



COMPSCI 280 S1 2011 Enterprise Software Development

ADO.NET
An Introduction to ADO.NET

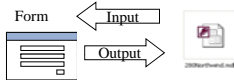


Agenda & Reading

- ▶ **Agenda:**
 - ▶ Relational Databases
 - ▶ ADO .NET Architecture
 - ▶ Overview of ADO.NET
 - ▶ Getting Started with ADO.NET
 - ▶ Using the Data Source Window
 - ▶ Overview of Data Providers
 - ▶ Understanding Connections
 - ▶ Creating Connections
 - ▶ Application Configuration File
 - ▶ Universal Data Link File
- ▶ **Recommended Reading:**
 - ▶ Microsoft ADO.NET 2.0 step by step, Rebecca M. Riordan
 - ▶ Chapter 1: Getting Started with ADO.NET
 - ▶ Chapter 2: Using Connections
- ▶ **ADO.NET**
 - ▶ [http://msdn2.microsoft.com/en-us/library/e80y5yhx\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/e80y5yhx(VS.80).aspx)
- ▶ **Namespace covered:**
 - ▶ System.Data, System.Data.OleDb
- ▶ **Hands-on Lab:**
 - ▶ Lecture 19 Lab: Creating Connections



Databases



- ▶ Projects can display and update the data from database files
 - ▶ VB.NET uses ADO.NET for database access

Database

Tables

Records

- Fields
- Primary Key
 - It is used to uniquely identify each record in a table.
- Foreign Key
 - It is a field in a relational table that matches the primary key column of another table.
 - Establishes a relationship between tables.

Company Name	Contact Name	Contact Title
ALFKI	Alfred Futterkiste	Maria Anders
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo
ANTON	Antonio Moreno Taquería	Antonio Moreno
AROUT	Around the Horn	Thomas Hardy
BEECH	Beverly Hills Country Club	Charles Schmitt

