Hands-On Lab
Lab Manual

DEV-HOL17: Data Access in Distributed Applications with the Data Access Application Block

Please do not remove this manual from the lab
The lab manual will be available from CommNet
Contents

ACCESSING DATA WITH THE DATA ACCESS APPLICATION BLOCK ..............................................................1
Lab Objectives..................................................................................................................................................1
Exercise 1: Retrieving a SqlDataReader.........................................................................................................2
  Task 1 – Adding a reference to the Data Access Application Block...........................................................2
  Task 2 – Adding code that retrieves a SqlDataReader .............................................................................3
  Task 3 – Testing your code.........................................................................................................................3
Exercise 2: Retrieving a DataSet ....................................................................................................................3
  Task 1 – Adding code that retrieves a DataSet ..........................................................................................4
  Task 2 – Testing your code.........................................................................................................................4
Exercise 3: Retrieving a Single Row Using Output Parameters .................................................................4
  Task 1 – Adding code that executes a stored procedure and retrieves output parameters.....................5
  Task 2 – Testing your code.........................................................................................................................6
Exercise 4: Retrieving a Single Item ...............................................................................................................6
  Task 1 – Adding code that executes a Transact-SQL statement that returns a single value.....................6
  Task 2 – Testing your code.........................................................................................................................7
Exercise 5: Performing a Transactional Update ............................................................................................7
  Task 1 – Adding code that performs a transactional update ...................................................................7
  Task 2 – Testing your code.........................................................................................................................8
Exercise 6: Retrieving XML Data ..................................................................................................................8
  Task 1 – Adding code that retrieves an XmlReader ..................................................................................8
  Task 2 – Testing your code.........................................................................................................................9
Lab Summary..................................................................................................................................................9
Accessing Data with the Data Access Application Block

In this lab, you learn how to use the Data Access Application Block to simplify data access in your .NET applications. The Microsoft® Data Access Application Block for .NET consists of a single .NET-based assembly, which contains all of the functionality necessary to perform the most common data access tasks against a Microsoft SQL Server™ 2000 database.

Specifically, the Data Access Application Block helps you to:

- Call stored procedures or SQL text commands.
- Specify parameter details.
- Return SqlDataReader, DataSet, XmlReader objects or single values.

The Data Access Application Block is designed to encapsulate the Microsoft recommended best practices for data access in .NET applications, as described in the Microsoft Data Access Architecture Guide. You can read the guide at http://msdn.microsoft.com/library/en-us/dnbda/html/daag.asp.

By using the Data Access Application Block in your applications you can:

- Minimize the data access code you need to write, often to a single line.
- Ensure that your data access logic is implemented in an efficient and effective manner.

You can learn more about the Data Access Application Block from the documentation provided in the Data Access folder in the Microsoft Application Blocks for .NET program group on the Start menu.

Lab Objectives

Estimated time to complete this lab: 30 minutes. Each exercise should be completed before proceeding to the next exercise.

After completing this lab, you will be able to:

- Add a reference to the Data Access Application Block.
- Retrieve a SqlDataReader with the Data Access Application Block.
- Retrieve a DataSet with the Data Access Application Block.
- Retrieve a single row with the Data Access Application Block.
- Retrieve a single item with the Data Access Application Block.
- Perform a transactional update with the Data Access Application Block.
- Retrieve an XML fragment with the Data Access Application Block.

Exercise 1: Retrieving a SqlDataReader.

In this exercise, you add a reference to the Data Access Application Block and use the ExecuteReader method of the SqlHelper class to retrieve a SqlDataReader filled with data from a Microsoft SQL Server 2000 database.

Exercise 2: Retrieving a DataSet.

In this exercise, you use the ExecuteDataset method of the SqlHelper class
to retrieve data in a **DataSet**.

**Exercise 3: Retrieving a Single Row using Output Parameters.**
In this exercise, you use the `ExecuteNonQuery` method of the `SqlHelper` class to retrieve data from a stored procedure that returns output parameters. To manage the stored procedure parameters, you use the `GetSpParameterSet` method of the `SqlHelperParameterCache` class.

