option has been fixed so that it does not generate an exception, it will be removed completely in version 6.4. (Bug #48290)

- Cloning of `MySqlCommand` was not typesafe. To clone a `MySqlCommand` it was necessary to do:

```
MySqlCommand clone = (MySqlCommand)((ICloneable)comm).Clone();
```

MySQL Connector/Net was changed so that it was possible to do:

```
MySqlCommand clone = comm.Clone();
```

(Bug #48460)

- An entity model created from a schema containing a table with a column of type `UNSIGNED BIGINT` and a view of the table did not behave correctly. When an entity was created and mapped to the view, the column that was of type `UNSIGNED BIGINT` was displayed as `BIGINT`. (Bug #47872)
- It was not possible to retrieve a value from a MySQL server table, if the value was larger than that supported by the .NET type `System.Decimal`.

MySQL Connector/Net was changed to expose the `MySqlDecimal` type, along with the supporting method `GetMySqlDecimal`. (Bug #48100)

- MySQL Connector/Net session support did not work with MySQL Server versions prior to 5.0, as the Session Provider used a call to `TIMESTAMPDIFF`, which was not available on servers prior to 5.0. (Bug #47219)

A.5.7. Changes in MySQL Connector/Net 6.2.0 (2009-10-21, Alpha)

The first alpha release of 6.2.

Bugs Fixed

- When using a `BINARY(16)` column to represent a GUID and having specified “old guids = true” in the connection string, the values were returned correctly until a null value was encountered in that field. After the null value was encountered a format exception was thrown with the following message:

```
Guid should contain 32 digits with 4 dashes (xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx).
```

(Bug #47928)

- The Session Provider created invalid “session expires” on a random basis.

This was due to the fact that the Session Provider was incorrectly reading from the root `web.config`, rather than from the application specific `web.config`. (Bug #47815)

- When loading the `MySQLClient-mono.sln` file included with the Connector/Net source into Mono Develop, the following error occurred:

```
/home/tbedford/connector-net-src/6.1/MySQLClient-mono.sln(22):
Unsupported or unrecognized project:
'/home/tbedford/connector-net-src/6.1/Installer/Installer.wixproj'
```

If the file was modified to remove this problem, then attempting to build the solution generated the following error:

```
/home/tbedford/connector-net-src/6.1/MySQL.Data/Provider/Source/Connection.cs(280,46):
error CS0115: 'MySQL.Data.MySqlClient.MySqlConnection.DbProviderFactory' is marked as an
override but no suitable property found to override
```

(Bug #47048)

A.6. Changes in MySQL Connector/Net Version 6.1

A.6.1. Changes in MySQL Connector/Net 6.1.7 (Not released, Generally Available)

This release fixes bugs since 6.1.6.

Version 6.1.7 has no changelog entries.

A.6.2. Changes in MySQL Connector/Net 6.1.6 (Not released, Generally Available)

This release fixes bugs since 6.1.5.

Version 6.1.6 has no changelog entries.

A.6.3. Changes in MySQL Connector/Net 6.1.5 (2010-08-30, Generally Available)

This release fixes bugs since 6.1.4.

Bugs Fixed

- In the ADO.NET Entity Data Model Wizard, the time to update a model scaled abnormally as the number of entities increased. (Bug #48791, Bug #12596237)
- Opening a connection in the Visual Studio Server Explorer and choosing to alter an existing routine required another authentication at the server. (Bug #44715)
- Stored procedure enumeration code generated an error if a procedure was used in a dataset that did not return any resultsets. (Bug #50671)
- EventLog was not disposed in the SessionState provider. (Bug #52550)
- Attempting to read `Double.MinValue` from a `DOUBLE` column in MySQL table generated the following exception:

```
System.OverflowException : Value was either too large or too small for a Double.

--OverflowException
at System.Number.ParseDouble(String value, NumberStyles options, NumberFormatInfo numfmt)
at MySql.Data.Types.MySqlDouble.MySql.Data.Types.IMySqlValue.ReadValue(MySqlPacket packet, Int64 length, Boolean nullVal)
at MySql.Data.MySqlClient.NativeDriver.ReadColumnValue(Int32 index, MySqlField field, IMySqlValue valObject)
at MySql.Data.MySqlClient.ResultSet.ReadColumnData(Boolean outputParms)
at MySql.Data.MySqlClient.ResultSet.NextRow(CommandBehavior behavior)
at MySql.Data.MySqlClient.MySqlDataReader.Read()
```

(Bug #55644)

- The calculation of `lockAge` in the Session Provider sometimes generated a `System.Data.SqlTypes.SqlNullValueException`. (Bug #55701)
- The `INSERT` command was significantly slower with MySQL Connector/Net 6.x compared to 5.x, when compression was enabled. (Bug #48243)
- If using MySQL Server 5.0.x it was not possible to alter stored routines in Visual Studio. If the stored routine was clicked, and the context sensitive menu option, Alter Routine, selected, the following error was generated:

```
Unable to load object with error: Object reference not
```

set to an instance of an object

(Bug #55170)

- Membership schema creation failed if the default schema collation was not Latin1. (Bug #53174)
- `CHAR(36)` columns were not recognized as GUIDs when used in views with entity models. (Bug #52085)
- When batching was used in `MySqlDataAdapter`, a connection was not opened automatically in `MySqlDataAdapter.Update()`. This resulted in an `InvalidOperationException` exception being generated, with the message text "connection must be valid and open".

MySQL Connector/Net has been changed to behave more like SQL Server: if the connection is closed, it is opened for the duration of update operation. (Bug #38411)

- MySQL Connector/Net did not throw an `EndOfStreamException` exception when `net_write_timeout` was exceeded. (Bug #53439)
- The icon for the MySQL Web Configuration Tool was not displayed in Visual Studio for Web Application Projects. (Bug #54571)
- Database name was emitted into typed datasets. This prevented users using the configured default database. (Bug #33870)
- `MySqlDataAdapter.Update()` generated concurrency violations for custom stored procedure driven update commands that used `UpdateRowSource.FirstReturnedRecord`. (Bug #54895)
- Several calls to `DataAdapter.Update()` with intervening changes to `DataTable` resulted in `ConcurrencyException` exceptions being generated. (Bug #54863)
- If `MySqlDataAdapter` was used with an `INSERT` command where the `VALUES` clause contained an expression with parentheses in it, and set the `adapter.UpdateBatchSize` parameter to be greater than one, then the call to `adapter.Update` either generated an exception or failed to batch the commands, executing each insert individually. (Bug #54386)
- The `MySqlHelper` object did not have an overloaded version of the `ExecuteReader` method that accepted a `MySqlConnection` object. (Bug #54570)
- Garbage Collector disposal of a `MySqlConnection` object caused the following exception:

```
System.IO.EndOfStreamException: Attempted to read past the end of the stream.
MySQL.Data.MySqlClient.MySqlStream.ReadFully(Stream stream, Byte[] buffer, Int32 offset,
Int32 count)
MySQL.Data.MySqlClient.MySqlStream.LoadPacket()
Outer Exception Reading from the stream has failed.
...
```

(Bug #53457)

- After a timeout exception, if an attempt was made to reuse a connection returned to the connection pool the following exception was generated:

```
[MySqlException (0x80004005): There is already an open DataReader associated with this
Connection which must be closed first.]
MySQL.Data.MySqlClient.MySqlCommand.CheckState() +278
MySQL.Data.MySqlClient.MySqlCommand.ExecuteReader(CommandBehavior behavior) +43
MySQL.Data.MySqlClient.MySqlCommand.ExecuteReader() +6
Controls.SimpleCommand.ExecuteReader(String SQL) in ...:323
Albums.GetImagesByAlbum(SimpleCommand Cmd, Int32 iAlbum, String Order, String Limit)
in ...:13
Forecast.Page_Load(Object sender, EventArgs e) in ...:70
System.Web.UI.Control.OnLoad(EventArgs e) +99
System.Web.UI.Control.LoadRecursive() +50
System.Web.UI.Page.ProcessRequestMain(Boolean includeStagesBeforeAsyncPoint, Boolean
includeStagesAfterAsyncPoint) +627
```


(Bug #53357)

- The method `MySQL.Data.Common.QueryNormalizer.CollapseValueList` generated an `ArgumentOutOfRangeException`. (Bug #54152, Bug #53865)
- When an application was subjected to increased concurrent load, MySQL Connector/Net generated the following error when calling stored procedures:

```
A DataTable named \'Procedure Parameters\'
already belongs to this DataSet.
```

(Bug #49118)

- When the connection string option "Connection Reset = True" was used, a connection reset used the previously used encoding for the subsequent authentication operation. This failed, for example, if UCS2 was used to read the last column before the reset. (Bug #47153)

A.6.4. Changes in MySQL Connector/Net 6.1.4 (2010-04-28, Generally Available)

This release fixes bugs since 6.1.3.

Functionality Added or Changed

- Procedure caching had a problem whereby if you created a procedure, dropped it, and recreated it with a different number of parameters an exception was generated.

MySQL Connector/Net has been changed so that if the procedure is recreated with a different number of parameters, it will still be recognized. (Bug #52562)

- MySQL Connector/Net has been changed to include `MySQLDataReader.GetFieldType(string columnname)`. Further, `MySQLDataReader.GetOrdinal()` now includes the name of the column in the exception if the column is not found. (Bug #47467)

Bugs Fixed

- In MySQL Connector/Net, the `MySQLConnection.Abort()` method contained a `try...catch` construct, with an empty `catch` block. This meant that any exception generated at this point would not be caught. (Bug #52769)
- If `FunctionsReturnString=true` was used in the connection string, the decimal separator (according to locale) was not interpreted. (Bug #52187)
- In MySQL Connector/Net, the `LoadCharsetMap()` function of the `CharSetMap` class set the following incorrect mapping:

```
mapping.Add("latin1", new CharSet("latin1", 1));
```

This meant that, for example, the Euro sign was not handled correctly.

The correct mapping should have been:

```
mapping.Add("latin1", new CharSet("windows-1252", 1));
```

This is because MySQL's `latin1` character set is the same as the `windows-cp1252` character set and it extends the official ISO 8859-1 or IANA latin1. (Bug #51927)

- A non-terminated string in SQL threw a CLR exception rather than a syntax exception. (Bug #51788)
- When using table per type inheritance and listing the contents of the parent table, the result of the query was a list of child objects, even though there was no related child record with the same parent Id. (Bug #49850)

- When calling `ExecuteNonQuery` on a command object, the following exception occurred:

```
Index and length must refer to a location within the string.  
Parameter name: length
```

(Bug #51610)

- Binary Columns were not displayed in the Query Builder of Visual Studio. (Bug #50171)
- `MySqlCommand.Parameters.Clear()` did not work. (Bug #50444)
- When the `UpdateBatchSize` property was set to a value greater than 1, only the first row was applied to the database. (Bug #50123)
- The method `Command.TrimSemicolons` used `StringBuilder`, and therefore allocated memory for the query even if it did not need to be trimmed. This led to excessive memory consumption when executing a number of large queries. (Bug #51149)
- A connection string set in `web.config` could not be reused after Visual Studio 2008 Professional was shut down. It continued working for the existing controls, but did not work for new controls added. (Bug #41629)
- When the `MySqlCommand.execute()` method was called, the following exception was generated:

```
InvalidOperationException : The CommandText property has not been properly initialized.
```

(Bug #50344)

- A `FormatException` was generated when an empty string was returned from a stored function. (Bug #49642)
- `MySqlDataReader.GetUInt64` returned an incorrect value when reading a `BIGINT UNSIGNED` column containing a value greater than 2147483647. (Bug #49794)
- Calling a User Defined Function using Entity SQL in the Entity Framework caused a `NullReferenceException`. (Bug #45277)
- The method `MySqlDataReader.GetSchemaTable()` returned 0 in the `NumericPrecision` field for decimal and newdecimal columns. (Bug #48171)
- Attempting to execute a load data local infile on a file where the user did not have write permissions, or the file was open in an editor gave an access denied error. (Bug #48944)
- When adding a data set in Visual Studio 2008, the following error was generated:

```
Relations couldn't be added. Column 'REFERENCED_TABLE_CATALOG' does not belong to table.
```

This was due to a 'REFERENCED_TABLE_CATALOG' column not being included in the foreign keys collection. (Bug #48974)

- When trying to create stored procedures from an SQL script, a `MySqlException` was thrown when attempting to redefine the `DELIMITER`:

```
MySql.Data.MySqlClient.MySqlException was unhandled  
Message="You have an error in your SQL syntax; check the manual that corresponds to your  
MySQL server version for the right syntax to use near 'DELIMITER' at line 1"  
Source="MySql.Data"  
ErrorCode=-2147467259  
Number=1064  
StackTrace:  
à MySql.Data.MySqlClient.MySqlStream.ReadPacket()  
à MySql.Data.MySqlClient.NativeDriver.ReadResult(UInt64& affectedRows, Int64&  
lastInsertId)  
à MySql.Data.MySqlClient.MySqlDataReader.GetResultSet()  
à MySql.Data.MySqlClient.MySqlDataReader.NextResult()  
à MySql.Data.MySqlClient.MySqlCommand.ExecuteReader(CommandBehavior behavior)
```

```

à MySql.Data.MySqlClient.MySqlCommand.ExecuteReader()
à MySql.Data.MySqlClient.MySqlCommand.ExecuteNonQuery()
à MySql.Data.MySqlClient.MySqlScript.Execute()

```

Note: The `MySqlScript` class has been fixed to support the delimiter statement as it is found in SQL scripts. (Bug #46429)

A.6.5. Changes in MySQL Connector/Net 6.1.3 (2009-11-16, Generally Available)

This release fixes bugs since 6.1.2.

Bugs Fixed

- When building the `MySql.Data` project with .NET Framework 3.5 installed, the following build output was displayed:

```

Project file contains ToolsVersion="4.0", which is not supported by this version of
MSBuild. Treating the project as if it had ToolsVersion="3.5".

```

The project had been created using the .NET Framework 4.0, which was beta, instead of using the 3.5 framework. (Bug #48271)

- Cloning of `MySqlCommand` was not typesafe. To clone a `MySqlCommand` it was necessary to do:

```

MySqlCommand clone = (MySqlCommand)((ICloneable)comm).Clone();

```

MySQL Connector/Net was changed so that it was possible to do:

```

MySqlCommand clone = comm.Clone();

```

(Bug #48460)

- An entity model created from a schema containing a table with a column of type `UNSIGNED BIGINT` and a view of the table did not behave correctly. When an entity was created and mapped to the view, the column that was of type `UNSIGNED BIGINT` was displayed as `BIGINT`. (Bug #47872)

- For some character sets such as UTF-8, a `CHAR` column would sometimes be incorrectly interpreted as a `GUID` by MySQL Connector/Net.

MySQL Connector/Net was changed so that a column would only be interpreted as a `GUID` if it had a character length of 36, as opposed to a byte length of 36. (Bug #47985)

- It was not possible to retrieve a value from a MySQL server table, if the value was larger than that supported by the .NET type `System.Decimal`.

MySQL Connector/Net was changed to expose the `MySqlDecimal` type, along with the supporting method `GetMySqlDecimal`. (Bug #48100)

- If `MySqlConnection.GetSchema` was called for "Indexes" on a table named "b`a`d" as follows:

```

DataTable schemaPrimaryKeys = connection.GetSchema(
    "Indexes",
    new string[] { null, schemaName, "b`a`d" });

```

Then the following exception was generated:

```

You have an error in your SQL syntax; check the manual that corresponds to
your MySQL server version for the right syntax to use near 'a`d`' at line 1

```

(Bug #48101)

- When using a `BINARY(16)` column to represent a GUID and having specified "old guids = true" in the connection string, the values were returned correctly until a null value was encountered in

that field. After the null value was encountered a format exception was thrown with the following message:

```
Guid should contain 32 digits with 4 dashes (xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx).
```

(Bug #47928)

- The Session Provider created invalid “session expires” on a random basis.

This was due to the fact that the Session Provider was incorrectly reading from the root `web.config`, rather than from the application specific `web.config`. (Bug #47815)

- Attempting to build MySQL Connector/Net 6.1 `MySQL.Data` from source code on Windows failed with the following error:

```
... \clones\6.1\MySQL.Data\Provider\Source\NativeDriver.cs(519,29): error CS0122:
'Mysql.Data.MySqlClient.MySqlPacket.MySqlPacket()' is inaccessible due to its protection level
```

(Bug #47354)

- When tables were auto created for the Session State Provider they were set to use the MySQL Server's default collation, rather than the default collation set for the containing database. (Bug #47332)
- When loading the `MySQLClient-mono.sln` file included with the Connector/Net source into Mono Develop, the following error occurred:

```
/home/tbedford/connector-net-src/6.1/MySQLClient-mono.sln(22):
Unsupported or unrecognized project:
'/home/tbedford/connector-net-src/6.1/Installer/Installer.wixproj'
```

If the file was modified to remove this problem, then attempting to build the solution generated the following error:

```
/home/tbedford/connector-net-src/6.1/MySQL.Data/Provider/Source/Connection.cs(280,46):
error CS0115: 'Mysql.Data.MySqlClient.MySqlConnection.DbProviderFactory' is marked as an
override but no suitable property found to override
```

(Bug #47048)

A.6.6. Changes in MySQL Connector/Net 6.1.2 (2009-09-08, Generally Available)

This is the first GA release of 6.1. This release fixes bugs since 6.1.1.

