Database db = DatabaseFactory.CreateDatabase("DataAccessQuickStart");

Database db = DatabaseFactory.CreateDatabase(); (Si no da ningun connection string le toma el default

**Archivo App.config (se agrega como un archivo XML**

<configuration>

<configSections>

<section name="dataConfiguration"

type="Microsoft.Practices.EnterpriseLibrary.Data.Configuration.DatabaseSettings, Microsoft.Practices.EnterpriseLibrary.Data" />

</configSections>

<connectionStrings>

<add

name="DataAccessQuickStart"

providerName="System.Data.SqlClient"

connectionString="server=(local)\SQLEXPRESS;database=EntLibQuickStarts;Integrated Security=true" />

</connectionStrings>

<dataConfiguration defaultDatabase="DataAccessQuickStart"/>

</configuration>

Agregue en referencias

* Microsoft.Practices.EnterpriseLibrary.Common.dll
* Microsoft.Practices.EnterpriseLibrary.Data.dll

Para Oracle y SQL Server

// DataReader that will hold the returned results

public string **GetCustomerList**()

{

**Database db = DatabaseFactory.CreateDatabase();**

DbCommand dbCommand = **db.GetSqlStringCommand(**"Select Name From Customers"**);**

**StringBuilder** readerData = new **StringBuilder**();

using (IDataReader dataReader = db.ExecuteReader(dbCommand))

{

while (dataReader.Read())

{

readerData.Append(dataReader["Name"]);

readerData.Append(**Environment.NewLine**);

}

}

return **readerData.ToString**();

}

private void retrieveUsingReaderButton\_Click(object sender, EventArgs e)

{

try

{

Cursor = Cursors.WaitCursor;

this.resultsTextBox.Text = **GetCustomerList**();

this.resultsDataGrid.Hide();

this.resultsTextBox.Show();

}

catch (Exception ex)

{

DisplayErrors(ex);

}

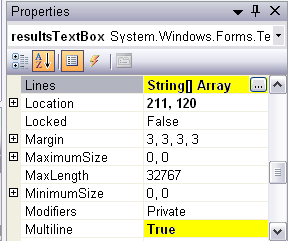
finally

{

Cursor = Cursors.Arrow;

}

}



using Microsoft.Practices.EnterpriseLibrary.Data;

using Microsoft.Practices.EnterpriseLibrary.Data.Sql;

using System.Data.Common;

// Return a dataset with the list of product in a category, using a store procedure

public DataSet GetProductsInCategory(int Category)

{

**Database db = DatabaseFactory.CreateDatabase();**

DbCommand Comm1 = db.GetStoredProcCommand("GetProductsByCategory");

db.AddInParameter(Comm1, "CategoryID", DbType.Int32, Category);

return db.ExecuteDataSet(Comm1);

}

private void retrieveUsingDatasetButton\_Click(object sender, EventArgs e)

{

try

{

Cursor = Cursors.WaitCursor;

DataSet customerDataSet = salesData.GetProductsInCategory(2);

// Bind the DataSet to the DataGrid for display.

this.resultsDataGrid.SetDataBinding(customerDataSet, "Table");

}

catch (Exception ex)

{

DisplayErrors(ex);

}

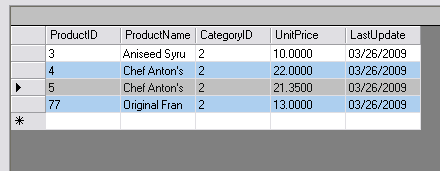
finally

{

Cursor = Cursors.Arrow;

}

}

public int UpdateProducts()

{

Database db = DatabaseFactory.CreateDatabase();

DataSet productsDataSet = new DataSet();

string sqlCommand = "Select ProductID, ProductName, CategoryID, UnitPrice, LastUpdate " +

"From Products";

DbCommand dbCommand = db.GetSqlStringCommand(sqlCommand);

// Retrieve the initial data

db.LoadDataSet(dbCommand, productsDataSet, "Products");

// Get the table that will be modified

DataTable table = productsDataSet.Tables["Products"];

// Add a new product to existing DataSet

DataRow addedRow = table.Rows.Add(new object[] {DBNull.Value, "New product", 11, 25});

// Modify an existing product

table.Rows[0]["ProductName"] = "Modified product";

// Establish our Insert, Delete, and Update commands

DbCommand insertCommand = db.**GetStoredProcCommand**("AddProduct");

db.AddInParameter(insertCommand, "ProductName", DbType.String, "ProductName", DataRowVersion.Current);

AddInParameter(insertCommand, "CategoryID", DbType.Int32, "CategoryID", DataRowVersion.Current);

db.AddInParameter(insertCommand, "UnitPrice", DbType.Currency, "UnitPrice", DataRowVersion.Current);

DbCommand deleteCommand = db.**GetStoredProcCommand**("DeleteProduct");

AddInParameter(deleteCommand, "ProductID", DbType.Int32, "ProductID", DataRowVersion.Current);

DbCommand updateCommand = db. **GetStoredProcCommand**("UpdateProduct");

db.AddInParameter(updateCommand, "ProductID", DbType.Int32, "ProductID", DataRowVersion.Current);

db.AddInParameter(updateCommand, "ProductName", DbType.String, "ProductName", DataRowVersion.Current);

db.AddInParameter(updateCommand, "LastUpdate", DbType.DateTime, "LastUpdate", DataRowVersion.Current);

// Submit the DataSet, capturing the number of rows that were affected

int rowsAffected = db.**UpdateDataSet**(productsDataSet, "Products", insertCommand, updateCommand, deleteCommand, UpdateBehavior.Standard);

return rowsAffected;

}

public string GetProductDetails(int productID)