**Exercise 4: Retrieving a Single Item.**
In this exercise, you use the `ExecuteScalar` method of the `SqlHelper` class to execute a Transact-SQL statement and retrieve a single value.

**Exercise 5: Performing a Transactional Update.**
In this exercise, you use the `ExecuteNonQuery` method of the `SqlHelper` class to execute transactional logic that requires two stored procedures to be called.

**Exercise 6: Retrieving XML Data.**
In this exercise, you use the `ExecuteXmlReader` method of the `SqlHelper` class to execute a command that returns an XML fragment.

**Exercise 1: Retrieving a SqlDataReader**
In this exercise, you add a reference to the Data Access Application Block and use the `ExecuteReader` method of the `SqlHelper` class to retrieve a `SqlDataReader` filled with data from a Microsoft SQL Server 2000 database.

There are nine overloads of the `ExecuteReader` method. In this exercise, you use one that accepts a connection string, allowing you to leave connection management issues to the Data Access Application Block.

In this exercise, you:
- Reset the database and client application.
- Use the `ExecuteReader` method of the `SqlHelper` class to retrieve a `SqlDataReader` by using a stored procedure called `GetProducts`. You use an overload of the `ExecuteReader` method that requires you to pass a connection string, the name of the stored procedure you want to call, and its parameters (if any).
- Test your code.

**Task 1 – Adding a reference to the Data Access Application Block**
- Start Microsoft Visual Studio .NET and open either the Visual Basic or C# version of the `DAABClient.sln` solution.
  - For the Visual Basic version, open `C:\Microsoft Hands-On-Lab\DEV-H17\Source\Exercises\vb\DAABClient.sln`.
  - For the C# version, open `C:\Microsoft Hands-On-Lab\DEV-H17\Source\Exercises\cs\DAABClient.sln`.
• Make sure that the Solution Explorer window is visible by choosing Solution Explorer on the View menu.

• In the Solution Explorer, right-click the DAABClient project, and then click Add Reference.

• In the Add Reference dialog box, click Browse and then navigate to C:\Microsoft Hands-On-Lab\DEV-HOL17\Source\Exercises\bin.

• Click Microsoft.ApplicationBlocks.Data.dll, click Open, and then click OK.

• In the Solution Explorer, right-click MainForm and then click View Code.

• View the code behind the MainForm form, and note the existing Imports (Visual Basic) or using (C#) statements at the beginning of the form. Add a statement to import or use the Microsoft.ApplicationBlocks.Data namespace.

  ' Visual Basic
  Imports Microsoft.ApplicationBlocks.Data

  // C#
  using Microsoft.ApplicationBlocks.Data;

Task 2 – Adding code that retrieves a SqlDataReader

• In MainForm view the code in GetProducts_Click. Note that some exception handling code has already been added for you.

• Under the comments in the Try block, add the following code to retrieve a SqlDataReader and write data from it to the ProductsList list box (this code uses a constant named connectionString), which has been declared for you):

  ' Visual Basic
  Dim productsReader As SqlDataReader = _
  SqlHelper.ExecuteReader(connectionString, "GetProducts")
  Do While productsReader.Read()
    ProductsList.Items.Add(productsReader.GetString(1))
  Loop
  productsReader.Close()

  // C#
  SqlDataReader productsReader =
  SqlHelper.ExecuteReader(connectionString,"GetProducts");
  while(productsReader.Read())
  {
    ProductsList.Items.Add(productsReader.GetString(1));
  }
  productsReader.Close();

Task 3 – Testing your code

• Select the Debug | Start menu command.

• Click the Get Products button. The list box beneath it should be filled with a list of product names.

• Close the MainForm window to stop debugging.

Exercise 2: Retrieving a DataSet

In this exercise, you use the ExecuteDataset method of the SqlHelper class to retrieve data in a DataSet. Of the nine ExecuteDataset overloads, you use one that accepts a SqlConnection object, giving you control over connection management and lifetime.
In this exercise, you:

- Write code that calls the `ExecuteDataset` method of the `SqlHelper` class. You use an overload of the `ExecuteDataset` method that requires you to pass a `SqlConnection` object, the name of the stored procedure you want to call, and its parameters (if any).
- Test your code.