Bugs Fixed

- The MySQL Connector/Net Session State Provider truncated session data to 64KB, due to its column types being set to `BLOB`. (Bug #47339)
- Input parameters were missing from Stored Procedures when using them with ADO.NET Data Entities. (Bug #44985)
- MySQL Connector/Net generated the following exception when using the Session State provider:

```
You have an error in your SQL syntax; check the manual that corresponds to your MySQL
server version for the right syntax to use near 'MINUTEWHERE SessionId =
'dtmgga55x35oi255nrfrxe45' AND ApplicationId = 1 AND Loc' at line 1
Description: An unhandled exception occurred during the execution of the current web
request. Please review the stack trace for more information about the error and where it
originated in the code.
```

```
Exception Details: Mysql.Data.MySqlClient.MySqlException: You have an error in your SQL
syntax; check the manual that corresponds to your MySQL server version for the right
syntax to use near 'MINUTEWHERE SessionId = 'dtmga55x35oi255nrfrxe45' AND ApplicationId =
```

```
1 AND Loc' at line 1
```

(Bug #46939)

- If an error occurred during connection to a MySQL Server, deserializing the error message from the packet buffer caused a `NullReferenceException` to be thrown. When the method `MySqlPacket::ReadString()` attempted to retrieve the error message, the following line of code threw the exception:

```
string s = encoding.GetString(bits, (int)buffer.Position, end - (int)buffer.Position);
```

This was due to the fact that the encoding field had not been initialized correctly. (Bug #46844)

- MySQL Connector/Net did not time out correctly. The command timeout was set to 30 secs, but MySQL Connector/Net hung for several hours. (Bug #43761)

A.6.7. Changes in MySQL Connector/Net 6.1.1 (2009-08-20, Beta)

This is the first Beta release of 6.1.

Bugs Fixed

- In the `MySqlDataReader` class the `GetSByte` function returned a `byte` value instead of an `sbyte` value. (Bug #46620)
- Calling a Stored Procedure with an output parameter through MySQL Connector/Net resulted in a memory leak. Calling the same Stored Procedure without an output parameter did not result in a memory leak. (Bug #36027)
- An exception was generated when using `TIMESTAMP` columns with the Entity Framework. (Bug #46311)
- Errors occurred when using the Entity Framework with cultures that used a comma as the decimal separator. This was because the formatting for `SINGLE`, `DOUBLE` and `DECIMAL` values was not handled correctly. (Bug #44455)
- When reading data, such as with a `MySqlDataAdapter` on a `MySqlConnection`, MySQL Connector/Net could potentially enter an infinite loop in `CompressedStream.ReadNextpacket()` if compression was enabled. (Bug #43678)
- Insert into two tables failed when using the Entity Framework. The exception generated was:

```
The value given is not an instance of type 'Edm.Int32'
```

(Bug #45077)

- Conversion of MySQL `TINYINT(1)` to `boolean` failed. (Bug #46205, Bug #46359, Bug #41953)
- Calling the Entity Framework `SaveChanges()` method of any MySQL ORM Entity with a column type `TIME`, generated an error message:

```
Unknown PrimitiveKind Time
```

(Bug #45457)

- When attempting to connect to MySQL using the Compact Framework version of MySQL Connector/Net, an `IndexOutOfRangeException` exception was generated on trying to open the connection. (Bug #43736)
- The Entity Framework provider was not calling `DBSortExpression` correctly when the `Skip` and `Take` methods were used, such as in the following statement:

```
TestModel.tblquarantine.OrderByDescending(q => q.MsgDate).Skip(100).Take(100).ToList();
```

This resulted in the data being unsorted. (Bug #45723)

- MySQL Connector/Net sometimes hung, without generating an exception. This happened if a read from a stream failed returning a 0, causing the code in `LoadPacket()` to enter an infinite loop. (Bug #46308)
- In the case of long network inactivity, especially when connection pooling was used, connections were sometimes dropped, for example, by firewalls.

Note: The bugfix introduced a new `keepalive` parameter, which prevents disconnects by sending an empty TCP packet after a specified timeout. (Bug #40684)

- The MySQL Connector/Net Profile Provider, `MySql.Web.Profile.MySQLProfileProvider`, generated an error when running on Mono. When an attempt was made to save a string in `Profile.Name` the string was not saved to the `my_aspnet_Profiles` table. If an attempt was made to force the save with `Profile.Save()` the following error was generated:

```
Server Error in '/mono' Application
-----

The requested feature is not implemented.
Description: HTTP 500. Error processing request.

Stack Trace:

System.NotImplementedException: The requested feature is not implemented.
at MySql.Data.MySqlClient.MySqlConnection.EnlistTransaction
(System.Transactions.Transaction transaction) [0x00000]
at MySql.Data.MySqlClient.MySqlConnection.Open () [0x00000]
at MySql.Web.Profile.MySQLProfileProvider.SetPropertyValues
(System.Configuration.SettingsContext context,
System.Configuration.SettingsPropertyValueCollection collection) [0x00000]
-----

Version information: Mono Version: 2.0.50727.1433; ASP.NET Version: 2.0.50727.1433
```

(Bug #46375)

- MySQL Connector/Net CHM documentation stated that MySQL Server 3.23 was supported. (Bug #42110)
- When populating a MySQL database table in Visual Studio using the Table Editor, if a `VARCHAR(10)` column was changed to a `VARCHAR(20)` column an exception was generated:

```
System.ArgumentException: DataGridViewComboBoxCell value is not valid.
To replace this default dialog please handle the DataError Event.
```

(Bug #46100)

- The MySQL Connector/Net 6.0.4 installer failed with an error. The error message generated was:

```
There is a problem with this Windows Installer package. A DLL required for this
install to complete could not be run. Contact your support personnel or package vendor.
```

When **OK** was clicked to acknowledge the error the installer exited. (Bug #45474)

- An error occurred when building MySQL Connector/Net from source code checked out from the public SVN repository. This happened on Linux using Mono and Nant. The Mono JIT compiler version was 1.2.6.0. The Nant version was 0.85.

When an attempt was made to build (for example) the MySQL Connector/Net 5.2 branch using the command:

```
$ nant -buildfile:Client.build
```

The following error occurred:

```
BUILD FAILED

Error loading buildfile.
Encoding name 'Windows-1252' not supported.
Parameter name: name
```

(Bug #42411)

- When using MySQL Connector/Net 6.0.4 and a MySQL Server 4.1 an exception was generated when trying to execute:

```
connection.GetSchema("Columns", ...);
```

The exception generated was:

```
'connection.GetSchema("Columns")' threw an exception of type
'System.ArgumentException' System.Data.DataTable {System.ArgumentException}
base{"Input string was not in a correct format.Couldn't store <'Select'> in
NUMERIC_PRECISION Column. Expected type is UInt64."}System.Exception
{System.ArgumentException}
```

(Bug #46270)

- The MySQL Connector/Net method `StoredProcedure.GetParameters(string)` ignored the programmer's setting of the `UseProcedureBodies` option. This broke any application for which the application's parameter names did not match the parameter names in the Stored Procedure, resulting in an `ArgumentException` with the message "Parameter 'foo' not found in the collection." and the following stack trace:

```
MySQL.Data.dll!MySQL.Data.MySqlClient.MySqlParameterCollection.GetParameterFlexible(string
parameterName = "pStart", bool throwOnNotFound = true) Line 459C#
MySQL.Data.dll!MySQL.Data.MySqlClient.StoredProcedure.Resolve() Line 157 + 0x25
bytesC#
MySQL.Data.dll!MySQL.Data.MySqlClient.MySqlCommand.ExecuteReader(System.Data.CommandBeha
vior behavior = SequentialAccess) Line 405 + 0xb bytesC#
MySQL.Data.dll!MySQL.Data.MySqlClient.MySqlCommand.ExecuteDbDataReader(System.Data.Comma
ndBehavior behavior = SequentialAccess) Line 884 + 0xb bytesC#
System.Data.dll!System.Data.Common.DbCommand.System.Data.IDbCommand.ExecuteReader(System
.Data.CommandBehavior behavior) + 0xb bytes
System.Data.dll!System.Data.Common.DbDataAdapter.FillInternal(System.Data.DataSet
dataset = {System.Data.DataSet}, System.Data.DataTable[] datatables = null, int
startRecord = 0, int maxRecords = 0, string srcTable = "Table", System.Data.IDbCommand
command = {MySQL.Data.MySqlClient.MySqlCommand}, System.Data.CommandBehavior behavior) +
0x83 bytes
System.Data.dll!System.Data.Common.DbDataAdapter.Fill(System.Data.DataSet dataSet, int
startRecord, int maxRecords, string srcTable, System.Data.IDbCommand command,
System.Data.CommandBehavior behavior) + 0x120 bytes
System.Data.dll!System.Data.Common.DbDataAdapter.Fill(System.Data.DataSet dataSet) +
0x5f bytes
```

(Bug #46213)

A.6.8. Changes in MySQL Connector/Net 6.1.0 (2009-07-15, Alpha)

This is the first Alpha release of 6.1.

Functionality Added or Changed

- Website Configuration Dialog - This is a new wizard that is activated by clicking a button on the toolbar at the top of the Visual Studio Solution Explorer. It works in conjunction with the ASP.Net administration pages, making it easier to activate and set advanced options for the different MySQL web providers included.
- Session State Provider - This enables you to store the state of your website in a MySQL server.

- Support for native output parameters - This is supported when connected to a server that supports native output parameters. This includes servers as of 5.5.3 and 6.0.8.
- Changed GUID type - The backend representation of a guid type has been changed to be CHAR(36). This is so you can use the server UUID() function to populate a GUID table. UUID generates a 36 character string. Developers of older applications can add `old_guids=true` to the connection string and the old BINARY(16) type will be used instead.

A.7. Changes in MySQL Connector/Net Version 6.0

A.7.1. Changes in MySQL Connector/Net 6.0.8 (Not released)

Fixes bugs since 6.0.7.

Version 6.0.8 has no changelog entries.

A.7.2. Changes in MySQL Connector/Net 6.0.7 (2010-08-30)

Fixes bugs since 6.0.6.

Bugs Fixed

- In the ADO.NET Entity Data Model Wizard, the time to update a model scaled abnormally as the number of entities increased. (Bug #48791, Bug #12596237)
- Opening a connection in the Visual Studio Server Explorer and choosing to alter an existing routine required another authentication at the server. (Bug #44715)
- Stored procedure enumeration code generated an error if a procedure was used in a dataset that did not return any resultsets. (Bug #50671)
- EventLog was not disposed in the SessionState provider. (Bug #52550)
- Attempting to read `Double.MinValue` from a `DOUBLE` column in MySQL table generated the following exception:

```
System.OverflowException : Value was either too large or too small for a Double.

--OverflowException
at System.Number.ParseDouble(String value, NumberStyles options, NumberFormatInfo numfmt)
at MySql.Data.Types.MySqlDouble.MySql.Data.Types.IMySqlValue.ReadValue(MySqlPacket packet, Int64 length, Boolean nullVal)
at MySql.Data.MySqlClient.NativeDriver.ReadColumnValue(Int32 index, MySqlField field, IMySqlValue valObject)
at MySql.Data.MySqlClient.ResultSet.ReadColumnData(Boolean outputParms)
at MySql.Data.MySqlClient.ResultSet.NextRow(CommandBehavior behavior)
at MySql.Data.MySqlClient.MySqlDataReader.Read()
```

(Bug #55644)

- The `INSERT` command was significantly slower with MySQL Connector/Net 6.x compared to 5.x, when compression was enabled. (Bug #48243)
- Membership schema creation failed if the default schema collation was not Latin1. (Bug #53174)
- When batching was used in `MySqlDataAdapter`, a connection was not opened automatically in `MySqlDataAdapter.Update()`. This resulted in an `InvalidOperationException` exception being generated, with the message text "connection must be valid and open".

MySQL Connector/Net has been changed to behave more like SQL Server: if the connection is closed, it is opened for the duration of update operation. (Bug #38411)

- Database name was emitted into typed datasets. This prevented users using the configured default database. (Bug #33870)
- `MySqlDataAdapter.Update()` generated concurrency violations for custom stored procedure driven update commands that used `UpdateRowSource.FirstReturnedRecord`. (Bug #54895)
- Several calls to `DataAdapter.Update()` with intervening changes to `DataTable` resulted in `ConcurrencyException` exceptions being generated. (Bug #54863)
- If `MySqlDataAdapter` was used with an `INSERT` command where the `VALUES` clause contained an expression with parentheses in it, and set the `adapter.UpdateBatchSize` parameter to be greater than one, then the call to `adapter.Update` either generated an exception or failed to batch the commands, executing each insert individually. (Bug #54386)
- The `MySqlHelper` object did not have an overloaded version of the `ExecuteReader` method that accepted a `MySqlConnection` object. (Bug #54570)
- Garbage Collector disposal of a `MySqlConnection` object caused the following exception:

```
System.IO.EndOfStreamException: Attempted to read past the end of the stream.
MySql.Data.MySqlClient.MySqlStream.ReadFully(Stream stream, Byte[] buffer, Int32 offset,
Int32 count)
MySql.Data.MySqlClient.MySqlStream.LoadPacket()
Outer Exception Reading from the stream has failed.
...
```

(Bug #53457)

- After a timeout exception, if an attempt was made to reuse a connection returned to the connection pool the following exception was generated:

```
[MySqlException (0x80004005): There is already an open DataReader associated with this
Connection which must be closed first.]
    MySql.Data.MySqlClient.MySqlCommand.CheckState() +278
    MySql.Data.MySqlClient.MySqlCommand.ExecuteReader(CommandBehavior behavior) +43
    MySql.Data.MySqlClient.MySqlCommand.ExecuteReader() +6
    Controls.SimpleCommand.ExecuteReader(String SQL) in ...:323
    Albums.GetImagesByAlbum(SimpleCommand Cmd, Int32 iAlbum, String Order, String Limit)
in ...:13
    Forecast.Page_Load(Object sender, EventArgs e) in ...:70
    System.Web.UI.Control.OnLoad(EventArgs e) +99
    System.Web.UI.Control.LoadRecursive() +50
    System.Web.UI.Page.ProcessRequestMain(Boolean includeStagesBeforeAsyncPoint, Boolean
includeStagesAfterAsyncPoint) +627
```

(Bug #53357)

- The method `MySql.Data.Common.QueryNormalizer.CollapseValueList` generated an `ArgumentOutOfRangeException`. (Bug #54152, Bug #53865)
- When an application was subjected to increased concurrent load, MySQL Connector/Net generated the following error when calling stored procedures:

```
A DataTable named \'Procedure Parameters\'
already belongs to this DataSet.
```

(Bug #49118)

- When the connection string option "Connection Reset = True" was used, a connection reset used the previously used encoding for the subsequent authentication operation. This failed, for example, if UCS2 was used to read the last column before the reset. (Bug #47153)

A.7.3. Changes in MySQL Connector/Net 6.0.6 (2010-04-28)

Fixes bugs since 6.0.5.

Functionality Added or Changed

- Procedure caching had a problem whereby if you created a procedure, dropped it, and recreated it with a different number of parameters an exception was generated.

MySQL Connector/Net has been changed so that if the procedure is recreated with a different number of parameters, it will still be recognized. (Bug #52562)

- MySQL Connector/Net has been changed to include `MySqlDataReader.GetFieldType(string columnname)`. Further, `MySqlDataReader.GetOrdinal()` now includes the name of the column in the exception if the column is not found. (Bug #47467)

Bugs Fixed

- If using MySQL Server 5.0.x it was not possible to alter stored routines in Visual Studio. If the stored routine was clicked, and the context sensitive menu option, Alter Routine, selected, the following error was generated:

```
Unable to load object with error: Object reference not
set to an instance of an object
```

(Bug #55170)

- In MySQL Connector/Net, the `MySqlConnection.Abort()` method contained a `try...catch` construct, with an empty `catch` block. This meant that any exception generated at this point would not be caught. (Bug #52769)
- If `FunctionsReturnString=true` was used in the connection string, the decimal separator (according to locale) was not interpreted. (Bug #52187)
- In MySQL Connector/Net, the `LoadCharsetMap()` function of the `CharsetMap` class set the following incorrect mapping:

```
mapping.Add("latin1", new CharacterSet("latin1", 1));
```

This meant that, for example, the Euro sign was not handled correctly.

