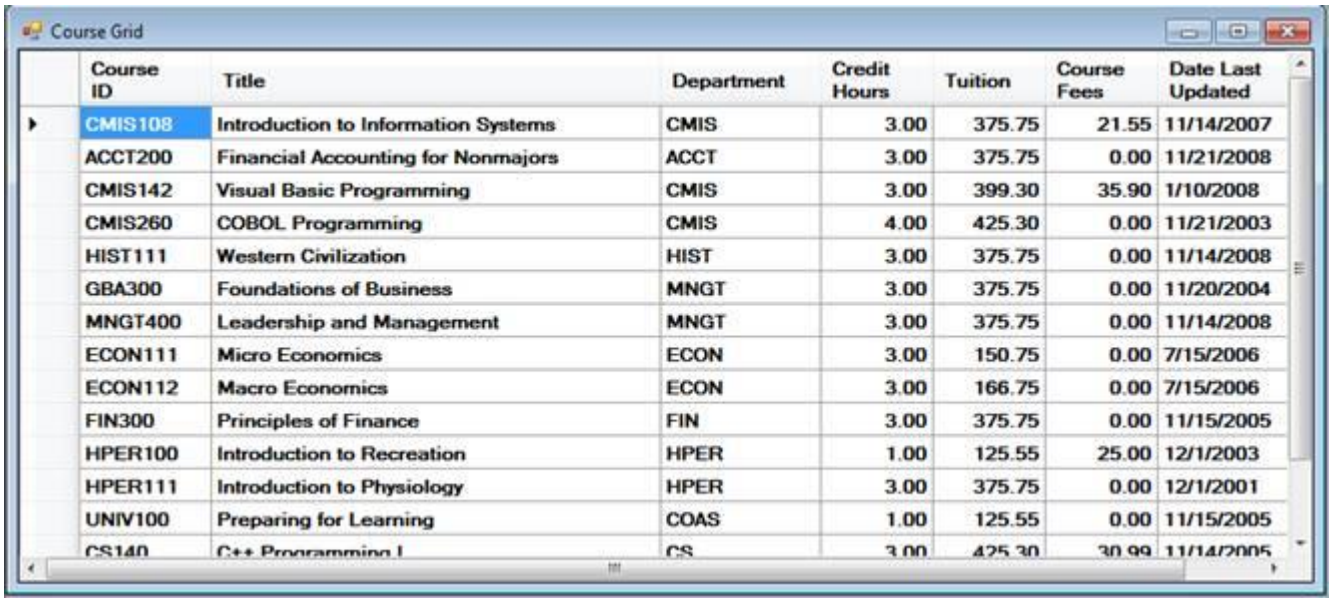
Windows Database Programming
Multiple Document Interface

This chapter teaches you to store data to and retrieve data from databases. The chapter focuses on Microsoft Access and Microsoft SQL Server databases; however, the programming techniques covered also work with other database management systems (DBMS) such as Oracle Corporation's Oracle DBMS and Oracle databases. You will learn to display data in various Windows-based controls such as TextBox and DataGridView controls.

INTRODUCTION

In this chapter you will develop a project for the VB University that enables administrators to store new student records and retrieve detailed information about courses offered and students enrolled in courses. You will develop several different forms that retrieve and store data.

The first form displays information from the **Course** table of the **VBUniversity** database through use of a **DataGridView** control.



Course ID	Title	Department	Credit Hours	Tuition	Course Fees	Date Last Updated
CMIS108	Introduction to Information Systems	CMIS	3.00	375.75	21.55	11/14/2007
ACCT200	Financial Accounting for Nonmajors	ACCT	3.00	375.75	0.00	11/21/2008
CMIS142	Visual Basic Programming	CMIS	3.00	399.30	35.90	1/10/2008
CMIS260	COBOL Programming	CMIS	4.00	425.30	0.00	11/21/2003
HIST111	Western Civilization	HIST	3.00	375.75	0.00	11/14/2008
GBA300	Foundations of Business	MNGT	3.00	375.75	0.00	11/20/2004
MNGT400	Leadership and Management	MNGT	3.00	375.75	0.00	11/14/2008
ECON111	Micro Economics	ECON	3.00	150.75	0.00	7/15/2006
ECON112	Macro Economics	ECON	3.00	166.75	0.00	7/15/2006
FIN300	Principles of Finance	FIN	3.00	375.75	0.00	11/15/2005
HPER100	Introduction to Recreation	HPER	1.00	125.55	25.00	12/1/2003
HPER111	Introduction to Physiology	HPER	3.00	375.75	0.00	12/1/2001
UNIV100	Preparing for Learning	COAS	1.00	125.55	0.00	11/15/2005
CS140	C++ Programming I	CS	3.00	425.30	30.99	11/14/2005

Databases and ADO.NET

Concepts and Terminology

Terminology – learn these terms:

- **Database** – a special repository—consists of one or more physical files—used to store and retrieve data.
- **Relational database** – a specific type of database where data rows are stored in separate tables, and the tables are related to each other by key values (see figures showing table data later in these notes).
- **Table** – basic database object that stores data – looks like a spreadsheet when you're viewing data. This figure shows a **Student** table diagram from a **Microsoft SQL Server**database.