Task 1 – Adding code that retrieves a DataSet

- In `MainForm` view the code in `GetProductsByCategory_Click`. Note that some exception handling code has already been added for you.
- Above the `Try` block, add the following code to declare and open a `SqlConnection` object:
  ```
  Visual Basic
  Dim conn As New SqlConnection(connectionString)
  conn.Open()
  // C#
  SqlConnection conn = new SqlConnection(connectionString);
  conn.Open();
  ```
- In the `Try` block, add the following code to retrieve a `DataSet` and bind it to a `DataGrid`:
  ```
  Visual Basic
  Dim productsDataSet As DataSet = _
  SqlHelper.ExecuteDataset(conn, "GetProductsByCategory", _
  CInt(CategoryID.Text))
  ProductByCategoryGrid.DataSource = productsDataSet
  ProductByCategoryGrid.DataMember = "Table"
  // C#
  DataSet productsDataSet =
  SqlHelper.ExecuteDataset(conn,"GetProductsByCategory",
  int.Parse(CategoryID.Text));
  ProductByCategoryGrid.DataSource = productsDataSet;
  ProductByCategoryGrid.DataMember = "Table";
  ```
- In the `Finally` block, add the following code to close the connection:
  ```
  Visual Basic
  conn.Close()
  // C#
  conn.Close();
  ```

Task 2 – Testing your code

- Select the `Debug | Start` menu command.
- Click the `Get Products By Category` button, ensuring that the text box next to it contains the number 1. The `DataGrid` beneath it should be filled with products in category 1.
- Close the `MainForm` window to stop debugging.

Exercise 3: Retrieving a Single Row Using Output Parameters

In this exercise, you use the `ExecuteNonQuery` method of the `SqlHelper` class to retrieve data from a
stored procedure that returns output parameters. To manage the stored procedure parameters, you use the GetSpParameterSet method of the SqlHelperParameterCache class.

Of the nine ExecuteDataset overloads, you use one that accepts an array of SqlParameter objects, allowing you to retrieve output parameter values. In addition, you use the GetSpParameterSet method of the SqlParameterCache class. GetSpParameterSet derives and caches an array of parameters for a specified stored procedure, automatically setting the appropriate parameter name, type, direction, and so on for you.

In this exercise, you:

- Write code that calls the ExecuteNonQuery method of the SqlHelper class. You use an overload of the ExecuteNonQuery method that requires you to pass a connection string, the command you want to execute, the command type, and its parameters.
- Test your code.

Task 1 – Adding code that executes a stored procedure and retrieves output parameters

- In MainForm view the code in GetProductDetails_Click. Note that some exception handling code has already been added for you.
- Add the following code to the Try block to create an array of parameters for the GetProductDetails stored procedure, and set the value of the input parameter:

  // Visual Basic
  Dim GetProductDetailsParams() As SqlParameter = _
  SqlHelperParameterCache.GetSpParameterSet(connectionString, _
  "GetProductDetails")
  GetProductDetailsParams(0).Value = CInt(ProductID.Text)

  // C#
  SqlParameter[] GetProductDetailsParams =
  SqlHelperParameterCache.GetSpParameterSet(connectionString, _
  "GetProductDetails");
  GetProductDetailsParams[0].Value = int.Parse(ProductID.Text);

- After the existing code in the Try block, add the following to execute the GetProductDetails stored procedure:

  // Visual Basic
  SqlHelper.ExecuteNonQuery(connectionString, _
  CommandType.StoredProcedure, _
  "GetProductDetails", GetProductDetailsParams)

  // C#
  SqlHelper.ExecuteNonQuery(connectionString, _
  CommandType.StoredProcedure, _
  "GetProductDetails", GetProductDetailsParams);

- After the existing code in the Try block, add the following to retrieve the output parameter values:

  // Visual Basic
  ProductNameLabel.Text = GetProductDetailsParams(1).Value.ToString()
  Dim unitPrice As Decimal = CDec(GetProductDetailsParams(2).Value)
  UnitPriceLabel.Text = unitPrice.ToString("c")
CategoryIDLabel.Text = GetProductDetailsParams(3).Value.ToString()

// C#
decimal unitPrice = (decimal)GetProductDetailsParams[2].Value;
UnitPriceLabel.Text = unitPrice.ToString("c");

Task 2 – Testing your code

- Select the Debug | Start menu command.
- Click the Get Product Details button, ensuring that the text box next to it contains the number 1. The labels beneath it should display the name, price, and category for product 1.
- Close the MainForm window to stop debugging.