The correct mapping should have been:

```
mapping.Add("latin1", new CharacterSet("windows-1252", 1));
```

This is because MySQL's `latin1` character set is the same as the `windows-cp1252` character set and it extends the official ISO 8859-1 or IANA latin1. (Bug #51927)

- A non-terminated string in SQL threw a CLR exception rather than a syntax exception. (Bug #51788)
- When using table per type inheritance and listing the contents of the parent table, the result of the query was a list of child objects, even though there was no related child record with the same parent Id. (Bug #49850)
- When calling `ExecuteNonQuery` on a command object, the following exception occurred:

```
Index and length must refer to a location within the string.
Parameter name: length
```

(Bug #51610)

- Binary Columns were not displayed in the Query Builder of Visual Studio. (Bug #50171)
- `MySqlCommand.Parameters.Clear()` did not work. (Bug #50444)
- When the `UpdateBatchSize` property was set to a value greater than 1, only the first row was applied to the database. (Bug #50123)

- ```
InvalidOperationException : The CommandText property has not been properly initialized.
```

Relations couldn't be added. Column 'REFERENCED\_TABLE\_CATALOG' does not belong to table.

```
MySQL.Data.MySqlClient.MySQLException was unhandled
Message="You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'DELIMITER' at line 1"
Source="MySql.Data"
ErrorCode=-2147467259
Number=1064
StackTrace:
à MySql.Data.MySqlClient.MySqlCommand.ExecuteReader(CommandBehavior behavior)
à MySql.Data.MySqlClient.NativeDriver.ReadResult(UInt64& affectedRows, Int64&
lastInsertId)
à MySql.Data.MySqlClient.MySqlDataReader.GetResultSet()
à MySql.Data.MySqlClient.MySqlDataReader.NextResult()
à MySql.Data.MySqlClient.MySqlCommand.ExecuteNonQuery()
à MySql.Data.MySqlClient.MySqlCommand.ExecuteNonQuery()
à MySql.Data.MySqlClient.MySqlCommand.ExecuteNonQuery()
à MySql.Data.MySqlClient.MySqlCommand.ExecuteNonQuery()
```

#### A.7.4. Changes in MySQL Connector/Net 6.0.5 (2009-11-12)

---

283

```
MySQLCommand clone = (MySQLCommand)((ICloneable)comm).Clone();
```

MySQL Connector/Net was changed so that it was possible to do:

```
MySQLCommand clone = comm.Clone();
```

(Bug #48460)

- An entity model created from a schema containing a table with a column of type `UNSIGNED BIGINT` and a view of the table did not behave correctly. When an entity was created and mapped to the view, the column that was of type `UNSIGNED BIGINT` was displayed as `BIGINT`. (Bug #47872)
- It was not possible to retrieve a value from a MySQL server table, if the value was larger than that supported by the .NET type `System.Decimal`.

MySQL Connector/Net was changed to expose the `MySQLDecimal` type, along with the supporting method `GetMySQLDecimal`. (Bug #48100)

- If `MySQLConnection.GetSchema` was called for "Indexes" on a table named "b`a`d" as follows:

```
DataTable schemaPrimaryKeys = connection.GetSchema(
 "Indexes",
 new string[] { null, schemaName, "b`a`d" });
```

Then the following exception was generated:

```
You have an error in your SQL syntax; check the manual that corresponds to
your MySQL server version for the right syntax to use near 'a`d`' at line 1
```

(Bug #48101)

- When loading the `MySQLClient-mono.sln` file included with the Connector/Net source into Mono Develop, the following error occurred:

```
/home/tbedford/connector-net-src/6.1/MySQLClient-mono.sln(22):
Unsupported or unrecognized project:
'/home/tbedford/connector-net-src/6.1/Installer/Installer.wixproj'
```

If the file was modified to remove this problem, then attempting to build the solution generated the following error:

```
/home/tbedford/connector-net-src/6.1/MySQL.Data/Provider/Source/Connection.cs(280,46):
error CS0115: 'MySQL.Data.MySqlClient.MySqlConnection.DbProviderFactory' is marked as an
override but no suitable property found to override
```

(Bug #47048)

- Input parameters were missing from Stored Procedures when using them with ADO.NET Data Entities. (Bug #44985)
- Using a `DataAdapter` with a linked `MySQLCommandBuilder` the following exception was thrown when trying to call `da.Update(DataRow[] rows)`:

```
Connection must be valid and open
```

(Bug #34657)

- If an error occurred during connection to a MySQL Server, deserializing the error message from the packet buffer caused a `NullReferenceException` to be thrown. When the method `MySQLPacket::ReadString()` attempted to retrieve the error message, the following line of code threw the exception:

```
string s = encoding.GetString(bits, (int)buffer.Position, end - (int)buffer.Position);
```

This was due to the fact that the encoding field had not been initialized correctly. (Bug #46844)

- In the `MySqlDataReader` class the `GetSByte` function returned a `byte` value instead of an `sbyte` value. (Bug #46620)
- Calling a Stored Procedure with an output parameter through MySQL Connector/Net resulted in a memory leak. Calling the same Stored Procedure without an output parameter did not result in a memory leak. (Bug #36027)
- An exception was generated when using `TIMESTAMP` columns with the Entity Framework. (Bug #46311)
- Errors occurred when using the Entity Framework with cultures that used a comma as the decimal separator. This was because the formatting for `SINGLE`, `DOUBLE` and `DECIMAL` values was not handled correctly. (Bug #44455)
- When reading data, such as with a `MySqlDataAdapter` on a `MySqlConnection`, MySQL Connector/Net could potentially enter an infinite loop in `CompressedStream.ReadNextpacket()` if compression was enabled. (Bug #43678)
- Insert into two tables failed when using the Entity Framework. The exception generated was:

```
The value given is not an instance of type 'Edm.Int32'
```

(Bug #45077)

- Conversion of MySQL `TINYINT(1)` to `boolean` failed. (Bug #46205, Bug #46359, Bug #41953)
- Calling the Entity Framework `SaveChanges()` method of any MySQL ORM Entity with a column type `TIME`, generated an error message:

```
Unknown PrimitiveKind Time
```

(Bug #45457)

- When attempting to connect to MySQL using the Compact Framework version of MySQL Connector/Net, an `IndexOutOfRangeException` exception was generated on trying to open the connection. (Bug #43736)
- The Entity Framework provider was not calling `DBSortExpression` correctly when the `Skip` and `Take` methods were used, such as in the following statement:

```
TestModel.tblquarantine.OrderByDescending(q => q.MsgDate).Skip(100).Take(100).ToList();
```

This resulted in the data being unsorted. (Bug #45723)

- MySQL Connector/Net sometimes hung, without generating an exception. This happened if a read from a stream failed returning a 0, causing the code in `LoadPacket()` to enter an infinite loop. (Bug #46308)
- In the case of long network inactivity, especially when connection pooling was used, connections were sometimes dropped, for example, by firewalls.

Note: The bugfix introduced a new `keepalive` parameter, which prevents disconnects by sending an empty TCP packet after a specified timeout. (Bug #40684)

- The MySQL Connector/Net Profile Provider, `MySql.Web.Profile.MySQLProfileProvider`, generated an error when running on Mono. When an attempt was made to save a string in `Profile.Name` the string was not saved to the `my_aspnet_Profiles` table. If an attempt was made to force the save with `Profile.Save()` the following error was generated:

```
Server Error in '/mono' Application
```

```

```

The requested feature is not implemented.  
Description: HTTP 500. Error processing request.

Stack Trace:

```
System.NotImplementedException: The requested feature is not implemented.
at MySql.Data.MySqlClient.MySqlConnection.EnlistTransaction
(System.Transactions.Transaction transaction) [0x00000]
at MySql.Data.MySqlClient.MySqlConnection.Open () [0x00000]
at MySql.Web.Profile.MySQLProfileProvider.SetPropertyValues
(System.Configuration.SettingsContext context,
System.Configuration.SettingsPropertyValueCollection collection) [0x00000]
```

-----  
Version information: Mono Version: 2.0.50727.1433; ASP.NET Version: 2.0.50727.1433

(Bug #46375)

- MySQL Connector/Net CHM documentation stated that MySQL Server 3.23 was supported. (Bug #42110)
- When populating a MySQL database table in Visual Studio using the Table Editor, if a `VARCHAR(10)` column was changed to a `VARCHAR(20)` column an exception was generated:

```
SystemArgumentException: DataGridViewComboBoxCell value is not valid.
To replace this default dialog please handle the DataError Event.
```

(Bug #46100)

- The MySQL Connector/Net 6.0.4 installer failed with an error. The error message generated was:

```
There is a problem with this Windows Installer package. A DLL required for this
install to complete could not be run. Contact your support personnel or package vendor.
```

When **OK** was clicked to acknowledge the error the installer exited. (Bug #45474)

- An error occurred when building MySQL Connector/Net from source code checked out from the public SVN repository. This happened on Linux using Mono and Nant. The Mono JIT compiler version was 1.2.6.0. The Nant version was 0.85.

When an attempt was made to build (for example) the MySQL Connector/Net 5.2 branch using the command:

```
$ nant -buildfile:Client.build
```

The following error occurred:

```
BUILD FAILED

Error loading buildfile.
Encoding name 'Windows-1252' not supported.
Parameter name: name
```

(Bug #42411)

- When using MySQL Connector/Net 6.0.4 and a MySQL Server 4.1 an exception was generated when trying to execute:

```
connection.GetSchema("Columns", ...);
```

The exception generated was:

```
'connection.GetSchema("Columns")' threw an exception of type
'System.ArgumentException' System.Data.DataTable {System.ArgumentException}
base{"Input string was not in a correct format.Couldn't store <'Select'> in
NUMERIC_PRECISION Column. Expected type is UInt64."}System.Exception
{System.ArgumentException}
```

(Bug #46270)

- The MySQL Connector/Net method `StoredProcedure.GetParameters(string)` ignored the programmer's setting of the `UseProcedureBodies` option. This broke any application for which the application's parameter names did not match the parameter names in the Stored Procedure, resulting in an `ArgumentException` with the message "Parameter 'foo' not found in the collection." and the following stack trace:

```

MySQL.Data.dll!MySQL.Data.MySqlClient.MySqlParameterCollection.GetParameterFlexible(string parameterName = "pStart", bool throwOnNotFound = true) Line 459C#
MySQL.Data.dll!MySQL.Data.MySqlClient.StoredProcedure.Resolve() Line 157 + 0x25 bytesC#
MySQL.Data.dll!MySQL.Data.MySqlClient.MySqlCommand.ExecuteReader(System.Data.CommandBehavior behavior = SequentialAccess) Line 405 + 0xb bytesC#
MySQL.Data.dll!MySQL.Data.MySqlClient.MySqlCommand.ExecuteDbDataReader(System.Data.CommandBehavior behavior = SequentialAccess) Line 884 + 0xb bytesC#
System.Data.dll!System.Data.Common.DbCommand.System.Data.IDbCommand.ExecuteReader(System.Data.CommandBehavior behavior) + 0xb bytes
System.Data.dll!System.Data.Common.DbDataAdapter.FillInternal(System.Data.DataSet dataset = {System.Data.DataSet}, System.Data.DataTable[] datatables = null, int startRecord = 0, int maxRecords = 0, string srcTable = "Table", System.Data.IDbCommand command = {MySQL.Data.MySqlClient.MySqlCommand}, System.Data.CommandBehavior behavior) + 0x83 bytes
System.Data.dll!System.Data.Common.DbDataAdapter.Fill(System.Data.DataSet dataSet, int startRecord, int maxRecords, string srcTable, System.Data.IDbCommand command, System.Data.CommandBehavior behavior) + 0x120 bytes
System.Data.dll!System.Data.Common.DbDataAdapter.Fill(System.Data.DataSet dataSet) + 0x5f bytes

```

(Bug #46213)

- The `EscapeString` code carried out escaping by calling `string.Replace` multiple times. This resulted in a performance bottleneck, as for every line a new string was allocated and another was disposed of by the garbage collector. (Bug #45699)
- In MySQL Connector/Net 6.0.4 using `GetProcData` generated an error because the `parameters` data table was only created if MySQL Server was at least version 6.0.6, or if the `UseProcedureBodies` connection string option was set to true.

Also the `DeriveParameters` command generated a null reference exception. This was because the `parameters` data table, which was null, was used in a `for each` loop. (Bug #45952)

- MySQL Connector/Net generated the following exception:

```

System.NullReferenceException: Object reference not set to an instance of an object.
 bei MySQL.Data.MySqlClient.MySqlCommand.TimeoutExpired(Object commandObject)
 bei System.Threading._TimerCallback.TimerCallback_Context(Object state)
 bei System.Threading.ExecutionContext.runTryCode(Object userData)
 bei
 System.Runtime.CompilerServices.RuntimeHelpers.ExecuteCodeWithGuaranteedCleanup(TryCode code, CleanupCode backoutCode, Object userData)
 bei System.Threading.ExecutionContext.RunInternal(ExecutionContext executionContext, ContextCallback callback, Object state)
 bei System.Threading.ExecutionContext.Run(ExecutionContext executionContext, ContextCallback callback, Object state)
 bei System.Threading._TimerCallback.PerformTimerCallback(Object state)

```

(Bug #40005)

- After a Reference to "C:\Program Files\MySQL\MySQL Connector Net 5.2.4\Compact Framework\MySql.Data.CF.dll" was added to a Windows Mobile 5.0 project, the project then failed to build, generating a Microsoft Visual C# compiler error.

The error generated was:

```

Error 2 The type 'System.Runtime.CompilerServices.CompilerGeneratedAttribute'
has no constructors defined MySqlTest

```



```
Error 3 Internal Compiler Error (0xc0000005 at address 5A7E3714):
likely culprit is 'COMPILE'.
```

(Bug #42261)

- Adding the `Allow Batch=False` option to the connection string caused MySQL Connector/Net to generate the error:

```
You have an error in your SQL syntax; check the manual that corresponds to your MySQL
server version for the right syntax to use near 'SET character_set_results=NULL' at line 1
```

(Bug #45502)

- A MySQL Connector/Net test program that connected to MySQL Server using the connection string option `compress=true` crashed, but only when running on Mono. The program worked as expected when running on Microsoft Windows.

This was due to a bug in Mono. MySQL Connector/Net was modified to avoid using `WeakReferences` in the `Compressed` stream class, which was causing the crash. (Bug #45463)

## A.7.5. Changes in MySQL Connector/Net 6.0.4 (2009-06-16)

This is the first post-GA release, fixing recently discovered bugs.

### Bugs Fixed

- If a certain socket exception occurred when trying to establish a MySQL database connection, MySQL Connector/Net displayed an exception message that appeared to be unrelated to the underlying problem. This masked the problem and made diagnosing problems more difficult.

For example, if, when establishing a database connection using TCP/IP, Windows on the local machine allocated an ephemeral port that conflicted with a socket address still in use, then Windows/.NET would throw a socket exception with the following error text:

```
Only one usage of each socket address (protocol/network address/port) is normally
permitted IP ADDRESS/PORT.
```

However, MySQL Connector/Net masked this socket exception and displayed an exception with the following text:

```
Unable to connect to any of the specified MySQL hosts.
```

(Bug #45021)

- The Data Set editor generated an error when attempts were made to modify insert, update or delete commands:

```
Error in WHERE clause near '@'.
Unable to parse query text.
```

(Bug #44512)

- MySQL Connector/Net was missing the capability to validate the server's certificate when using encryption. This made it possible to conduct a man-in-the-middle attack against the connection, which defeated the security provided by SSL. (Bug #38700)
- A SQL query string containing an escaped backslash caused an exception to be generated:

```
Index and length must refer to a location within the string.
Parameter name: length
at System.String.InternalSubStringWithChecks(Int32 startIndex, Int32 length, Boolean
fAlwaysCopy)
at MySql.Data.MySqlClient.MySqlTokenizer.NextParameter()
at MySql.Data.MySqlClient.Statement.InternalBindParameters(String sql,
MySqlParameterCollection parameters, MySqlPacket packet)
```



```
at MySql.Data.MySqlClient.Statement.BindParameters()
at MySql.Data.MySqlClient.PreparableStatement.Execute()
at MySql.Data.MySqlClient.MySqlCommand.ExecuteReader(CommandBehavior behavior)
at MySql.Data.MySqlClient.MySqlCommand.ExecuteNonQuery()
```

(Bug #44960)

- The Microsoft Visual Studio solution file [MySQL-VS2005.sln](#) was invalid. Several projects could not be loaded and thus it was not possible to build MySQL Connector/Net from source. (Bug #44822)
- When creating a new DataSet the following error was generated:

```
Failed to open a connection to database.
Cannot load type with name 'MySQL.Data.VisualStudio.StoredProcedureColumnEnumerator'
```

(Bug #44460)

- The DataReader in MySQL Connector/Net 6.0.3 considered a BINARY(16) field as a GUID with a length of 16. (Bug #44507)
- The MySQL Connector/Net MySQLRoleProvider reported that there were no roles, even when roles existed. (Bug #44414)

## A.7.6. Changes in MySQL Connector/Net 6.0.3 (2009-04-28)

First GA release.

### Functionality Added or Changed

- The [MySqlTokenizer](#) failed to split fieldnames from values if they were not separated by a space. This also happened if the string contained certain characters. As a result [MySqlCommand.ExecuteNonQuery](#) raised an index out of range exception.

The resulting errors are illustrated by the following examples. Note, the example statements do not have delimiting spaces around the = operator.

```
INSERT INTO anytable SET Text='test--test';
```

The tokenizer incorrectly interpreted the value as containing a comment.

```
UPDATE anytable SET Project='123-456',Text='Can you explain this ?',Duration=15 WHERE
ID=4711;'
```

A [MySqlException](#) was generated, as the ? in the value was interpreted by the tokenizer as a parameter sign. The error message generated was:

```
Fatal error encountered during command execution.
EXCEPTION: MySqlException - Parameter '?' must be defined.
```

(Bug #44318)

### Bugs Fixed

- Column types for [SchemaProvider](#) and [ISSchemaProvider](#) did not match.

