CUBRID ADO.NET Data Provider

Tutorial– Part I

December 2011- 1st release

November 2012–Updated for 8.4.3 release

The scope of this tutorial is to give you an overview of the **CUBRID ADO.NET Data Provider**. We will show you how to install the provider library, how to use the provider features and how to develop somesimple ADO.NET applications code.

*Note: This tutorial uses****CUBRID 8.4.3***(<http://www.cubrid.org/?mid=downloads&item=cubrid&os=detect>), ***Windows 7, and Visual Studio 2010 Express Edition***(<http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-csharp-express>)*.*

# Prerequisites

Before you start to developing .NET applications with CUBRID, you will need the CUBRID ADO.NETData provider library (*Cubrid.Data.dll*). You have the options to:

* Download the compiled library, along with other files(for example, the help file) from: <http://www.cubrid.org/?mid=downloads&item=ado_dot_net_driver&os=windows>

or

* Compile it yourself from the source code; checkout the code from the CUBRID SVN repository the folder [http://svn.cubrid.org/cubridapis/adodotnet/branches/RB-8.4.3/](http://svn.cubrid.org/cubridapis/adodotnet/branches/RB-8.4.1/)

or

* Use the CUBRID ADO.NET Data provider **Installer**. This is the easiest and the recommended way to get the driver installed on your machine. The installer can be downloaded from here: <http://www.cubrid.org/?mid=downloads&item=ado_dot_net_driver&os=windows>

You will also need:

* A Windows OS; recommended is Windows Vista or **Windows 7**
* .NET framework 2. or above (recommended **4.x** (<http://www.microsoft.com/net>))

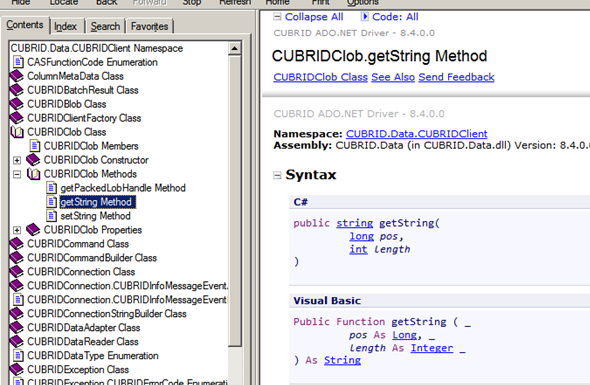
|  |
| --- |
| The CUBRID .NET Provider is 100% full-managed .NET code, and it does not rely on any CUBRID library files. **This means that the usage of the driver does not require any kind of CUBRID installation or files on the local machine.** |

*For compiling the source code and the example in this tutorial, you can use Microsoft Visual Studio Express edition – download it from:* <http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-csharp-express>*.*

# Help file

The CUBRID ADO.NET Provider comes with a full documentation help file, available in the **CHM** format (<http://en.wikipedia.org/wiki/Microsoft_Compiled_HTML_Help>).

Support for CHM files format is available by default in any Windows OS, so all you have to do is to open the help file and search for the information you need:



|  |
| --- |
| Remember, the full source code of the driver is available, under BSD license  (<http://www.cubrid.org/bsd_license>), and all the public methods are documented online.  For example:  ///<summary>  /// Gets the name of the current database after a connection is opened, or the database name specified in the connection string before the connection is opened.  ///</summary>  ///<returns>  /// The name of the current database or the name of the database to be used after a connection is opened.  /// The default value is an empty string.  ///</returns>  [Browsable(true)]  publicoverridestring Database |

# Installation & Configuration

As mentioned before, you can install the driver by using the official **Installer** (<http://www.cubrid.org/?mid=downloads&item=ado_dot_net_driver&os=windows>).

If you choose to install using the default options (x86), the driver will be installed into this folder:…**\Program Files\CUBRID\CUBRID ADO.NET Data Provider 8.4.1**

You can also install it by yourself, just choose the files you need, and copy them to your computer, in a folder location of your choice.

Remember, you can always choose to install the driver into **GAC** (<http://en.wikipedia.org/wiki/Global_Assembly_Cache>).

The best way to install the driver library in GAC is to use the **tblimp** tool (<http://msdn.microsoft.com/en-us/library/tt0cf3sx%28v=vs.80%29.aspx>) and to execute:

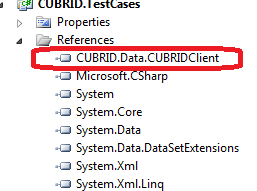
>**tlbimp.exe file.dll /out:Cubrid.Data.dll /keyfile:CUBRID.snk**

Once the driver is installed, you will reference it in the same way as a local copy installation.

