An Introduction to ADO.Net
Contents

• What is ADO.Net?
• What happened to ADO?
• The ADO.Net object structure
• Connecting
• Commanding
• Readers and DataSets
What is ADO.Net?

- The data access classes for the .Net framework
- Designed for highly efficient data access
- Support for XML and disconnected record sets
And the .Net framework?

- A standard cross language interface
- Encapsulation of services, classes and data types
- Uses XML for data representation
What happened to ADO?

- ADO still exists.
- ADO is tightly coupled to client server architectures.
- Needs COM marshalling to pass data between tiers.
- Connections and locks are typically persisted.
# ADO / ADO.Net Comparisons

<table>
<thead>
<tr>
<th>Feature</th>
<th>ADO</th>
<th>ADO.Net</th>
</tr>
</thead>
<tbody>
<tr>
<td>In memory data storage</td>
<td>Recordset object Mimics single table</td>
<td>Dataset object Contains DataTables</td>
</tr>
<tr>
<td>Data Reads</td>
<td>Sequential</td>
<td>Sequential or non-sequential</td>
</tr>
<tr>
<td>Data Sources</td>
<td>OLE/DB via the Connection object</td>
<td>Managed provider calls the SQL APIs</td>
</tr>
<tr>
<td>Feature</td>
<td>ADO</td>
<td>ADO.Net</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>Disconnected data</td>
<td>Limited support, suitable for R/O</td>
<td>Strong support, with updating</td>
</tr>
<tr>
<td>Passing datasets</td>
<td>COM marshalling</td>
<td>DataSet support for XML passing</td>
</tr>
<tr>
<td>Scalability</td>
<td>Limited</td>
<td>Disconnected access provides scalability</td>
</tr>
</tbody>
</table>
The ADO.NET Object Model

SQL Server .NET Data Provider

SqlConnection

SqlConnection

OLE DB .NET Data Provider

OleDbConnection

OleDbConnection

SqlDataAdapter

OleDbDataAdapter

SqlDataAdapter

Sql Server 7.0 (and later)

OLEDB sources (SQL Server 6.5)
What is a Dataset?
Accessing Data with ADO.NET

1. Client makes request
2. Create the SqlConnection and SqlDataAdapter objects
3. Fill the DataSet from the DataAdapter and close the connection
4. Return the DataSet to the Client
5. Client manipulates the data
6. Update the DataSet
7. Use the SqlDataAdapter to open the SqlConnection, update the database, and close the connection
Using Server Explorer to Generate a Connection

- Create a new data connection by dragging a Table from Server Explorer.
- Create a new data connection using the Data Links dialog box.
Namespaces

- System.Data.SqlTypes
- System.XML & System.XML.Schema
Using Namespaces

• VB.Net
  Imports System.Data
  Imports System.Data.SqlClient
  Dim sqlAdp as SqlDataAdapter

• C#
  using System.Data;
  using System.Data.SqlClient;
  SqlDataAdapter sqlAdp = new SqlDataAdapter();
SQL Namespace Objects

• using System.Data.SqlClient;
• SqlConnection
• SqlCommand
• SqlDataReader
• SqlDataAdapter
• SqlParameter
• SqlParameterCollection
• SqlError
• SqlErrorCollection
• SqlException
• SqlTransaction
• SqlDbType
# ADO.NET Core Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connection</strong></td>
<td>Establishes a connection to a specific data source. (Base class: DbConnection)</td>
</tr>
<tr>
<td><strong>Command</strong></td>
<td>Executes a command against a data source. Exposes <strong>Parameters</strong> and can execute within the scope of a <strong>Transaction</strong> from a <strong>Connection</strong>. (The base class: DbCommand)</td>
</tr>
<tr>
<td><strong>DataReader</strong></td>
<td>Reads a forward-only, read-only stream of data from a data source. (Base class: DbDataReader)</td>
</tr>
<tr>
<td><strong>DataAdapter</strong></td>
<td>Populates a <strong>DataSet</strong> and resolves updates with the data source. (Base class: DbDataAdapter)</td>
</tr>
<tr>
<td><strong>DataTable</strong></td>
<td>Has a collection of DataRows and DataColumns representing table data, used in disconnected model</td>
</tr>
<tr>
<td><strong>DataSet</strong></td>
<td>Represents a cache of data. Consists of a set of DataTables and relations among them</td>
</tr>
</tbody>
</table>
Connected Data Access Model

1. makes a connection
2. passes some query and read result from DB
3. passes another query and read result from DB
4. disconnects from the DB system
Disconnected Data Access Model