When the source code in [SchemaProvider.cs](#) and [ISSchemaProvider.cs](#) were compared it was apparent that they were not using the same column types. The base provider used SQL such as [SHOW CREATE TABLE](#), while [ISSchemaProvider](#) used the schema information tables. Column types used by the base class were [INT64](#) and the column types used by [ISSchemaProvider](#) were [UNSIGNED](#). (Bug #44123)

- [MySQL.Data](#) was not displayed as a Reference inside Microsoft Visual Studio 2008 Professional.

When a new C# project was created in Microsoft Visual Studio 2008 Professional, [MySQL.Data](#) was not displayed when [References](#), [Add Reference](#) was selected. (Bug #44141)

### A.7.7. Changes in MySQL Connector/Net 6.0.2 (2009-04-07, Beta)

This is a new development release, fixing recently discovered bugs.

#### Bugs Fixed

- MySQL Connector/Net 6.0.1 did not load in Microsoft Visual Studio 2008 and Visual Studio 2005 Pro.

The following error message was generated:

```
.NET Framework Data Provider for MySQL: The data provider object factory service was not found.
```

(Bug #44064)

### A.7.8. Changes in MySQL Connector/Net 6.0.1 (2009-04-02, Beta)

This is a new Beta development release, fixing recently discovered bugs.

#### Bugs Fixed

- Generating an Entity Data Model (EDM) schema with a table containing columns with data types `MEDIUMTEXT` and `LONGTEXT` generated a runtime error message "Max value too long or too short for Int32". (Bug #43480)
- An insert and update error was generated by the decimal data type in the Entity Framework, when a German collation was used. (Bug #43574)

### A.7.9. Changes in MySQL Connector/Net 6.0.0 (2009-03-02, Alpha)

This is a new Alpha development release.

#### Bugs Fixed

- A null reference exception was generated when `MySqlConnection.ClearPool(connection)` was called. (Bug #42801)

## A.8. Changes in MySQL Connector/Net Version 5.3

### A.8.1. Changes in MySQL Connector/Net 5.3.0 (Not released)

Version 5.3.0 has no changelog entries.

## A.9. Changes in MySQL Connector/Net Version 5.2

### A.9.1. Changes in MySQL Connector/Net 5.2.8 (Not released)

Version 5.2.8 has no changelog entries.

### A.9.2. Changes in MySQL Connector/Net 5.2.7 (2009-07-15)

#### Bugs Fixed

- The `EscapeString` code carried out escaping by calling `string.Replace` multiple times. This resulted in a performance bottleneck, as for every line a new string was allocated and another was disposed of by the garbage collector. (Bug #45699)
- MySQL Connector/Net generated the following exception:

```
System.NullReferenceException: Object reference not set to an instance of an object.
 bei MySql.Data.MySqlClient.MySqlCommand.TimeoutExpired(Object commandObject)
 bei System.Threading._TimerCallback.TimerCallback_Context(Object state)
 bei System.Threading.ExecutionContext.RunTryCode(Object userData)
```

```

bei
System.Runtime.CompilerServices.RuntimeHelpers.ExecuteCodeWithGuaranteedCleanup(TryCode
code, CleanupCode backoutCode, Object userData)
bei System.Threading.ExecutionContext.RunInternal(ExecutionContext executionContext,
ContextCallback callback, Object state)
bei System.Threading.ExecutionContext.Run(ExecutionContext executionContext,
ContextCallback callback, Object state)
bei System.Threading._TimerCallback.PerformTimerCallback(Object state)

```

(Bug #40005)

- After a Reference to "C:\Program Files\MySQL\MySQL Connector Net 5.2.4\Compact Framework\MySql.Data.CF.dll" was added to a Windows Mobile 5.0 project, the project then failed to build, generating a Microsoft Visual C# compiler error.

The error generated was:

```

Error 2 The type 'System.Runtime.CompilerServices.CompilerGeneratedAttribute'
has no constructors defined MySqlTest
Error 3 Internal Compiler Error (0xc0000005 at address 5A7E3714):
likely culprit is 'COMPILE'.

```

(Bug #42261)

- If a certain socket exception occurred when trying to establish a MySQL database connection, MySQL Connector/Net displayed an exception message that appeared to be unrelated to the underlying problem. This masked the problem and made diagnosing problems more difficult.

For example, if, when establishing a database connection using TCP/IP, Windows on the local machine allocated an ephemeral port that conflicted with a socket address still in use, then Windows/.NET would throw a socket exception with the following error text:

```

Only one usage of each socket address (protocol/network address/port) is normally
permitted IP ADDRESS/PORT.

```

However, MySQL Connector/Net masked this socket exception and displayed an exception with the following text:

```

Unable to connect to any of the specified MySQL hosts.

```

(Bug #45021)

- A MySQL Connector/Net test program that connected to MySQL Server using the connection string option `compress=true` crashed, but only when running on Mono. The program worked as expected when running on Microsoft Windows.

This was due to a bug in Mono. MySQL Connector/Net was modified to avoid using [WeakReferences](#) in the [Compressed](#) stream class, which was causing the crash. (Bug #45463)

- The Microsoft Visual Studio solution file [MySQL-VS2005.sln](#) was invalid. Several projects could not be loaded and thus it was not possible to build MySQL Connector/Net from source. (Bug #44822)
- The MySQL Connector/Net [MySQLRoleProvider](#) reported that there were no roles, even when roles existed. (Bug #44414)
- When a [TableAdapter](#) was created on a [DataSet](#), it was not possible to use a stored procedure with variables. The following error was generated:

```

The method or operation is not implemented

```

(Bug #39409)

### A.9.3. Changes in MySQL Connector/Net 5.2.6 (2009-04-28)

#### Functionality Added or Changed

- A new connection string option has been added: `use affected rows`. When `true` the connection will report changed rows instead of found rows. (Bug #44194)

### Bugs Fixed

- Calling `GetSchema()` on `Indexes` or `IndexColumns` failed where index or column names were restricted.

In `SchemaProvider.cs`, methods `GetIndexes()` and `GetIndexColumns()` passed their restrictions directly to `GetTables()`. This only worked if the restrictions were no more specific than `schemaName` and `tableName`. If `IndexName` was given, this was passed to `GetTables()` where it was treated as `TableType`. As a result no tables were returned, unless the index name happened to be `BASE TABLE` or `VIEW`. This meant that both methods failed to return any rows. (Bug #43991)

- `GetSchema("MetaDataCollections")` should have returned a table with a column named "NumberOfRestrictions" not "NumberOfRestriction".

This can be confirmed by referencing the [Microsoft Documentation](#). (Bug #43990)

- Requests sent to the MySQL Connector/Net role provider to remove a user from a role failed. The query log showed the query was correctly executed within a transaction which was immediately rolled back. The rollback was caused by a missing call to the `Complete` method of the transaction. (Bug #43553)
- A null reference exception was generated when `MySqlConnection.ClearPool(connection)` was called. (Bug #42801)
- The `GetGuid()` method of `MySqlDataReader` did not treat `BINARY(16)` column data as a GUID. When operating on such a column a `FormatException` exception was generated. (Bug #41452)
- When using `MySqlBulkLoader.Load()`, the text file is opened by `NativeDriver.SendFileToServer`. If it encountered a problem opening the file as a stream, an exception was generated and caught. An attempt to clean up resources was then made in the `finally{}` clause by calling `fs.Close()`, but since the stream was never successfully opened, this was an attempt to execute a method of a null reference. (Bug #43332)
- `MySQLMembershipProvider.ValidateUser` only used the `userId` to validate. However, it should also use the `applicationId` to perform the validation correctly.

The generated query was, for example:

```
SELECT Password, PasswordKey, PasswordFormat, IsApproved, Islockedout
FROM my_aspnet_Membership WHERE userId=13
```

Note that `applicationId` is not used. (Bug #42574)

- When ASP.NET membership was configured to not require password question and answer using `requiresQuestionAndAnswer="false"`, a `SqlNullValueException` was generated when using `MembershipUser.ResetPassword()` to reset the user password. (Bug #41408)
- There was an error in the `ProfileProvider` class in the `private ProfileInfoCollection GetProfiles()` function. The column of the final table was named "lastUpdatdDate" ('e' is missing) instead of the correct "lastUpdatedDate". (Bug #41654)
- When `MySql.Web.Profile.MySQLProfileProvider` was configured, it was not possible to assign a name other than the default name `MySQLProfileProvider`.

If the name `SCC_MySQLProfileProvider` was assigned, an exception was generated when attempting to use `Page.Context.Profile['custom prop']`.

The exception generated was:

```
The profile default provider was not found.
```

Note that the exception stated: 'the profile **default provider...**', even though a different name was explicitly requested. (Bug #40871)

- If a [Stored Procedure](#) contained spaces in its parameter list, and was then called from MySQL Connector/Net, an exception was generated. However, the same [Stored Procedure](#) called from the MySQL Query Analyzer or the MySQL Client worked correctly.

The exception generated was:

```
Parameter '0' not found in the collection.
```

(Bug #41034)

- The [DATETIME](#) format contained an erroneous space. (Bug #41021)
- When [ExecuteNonQuery](#) was called with a command type of [Stored Procedure](#) it worked for one user but resulted in a hang for another user with the same database permissions.

However, if [CALL](#) was used in the command text and [ExecuteNonQuery](#) was used with a command type of [Text](#), the call worked for both users. (Bug #40139)

#### A.9.4. Changes in MySQL Connector/Net 5.2.5 (2008-11-19)

##### Bugs Fixed

- Visual Studio 2008 displayed the following error three times on start-up:

```
"Package Load Failure

Package 'MySql.Data.VisualStudio.MySqlDataProviderPackage, MySql.VisualStudio,
Version=5.2.4, Culture=neutral, PublicKeyToken=null' has failed to load properly (GUID =
{79A115C9-B133-4891-9E7B-242509DAD272}). Please contact the package vendor for
assistance. Application restart is recommended, due to possible environment corruption.
Would you like to disable loading the package in the future? You may use
'devenve/resetskipkgs' to re-enable package loading."
```

(Bug #40726)

#### A.9.5. Changes in MySQL Connector/Net 5.2.4 (2008-11-13)

##### Bugs Fixed

- [MySqlDataReader](#) did not feature a [GetSByte](#) method. (Bug #40571)
- [GetDefaultCollation](#) and [GetMaxLength](#) were not thread safe. These functions called the database to get a set of parameters and cached them in two static dictionaries in the function [InitCollections](#). However, if many threads called them they would try to insert the same keys in the collections resulting in duplicate key exceptions. (Bug #40231)
- The connection string option [Functions Return String](#) did not set the correct encoding for the result string. Even though the connection string option [Functions Return String=true](#) is set, the result of [SELECT DES\\_DECRYPT\(\)](#) contained “??” instead of the correct national character symbols. (Bug #40076)
- When working with stored procedures MySQL Connector/Net generated an exception [Unknown "table parameters" in information\\_schema](#). (Bug #40382)
- If connection pooling was not set explicitly in the connection string, MySQL Connector/Net added “;Pooling=False” to the end of the connection string when [MySqlCommand.ExecuteReader\(\)](#) was called.

If connection pooling was explicitly set in the connection string, when [MySqlConnection.Open\(\)](#) was called it converted “Pooling=True” to “pooling=True”.

If `MySqlCommand.ExecuteReader()` was subsequently called, it concatenated “;Pooling=False” to the end of the connection string. The resulting connection string was thus terminated with “pooling=True;Pooling=False”. This disabled connection pooling completely. (Bug #40091)

- After the `ConnectionString` property was initialized using the public setter of `DbConnectionStringBuilder`, the `GetConnectionString` method of `MySqlConnectionStringBuilder` incorrectly returned `null` when `true` was assigned to the `includePass` parameter. (Bug #39728)
- If, when using the `MySqlTransaction` transaction object, an exception was thrown, the transaction object was not disposed of and the transaction was not rolled back. (Bug #39817)
- When using `ProfileProvider`, attempting to update a previously saved property failed. (Bug #39330)
- Reading a negative time value greater than -01:00:00 returned the absolute value of the original time value. (Bug #39294)
- Inserting a negative time value (negative `TimeSpan`) into a `Time` column through the use of `MySqlParameter` caused `MySqlException` to be thrown. (Bug #39275)
- The Web Provider did not work at all on a remote host, and did not create a database when using `autogenerateschema="true"`. (Bug #39072)
- MySQL Connector/Net called hashed password methods not supported in Mono 2.0 Preview 2. (Bug #38895)
- When a data connection was created in the server explorer of Visual Studio 2008 Team, an error was generated when trying to expand stored procedures that had parameters.

Also, if **TableAdapter** was right-clicked and then Add, Query, Use Existing Stored Procedures selected, if you then attempted to select a stored procedure, the window would close and no error message would be displayed. (Bug #39252)

## A.9.6. Changes in MySQL Connector/Net 5.2.3 (2008-08-19)

### Functionality Added or Changed

- String escaping functionality has been moved from the `MySqlString` class to the `MySqlHelper` class, where it can be accessed by the `EscapeString` method. (Bug #36205)
- Error string was returned after a 28000 second `wait_timeout`. This has been changed to generate a `ConnectionState.Closed` event. (Bug #38119)
- Changed how the procedure schema collection is retrieved. If `use procedure bodies=true` then the `mysql.proc` table is selected directly as this is up to 50 times faster than the current `information_schema` implementation. If `use procedure bodies=false`, then the `information_schema` collection is queried. (Bug #36694)

### Bugs Fixed

- MySQL Connector/Net uninstaller did not clean up all installed files. (Bug #38534)
- The `GetOrdinal()` method failed to return the ordinal if the column name string contained an accent. (Bug #38721)
- `GetSchema` did not work correctly when querying for a collection, if using a non-English locale. (Bug #35459)
- There was a short circuit evaluation error in the `MySqlCommand.CheckState()` method. When the statement `connection == null` was true a `NullReferenceException` was thrown and not the expected `InvalidOperationException`. (Bug #38276)



- The provider did not silently create the user if the user did not exist. (Bug #38243)
- When reading back a stored double or single value using the .NET provider, the value had less precision than the one stored. (Bug #33322)
- Unnecessary network traffic was generated for the normal case where the web provider schema was up to date. (Bug #37469)
- Executing a command that resulted in a fatal exception did not close the connection. (Bug #37991)
- When a prepared insert query is run that contains an `UNSIGNED TINYINT` in the parameter list, the complete query and data that should be inserted is corrupted and no error is thrown. (Bug #37968)
- In a .NET application MySQL Connector/Net modifies the connection string so that it contains several occurrences of the same option with different values. This is illustrated by the example that follows.

The original connection string:

```
host=localhost;database=test;uid=****;pwd=****;
connect timeout=25; auto enlist=false;pooling=false;
```

The connection string after closing `MySqlDataReader`:

```
host=localhost;database=test;uid=****;pwd=****;
connect timeout=25;auto enlist=false;pooling=false;
Allow User Variables=True;Allow User Variables=False;
Allow User Variables=True;Allow User Variables=False;
```

(Bug #37955)

- `MySqlReader.GetOrdinal()` performance enhancements break existing functionality. (Bug #37239)
- The `autogenerateschema` option produced tables with incorrect collations. (Bug #36444)
- Using the MySQL Visual Studio plugin and a MySQL 4.1 server, certain field types (`ENUM`) would not be identified correctly. Also, when looking for tables, the plugin would list all tables matching a wildcard pattern of the database name supplied in the connection string, instead of only tables within the specified database. (Bug #30603)

## A.9.7. Changes in MySQL Connector/Net 5.2.2 (2008-05-12)

### Bugs Fixed

- Product documentation incorrectly stated '?' is the preferred parameter marker. (Bug #37349)
- There was a high level of contention in the connection pooling code that could lead to delays when opening connections and submitting queries. The connection pooling code has been modified to try and limit the effects of the contention issue. (Bug #34001)
- An incorrect value for a bit field would returned in a multi-row query if a preceding value for the field returned `NULL`. (Bug #36313)
- When using the `MySQLProfileProvider`, setting profile details and then reading back saved data would result in the default values being returned instead of the updated values. (Bug #36000)
- The `DbCommandBuilder.QuoteIdentifier` method was not implemented. (Bug #35492)
- Tables with `GEOMETRY` field types would return an unknown data type exception. (Bug #36081)
- When creating a connection, setting the `ConnectionString` property of `MySQLConnection` to `NULL` would throw an exception. (Bug #35619)
- When using encrypted passwords, the `GetPassword()` function would return the wrong string. (Bug #35336)

- An error would be raised when calling `GetPassword()` with a `NULL` value. (Bug #35332)
- When retrieving data where a field has been identified as containing a GUID value, the incorrect value would be returned when a previous row contained a `NULL` value for that field. (Bug #35041)
- When using `SqlDataSource` to open a connection, the connection would not automatically be closed when access had completed. (Bug #34460)
- When creating a new stored procedures, the new parameter code which permits the use of the `@` symbol would interfere with the specification of a `DEFINER`. (Bug #34940)
- Using the `TableAdapter Wizard` would fail when generating commands that used stored procedures due to the change in supported parameter characters. (Bug #34941)
- Using the `TableAdapter` wizard in combination with a suitable `SELECT` statement, only the associated `INSERT` statement would also be created, rather than the required `DELETE` and `UPDATE` statements. (Bug #31338)
- Fixed profile provider that would throw an exception if you were updating a profile that already existed.
- Fixed problem in datagrid code related to creating a new table. This problem may have been introduced with .NET 2.0 SP1.