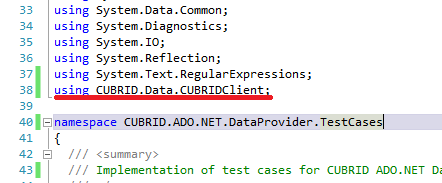
***Be aware****: You need to have Administrator rights during installation if you want to install in into GAC.*

Once the driver is installed, you will need to do two more things:

* Reference the driver library from your projects:



* Import the required namespaces:



# Connection string

The first thing you need to know in order to open a CUBRID connection from a.NET application is how to build the database connection string.

|  |
| --- |
| The format of the CUBRID [ADO.NET](http://ADO.NET)connection string is:    **ConnectionString** = "server=<**server address**>;database=<**database name**>;port=<**port number to use for connection to broker**>;user=<**user name**>;password=<**user password**>;" |

Notes:

- All parameters are mandatory, except for the (CUBRID broker) **port number**

- If you don't specify the broker port number, the default value assumed is **30000**.

## **Examples**

* Connect to a local server, using the default demodb database:

ConnectionString = "server=**127.0.0.1**;database=**demodb**;port=30000;user=public;password=";

* Connect to a remote server, using the default demodb database, as user dba:

ConnectionString = "server=**10.50.88.1**;database=**demodb**;user=public;password=";

* Connect to a remote server, using the default demodb database, as user dba, using password "secret":

ConnectionString = "server=**10.50.99.1**;database=**demodb**;port=30000;user=public;password=secret";

As alternative, you can use the **CubridConnectionStringBuilder**class to build easily a connection string in the correct format. For example:

CUBRIDConnectionStringBuildersb = newCUBRIDConnectionStringBuilder("localhost", 33000, "demodb", "public", "");

using (CUBRIDConnection conn = newCUBRIDConnection(sb.GetConnectionString()))

{

conn.Open();

}

or:

sb = newCUBRIDConnectionStringBuilder();

sb.User = "public";

sb.Database = "demodb";

sb.Port = "33000";

sb.Server = "localhost";

using (CUBRIDConnection conn = newCUBRIDConnection(sb.GetConnectionString()))

{

conn.Open();

}

# Connection options

When connecting to a CUBRID database, you can setup options to be used for the connection lifecycle (unless explicitly changed later):

* The **Auto-Commit**state
* The **Isolation**Level
* The **Connection timeout**
* The **Lock timeout**

For example, by default, the auto-commit is set to **ON**.(see **SELECT\_AUTO\_COMMIT**

at<http://www.cubrid.org/manual/831/en/Parameter%20by%20Broker>), in the CUBRID .NET provider. If you want to set it to **OFF**, you should use the following code:

conn.Open();

conn.SetAutoCommit(false);

|  |
| --- |
| **Note**: The connection parameters are also accessible from the Visual Studio Properties window: |

# A simple Query/Retrieve code

Let’s take a look at a simple code which retrieves value from a CUBRID table, to get a quick insight into how coding will look like. We will assume that the connection is already opened:

String sql = "select \* from nation order by `code` asc";

using (CUBRIDCommand cmd = newCUBRIDCommand(sql, conn))

{

using (DbDataReader reader = cmd.ExecuteReader())

{

reader.Read();

//… (read the values using: reader.Get…() methods)

}

}

Once you have a Connection open and a valid Command which provides a **DbDataReader**(<http://msdn.microsoft.com/en-us/library/system.data.common.dbdatareader.aspx>) object, all you have to do is to use the **Get…()** methods to retrieve any column data. The CUBRID ADO.NET Driver implements all the methods required to read any CUBRID data types (see <http://www.cubrid.org/manual/840/en/Data%20Types>).

For example:

reader.GetString(3)

or

reader.GetDecimal(1)

|  |
| --- |
| The **Get….()** methods will use as an input parameter the 0-based index position of the retrieved column. |

To retrieve specific CUBRID data types, you will need to use **CUBRIDDataReader**extension, instead of the generic **DbDataReader** interface:

using (CUBRIDCommand cmd = newCUBRIDCommand("select \* from t", conn))

{

CUBRIDDataReader reader = (CUBRIDDataReader)cmd.ExecuteReader();

reader.Read();

Debug.Assert(reader.GetDateTime(0) == newDateTime(2008, 10, 31, 10, 20, 30, 040));

Debug.Assert(reader.GetDate(0) == "2008-10-31");

Debug.Assert(reader.GetDate(0, "yy/MM/dd") == "08-10-31");

Debug.Assert(reader.GetTime(0) == "10:20:30");

Debug.Assert(reader.GetTime(0, "HH") == "10");

Debug.Assert(reader.GetTimestamp(0) == "2008-10-31 10:20:30.040");

Debug.Assert(reader.GetTimestamp(0, "yyyy HH") == "2008 10");

}

# Working with parameters

The CUBRID .NET Driver provides support for using **parameters** in SQL commands.

As you probably know, in CUBRID there is no support for named parameters, but only for position-based parameters.