Exercise 4: Retrieving a Single Item

In this exercise, you use the ExecuteScalar method of the SqlHelper class to execute a Transact-SQL statement and retrieve a single value.

Of the nine ExecuteDataset overloads, you use one that accepts a CommandType enumeration, allowing you to execute Transact-SQL or a stored procedure as required, giving you control over connection management and lifetime.

In this exercise, you:
- Write code that calls the ExecuteScalar method of the SqlHelper class. You use an overload of the ExecuteScalar method that requires you to pass a connection string, the command you want to execute, and the command type.
- Test your code.

Task 1 – Adding code that executes a Transact-SQL statement that returns a single value

- In MainForm view the code in GetAveragePrice_Click. Note that some exception handling code has already been added for you.
- In the Try block, add the following code to retrieve the average price and display the value returned formatted as currency:

  ' Visual Basic
  Dim averagePrice As Decimal = _
  CDec(SqlHelper.ExecuteScalar(connectionString, _
  CommandType.Text, _
  "SELECT AVG(UnitPrice) AveragePrice FROM Products"))
  AvgPrice.Text = averagePrice.ToString("c")

  // C#
  decimal averagePrice = _
  (decimal)SqlHelper.ExecuteScalar(connectionString, _
  CommandType.Text, _
  "SELECT AVG(UnitPrice) AveragePrice FROM Products");
  AvgPrice.Text = averagePrice.ToString("c");
Task 2 – Testing your code

- Select the Debug | Start menu command.
- Click the Get Average Price button. The text box beside it should display the average price of a product.
- Close the MainForm window to stop debugging.

Exercise 5: Performing a Transactional Update

In this exercise you use the ExecuteNonQuery method of the SqlHelper class to execute transactional logic that requires two stored procedures to be called.

Of the nine ExecuteDataset overloads, you use one that accepts a SqlTransaction object, allowing you to execute multiple commands in the same transaction.

In this exercise, you:

- Write code that calls the ExecuteNonQuery method of the SqlHelper class. You use an overload of the ExecuteScalar method that requires you to pass a SqlTransaction object, the name of the stored procedure you want to execute, and the stored procedure parameters (if any).
- Test your code.

Task 1 – Adding code that performs a transactional update

- In MainForm view the code in InsertCategoryAndProduct_Click. Note that some exception handling code has already been added for you.
- Before the Try block, add the following code to open a connection and start a transaction:

  ```visual-basic
  Dim conn As New SqlConnection(connectionString)
  conn.Open()
  Dim tran As SqlTransaction = conn.BeginTransaction()
  ```

  ```csharp
  SqlConnection conn = new SqlConnection(connectionString);
  conn.Open();
  SqlTransaction tran = conn.BeginTransaction();
  ```

- In the Try block, add the following code to execute the AddCategory and AddProduct stored procedures in a transaction:

  ```visual-basic
  SqlHelper.ExecuteNonQuery(tran, "AddCategory", _
  CInt(CategoryToAdd.Text), CategoryNameToAdd.Text)
  SqlHelper.ExecuteNonQuery(tran, "AddProduct", _
  CInt(ProductIDToAdd.Text), ProductNameToAdd.Text, _
  CDec(UnitPriceToAdd.Text), CInt(CategoryToAdd.Text))
  tran.Commit()
  MessageBox.Show("Transaction Completed Successfully")
  ```

  ```csharp
  SqlHelper.ExecuteNonQuery(tran,"AddCategory",
  int.Parse(CategoryToAdd.Text), CategoryNameToAdd.Text);
  SqlHelper.ExecuteNonQuery(tran, "AddProduct",
  int.Parse(ProductIDToAdd.Text), ProductNameToAdd.Text,
  decimal.Parse(UnitPriceToAdd.Text) ,int.Parse(CategoryToAdd.Text));
  ```
tran.Commit();
MessageBox.Show("Transaction Completed Successfully");