## A.9.8. Changes in MySQL Connector/Net 5.2.1 (2008-02-27)

### Bugs Fixed

- A race condition could occur within the procedure cache resulting the cache contents overflowing beyond the configured cache size. (Bug #34338)
- When altering a stored procedure within Visual Studio, the parameters to the procedure could be lost. (Bug #34359)
- Attempting to use an isolation level other than the default with a transaction scope would use the default isolation level. (Bug #34448)
- A number of issues were identified in the case, connection and schema areas of the code for `MembershipProvider`, `RoleProvider`, `ProfileProvider`. (Bug #34495)
- When executing statements that used stored procedures and functions, the new parameter code could fail to identify the correct parameter format. (Bug #34699)
- When using web providers, the MySQL Connector/Net would check the schema and cache the application id, even when the connection string had been set. The effect would be to break the membership provider list. (Bug #34451)
- The installer would fail to the DDEX provider binary if the Visual Studio 2005 component was not selected. The result would lead to MySQL Connector/Net not loading properly when using the interface to a MySQL server within Visual Studio. (Bug #34674)
- When using the provider to generate or update users and passwords, the password checking algorithm would not validate the password strength or requirements correctly. (Bug #34792)
- The provider code has been updated to fix a number of outstanding issues.
- Fixed problem with Visual Studio 2008 integration that caused pop-up menus on server explorer nodes to not function

## A.9.9. Changes in MySQL Connector/Net 5.2.0 (2008-02-11)

### Functionality Added or Changed



- Performing `GetValue()` on a field `TINYINT(1)` returned a `BOOLEAN`. While not a bug, this caused problems in software that expected an `INT` to be returned. A new connection string option `TreatTiny As Boolean` has been added with a default value of `true`. If set to `false` the provider will treat `TINYINT(1)` as `INT`. (Bug #34052)
- Added `ClearPool` and `ClearAllPools` features.
- DDEX provider now works under Visual Studio 2008 beta 2.
- Added support for `DbDataAdapter.UpdateBatchSize`. Batching is fully supported including collapsing inserts down into the multi-value form if possible.

#### Bugs Fixed

- In an open connection where the server had disconnected unexpectedly, the status information of the connection would not be updated properly. (Bug #33909)
- The status of connections reported through the state change handler was not being updated correctly. (Bug #34082)
- When accessing tables from different databases within the same `TransactionScope`, the same user/password combination would be used for each database connection. MySQL Connector/Net does not handle multiple connections within the same transaction scope. An error is now returned if you attempt this process, instead of using the incorrect authorization information. (Bug #34204)
- Some speed improvements have been implemented in the `TokenizeSql` process used to identify elements of SQL statements. (Bug #34220)
- Incorporated some connection string cache optimizations sent to us by Maxim Mass. (Bug #34000)
- Using compression in the MySQL connection with MySQL Connector/Net would be slower than using native (uncompressed) communication. (Bug #27865)
- Commands executed from within the state change handler would fail with a `NULL` exception. (Bug #30964)
- Memory usage could increase and decrease significantly when updating or inserting a large number of rows. (Bug #31090)
- Column name metadata was not using the character set as defined within the connection string being used. (Bug #31185)
- Data cached from the connection string could return invalid information because the internal routines were not using case-sensitive semantics. This lead to updated connection string options not being recognized if they were of a different case than the existing cached values. (Bug #31433)
- When running a stored procedure multiple times on the same connection, the memory usage could increase indefinitely. (Bug #30116)
- The `MySqlDbType.Datetime` has been replaced with `MySqlDbType.DateTime`. The old format has been obsoleted. (Bug #26344)

## A.10. Changes in MySQL Connector/Net Version 5.1

### A.10.1. Changes in MySQL Connector/Net 5.1.8 (Not released)

Version 5.1.8 has no changelog entries.

### A.10.2. Changes in MySQL Connector/Net 5.1.7 (2008-08-21)

#### Bugs Fixed

- `GetSchema` did not work correctly when querying for a collection, if using a non-English locale. (Bug #35459)
- There was a short circuit evaluation error in the `MySqlCommand.CheckState()` method. When the statement `connection == null` was true a `NullReferenceException` was thrown and not the expected `InvalidOperationException`. (Bug #38276)
- When reading back a stored double or single value using the .NET provider, the value had less precision than the one stored. (Bug #33322)
- Documentation incorrectly stated that “the DataColumn class in .NET 1.0 and 1.1 does not permit columns with type of UInt16, UInt32, or UInt64 to be autoincrement columns”. (Bug #37350)
- As `MySqlDbType.DateTime` is not available in `VB.Net` the warning `The datetime enum value is obsolete` was always shown during compilation. (Bug #37406)
- An unknown `MySqlErrorCode` was encountered when opening a connection with an incorrect password. (Bug #37398)
- Executing a command that resulted in a fatal exception did not close the connection. (Bug #37991)
- In a .NET application MySQL Connector/Net modifies the connection string so that it contains several occurrences of the same option with different values. This is illustrated by the example that follows.

The original connection string:

```
host=localhost;database=test;uid=****;pwd=****;
connect timeout=25; auto enlist=false;pooling=false;
```

The connection string after closing `MySqlDataReader`:

```
host=localhost;database=test;uid=****;pwd=****;
connect timeout=25;auto enlist=false;pooling=false;
Allow User Variables=True;Allow User Variables=False;
Allow User Variables=True;Allow User Variables=False;
```

(Bug #37955)

- `SemaphoreFullException` is generated when application is closed. (Bug #36688)
- Using the MySQL Visual Studio plugin and a MySQL 4.1 server, certain field types (`ENUM`) would not be identified correctly. Also, when looking for tables, the plugin would list all tables matching a wildcard pattern of the database name supplied in the connection string, instead of only tables within the specified database. (Bug #30603)

### A.10.3. Changes in MySQL Connector/Net 5.1.6 (2008-05-12)

#### Bugs Fixed

- An incorrect value for a bit field would returned in a multi-row query if a preceding value for the field returned `NULL`. (Bug #36313)
- The `DbCommandBuilder.QuoteIdentifier` method was not implemented. (Bug #35492)
- When creating a connection pool, specifying an invalid IP address will cause the entire application to crash, instead of providing an exception. (Bug #36432)
- The `MembershipProvider` will raise an exception when the connection string is configured with `enablePasswordRetrival = true` and `RequireQuestionAndAnswer = false`. (Bug #36159)
- When calling `GetNumberOfUsersOnline` an exception is raised on the submitted query due to a missing parameter. (Bug #36157)

- Tables with `GEOMETRY` field types would return an unknown data type exception. (Bug #36081)
- When creating a connection, setting the `ConnectionString` property of `MySQLConnection` to `NULL` would throw an exception. (Bug #35619)
- A race condition could occur within the procedure cache resulting the cache contents overflowing beyond the configured cache size. (Bug #34338)
- When altering a stored procedure within Visual Studio, the parameters to the procedure could be lost. (Bug #34359)
- Attempting to use an isolation level other than the default with a transaction scope would use the default isolation level. (Bug #34448)
- When using `SqlDataSource` to open a connection, the connection would not automatically be closed when access had completed. (Bug #34460)
- Using the `TableAdapter` wizard in combination with a suitable `SELECT` statement, only the associated `INSERT` statement would also be created, rather than the required `DELETE` and `UPDATE` statements. (Bug #31338)

#### A.10.4. Changes in MySQL Connector/Net 5.1.5 (Not released)

Version 5.1.5 has no changelog entries.

#### A.10.5. Changes in MySQL Connector/Net 5.1.4 (2007-11-20)

##### Bugs Fixed

- Trying to use a connection that was not open could return an ambiguous and misleading error message. (Bug #31262)
- A syntax error in a set of batch statements could leave the data adapter in a state that appears hung. (Bug #31930)
- When accessing certain statements, the command would timeout before the command completed. Because this cannot always be controlled through the individual command timeout options, a `default command timeout` has been added to the connection string options. (Bug #27958)
- MySQL Connector/Net would incorrectly report success when enlisting in a distributed transaction, although distributed transactions are not supported. (Bug #31703)
- A date string could be returned incorrectly by `MySQLDateTime.ToString()` when the date returned by MySQL was `0000-00-00 00:00:00`. (Bug #32010)
- Commands executed from within the state change handler would fail with a `NULL` exception. (Bug #30964)
- Memory usage could increase and decrease significantly when updating or inserting a large number of rows. (Bug #31090)
- Column name metadata was not using the character set as defined within the connection string being used. (Bug #31185)
- Data cached from the connection string could return invalid information because the internal routines were not using case-sensitive semantics. This lead to updated connection string options not being recognized if they were of a different case than the existing cached values. (Bug #31433)
- Installing over a failed uninstall of a previous version could result in multiple clients being registered in the `machine.config`. This would prevent certain aspects of the MySQL connection within Visual Studio to work properly. (Bug #31731)

- Creation of parameter objects with noninput direction using a constructor would fail. This was caused by some old legacy code preventing their use. (Bug #32093)
- Setting the size of a string parameter after the value could cause an exception. (Bug #32094)
- Changing the connection string of a connection to one that changes the parameter marker after the connection had been assigned to a command but before the connection is opened could cause parameters to not be found. (Bug #13991)
- Column types with only 1-bit (such as `BOOLEAN` and `TINYINT(1)`) were not returned as boolean fields. (Bug #27959)
- When running a stored procedure multiple times on the same connection, the memory usage could increase indefinitely. (Bug #30116)
- The `MySqlDbType.Datetime` has been replaced with `MySqlDbType.DateTime`. The old format has been obsoleted. (Bug #26344)
- The server error code was not updated in the `Data[]` hash, which prevented `DbProviderFactory` users from accessing the server error code. (Bug #27436)
- Extracting data through XML functions within a query returns the data as `System.Byte[]`. This was due to MySQL Connector/Net incorrectly identifying `BLOB` fields as binary, rather than text. (Bug #30233)

### A.10.6. Changes in MySQL Connector/Net 5.1.3 (2007-09-21, Beta)

This is a new Beta development release, fixing recently discovered bugs.

#### Bugs Fixed

- An incorrect `ConstraintException` could be raised on an `INSERT` when adding rows to a table with a multiple-column unique key index. (Bug #30204)
- The availability of a MySQL server would not be reset when using pooled connections (`pooling=true`). This would lead to the server being reported as unavailable, even if the server became available while the application was still running. (Bug #29409)
- Calling `SHOW CREATE PROCEDURE` for routines with a hyphen in the catalog name produced a syntax error. (Bug #29526)
- Using the membership/role providers when `validationKey` or `decryptionKey` parameters are set to `AutoGenerate`, an exception would be raised when accessing the corresponding values. (Bug #29235)
- *Visual Studio Plugin*: Adding a new query based on a stored procedure that uses the `SELECT` statement would terminate the query/TableAdapter wizard. (Bug #29098)
- An exception would be thrown when using the Manage Role functionality within the web administrator to assign a role to a user. (Bug #29236)
- Using `TransactionScope` would cause an `InvalidOperationException`. (Bug #28709)
- The Saudi Hijri calendar was not supported. (Bug #29931)
- A `DATE` field would be updated with a date/time value, causing a `MySqlDataAdapter.Update()` exception. (Bug #30077)
- Connecting to a MySQL server earlier than version 4.1 would raise a `NullException`. (Bug #29476)
- Certain operations would not check the `UsageAdvisor` setting, causing log messages from the Usage Advisor even when it was disabled. (Bug #29124)

- Using the same connection string multiple times would result in `Database=dbname` appearing multiple times in the resulting string. (Bug #29123)
- A `FormatException` error would be raised if a parameter had not been found, instead of `Resources.ParameterMustBeDefined`. (Bug #29312)

### A.10.7. Changes in MySQL Connector/Net 5.1.2 (2007-06-18)

This is a new Beta development release, fixing recently discovered bugs.

#### Bugs Fixed

- Creating a user would fail due to the application name being set incorrectly. (Bug #28648)
- Accessing the results from a large query when using data compression in the connection would fail to return all the data. (Bug #28204)
- *Visual Studio Plugin*: Query Builder would fail to show `TINYTEXT` columns, and any columns listed after a `TINYTEXT` column correctly. (Bug #28437)
- *Visual Studio Plugin*: Adding a new query based on a stored procedure that used a `UPDATE`, `INSERT` or `DELETE` statement would terminate the query/TableAdapter wizard. (Bug #28536)
- Log messages would be truncated to 300 bytes. (Bug #28706)
- *Visual Studio Plugin*: Update commands would not be generated correctly when using the TableAdapter wizard. (Bug #26347)

### A.10.8. Changes in MySQL Connector/Net 5.1.1 (2007-05-23)

#### Bugs Fixed

- Running the statement `SHOW PROCESSLIST` would return columns as byte arrays instead of native columns. (Bug #28448)
- Building a connection string within a tight loop would show slow performance. (Bug #28167)
- Using `MySQLDataAdapter.FillSchema()` on a stored procedure would raise an exception: `Invalid attempt to access a field before calling Read()`. (Bug #27668)
- Fixed password property on `MySqlConnectionStringBuilder` to use `PasswordPropertyText` attribute. This causes dots to show instead of actual password text.
- Installation of the MySQL Connector/Net on Windows would fail if VisualStudio had not already been installed. (Bug #28260)
- `DATETIME` fields from versions of MySQL before 4.1 would be incorrectly parsed, resulting in a exception. (Bug #23342)
- The `UNSIGNED` flag for parameters in a stored procedure would be ignored when using `MySqlCommandBuilder` to obtain the parameter information. (Bug #27679)
- MySQL Connector/Net would look for the wrong table when executing `User.IsRole()`. (Bug #28251)

### A.10.9. Changes in MySQL Connector/Net 5.1.0 (2007-05-01)

#### Functionality Added or Changed

- Added Membership and Role provider contributed by Sean Wright (thanks!).

- GetSchema will now report objects relative to the currently selected database. What this means is that passing in null as a database restriction will report objects on the currently selected database only.
- Rewrote stored procedure parsing code using a new SQL tokenizer. Really nasty procedures including nested comments are now supported.
- Now compiles for .NET CF 2.0.

## A.11. Changes in MySQL Connector/Net Version 5.0

### A.11.1. Changes in MySQL Connector/Net 5.0.10 (Not released)

Version 5.0.10 has no changelog entries.

### A.11.2. Changes in MySQL Connector/Net 5.0.9 (Not released)

Version 5.0.9 has no changelog entries.

### A.11.3. Changes in MySQL Connector/Net 5.0.8 (2007-08-21)

#### Note

This version introduces a new installer technology.