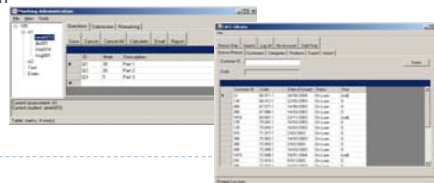
Column

Row

- ▶ Constraints
- ▶ Relationships

Examples

- ▶ Administration
- ▶ Library



Primary key in Table Orders

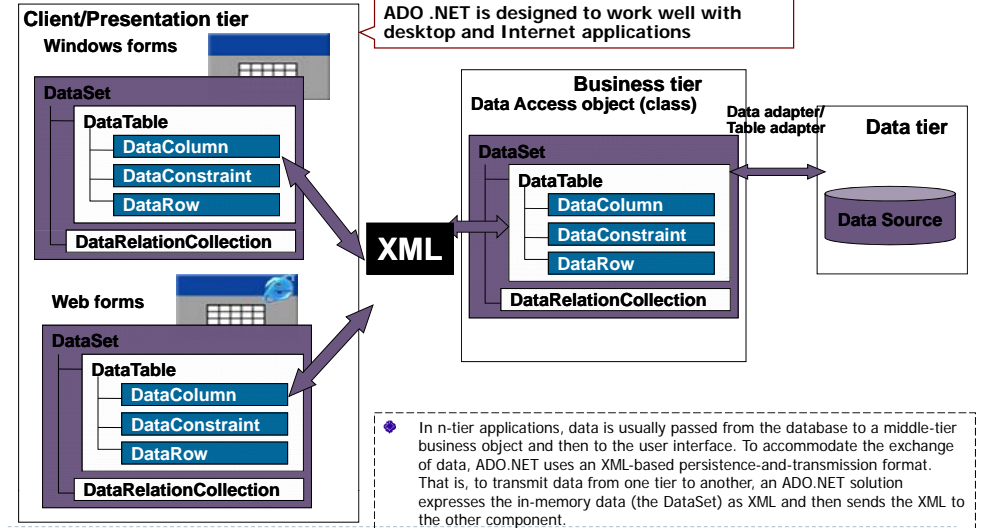
Foreign key in Table Orders

Primary key in Table Customers

One to many relationship

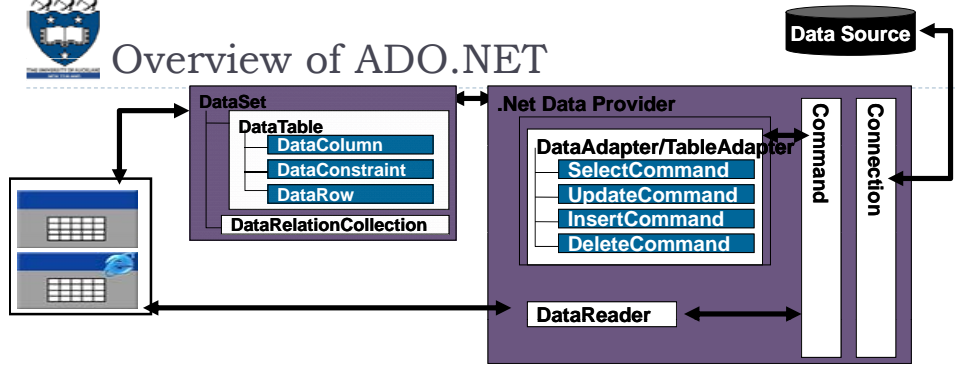


ADO .NET Architecture





Overview of ADO.NET

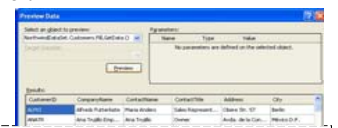
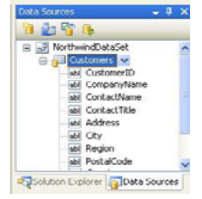


- ADO.NET is not a revision of Microsoft ActiveX Data Objects (ADO) but a new way to manipulate data that is based on disconnected data and Extensible Markup Language (XML).
- In a traditional client/server application, components usually open connections to data sources, and the connections remain open as long as the application is running. Open connections to data sources not only consume resources and increase network traffic but also reduce application performance and scalability.
- ADO.NET has been designed to work with disconnected DataSets to help overcome the problems listed earlier. Applications are connected to the data source only long enough to fetch and update data.
- ADO.NET uses XML as the universal transmission format. This guarantees platform interoperability as long as the receiving component runs on a platform on which an XML parser is available. Any software component can share ADO.NET data as long as it uses the same XML schema as the format for the transmitted data.



Getting Started with ADO.NET

- The process of connecting data to the controls on a form is called data binding. Data binding can be performed in code, but it can also be performed at design time.
 - The Data Sources Window Offers Drag-and-Drop Data-Binding
- Configuring a Data Source Using the Data Source Window
 - Add a Data Source to a Project
 - Choose **Data | Add New Data Source** menu command
 - Choose the data connection
 - Save the Connection String to the Application Configuration File
 - Choose the Database objects (table) (Demo: 19.1_01AddDataSource.wmv)
 - Preview the Contents of a table
 - Double-click the NorthwindDataSet.xsd file in the **Solution Explorer**
 - Right-click the Customer table, choose **Preview Data** from the context menu (Demo: 19.1_02PreviewData.wmv)

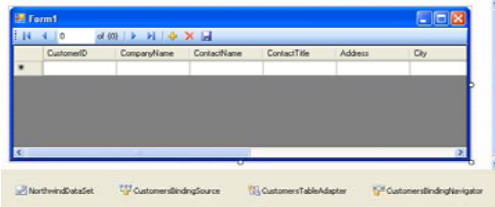


- The Data Sources Window is the main window for viewing the data available to your project. You can drag objects from the Data Sources window onto forms to create data-bound controls that display and navigate data.
- The Data Source Configuration Wizard simplifies connecting your application to data in databases. You can run this wizard to add or edit a data source in your project.



Data Source Window

- Creating a Data-Bound Control Using the Data Source Window
 - Add a Data-Bound Control to a Form
 - Open the Data Source Window
 - Drag the Customer table from the **Data Source window** to the form
 - A DataGridView control and a BindingNavigator control are added onto the form (Demo: 19.1_03AddDataGridViewControl.wmv)



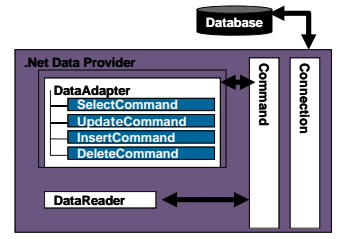
- Step 1: 19.1_01AddDataSource.wmv
- Step 2: 19.1_02PreviewData.wmv
- Step 3: 19.1_03AddDataGridViewControl.wmv

CustomerID	CompanyName	ContactName	ContactTitle	Address	City
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Represent...	Obere Str. 57	Berlin
ANATR	Ana Trujillo Empa...	Ana Trujillo	Owner	Avda. de la Cons...	México
ANTON	Antonio Moreno ...	Antonio Moreno	Owner	Mataderos 2312	México
AROUT	Around the Horn	Thomas Hardy	Sales Represent...	120 Hanover Sq.	London
BERGS	Berglunds snabb...	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå



Overview of Data Providers

- A .NET Framework data provider is used for connecting to a database, executing commands, and retrieving results. Manage access to data stores.
- The .NET Framework data providers that are included in the .NET Framework are:
 - OLE DB .NET Data Provider
 - Access to OLEDB data store
 - SQL Server .NET Data Provider
 - Access to Microsoft SQL Server database
 - ODBC .NET Data Provider
 - Access to native ODBC drivers
 - Oracle .NET Data Provider
 - Access to Oracle database
- The Core Objects of .NET Framework Data Providers
 - Connection
 - Establishes a connection to a specific data source.
 - Command
 - Executes a command at a data source. Exposes Parameters and can enlist a Transaction from a Connection.
 - DataReader
 - Reads a forward-only, read-only stream of data from a data source.
 - DataAdapter
 - Populates a DataSet and resolves updates with the data source





Understanding Connections

- ▶ **Connections**
 - ▶ Represent the physical connection to a data source
- ▶ **Types**
 - ▶ OleDbConnection
 - ▶ SqlConnection
 - ▶ OdbcConnection
 - ▶ OracleConnection
- ▶ **Properties**
 - ▶ **ConnectionString:** Gets or sets the string that is used to open the connection. It contains Provider, Data Source, username and password information.
 - ▶ **State:** Gets a string that describes the state of the connection.
- ▶ **Methods:**
 - ▶ **Open:** Opens a database connection with the settings specified by the ConnectionString.
 - ▶ **Close:** Closes the connection to the database. This is the preferred method of closing any open connection.
- ▶ **Event**
 - ▶ **StateChange:** Occurs when the state of the connection changes
- ▶ **Creating connections**
 - ▶ Method 1: Using the Toolbox
 - ▶ Method 2: Creating at Run Time

The Connection object is used to establish a connection to a database. The Connection object has properties, such as DataSource, UserID, and Password that are needed to access a particular DataSource. Once a connection to the data source has been established, Commands can be run against the data source, and result sets that are returned in the form of streams can be read by DataReaders or stored in DataSet objects.



Method 1: Using the Toolbox

Example: ConnDemo

- ▶ **Add a Connection to the Toolbox**
 - ▶ Open the **Toolbox**, and right-click the **Data** tab
 - ▶ Select **Choose Items** from the Context menu.
 - ▶ Select the **OleDbConnection** check box
 - ▶ (Demo: 19.2_01CreateConnection.wmv)
- ▶ **Add a Connection to a Form**
 - ▶ Drag the **OleDbConnection** control onto the form
 - ▶ Set the **ConnectionString** property of the control
 - ▶ Open the Server Explorer, and click the Add DataConnection button
 - ▶ In the Data Link Properties dialog box, click the Provider tab.
 - Click Microsoft.ACE.OLEDB.12.0 Provider, and then click Next.
 - ▶ Click the Connection tab, and then click the ellipses button (...) to select the database file
 - ▶ Click OK.
 - ▶ (Demo: 19.2_02SetConnection.wmv)
- ▶ **Open the Connection**



```
try {
    cnMyConnection.Open();
    MessageBox.Show(cnMyConnection.ConnectionString + " " + cnMyConnection.State);
} ...
```



Method 2: Creating at Run Time

- ▶ **Using Constructors**
 - ▶ New(), or
 - ▶ New(ConnectionString)
- ▶ **Creating connection at run time**
 - ▶ Declare a connection object

```
private OleDbConnection cnMyConnection;
```

- ▶ Create the connection object with the ConnectionString

```
try {
    string DBconnString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=H:\compsci280\Northwind.mdb;";
    cnMyConnection = new OleDbConnection(DBconnString);
    cnMyConnection.Open();
    MessageBox.Show(cnMyConnection.ConnectionString + " " + cnMyConnection.State);
} catch (Exception Ex) {
    MessageBox.Show(Ex.ToString(), "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Error);
} finally {
    if (cnMyConnection != null)
        cnMyConnection.Close();
}
```

Print the connection status

CAUTION: It is recommended that you always close the Connection when you are finished using it in order for the connection to be returned to the pool.



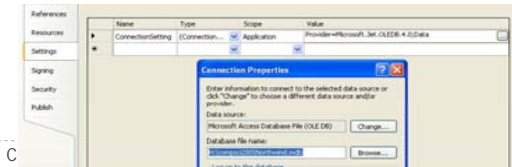
Application Configuration File



- ▶ **Understanding Application Configuration File**
 - ▶ A configuration file is generated in the same folder as the application's executable file and has the following naming convention: ConnDemo.exe.config

application's main namespace

- ▶ To store a connection in the Application Configuration File at design time
 - ▶ Select on the **Project** menu, click **Properties**.
 - ▶ Select the **Settings** pane.
 - ▶ Enter a name (ConnectionString) in the **Name** column. The name cannot contain spaces.
 - ▶ Select (connectionStrng) from the **Type** drop-down list.
 - ▶ Select **Application** from the **Scope** drop-down list.
 - ▶ Enter a ConnectionString or click the ... button to select the database file.
 - ▶ (Demo: 19.3AppConfig.wmv_)