- In the Catch block (before the MessageBox.Show statement), add the following code to roll back the transaction if an exception occurred:
  
  ```csharp
  tran.Rollback();
  ```

- Add the following code in the Finally block to close the connection:
  
  ```csharp
  conn.Close();
  ```

**Task 2 – Testing your code**

- Select the Debug | Start menu command.
- Click the Add Category & Product button, ensuring that the text boxes next to it contain the values 101, Scottish Delicacies, 999, Traditional Haggis, and 5.99. The first time the button is clicked, a message box should confirm that the transaction completed successfully. Subsequent clicks with the same values should result in an exception, and the transaction will be rolled back.
- Close the MainForm window to stop debugging.

**Exercise 6: Retrieving XML Data**

In this exercise you will use the `ExecuteXmlReader` method of the `SqlHelper` class to execute a command that returns an XML fragment.

In this exercise, you:

- Write code that calls the `ExecuteXmlReader` method of the `SqlHelper` class and retrieves an XML fragment in an `XmlReader` object.
- Test your code.

**Task 1 – Adding code that retrieves an XmlReader**

- In MainForm view the code in `GetCategoryXML_Click`. Note that some exception handling code has already been added for you.
- Before the Try block, add the following code to create a connection and open it:
  
  ```csharp
  SqlConnection conn = new SqlConnection(connectionString);
  conn.Open();
  ```
• In the **Try** block, add the following code to call the **GetCategoryXML** stored procedure, return an **XmlReader**, and then move to the contents of the **XmlReader** and display its XML:

```vba
' Visual Basic
Dim xmlCategoryData As XmlReader
xmlCategoryData = SqlHelper.ExecuteXmlReader(conn, 
    "GetCategoryXML", CInt(CategoryToGetAsXml.Text))
xmlCategoryData.MoveToContent()
CategoryXML.Text = xmlCategoryData.ReadOuterXml()
xmlCategoryData.Close()
```

```csharp
// C#
XmlReader xmlCategoryData;
xmlCategoryData = SqlHelper.ExecuteXmlReader(conn,
    "GetCategoryXML", int.Parse(CategoryToGetAsXml.Text));
xmlCategoryData.MoveToContent();
CategoryXML.Text = xmlCategoryData.ReadOuterXml();
xmlCategoryData.Close();
```

• In the **Finally** block, add the following code to close the connection:

```vba
' Visual Basic
conn.Close()
```

```csharp
// C#
conn.Close();
```

**Task 2 – Testing your code**

• Select the **Debug | Start** menu command.

• Click the **Get Category XML** button, ensuring that the text box next to it contains the value `1`. The XML representation of the category should be displayed in the text box beneath the button.

• Close the **MainForm** window to stop debugging.

**Lab Summary**

In this lab you learned how to work with the Data Access Application Block. You performed the following exercises:

**Exercise 1: Retrieving a SqlDataReader.**

In this exercise, you add a reference to the Data Access Application Block and use the **ExecuteReader** method of the **SqlHelper** class to retrieve a **SqlDataReader** filled with data from a Microsoft SQL Server 2000 database.

**Exercise 2: Retrieving a DataSet.**

In this exercise, you use the **ExecuteDataset** method of the **SqlHelper** class to retrieve data in a **DataSet**.

**Exercise 3: Retrieving a Single Row using Output Parameters.**

In this exercise, you use the **ExecuteNonQuery** method of the **SqlHelper** class to retrieve data from a stored procedure that returns output parameters. To manage the stored procedure parameters, you use the **GetSpParameterSet** method of the **SqlHelperParameterCache** class.
Exercise 4: Retrieving a Single Item.
In this exercise, you use the `ExecuteScalar` method of the `SqlHelper` class to execute a Transact-SQL statement and retrieve a single value.

Exercise 5: Performing a Transactional Update.
In this exercise, you use the `ExecuteNonQuery` method of the `SqlHelper` class to execute transactional logic that requires two stored procedures to be called.

Exercise 6: Retrieving XML Data.
In this exercise, you use the `ExecuteXmlReader` method of the `SqlHelper` class to execute a command that returns an XML fragment.