#### Bugs Fixed

- Fixed some serious issues with command timeout and cancel that could present as exceptions about thread ownership. The issue was that not all queries cancel the same. Some produce resultsets while others don't. ExecuteReader had to be changed to check for this.
- An incorrect `ConstraintException` could be raised on an `INSERT` when adding rows to a table with a multiple-column unique key index. (Bug #30204)
- The availability of a MySQL server would not be reset when using pooled connections (`pooling=true`). This would lead to the server being reported as unavailable, even if the server become available while the application was still running. (Bug #29409)
- Fixed bug where MySQL Connector/Net was hand building some date time patterns rather than using the patterns provided under CultureInfo. This caused problems with some calendars that do not support the same ranges as Gregorian.. (Bug #29931)
- Fixed problem where a command timing out just after it actually finished would cause an exception to be thrown on the command timeout thread which would then be seen as an unhandled exception.
- Fixed problem where any attempt to not read all the records returned from a select where each row of the select is greater than 1024 bytes would hang the driver.
- Accessing the results from a large query when using data compression in the connection will fail to return all the data. (Bug #28204)
- Calling `SHOW CREATE PROCEDURE` for routines with a hyphen in the catalog name produced a syntax error. (Bug #29526)
- Fixed the database schema collection so that it works on servers that are not properly respecting the `lower_case_table_names` setting.
- A `DATE` field would be updated with a date/time value, causing a `MySqlDataAdapter.Update()` exception. (Bug #30077)
- Log messages would be truncated to 300 bytes. (Bug #28706)



- Certain operations would not check the `UsageAdvisor` setting, causing log messages from the Usage Advisor even when it was disabled. (Bug #29124)
- Fixed problem where we were not closing prepared statement handles when commands are disposed. This could lead to using up all prepared statement handles on the server.
- Fixed problem where `MySqlConnection.BeginTransaction` checked the drivers status var before checking if the connection was open. The result was that the driver could report an invalid condition on a previously opened connection.
- Extracting data through XML functions within a query returns the data as `System.Byte[]`. This was due to MySQL Connector/Net incorrectly identifying `BLOB` fields as binary, rather than text. (Bug #30233)
- Using the same connection string multiple times would result in `Database=dbname` appearing multiple times in the resulting string. (Bug #29123)
- A `FormatException` error would be raised if a parameter had not been found, instead of `Resources.ParameterMustBeDefined`. (Bug #29312)

#### A.11.4. Changes in MySQL Connector/Net 5.0.7 (2007-05-18)

##### Bugs Fixed

- When cloning an open `MySqlClient.MySqlConnection` with the `Persist Security Info=False` option set, the cloned connection is not usable because the security information has not been cloned. (Bug #27269)
- Running the statement `SHOW PROCESSLIST` would return columns as byte arrays instead of native columns. (Bug #28448)
- Enlisting a null transaction would affect the current connection object, such that further enlistment operations to the transaction are not possible. (Bug #26754)
- The `CreateFormat` column of the `DataTypes` collection did not contain a format specification for creating a new column type. (Bug #25947)
- Using logging (with the `logging=true` parameter to the connection string) would not generate a log file. (Bug #27765)
- Building a connection string within a tight loop would show slow performance. (Bug #28167)
- The `characterset` property would not be identified during a connection (also affected Visual Studio Plugin). (Bug #26147, Bug #27240)
- Using `MySQLDataAdapter.FillSchema()` on a stored procedure would raise an exception: `Invalid attempt to access a field before calling Read()`. (Bug #27668)
- Attempting to change the `Connection Protocol` property within a `PropertyGrid` control would raise an exception. (Bug #26472)
- `DATETIME` fields from versions of MySQL before 4.1 would be incorrectly parsed, resulting in a exception. (Bug #23342)
- The `UNSIGNED` flag for parameters in a stored procedure would be ignored when using `MySqlCommandBuilder` to obtain the parameter information. (Bug #27679)
- If you close an open connection with an active transaction, the transaction is not automatically rolled back. (Bug #27289)

#### A.11.5. Changes in MySQL Connector/Net 5.0.6 (2007-03-22)

##### Bugs Fixed

- `cmd.Parameters.RemoveAt( "Id" )` will cause an error if the last item is requested. (Bug #27187)
- `MySQLParameterCollection` and parameters added with `Insert` method can not be retrieved later using `ParameterName`. (Bug #27135)
- MySQL Visual Studio Plugin 1.1.2 does not work with MySQL Connector/Net 5.0.5. (Bug #26960)
- Exception thrown when using large values in `UInt64` parameters. (Bug #27093)
- Publisher listed in "Add/Remove Programs" is not consistent with other MySQL products. (Bug #27253)
- `DESCRIBE ....` SQL statement returns byte arrays rather than data on MySQL versions older than 4.1.15. (Bug #27221)

## A.11.6. Changes in MySQL Connector/Net 5.0.5 (2007-03-07)

### Functionality Added or Changed

- Added `Use Procedure Bodies` connection string option to enable calling procedures without using procedure metadata.
- Fixed problem with calling stored functions when a return parameter was not given.
- Fixed problem with parameter name hashing where the hashes were not getting updated when parameters were removed from the collection.
- Return parameters created with `DeriveParameters` now have the name `RETURN_VALUE`.
- Added `MySQLParameterCollection.AddWithValue` and marked the `Add(name, value)` method as obsolete.
- Fixed problem that prevented use of `SchemaOnly` or `SingleRow` command behaviors with stored procedures or prepared statements.
- Assembly now properly appears in the Visual Studio 2005 Add/Remove Reference dialog.
- Reverted behavior that required parameter names to start with the parameter marker. We apologize for this back and forth but we mistakenly changed the behavior to not match what `SqlClient` supports. We now support using either syntax for adding parameters however we also respond exactly like `SqlClient` in that if you ask for the index of a parameter using a syntax different from when you added the parameter, the result will be -1.

### Bugs Fixed

- Applications would crash when calling with `CommandType` set to `StoredProcedure`.
- `BINARY` and `VARBINARY` columns would be returned as a string, not binary, data type. (Bug #25605)
- `MySQLConnection` throws an exception when connecting to MySQL v4.1.7. (Bug #25726)
- Incorrect values/formats would be applied when the `OldSyntax` connection string option was used. (Bug #25950)
- High CPU utilization would be experienced when there is no idle connection waiting when using pooled connections through `MySQLPool.GetConnection`. (Bug #24373)
- Opening a connection would be slow due to host name lookup. (Bug #26152)
- The `UpdateRowSource.FirstReturnedRecord` method does not work. (Bug #25569)
- When closing and then re-opening a connection to a database, the character set specification is lost. (Bug #25614)



- Filling a table schema through a stored procedure triggers a runtime error. (Bug #25609)
- Returned data types of a [DataTypes](#) collection do not contain the right correct CLR data type. (Bug #25907)
- When connecting to a MySQL Server earlier than version 4.1, the connection would hang when reading data. (Bug #25458)
- Registry would be incorrectly populated with installation locations. (Bug #25928)
- [MySQLConnection.GetSchema](#) fails with [NullReferenceException](#) for Foreign Keys. (Bug #26660)
- Times with negative values would be returned incorrectly. (Bug #25912)
- MySQL Connector/Net would not compile properly when used with Mono 1.2. (Bug #24263)
- MySQL Connector/Net would fail to install under Windows Vista. (Bug #26430)
- Using [ExecuteScalar\(\)](#) with more than one query, where one query fails, will hang the connection. (Bug #25443)
- When a [MySQLConversionException](#) is raised on a remote object, the client application would receive a [SerializationException](#) instead. (Bug #24957)
- A critical [ConnectionPool](#) error would result in repeated [System.NullReferenceException](#). (Bug #25603)
- When connecting to a server, the return code from the connection could be zero, even though the host name was incorrect. (Bug #24802)
- [GetSchema](#) and [DataTypes](#) would throw an exception due to an incorrect table name. (Bug #25906)
- [SELECT](#) did not work correctly when using a [WHERE](#) clause containing a UTF-8 string. (Bug #25651)

### A.11.7. Changes in MySQL Connector/Net 5.0.4 (Not released)

This is a new Beta development release, fixing recently discovered bugs.

Version 5.0.4 has no changelog entries.

### A.11.8. Changes in MySQL Connector/Net 5.0.3 (2007-01-05)

#### Functionality Added or Changed

- SSL support has been updated.
- The ShapZipLib library has been replaced with the deflate support provided within .NET 2.0.
- Support for the embedded server and client library have been removed from this release. Support will be added back to a later release.
- Improved speed and performance by re-architecting certain sections of the code.
- The [ViewColumns](#) [GetSchema](#) collection has been updated.
- The [CommandBuilder.DeriveParameters](#) function has been updated to the procedure cache.
- Metadata from stored procedures and stored function execution are cached.
- The [MySQLCommand](#) object now supports asynchronous query methods. This is implemented using the [BeginExecuteNonQuery](#) and [EndExecuteNonQuery](#) methods.

- PerfMon hooks have been added to monitor the stored procedure cache hits and misses.
- Usage Advisor has been implemented. The Usage Advisor checks your queries and will report if you are using the connection inefficiently.

#### Bugs Fixed

- When using a `DBNull.Value` as the value for a parameter value, and then later setting a specific value type, the command would fail with an exception because the wrong type was implied from the `DBNull.Value`. (Bug #24565)
- Using `Driver.IsTooOld()` would return the wrong value. (Bug #24661)
- Nested transactions (which are unsupported) do not raise an error or warning. (Bug #22400)
- When adding parameter objects to a command object, if the parameter direction is set to `ReturnValue` before the parameter is added to the command object then when the command is executed it throws an error. (Bug #25013)
- Deleting a connection to a disconnected server when using the Visual Studio Plugin would cause an assertion failure. (Bug #23687)
- Additional text added to error message (Bug #25178)
- An exception would be raised, or the process would hang, if `SELECT` privileges on a database were not granted and a stored procedure was used. (Bug #25033)
- Stored procedure executions are not thread safe. (Bug #23905)

### A.11.9. Changes in MySQL Connector/Net 5.0.2 (2006-11-06)

#### Functionality Added or Changed

- **Important change:** Due to a number of issues with the use of server-side prepared statements, MySQL Connector/Net 5.0.2 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any way.

To enable server-side prepared statements you must add the following configuration property to your connector string properties:

```
ignore_prepare=false
```

The default value of this property is true.

- Implemented a stored procedure cache. By default, the connector caches the metadata for the last 25 procedures that are seen. You can change the number of procedures that are cached by using the `procedure cache` connection string.
- An `Ignore Prepare` option has been added to the connection string options. If enabled, prepared statements will be disabled application-wide. The default for this option is true.

#### Bugs Fixed

- Within Mono, using the `PreparedStatement` interface could result in an error due to a `BitArray` copying error. (Bug #18186)
- An exception would be thrown when calling `GetSchemaTable` and `fields` was null. (Bug #23538)
- During installation, an antivirus error message would be raised (indicating a malicious script problem). (Bug #23245)
- MySQL Connector/Net did not work as a data source for the `SqlDataSource` object used by ASP.NET 2.0. (Bug #16126)

- One system where IPv6 was enabled, MySQL Connector/Net would incorrectly resolve host names. (Bug #23758)
- A `System.FormatException` exception would be raised when invoking a stored procedure with an `ENUM` input parameter. (Bug #23268)
- Using Windows Vista (RC2) as a nonprivileged user would raise a `Registry key 'Global' access denied`. (Bug #22882)
- Column names with accented characters were not parsed properly causing malformed column names in result sets. (Bug #23657)
- Creating a connection through the Server Explorer when using the Visual Studio Plugin would fail. The installer for the Visual Studio Plugin has been updated to ensure that MySQL Connector/Net 5.0.2 must be installed. (Bug #23071)

### A.11.10. Changes in MySQL Connector/Net 5.0.1 (2006-10-01)

#### Bugs Fixed

- Submitting an empty string to a command object through `prepare` raises an `System.IndexOutOfRangeException`, rather than a MySQL Connector/Net exception. (Bug #18391)
- Executing multiple queries as part of a transaction returns `There is already an openDataReader associated with this Connection which must be closed first`. (Bug #7248)
- Using `ExecuteScalar` with a datetime field, where the value of the field is "0000-00-00 00:00:00", a `MySQLConversionException` exception would be raised. (Bug #11991)
- The `MySqlException` class is now derived from the `DbException` class. (Bug #21874)
- Incorrect field/data lengths could be returned for `VARCHAR` UTF8 columns. (Bug #14592)
- You can now install the MySQL Connector/Net MSI package from the command line using the `/passive`, `/quiet`, `/q` options. (Bug #19994)
- An `MySql.Data.Types.MySqlConversionException` would be raised when trying to update a row that contained a date field, where the date field contained a zero value (0000-00-00 00:00:00). (Bug #9619)
- Starting a transaction on a connection created by `MySql.Data.MySqlClient.MySqlClientFactory`, using `BeginTransaction` without specifying an isolation level, causes the SQL statement to fail with a syntax error. (Bug #22042)
- The `#` would not be accepted within column/table names, even though it was valid. (Bug #21521)
- MySQL Connector/Net on a Turkish operating system, may fail to execute certain SQL statements correctly. (Bug #22452)

### A.11.11. Changes in MySQL Connector/Net 5.0.0 (2006-08-08)

#### Functionality Added or Changed

- Implemented `MySqlConnectionBuilder` class.
- Completely refactored how column values are handled to avoid boxing in some cases.
- Implemented Usage Advisor.
- Added Async query methods.

- Implemented classes and interfaces for ADO.Net 2.0 support.
- Added perfmon hooks for stored procedure cache hits and misses.
- Implemented [MySqlClientFactory](#) class.
- Added internal implementation of SHA1 so we don't have to distribute the OpenNetCF on mobile devices.
- Added procedure metadata caching.
- Reworked connection string classes to be simpler and faster.
- Reimplemented PacketReader/PacketWriter support into [MySqlStream](#) class.
- Added usage advisor warnings for requesting column values by the wrong type.
- Refactored test suite to test all protocols in a single pass.
- Replaced use of ICSharpCode with .NET 2.0 internal deflate support.

#### Bugs Fixed

- CommandText: Question mark in comment line is being parsed as a parameter. (Bug #6214)

## A.12. Changes in MySQL Connector/Net Version 1.0

### A.12.1. Changes in MySQL Connector/Net 1.0.11 (Not released)

Version 1.0.11 has no changelog entries.

### A.12.2. Changes in MySQL Connector/Net 1.0.10 (2007-08-24)

#### Bugs Fixed

- [BINARY](#) and [VARBINARY](#) columns would be returned as a string, not binary, data type. (Bug #25605)
- An incorrect [ConstraintException](#) could be raised on an [INSERT](#) when adding rows to a table with a multiple-column unique key index. (Bug #30204)
- [MySqlParameterCollection](#) and parameters added with [Insert](#) method can not be retrieved later using [ParameterName](#). (Bug #27135)
- The availability of a MySQL server would not be reset when using pooled connections ([pooling=true](#)). This would lead to the server being reported as unavailable, even if the server become available while the application was still running. (Bug #29409)
- High CPU utilization would be experienced when there is no idle connection waiting when using pooled connections through [MySqlConnection.GetConnection](#). (Bug #24373)
- Publisher listed in "Add/Remove Programs" is not consistent with other MySQL products. (Bug #27253)
- When a [MySQLConversionException](#) is raised on a remote object, the client application would receive a [SerializationException](#) instead. (Bug #24957)
- A critical [ConnectionPool](#) error would result in repeated [System.NullReferenceException](#). (Bug #25603)

### A.12.3. Changes in MySQL Connector/Net 1.0.9 (2007-02-02)

#### Functionality Added or Changed

- **Important change:** Due to a number of issues with the use of server-side prepared statements, MySQL Connector/Net 5.0.2 has disabled their use by default. The disabling of server-side prepared statements does not affect the operation of the connector in any way.

To enable server-side prepared statements you must add the following configuration property to your connector string properties:

```
ignore_prepare=false
```

The default value of this property is true.

- Implemented a stored procedure cache. By default, the connector caches the metadata for the last 25 procedures that are seen. You can change the number of procedures that are cached by using the `procedure cache` connection string.
- An `Ignore Prepare` option has been added to the connection string options. If enabled, prepared statements will be disabled application-wide. The default for this option is true.
- Improved `CommandBuilder.DeriveParameters` to first try and use the procedure cache before querying for the stored procedure metadata. Return parameters created with `DeriveParameters` now have the name `RETURN_VALUE`.
- **Important change:** Binaries for .NET 1.0 are no longer supplied with this release. If you need support for .NET 1.0, you must build from source.
- The ICSharpCode ZipLib is no longer used by the Connector, and is no longer distributed with it.

### Bugs Fixed

- When using a `DBNull.Value` as the value for a parameter value, and then later setting a specific value type, the command would fail with an exception because the wrong type was implied from the `DBNull.Value`. (Bug #24565)
- Within Mono, using the `PreparedStatement` interface could result in an error due to a `BitArray` copying error. (Bug #18186)
- An `System.OverflowException` would be raised when accessing a varchar field over 255 bytes. (Bug #23749)
- Using `Driver.IsTooOld()` would return the wrong value. (Bug #24661)
- Nested transactions do not raise an error or warning. (Bug #22400)
- Additional text added to error message. (Bug #25178)
- When adding parameter objects to a command object, if the parameter direction is set to `ReturnValue` before the parameter is added to the command object then when the command is executed it throws an error. (Bug #25013)
- `MySQLConnection` throws a `NullReferenceException` and `ArgumentNullException` when connecting to MySQL v4.1.7. (Bug #25726)
- When closing and then re-opening a connection to a database, the character set specification is lost. (Bug #25614)
- Trying to fill a table schema through a stored procedure triggers a runtime error. (Bug #25609)
- One system where IPv6 was enabled, MySQL Connector/Net would incorrectly resolve host names. (Bug #23758)
- Times with negative values would be returned incorrectly. (Bug #25912)
- The `CommandBuilder` would mistakenly add insert parameters for a table column with auto incrementation enabled. (Bug #23862)

- Using `ExecuteScalar()` with more than one query, where one query fails, will hang the connection. (Bug #25443)
- When connecting to a server, the return code from the connection could be zero, even though the host name was incorrect. (Bug #24802)
- Stored procedure executions are not thread safe. (Bug #23905)
- `SELECT` did not work correctly when using a `WHERE` clause containing a UTF-8 string. (Bug #25651)

## A.12.4. Changes in MySQL Connector/Net 1.0.8 (2006-10-20)

### Functionality Added or Changed

- The method for retrieving stored procedure metadata has been changed so that users without `SELECT` privileges on the `mysql.proc` table can use a stored procedure.
- Stored procedures are now cached.