{

Database db = DatabaseFactory.CreateDatabase();

string sqlCommand = "GetProductDetails";

DbCommand dbCommand = db.**GetStoredProcCommand**(sqlCommand);

// Input parameters can specify the input value

db.AddInParameter(dbCommand, "ProductID", DbType.Int32, productID);

// Output parameters specify the size of the return data

db.AddOutParameter(dbCommand, "ProductName", DbType.String, 50);

db.AddOutParameter(dbCommand, "UnitPrice", DbType.Currency, 8);

db.ExecuteNonQuery(dbCommand);

// Row of data is captured via output parameters

string results = string.Format(CultureInfo.CurrentCulture, "{0}, {1}, {2:C} ",

db.GetParameterValue(dbCommand, "ProductID"),

db.GetParameterValue(dbCommand, "ProductName"),

db.GetParameterValue(dbCommand, "UnitPrice"));

return results;

}

public string GetProductName(int productID)

{

Database db = DatabaseFactory.CreateDatabase();

// Passing the productID value to the GetStoredProcCommand

// results in parameter discovery being used to correctly establish the parameter

// information for the productID. Subsequent calls to this method will

// cause the block to retrieve the parameter information from the

// cache, and not require rediscovery.

DbCommand dbCommand = db.**GetStoredProcCommand**("GetProductName", productID);

// Retrieve ProdcutName. ExecuteScalar returns an object, so we cast to the correct type (string).

string productName = (string) db.**ExecuteScalar**(dbCommand);

return productName;

}

/// Retrieves a list of products.

/// <returns>A list of products as an XML string.</returns>

/// <remarks>Demonstrates retrieving multiple rows of data as XML. This

/// method is not portable across database providers, but is specific to the SqlDatabase.

public string GetProductList()

{

SqlDatabase dbSQL = DatabaseFactory.CreateDatabase() as SqlDatabase;

// Use "FOR XML AUTO" to have SQL return XML data

string sqlCommand = "Select ProductID, ProductName, CategoryID, UnitPrice, LastUpdate " +

"From Products FOR XML AUTO";

DbCommand dbCommand = dbSQL.**GetSqlStringCommand**(sqlCommand);

XmlReader productsReader = null;

StringBuilder productList = new StringBuilder();

try

{

productsReader = dbSQL.**ExecuteXmlReader**(dbCommand);

// Iterate through the XmlReader and put the data into our results.

while (!productsReader.EOF)

{

if (products**Reader**.IsStartElement())

{

productList.Append(productsReader.ReadOuterXml());

productList.Append(Environment.NewLine);

}

}

}

finally

{

// Close the Reader.

if (productsReader != null) productsReader.Close();

// Close the connection. The connection is not closed when the XmlReader is closed.

if (dbCommand.Connection != null)

{

dbCommand.Connection.Close();

}

}

return productList.ToString();

}

/// Transfers an amount between two accounts.

public bool Transfer(int transactionAmount, int sourceAccount, int destinationAccount)

{

bool result = false;

Database db = DatabaseFactory.CreateDatabase();

// Two operations, one to credit an account, and one to debit another account.

DbCommand creditCommand = db.**GetStoredProcCommand**("CreditAccount");

db.AddInParameter(creditCommand, "AccountID", DbType.Int32, sourceAccount);

db.AddInParameter(creditCommand, "Amount", DbType.Int32, transactionAmount);

DbCommand debitCommand = db. **GetStoredProcCommand**("DebitAccount");

db.AddInParameter(debitCommand, "AccountID", DbType.Int32, destinationAccount);

db.AddInParameter(debitCommand, "Amount", DbType.Int32, transactionAmount);

using (DbConnection connection = db.CreateConnection())

{

connection.Open();

DbTransaction transaction = connection.BeginTransaction();

try

{

// Credit the first account

db.**ExecuteNonQuery**(creditCommand, transaction);

// Debit the second account

db.**ExecuteNonQuery**(debitCommand, transaction);

// Commit the transaction

transaction.Commit();

result = true;

}

catch

{

transaction.Rollback();

}

connection.Close();

return result;

}

}

**PASANDO PARAMETROS**

db = DatabaseFactory.CreateDatabase(“base”)

dbCommand = db.**GetSqlStringCommand**(“select \* from tabla where dtInter > @begin “)

db.AddInParameter(dbCommand, "@begin", DbType.DateTime, dtLastIntervalBegin)

dsCalls = db.**ExecuteDataSet**(dbCommand)

If Not dsCalls Is Nothing Then CSVFileWrite(**dsCalls.Tables(0))**

CREATE PROCEDURE dbo.GetProdDetails

(

@ProductID int,

@ProductName nvarchar(40) output,

@UnitPrice money output

)

AS

SET NOCOUNT ON

Select @ProductName = ProductName,

@UnitPrice = UnitPrice

From Products

Where ProductID = @ProductID

private void btnRetrieveSingleRow\_Click(object sender, System.EventArgs e)

{

Database db = DatabaseFactory.CreateDatabase("Northwind");

DBCommandWrapper dbc = db.GetStoredProcCommandWrapper("GetProdDetails");

dbc.AddInParameter("@ProductID", DbType.Int32, 1);

dbc.AddOutParameter("@ProductName", DbType.String, 50);

dbc.AddOutParameter("@UnitPrice", DbType.Currency, 8);

db.ExecuteNonQuery(dbc);

lblResult.Text = "Product Name:"+ dbc.GetParameterValue("@ProductName") + " ";

lblResult.Text += "Unit Price:" + dbc.GetParameterValue("@UnitPrice");

}

DbCommand comm1 = db.GetStoreProcCommand("GetProdDetails");

db.ExecuteNonQuery(comm1);