1. makes a connection
2. passes some query and get result back
3. disconnects from the DB system
4. makes connection again
5. passes another query and get result back
6. disconnects from the DB system
• using System.Data.SqlClient;

    string sConnectionString = 
        "Initial Catalog=Northwind; 
        Data Source=localhost; 
        Integrated Security=SSPI;";

    SqlDataAdapter sqlAdp = new SqlDataAdapter(sConnectionString);

    sqlAdp.Close();
    sqlAdp.Dispose();
Connection Pooling

- ADO.Net pools connections. When you close a connection it is released back into a pool.
  - SqlConnection conn = new SqlConnection();
    conn.ConnectionString =
    "Integrated Security=SSPI;Initial Catalog=northwind";
    conn.Open(); // Pool A is created.
  - SqlConnection conn = new SqlConnection();
    conn.ConnectionString =
    "Integrated Security=SSPI;Initial Catalog=pubs";
    conn.Open(); // Pool B is created because the connection strings differ.
  - SqlConnection conn = new SqlConnection();
    conn.ConnectionString =
    "Integrated Security=SSPI;Initial Catalog=northwind";
    conn.Open(); // The connection string matches pool A.
Steps of Data Access: Disconnected Environment

- Defining the connection string
- Defining the connection
- Defining the command
- Defining the data adapter
- Creating a new DataSet object
- SELECT -> fill the dataset object with the result of the query through the data adapter
- Reading the records from the DataTables in the datasets using the DataRow and DataColumn objects
- UPDATE, INSERT or DELETE -> update the database through the data adapter
using System;
using System.Data;
using System.Data.SqlClient;

namespace SampleClass
{
    class Program
    {
        static void Main(string[] args)
        {
            string connStr = Properties.Settings.Default.connStr;
            SqlConnection conn = new SqlConnection(connStr);
            string queryString = "SELECT * from titles;";
            SqlDataAdapter da = new SqlDataAdapter(queryString, conn);
            DataSet ds = new DataSet();
            da.Fill(ds);
            // Work on the data in memory using
            // the DataSet (ds) object
        }
    }
}
Disconnected –
Update, Delete, Insert

```csharp
SqlDataAdapter da = new SqlDataAdapter();
DataSet ds = new DataSet();
SqlCommandBuilder cmdBuilder = new SqlCommandBuilder(da);
da.Fill(ds);

DataRow dr = ds.Tables[0].Rows[0];
dr.Delete();
da.UpdateCommand = builder.GetUpdateCommand();
da.Update(ds);

DataRow dr = ds.Tables[0].Rows[0];
dr["CustomerName"] = "John";
da.UpdateCommand = builder.GetUpdateCommand();
da.Update(ds);

DataRow dr = ds.Tables[0].NewRow();
dr["CustomerName"] = "John";
dr["CustomerSurName"] = "Smith";
 ds.Tables[0].Rows.Add(dr);
da.UpdateCommand = builder.GetUpdateCommand();
da.Update(ds);
```

**INITIAL CODE**

**DELETE**

**UPDATE**

**INSERT**
Steps of Data Access: Connected Environment

• Create connection
• Create command (select-insert-update-delete)
• Open connection
• If SELECT -> use a `DataReader` to fetch data
• If UPDATE, DELETE, INSERT -> use command object’s methods
• Close connection
static void Main()
{
    string connectionString =
        Properties.Settings.Default.connStr;
    string queryString = "SELECT CategoryID, CategoryName FROM
dbo.Categories;";
    SqlConnection connection = new
            SqlConnection(connectionString);

    SqlCommand command = new SqlCommand(queryString, connection);
    try
    {
        connection.Open();
        SqlDataReader reader = command.ExecuteReader();
        while (reader.Read())
        {
            Console.WriteLine("\t{0}\t{1}" , reader[0], reader[1]);
        }
        reader.Close();
        connection.close();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}
Connected – Update, Delete, Insert

• Command class core methods:
  – **ExecuteNonQuery**: Executes a SQL statement against a connection object
  – **ExecuteReader**: Executes the CommandText against the Connection and returns a **DbDataReader**
  – **ExecuteScalar**: Executes the query and returns the first column of the first row in the result set returned by the query
string connString = Properties.Settings.Default.connStr;
SqlConnection conn = new SqlConnection(connString);
SqlCommand cmd = new SqlCommand("delete from Customers" + "where custID=12344", conn);
conn.Open();
cmd.ExecuteNonQuery();
conn.Close();

Can be an update or insert command