### Bugs Fixed

- When working with multiple threads, character set initialization would generate errors. (Bug #17106)
- When using `MySqlDataAdapter`, connections to a MySQL server may remain open and active, even though the use of the connection has been completed and the data received. (Bug #8131)
- Submitting an empty string to a command object through `prepare` raises an `System.IndexOutOfRangeException`, rather than a MySQL Connector/Net exception. (Bug #18391)
- Called `MySqlCommandBuilder.DeriveParameters` for a stored procedure that has no parameters would cause an application crash. (Bug #15077)
- The connection string parser did not permit single or double quotation marks in the password. (Bug #16659)
- The `CommandBuilder` ignored `Unsigned` flag at `Parameter` creation. (Bug #17375)
- Executing multiple queries as part of a transaction returns `There is already an openDataReader associated with this Connection which must be closed first.` (Bug #7248)
- `CHAR` type added to `MySqlDbType`. (Bug #17749)
- Using `ExecuteScalar` with a datetime field, where the value of the field is "0000-00-00 00:00:00", a `MySqlConversionException` exception would be raised. (Bug #11991)
- Parameter substitution in queries where the order of parameters and table fields did not match would substitute incorrect values. (Bug #19261)
- The `MySqlDateTime` class did not contain constructors. (Bug #15112)
- When running a query that included a date comparison, a `DateReader` error would be raised. (Bug #19481)
- When using an unsigned 64-bit integer in a stored procedure, the unsigned bit would be lost stored. (Bug #16934)
- Incorrect field/data lengths could be returned for `VARCHAR` UTF8 columns. (Bug #14592)
- A `SELECT` query on a table with a date with a value of '0000-00-00' would hang the application. (Bug #17736)

- You can now install the MySQL Connector/Net MSI package from the command line using the `/passive`, `/quiet`, `/q` options. (Bug #19994)
- An `MySql.Data.Types.MySqlConversionException` would be raised when trying to update a row that contained a date field, where the date field contained a zero value (0000-00-00 00:00:00). (Bug #9619)
- The `DiscoverParameters` function would fail when a stored procedure used a `NUMERIC` parameter type. (Bug #19515)
- Unsigned data types were not properly supported. (Bug #16788)
- The `#` would not be accepted within column/table names, even though it was valid. (Bug #21521)
- `DataReader` would show the value of the previous row (or last row with nonnull data) if the current row contained a `datetime` field with a null value. (Bug #16884)
- `IDataRecord.GetString` would raise `NullPointerException` for null values in returned rows. Method now throws `SqlNullValueException`. (Bug #19294)
- MySQL Connector/Net on a Turkish operating system, may fail to execute certain SQL statements correctly. (Bug #22452)
- Calling `Close` on a connection after calling a stored procedure would trigger a `NullReferenceException`. (Bug #20581)
- An exception would be raised when using an output parameter to a `System.String` value. (Bug #17814)

### A.12.5. Changes in MySQL Connector/Net 1.0.7 (2005-11-21)

#### Bugs Fixed

- Calling a stored procedure where a parameter contained special characters (such as `'@'`) would produce an exception. Note that `ANSI_QUOTES` had to be enabled to make this possible. (Bug #13753)
- A `#42000Query was empty` exception occurred when executing a query built with `MySqlCommandBuilder`, if the query string ended with a semicolon. (Bug #14631)
- The parameter collection object's `Add()` method added parameters to the list without first checking to see whether they already existed. Now it updates the value of the existing parameter object if it exists. (Bug #13927)
- Added support for the `cp932` character set. (Bug #13806)
- The `Ping()` method did not update the `State` property of the `Connection` object. (Bug #13658)
- Implemented the `MySqlCommandBuilder.DeriveParameters` method that is used to discover the parameters for a stored procedure. (Bug #13632)
- A statement that contained multiple references to the same parameter could not be prepared. (Bug #13541)
- Unsigned `tinyint` (NET byte) would lead to and incorrectly determined parameter type from the parameter value. (Bug #18570)

### A.12.6. Changes in MySQL Connector/Net 1.0.6 (2005-10-03)

#### Bugs Fixed

- Field names that contained the following characters caused errors: `()%<>/` (Bug #13036)



- MySQL Connector/Net 1.0.5 could not connect on Mono. (Bug #13345)
- The MySQL Connector/Net 1.0.5 installer would not install alongside MySQL Connector/Net 1.0.4. (Bug #12835)
- Serializing a parameter failed if the first value passed in was `NULL`. (Bug #13276)
- The `nant` build sequence had problems. (Bug #12978)

## A.12.7. Changes in MySQL Connector/Net 1.0.5 (2005-08-29)

### Bugs Fixed

- A call to a stored procedure caused an exception if the stored procedure had no parameters. (Bug #11542)
- Added support to call a stored function from MySQL Connector/Net. (Bug #10644)
- MySQL Connector/Net interpreted the new decimal data type as a byte array. (Bug #11294)
- Trying to use a stored procedure when `Connection.Database` was not populated generated an exception. (Bug #11450)
- The `ConnectionString` property could not be set when a `MySqlConnection` object was added with the designer. (Bug #12551, Bug #8724)
- With multiple hosts in the connection string, MySQL Connector/Net would not connect to the last host in the list. (Bug #12628)
- Connection could fail when .NET thread pool had no available worker threads. (Bug #10637)
- MySQL Connector/Net could not work properly with certain regional settings. (WL#8228)
- Certain malformed queries would trigger a `Connection must be valid and open` error message. (Bug #11490)
- MySQL Connector/Net could not connect to MySQL 4.1.14. (Bug #12771)
- Trying to read a `TIMESTAMP` column generated an exception. (Bug #7951)
- Decimal parameters caused syntax errors. (Bug #10152, Bug #11550, Bug #10486)
- Parameters were not recognized when they were separated by linefeeds. (Bug #9722)
- The `cp1250` character set was not supported. (Bug #11621)
- Calling `MySqlConnection.clone` when a connection string had not yet been set on the original connection would generate an error. (Bug #10281)
- The `MySqlCommandBuilder` class could not handle queries that referenced tables in a database other than the default database. (Bug #8382)

## A.12.8. Changes in MySQL Connector/Net 1.0.4 (2005-01-20)

### Bugs Fixed

- `MySqlCommand.Connection` returns an `IDbConnection`. (Bug #7258)
- Fixed problem that causes named pipes to not work with some blob functionality.
- Fixed another small problem with prepared statements.
- Quote character `\222` not quoted in `EscapeString`. (Bug #7724)



- Clone method bug in `MySqlCommand`. (Bug #7478)
- Added or filled out several more topics in the API reference documentation.
- Fixed problem with shared memory connections.
- Calling prepare causing exception. (Bug #7243)
- `MySqlDataReader.GetString(index)` returns non-Null value when field is `Null`. (Bug #7612)
- `MySqlAdapter.Fill` method throws error message `Non-negative number required`. (Bug #7345)
- Problem with Multiple resultsets. (Bug #7436)
- `MySqlReader.GetInt32` throws exception if column is unsigned. (Bug #7755)
- `GetBytes` was not working. (Bug #7704)

### A.12.9. Changes in MySQL Connector/Net 1.0.3 (2004-10-12, gamma)

#### Bugs Fixed

- Made MySQL the default named pipe name.
- Now `SHOW COLLATION` is used upon connection to retrieve the full list of charset ids.
- Inserting `DateTime` causes `System.InvalidCastException` to be thrown. (Bug #7132)
- Integer "out" parameter from stored procedure returned as string. (Bug #6668)
- Int64 Support in `MySqlCommand` Parameters. (Bug #6863)
- Changed the name of the test suite to `MySql.Data.Tests.dll`.
- Added Ping method to `MySqlConnection`.
- Invalid query string when using inout parameters (Bug #7133)
- Errors in parsing stored procedure parameters. (Bug #6902)
- Added `ServerThread` property to `MySqlConnection` to expose server thread id.
- InvalidCast when using `DATE_ADD`-function. (Bug #6879)
- Fixed problem where multiple resultsets having different numbers of columns would cause a problem.
- `MySqlDateTime` in Datatables sorting by Text, not Date. (Bug #7032)
- `MySqlDataReader.GetChar(int i)` throws `IndexOutOfRangeException` exception. (Bug #6770)
- An Open Connection has been Closed by the Host System. (Bug #6634)
- Fixed Invalid character set index: 200. (Bug #6547)
- Exception stack trace lost when re-throwing exceptions. (Bug #6983)
- Fixed major problem with detecting null values when using prepared statements.
- Installer now includes options to install into GAC and create Start Menu items.
- Connections now do not have to give a database on the connection string.

- Test suite fails with MySQL 4.0 because of case sensitivity of table names. (Bug #6831)

## A.12.10. Changes in MySQL Connector/Net 1.0.2 (2004-11-15, gamma)

### Bugs Fixed

- Fixed problem where calling stored procedures might cause an "Illegal mix of collations" problem.
- Fixed Objects not being disposed (Bug #6649)
- Fixed double type handling in `MySqlParameter(string parameterName, object value)`. (Bug #6428)
- Fixed #HY000 Illegal mix of collations (latin1\_swedish\_ci,IMPLICIT) and (utf8\_general\_ (Bug #6322)
- Fixed Installation directory ignored using custom installation (Bug #6329)
- Fixed Long inserts take very long time (Bu #5453)
- Fixed Charset-map for UCS-2 (Bug #6541)
- Fixed Zero date "0000-00-00" is returned wrong when filling Dataset (Bug #6429)
- Fixed problem where setting command text leaves the command in a prepared state
- Updated the installer to include the new samples
- Provider is now using character set specified by server as default
- Fixed problem with `MySqlBinary` where string values could not be used to update extended text columns
- Added charset connection string option
- Added the TableEditor CS and VB sample

## A.12.11. Changes in MySQL Connector/Net 1.0.1 (2004-10-27, Beta)

### Bugs Fixed

- Virtualized driver subsystem so future releases could easily support client or embedded server support
- Setting `DbType` threw a `NullReferenceException`. (Bug #5469)
- Fixed constructor initialize problems in `MySqlCommand()` (Bug #5613)
- Fixed `System.OverflowException` when using `YEAR` data type. (Bug #6036)
- Fixed Yet Another "object reference not set to an instance of an object" (Bug #5496)
- CP1252 is now used for Latin1 only when the server is 4.1.2 and later
- Refactored compression code into `CompressedStream` to clean up `NativeDriver`
- Fixed Parsing the ';' char (Bug #5876)
- Fixed problem where Min Pool Size was not being respected
- Fixed problem where connector was not issuing a `CMD_QUIT` before closing the socket
- Fixed `MySqlDataReader` and 'show tables from ...' behavior (Bug #5256)
- Fixed problem where `MySqlParameterCollection.Add()` would throw unclear exception when given a null value (Bug #5621)

- Cannot run a stored procedure populating `mysqlcommand.parameters` (Bug #5474)
- Fixed Russian character support as well
- Fixed problem with `ConnectionInternal` where a key might be added more than once
- Implemented `SequentialAccess`
- Fixed `IndexOutOfBounds` when reading `BLOB` with `DataReader` with `GetString(index)`. (Bug #6230)
- Fixed `MySqlDateTime` sets `IsZero` property on all subseq.records after first zero found (Bug #6006)
- Using `PacketWriter` instead of `Packet` for writing to streams
- Added Aggregate function test (wasn't really a bug)
- `MySqlCommand` saw instances of "?" as parameters in string literals. (Bug #5392)
- Field buffers being reused to decrease memory allocations and increase speed
- Possible bug in `MySqlParameter(string, object)` constructor (Bug #5602)
- Fixed NET Connector source missing `resx` files (Bug #6216)
- Fixed serializing of floating point parameters (double, numeric, single, decimal) (Bug #5900)
- Fixed `Method TokenizeSql()` uses only a limited set of valid characters for parameters (Bug #6217)
- Fixed problem where connection lifetime on the connect string was not being respected
- `DataReader` reported all rows as `NULL` if one row was `NULL`. (Bug #5388)
- Fixed Can't display Chinese correctly (Bug #5288)
- Fixed missing Reference in `DbType` setter (Bug #5897)
- Added test case for resetting the command text on a prepared command
- Fixed `GetBoolean` returns wrong values (Bug #6227)
- `IsNullable` error (Bug #5796)
- Fixed `DBNull` Values causing problems with retrieving/updating queries. (Bug #5798)
- Calling `GetChars` on a `LONGTEXT` column threw an exception. (Bug #5458)
- Fixed problem where using old syntax while using the interfaces caused problems
- Fixed problem in `PacketReader` where it could try to allocate the wrong buffer size in `EnsureCapacity`

## A.12.12. Changes in MySQL Connector/Net 1.0.0 (2004-09-01)

### Bugs Fixed

- Removed some last references to `ByteFX`.
- Fixed problem with using compression.
- Updated many of the test cases.
- Thai encoding not correctly supported. (Bug #3889)
- Added `COPYING.rtf` file for use in installer.

- Removed all of the XML comment warnings.
- Bumped version number to 1.0.0 for beta 1 release.

## A.13. Changes in MySQL Connector/Net Version 0.9.0 (30 August 2004)

- Added test fixture for prepared statements.
- All type classes now implement a `SerializeBinary` method for sending their data to a `PacketWriter`.
- Added `PacketWriter` class that will enable future low-memory large object handling.
- Fixed many small bugs in running prepared statements and stored procedures.
- Changed command so that an exception will not be thrown in executing a stored procedure with parameters in old syntax mode.
- `SingleRow` behavior now working right even with limit.
- `GetBytes` now only works on binary columns.
- Logger now truncates long SQL commands so blob columns do not blow out our log.
- Host and database now have a default value of "" unless otherwise set.
- Connection Timeout seems to be ignored. (Bug #5214)
- Added test case for bug# 5051: GetSchema not working correctly.
- Fixed problem where `GetSchema` would return false for `IsUnique` when the column is key.
- `MySqlDataReader` `GetXXX` methods now using the field level `MySqlValue` object and not performing conversions.
- `DataReader` returning `NULL` for time column. (Bug #5097)
- Added test case for `LOAD DATA LOCAL INFILE`.
- Added `replacetext` custom nant task.
- Added `CommandBuilderTest` fixture.
- Added Last One Wins feature to `CommandBuilder`.
- Fixed persist security info case problem.
- Fixed `GetBool` so that 1, true, "true", and "yes" all count as true.
- Make parameter mark configurable.
- Added the "old syntax" connection string parameter to enable use of @ parameter marker.
- `MySqlCommandBuilder`. (Bug #4658)
- `ByteFX.MySqlClient` caches passwords if `Persist Security Info` is false. (Bug #4864)
- Updated license banner in all source files to include FLOSS exception.
- Added new `.Types` namespace and implementations for most current MySql types.
- Added `MySqlField41` as a subclass of `MySqlField`.

- Changed many classes to now use the new `.Types` types.
- Changed type `enum int` to `Int32`, `short` to `Int16`, and `bigint` to `Int64`.
- Added dummy types `UInt16`, `UInt32`, and `UInt64` to allow an unsigned parameter to be made.
- Connections are now reset when they are pulled from the connection pool.
- Refactored auth code in driver so it can be used for both auth and reset.
- Added `UserReset` test in `PoolingTests.cs`.
- Connections are now reset using `COM_CHANGE_USER` when pulled from the pool.
- Implemented `SingleResultSet` behavior.
- Implemented support of unicode.
- Added char set mappings for utf-8 and ucs-2.
- Time fields overflow using `bytefx .net mysql driver` (Bug #4520)
- Modified time test in data type test fixture to check for time spans where hours > 24.
- Wrong string with backslash escaping in `ByteFx.Data.MySqlClient.MySqlParameter`. (Bug #4505)
- Added code to Parameter test case `TestQuoting` to test for backslashes.
- `MySqlCommandBuilder` fails with multi-word column names. (Bug #4486)
- Fixed bug in `TokenizeSql` where underscore would terminate character capture in parameter name.
- Added test case for spaces in column names.
- `MySqlDataReader.GetBytes` do not work correctly. (Bug #4324)
- Added `GetBytes()` test case to `DataReader` test fixture.
- Now reading all server variables in `InternalConnection.Configure` into `Hashtable`.
- Now using `string[]` for index map in `CharSetMap`.
- Added `CRInSQL` test case for carriage returns in SQL.
- Setting `maxPacketSize` to default value in `Driver.ctor`.
- Setting `MySqlDbType` on a parameter doesn't set generic type. (Bug #4442)
- Removed obsolete data types `Long` and `LongLong`.
- Overflow exception thrown when using "use pipe" on connection string. (Bug #4071)
- Changed "use pipe" keyword to "pipe name" or just "pipe".
- Enable reading multiple resultsets from a single query.
- Added flags attribute to `ServerStatusFlags` enum.
- Changed name of `ServerStatus` enum to `ServerStatusFlags`.
- Inserted data row doesn't update properly.
- Error processing show create table. (Bug #4074)

- Change `Packet.ReadLenInteger` to `ReadPackedLong` and added `packet.ReadPackedInteger` that always reads integers packed with 2,3,4.
- Added `syntax.cs` test fixture to test various SQL syntax bugs.
- Improper handling of time values. Now time value of 00:00:00 is not treated as null. (Bug #4149)
- Moved all test suite files into `TestSuite` folder.
- Fixed bug where null column would move the result packet pointer backward.
- Added new nant build script.
- Clear tablename so it will be regen'ed properly during the next `GenerateSchema`. (Bug #3917)
- `GetValues` was always returning zero and was also always trying to copy all fields rather than respecting the size of the array passed in. (Bug #3915)
- Implemented shared memory access protocol.
- Implemented prepared statements for MySQL 4.1.
- Implemented stored procedures for MySQL 5.0.
- Renamed `MySqlInternalConnection` to `InternalConnection`.
- SQL is now parsed as chars, fixes problems with other languages.
- Added logging and allow batch connection string options.
- `RowUpdating` event not set when setting the `DataAdapter` property. (Bug #3888)
- Fixed bug in char set mapping.
- Implemented 4.1 authentication.
- Improved open/auth code in driver.
- Improved how connection bits are set during connection.
- Database name is now passed to server during initial handshake.
- Changed namespace for client to `MySql.Data.MySqlClient`.
- Changed assembly name of client to `MySql.Data.dll`.
- Changed license text in all source files to GPL.
- Added the `MySqlClient.build` Nant file.
- Removed the mono batch files.
- Moved some of the unused files into notused folder so nant build file can use wildcards.
- Implemented shared memory access.
- Major revamp in code structure.
- Prepared statements now working for MySql 4.1.1 and later.
- Finished implementing auth for 4.0, 4.1.0, and 4.1.1.
- Changed namespace from `MySQL.Data.MySQLClient` back to `MySql.Data.MySqlClient`.
- Fixed bug in `CharSetMapping` where it was trying to use text names as ints.