The CUBRID ADO.NET Driver implements support for position-based parameters, with any name you want, as long as **the parameters are prefixed with the character** “**?**”. Remember, the parameter name is not important, as long as you declare and initialize them in the correct order. For example:

using (CUBRIDCommand cmd = newCUBRIDCommand("insert into t values(?, ?)", conn))

{

CUBRIDParameter p1 = newCUBRIDParameter("?p1", CUBRIDDataType.CCI\_U\_TYPE\_INT);

p1.Value = 1;

cmd.Parameters.Add(p1);

CUBRIDParameter p2 = newCUBRIDParameter("?p2", CUBRIDDataType.CCI\_U\_TYPE\_STRING);

p2.Value = "abc";

cmd.Parameters.Add(p2);

cmd.ExecuteNonQuery();

}

In the above example, the most important thing is the order in which the **Add()** methods are called!

# DataTable support

The DataTable is a central object in the ADO.NET library - see <http://msdn.microsoft.com/en-us/library/system.data.datatable.aspx>.

In the CUBRID .NET Provider, we have implemented support for:

* DataTable data populate
* Built-in commands construct: INSERT , UPDATE, DELETE
* Columns metadata/properties
* **DataSet** (<http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx>), **DataView** (<http://msdn.microsoft.com/en-us/library/system.data.dataview.aspx>) inter-connection

Let’s take a look at some coding examples.

## Getting columns properties:

String sql = "select \* from nation";

CUBRIDDataAdapter da = newCUBRIDDataAdapter();

da.SelectCommand = newCUBRIDCommand(sql, conn);

DataTable dt = newDataTable("nation");

da.FillSchema(dt, SchemaType.Source);//To retrieve all the column properties you have to use the FillSchema() method

Debug.Assert(dt.Columns[0].ColumnName == "code");

Debug.Assert(dt.Columns[0].AllowDBNull == false);

Debug.Assert(dt.Columns[0].DefaultValue.ToString() == "");

Debug.Assert(dt.Columns[0].Unique == true);

Debug.Assert(dt.Columns[0].DataType == typeof(System.String));

Debug.Assert(dt.Columns[0].Ordinal == 0);

Debug.Assert(dt.Columns[0].Table == dt);

## Insert values into a table, using the built-in CUBRID support for INSERT statements:

String sql = "select \* from nation order by `code` asc";

using (CUBRIDDataAdapter da = newCUBRIDDataAdapter(sql, conn))

{

using (CUBRIDDataAdapter daCmd = newCUBRIDDataAdapter(sql, conn))

{

CUBRIDCommandBuildercmdBuilder = newCUBRIDCommandBuilder(daCmd);

da.InsertCommand = cmdBuilder.GetInsertCommand();

}

DataTable dt = newDataTable("nation");

da.Fill(dt);

DataRow newRow = dt.NewRow();

newRow["code"] = "ZZZ";

newRow["name"] = "ABCDEF";

newRow["capital"] = "MyXYZ";

newRow["continent"] = "QWERTY";

dt.Rows.Add(newRow);

da.Update(dt);

# Transactions

A **database transaction** groups CUBRID queries into a unit of consistency (for ensuring valid results in multi-user environment) and restore (for making the results of committed transactions permanent and ensuring that the aborted transactions are canceled in the database despite any failure, such as system failure). A transaction is a collection of one or more queries that access and update the database. See more information here: <http://www.cubrid.org/manual/840/en/Database%20Transaction-Overview>.

Transactions are a fundamental concept in databases and our CUBRID .NET data provider implements support for transactions in a similar way with direct-SQL transactions support (<http://www.cubrid.org/manual/840/en/Transaction%20Commit>, <http://www.cubrid.org/manual/840/en/Transaction%20Rollback>).

Here is a code example showing how to use transactions:

conn.BeginTransaction();

string sql = "create table t(idx integer)";

using (CUBRIDCommand command = newCUBRIDCommand(sql, conn))

{

command.ExecuteNonQuery();

}

conn.Rollback();

conn.BeginTransaction();

sql = "create table t(idx integer)";

using (CUBRIDCommand command = newCUBRIDCommand(sql, conn))

{

command.ExecuteNonQuery();

}

conn.Commit();

# Batch commands

When using the ADO.NET Data CUBRID Provider library, you can execute more than one query against the data service in a single **batch**. For more information, see [Batching Operations (ADO.NET Data Services)](http://msdn.microsoft.com/en-us/library/dd744839%28v=vs.90%29.aspx).

For example, in CUBRID you can do:

string[] sql\_arr = newstring[3];

sql\_arr[0] = "insert into t values(1)";

sql\_arr[1] = "insert into t values(2)";

sql\_arr[2] = "insert into t values(3)";

conn.BatchExecute(sql\_arr);

or:

string[] sqls = newstring[3];

sqls[0] = "create table t(id int)";

sqls[1] = "insert into t values(1)";

sqls[2] = "insert into t values(2)";

conn.BatchExecuteNoQuery(sqls);

This concludes the1st part of the CUBRID **ADO.NET Data Provider tutorial**.

…Stay tuned for the 2nd part! ☺ …the 2nd part will focus on specific **CUBRID extensions**…

Remember, please let us know your feedback – we highly appreciate your feedback - and remember to periodically check the CUBRID web site – [www.cubrid.org/tutorials](http://www.cubrid.org/tutorials) - for moreCUBRID tutorials and resources.

**Thank you!**

The CUBRID Apps&Tools Team