Using an App.config file

```
<?xml version="1.0" encoding="utf-8" ?><configuration>
...
<connectionStrings>
  <add name="ConnDemo.Properties.Settings.ConnectionSetting"
  connectionString="Provider= Microsoft.ACE.OLEDB.12.0;Data Source=H:\compsci280\Northwind.mdb;"
  providerName="" />
</connectionStrings>
```

application's main namespace

Store settings as name(key)/value pairs

To get the ConnectionString from the Application Configuration File in code

- The Properties.Settings object provides access to the application's settings and allows you to dynamically store and retrieve property settings and other information for your application.

```
txtConfig.Text = Properties.Settings.Default.ConnectionSetting;
```

- When Visual Studio creates a new project it automatically generates a Properties folder and populates it with classes that wrap the configuration settings you generate with the Properties Editor. The Properties Editor updates the xxx.config file and the Settings.cs class file (the class wrapper around the generated properties) automatically.
- The ConnectionString properties we set in the previous examples are hard-coded. That's not a problem for little example. But in the real world, the location of a database is almost certain to change. We should store the database connection string in a single place for ease of application maintenance.



OLE DB Universal Data Link (.udl)

- UDL files
 - Store the connection string that an ADO .NET OleDbConnection object uses to establish a database connection



- Creating a UDL file
 - Create a text file in a folder, rename it as conn.udl
 - Double-click it to open the ADO connection string editor
 - Click the Provider tab
 - Choose Microsoft.ACE.OLEDB.12.0 Provider
 - Click the Connection tab
 - Click the ... button to select the database file
 - Enter username and password if needed
 - Click Test Connection button to test the connection

Select the database file

The selections you make are written as a connection string in the .udl file.

```
[oledb]
; Everything after this line is an OLE DB Initstring
Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=H:\compsci280\Northwind.mdb;
Persist Security Info=False
```



Using a UDL File

- Creating a connection using a UDL file
 - Use FileName in the ConnectionString
 - Sets the name of the Universal Data Link (UDL) file for connecting to the data source.

```
try {
  using (OpenFileDialog dlgOpenFile = new OpenFileDialog()) {
    dlgOpenFile.Filter = "UDL files (*.UDL)|*.udl";
    if (dlgOpenFile.ShowDialog() == DialogResult.OK) {
      string DBconnString = "File Name=" + dlgOpenFile.FileName;
      cnMyConnection = new OleDbConnection(DBconnString);
      cnMyConnection.Open();
      MessageBox.Show(cnMyConnection.ConnectionString + " " + cnMyConnection.State);
    }
  }
} catch ...
```

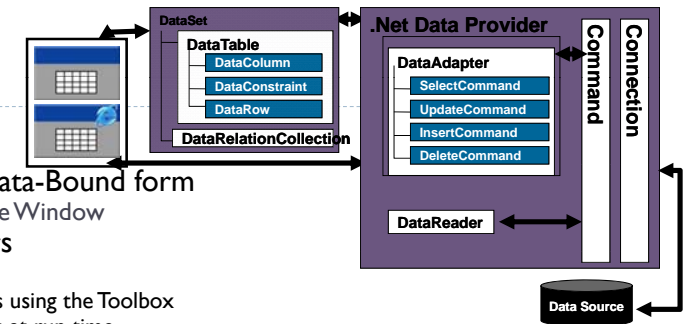
Use "File Name" to read from a udl file

- Because UDL files can be modified externally to any ADO.NET client program, connection strings that contain references to UDL files will be parsed every time the connection is opened.
- It is possible to supply connection information for an OleDbConnection in a Universal Data Link (UDL) file; however you should avoid doing so. UDL files are not encrypted and expose connection string information in clear text. Because a UDL file is an external file-based resource to your application, it cannot be secured using the .NET Framework.



Summary

- ADO.NET
 - Architecture
- Creating a Simple Data-Bound form
 - Using the Data Source Window
- Using Data Providers
 - Using Connections
 - Creating connections using the Toolbox
 - Creating connections at run time
 - Application Configuration files
 - UDL files
 - Path



```
... "Data Source=" + Application.StartupPath + "\\dbl.mdb;"
```

- The ConnectionString can be set at design-time; however, only the absolute path can be used. This poses a problem when you move the system to a different directory, a disk etc. We can set the ConnectionString using a relative path at run-time.
- Application.StartupPath: the path for the executable file that started the application, not including the executable name.