- Changed namespace to `MySQL.Data.MySQLClient`.
- Integrated auth changes from UC2004.
- Fixed bug where calling any of the GetXXX methods on a datareader before or after reading data would not throw the appropriate exception (thanks Luca Morelli).
- Added `TimeSpan` code in parameter.cs to properly serialize a timespan object to mysql time format (thanks Gianluca Colombo).
- Added `TimeStamp` to parameter serialization code. Prevented `DataAdapter` updates from working right (thanks Michael King).
- Fixed a misspelling in `MySqlHelper.cs` (thanks Patrick Kristiansen).

## A.14. Changes in MySQL Connector/Net Version 0.76

- Driver now using charset number given in handshake to create encoding.
- Changed command editor to point to `MySqlClient.Design`.
- Fixed bug in `Version.isAtLeast`.
- Changed `DBConnectionString` to support changes done to `MySqlConnectionString`.
- Removed `SqlCommandEditor` and `DataAdapterPreviewDialog`.
- Using new long return values in many places.
- Integrated new `CompressedStream` class.
- Changed `ConnectionString` and added attributes to permit it to be used in `MySqlClient.Design`.
- Changed `packet.cs` to support newer lengths in `ReadLenInteger`.
- Changed other classes to use new properties and fields of `MySqlConnectionString`.
- `ConnectionInternal` is now using PING to see whether the server is available.
- Moved toolbox bitmaps into resource folder.
- Changed `field.cs` to permit values to come directly from row buffer.
- Changed to use the new driver.Send syntax.
- Using a new packet queueing system.
- Started work handling the "broken" compression packet handling.
- Fixed bug in `StreamCreator` where failure to connect to a host would continue to loop infinitely (thanks Kevin Casella).
- Improved connectstring handling.
- Moved designers into Pro product.
- Removed some old commented out code from `command.cs`.
- Fixed a problem with compression.
- Fixed connection object where an exception throw prior to the connection opening would not leave the connection in the connecting state (thanks Chris Cline).

- Added GUID support.
- Fixed sequence out of order bug (thanks Mark Reay).

## A.15. Changes in MySQL Connector/Net Version 0.75

- Enum values now supported as parameter values (thanks Philipp Sumi).
- Year data type now supported.
- Fixed compression.
- Fixed bug where a parameter with a `TimeSpan` as the value would not serialize properly.
- Fixed bug where default constructor would not set default connection string values.
- Added some XML comments to some members.
- Work to fix/improve compression handling.
- Improved `ConnectionString` handling so that it better matches the standard set by `SqlClient`.
- A `MySqlException` is now thrown if a user name is not included in the connection string.
- Localhost is now used as the default if not specified on the connection string.
- An exception is now thrown if an attempt is made to set the connection string while the connection is open.
- Small changes to `ConnectionString` docs.
- Removed `MultiHostStream` and `MySqlStream`. Replaced it with `Common/StreamCreator`.
- Added support for Use Pipe connection string value.
- Added Platform class for easier access to platform utility functions.
- Fixed small pooling bug where new connection was not getting created after `IsAlive` fails.
- Added `Platform.cs` and `StreamCreator.cs`.
- Fixed `Field.cs` to properly handle 4.1 style timestamps.
- Changed `Common.Version` to `Common.DBVersion` to avoid name conflict.
- Fixed `field.cs` so that text columns return the right field type.
- Added `MySqlError` class to provide some reference for error codes (thanks Geert Veenstra).

## A.16. Changes in MySQL Connector/Net Version 0.74

- Added Unix socket support (thanks Mohammad DAMT).
- Only calling `Thread.Sleep` when no data is available.
- Improved escaping of quote characters in parameter data.
- Removed misleading comments from `parameter.cs`.
- Fixed pooling bug.
- Fixed `ConnectionString` editor dialog (thanks marco p (pomarc)).



- `UserId` now supported in connection strings (thanks Jeff Neeley).
- Attempting to create a parameter that is not input throws an exception (thanks Ryan Gregg).
- Added much documentation.
- Checked in new `MultiHostStream` capability. Big thanks to Dan Guisinger for this. he originally submitted the code and idea of supporting multiple machines on the connect string.
- Added a lot of documentation.
- Fixed speed issue with 0.73.
- Changed to `Thread.Sleep(0)` in `MySqlDataStream` to help optimize the case where it doesn't need to wait (thanks Todd German).
- Prepopulating the idlepools to `MinPoolSize`.
- Fixed `MySqlPool` deadlock condition as well as stupid bug where `CreateNewPooledConnection` was not ever adding new connections to the pool. Also fixed `MySqlStream.ReadBytes` and `ReadByte` to not use `TicksPerSecond` which does not appear to always be right. (thanks Matthew J. Peddlesden)
- Fix for precision and scale (thanks Matthew J. Peddlesden).
- Added `Thread.Sleep(1)` to stream reading methods to be more cpu friendly (thanks Sean McGinnis).
- Fixed problem where `ExecuteReader` would sometime return null (thanks Lloyd Dupont).
- Fixed major bug with null field handling (thanks Naucki).
- Enclosed queries for `max_allowed_packet` and `characteraset` inside try catch (and set defaults).
- Fixed problem where socket was not getting closed properly (thanks Steve!).
- Fixed problem where `ExecuteNonQuery` was not always returning the right value.
- Fixed `InternalConnection` to not use `@@session.max_allowed_packet` but use `@@max_allowed_packet`. (Thanks Miguel)
- Added many new XML doc lines.
- Fixed SQL parsing to not send empty queries (thanks Rory).
- Fixed problem where the reader was not unpeeking the packet on close.
- Fixed problem where user variables were not being handled (thanks Sami Vaaraniemi).
- Fixed loop checking in the `MySqlPool` (thanks Steve M. Brown)
- Fixed `ParameterCollection.Add` method to match `SqlClient` (thanks Joshua Mouch).
- Fixed `ConnectionString` parsing to handle no and yes for boolean and not lowercase values (thanks Naucki).
- Added `InternalConnection` class, changes to pooling.
- Implemented Persist Security Info.
- Added `security.cs` and `version.cs` to project
- Fixed `DateTime` handling in `Parameter.cs` (thanks Burkhard Perens-Golomb).

- Fixed parameter serialization where some types would throw a cast exception.
- Fixed `DataReader` to convert all returned values to prevent casting errors (thanks Keith Murray).
- Added code to `Command.ExecuteReader` to return null if the initial SQL statement throws an exception (thanks Burkhard Perkens-Golomb).
- Fixed `ExecuteScalar` bug introduced with restructure.
- Restructure to permit `LOCAL DATA INFILE` and better sequencing of packets.
- Fixed several bugs related to restructure.
- Early work done to support more secure passwords in MySQL 4.1. Old passwords in 4.1 not supported yet.
- Parameters appearing after system parameters are now handled correctly (Adam M. (adammil)).
- Strings can now be assigned directly to blob fields (Adam M.).
- Fixed float parameters (thanks Pent).
- Improved Parameter constructor and `ParameterCollection.Add` methods to better match `SqlClient` (thanks Joshua Mouch).
- Corrected `Connection.CreateCommand` to return a `MySqlCommand` type.
- Fixed connection string designer dialog box problem (thanks Abraham Guyt).
- Fixed problem with sending commands not always reading the response packet (thanks Joshua Mouch).
- Fixed parameter serialization where some blobs types were not being handled (thanks Sean McGinnis).
- Removed spurious `MessageBox.show` from `DataReader` code (thanks Joshua Mouch).
- Fixed a nasty bug in the split SQL code (thanks everyone!).

## A.17. Changes in MySQL Connector/Net Version 0.71

- Fixed bug in `MySqlStream` where too much data could attempt to be read (thanks Peter Belbin)
- Implemented `HasRows` (thanks Nash Pherson).
- Fixed bug where tables with more than 252 columns cause an exception (thanks Joshua Kessler).
- Fixed bug where SQL statements ending in ; would cause a problem (thanks Shane Krueger).
- Fixed bug in driver where error messages were getting truncated by 1 character (thanks Shane Krueger).
- Made `MySqlException` serializable (thanks Mathias Hasselmann).

## A.18. Changes in MySQL Connector/Net Version 0.70

- Updated some of the character code pages to be more accurate.
- Fixed problem where readers could be opened on connections that had readers open.
- Moved test to separate assembly `MySqlClientTests`.
- Fixed stupid problem in driver with sequence out of order (Thanks Peter Belbin).

- Added some pipe tests.
- Increased default max pool size to 50.
- Compiles with Mono 0-24.
- Fixed connection and data reader dispose problems.
- Added `String` data type handling to parameter serialization.
- Fixed sequence problem in driver that occurred after thrown exception (thanks Burkhard Perken-Golomb).
- Added support for `CommandBehavior.SingleRow` to `DataReader`.
- Fixed command SQL processing so quotation marks are better handled (thanks Theo Spears).
- Fixed parsing of double, single, and decimal values to account for non-English separators. You still have to use the right syntax if you using hard coded SQL, but if you use parameters the code will convert floating point types to use '.' appropriately internal both into the server and out.
- Added `MySqlStream` class to simplify timeouts and driver coding.
- Fixed `DataReader` so that it is closed properly when the associated connection is closed. [thanks smishra]
- Made client more `SqlClient` compliant so that `DataReaders` have to be closed before the connection can be used to run another command.
- Improved `DBNull.Value` handling in the fields.
- Added several unit tests.
- Fixed `MySqlException` base class.
- Improved driver coding
- Fixed bug where `NextResult` was returning false on the last resultset.
- Added more tests for MySQL.
- Improved casting problems by equating unsigned 32bit values to `Int64` and unsigned 16bit values to `Int32`, and so forth.
- Added new constructor for `MySqlParameter` for (name, type, size, srccol)
- Fixed bug in `MySqlDataReader` where it didn't check for null fieldlist before returning field count.
- Started adding `MySqlClient` unit tests (added `MySqlClient/Tests` folder and some test cases).
- Fixed some things in Connection String handling.
- Moved `INIT_DB` to `MySqlPool`. I may move it again, this is in preparation of the conference.
- Fixed bug inside `CommandBuilder` that prevented inserts from happening properly.
- Reworked some of the internals so that all three execute methods of `Command` worked properly.
- Fixed many small bugs found during benchmarking.
- The first cut of `CoonnectionPooling` is working. "min pool size" and "max pool size" are respected.
- Work to enable multiple resultsets to be returned.

- Character sets are handled much more intelligently now. The driver queries MySQL at startup for the default character set. That character set is then used for conversions if that code page can be loaded. If not, then the default code page for the current OS is used.
- Added code to save the inferred type in the name,value constructor of `Parameter`.
- Also, inferred type if value of null parameter is changed using `Value` property.
- Converted all files to use proper Camel case. MySQL is now MySql in all files. PostgreSQL is now Pgsql.
- Added attribute to Pgsql code to prevent designer from trying to show.
- Added `MySQLDbType` property to Parameter object and added proper conversion code to convert from `DbType` to `MySQLDbType`.
- Removed unused `ObjectToString` method from `MySQLParameter.cs`.
- Fixed `Add(...)` method in `ParameterCollection` so that it doesn't use `Add(name, value)` instead.
- Fixed `IndexOf` and `Contains` in `ParameterCollection` to be aware that parameter names are now stored without `@`.
- Fixed `Command.ConvertSQLToBytes` so it only permits characters that can be in MySQL variable names.
- Fixed `DataReader` and `Field` so that blob fields read their data from `Field.cs` and `GetBytes` works right.
- Added simple query builder editor to `CommandText` property of `MySQLCommand`.
- Fixed `CommandBuilder` and `Parameter` serialization to account for Parameters not storing `@` in their names.
- Removed `MySQLFieldType` enum from `Field.cs`. Now using `MySQLDbType` enum.
- Added `Designer` attribute to several classes to prevent designer view when using VS.Net.
- Fixed Initial catalog typo in `ConnectionString` designer.
- Removed 3 parameter constructor for `MySQLParameter` that conflicted with (name, type, value).
- Changed `MySQLParameter` so `paramName` is now stored without leading `@` (this fixed null inserts when using designer).
- Changed `TypeConverter` for `MySQLParameter` to use the constructor with all properties.

## A.19. Changes in MySQL Connector/Net Version 0.68

- Fixed sequence issue in driver.
- Added `DbParametersEditor` to make parameter editing more like `SqlClient`.
- Fixed `Command` class so that parameters can be edited using the designer
- Update connection string designer to support `Use Compression` flag.
- Fixed string encoding so that European characters will work correctly.
- Creating base classes to aid in building new data providers.
- Added support for UID key in connection string.

- Field, parameter, command now using `DBNull.Value` instead of `null`.
- `CommandBuilder` using `DBNull.Value`.
- `CommandBuilder` now builds insert command correctly when an `auto_insert` field is not present.
- Field now uses `typeof` keyword to return `System.Types` (performance).

## A.20. Changes in MySQL Connector/Net Version 0.65

- `MySQLCommandBuilder` now implemented.
- Transaction support now implemented (not all table types support this).
- `GetSchemaTable` fixed to not use `xsd` (for Mono).
- Driver is now Mono-compatible.
- `TIME` data type now supported.
- More work to improve `Timestamp` data type handling.
- Changed signatures of all classes to match corresponding `SqlClient` classes.

## A.21. Changes in MySQL Connector/Net Version 0.60

- Protocol compression using `SharpZipLib` ([www.icsharpcode.net](http://www.icsharpcode.net)).
- Named pipes on Windows now working properly.
- Work done to improve `Timestamp` data type handling.
- Implemented `IEnumerable` on `DataReader` so `DataGrid` would work.

## A.22. Changes in MySQL Connector/Net Version 0.50

- Speed increased dramatically by removing bugging network sync code.
- Driver no longer buffers rows of data (more ADO.Net compliant).
- Conversion bugs related to `TIMESTAMP` and `DATETIME` fields fixed.



---

## Appendix B. Licenses for Third-Party Components

### Table of Contents

|                                                                 |     |
|-----------------------------------------------------------------|-----|
| B.1. ANTLR 3.3 License .....                                    | 327 |
| B.2. RFC 3174 - US Secure Hash Algorithm 1 (SHA1) License ..... | 327 |
| B.3. <a href="#">zlib</a> License .....                         | 328 |
| B.4. ZLIB.NET License .....                                     | 328 |

### MySQL Connector/Net

- [Section B.1, “ANTLR 3.3 License”](#)
- [Section B.2, “RFC 3174 - US Secure Hash Algorithm 1 \(SHA1\) License”](#)
- [Section B.3, “\[zlib\]\(#\) License”](#)
- [Section B.4, “ZLIB.NET License”](#)

### B.1. ANTLR 3.3 License

The following software may be included in this product:

ANTLR 3.3

ANTLR 3.3 License

Copyright (c) 2010 Terence Parr

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of the author nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

### B.2. RFC 3174 - US Secure Hash Algorithm 1 (SHA1) License

The following software may be included in this product:

RFC 3174 - US Secure Hash Algorithm 1 (SHA1)

RFC 3174 - US Secure Hash Algorithm 1 (SHA1)

Copyright (C) The Internet Society (2001). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it

or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

#### Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

## B.3. zlib License

The following software may be included in this product:

**zlib**

Oracle gratefully acknowledges the contributions of Jean-loup Gailly and Mark Adler in creating the zlib general purpose compression library which is used in this product.

```
zlib.h -- interface of the 'zlib' general purpose compression library
Copyright (C) 1995-2004 Jean-loup Gailly and Mark Adler
```

```
zlib.h -- interface of the 'zlib' general purpose compression library
version 1.2.3, July 18th, 2005
Copyright (C) 1995-2005 Jean-loup Gailly and Mark Adler
```

```
zlib.h -- interface of the 'zlib' general purpose compression library
version 1.2.5, April 19th, 2010
Copyright (C) 1995-2010 Jean-loup Gailly and Mark Adler
```

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software. Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly jloup@gzip.org  
Mark Adler madler@alumni.caltech.edu

## B.4. ZLIB.NET License

The following software may be included in this product:

ZLIB.NET



Copyright (c) 2006-2007, ComponentAce  
<http://www.componentace.com>  
All rights reserved.

Redistribution and use in source and binary forms,  
with or without modification, are permitted provided  
that the following conditions are met:

- \* Redistributions of source code must retain the  
above copyright notice, this list of conditions and  
the following disclaimer.
- \* Redistributions in binary form must reproduce the  
above copyright notice, this list of conditions and  
the following disclaimer in the documentation and/or  
other materials provided with the distribution.
- \* Neither the name of ComponentAce nor the names of its  
contributors may be used to endorse or promote products  
derived from this software without specific prior written  
permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND  
CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,  
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF  
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE  
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR  
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,  
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,  
BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;  
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER  
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,  
STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF  
ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