Table - dbo.Student | Summary

StudentSSN	FirstName	MiddleInitial	LastName	Address	City	StateCode	ZipCode	Ph
111-11-1111	Andrea	A	Able	100 S. Andrews St.	Edwardsville	IL	62025	61
222-22-2222	Bobby	B	Best	15 Beauford Way	Alton	IL	63000	61
333-33-3333	Claudia	C	Charming	Clover Avenue North	Maryville	IL	610004400	NU
888-88-8888	Henrietta	H	Headstrong	S. Howard Road, #8	Maryville	IL	610005600	61
999-99-9999	Inga	I	Indigo	5203 Icon Way	Alton	IL	63000	61
444-44-4444	Donald	D	Domingo	1804 Drewey Drive	Alton	IL	63000	← Row
555-55-5555	Edwardo	E	Easterwing	East South Road 188	Edwardsville	IL	62025	61
666-66-6666	Fredericko	F	Freewaydrive	Four Points South #2	Edwardsville	IL	62025	61
777-77-7777	Greta	G	Greathouse	S. Grand St. #15	Alton	IL	63000	61
111-11-1112	Arnold	A	Airhead	101 S. Andrews	Edwardsville	IL	62025	61
555-55-5554	Claude	C	Charming	Northwest Avenue	Alton	Il	63000	NU
111-11-1113	Harvey	H	Headstrong	105 Northwest St.	Edwardsville	IL	62025	61
999-99-9988	Amy	A	Airhead	17 Southend Drive	St. Louis	Mo	63115	31
333-33-3334	Ariana	A	Airhead	52 West 14th St.	St. Louis	Mo	631155600	31
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NU

Key Column

Rows and Columns – a table consists of rows and columns.

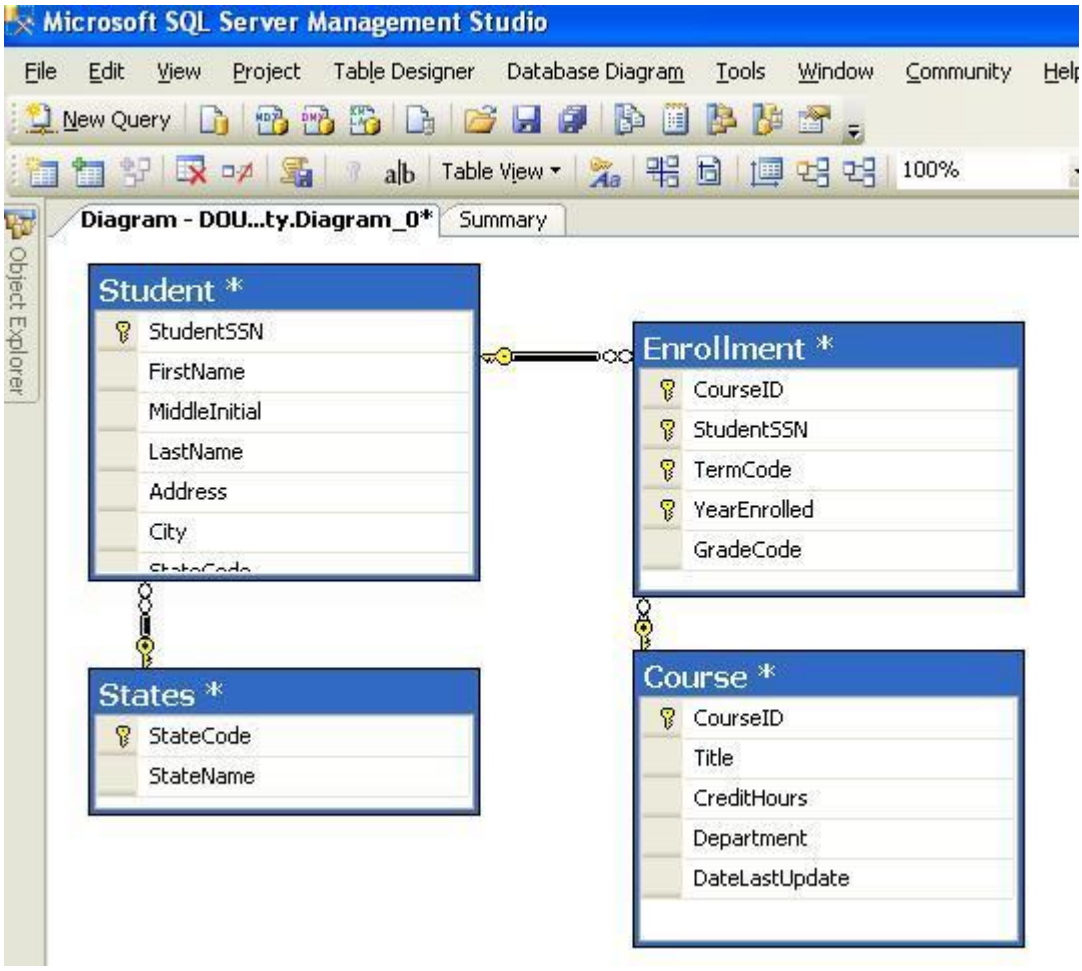
- **Row** = a row is a **record** for an individual course or student.
- **Column** = a column is a **field** of data stored for each course such as **CourseID**, **Title**, **Department**, or for each student such as the **StudentSSN**, **LastName**, and **FirstName**.
- **Key Column (Field)** – uniquely identifies a row in a table – almost all database tables require one or more columns that form the key column(s) to identify rows uniquely – eliminates the occurrence of duplicate rows.

Database Products – VB.NET stores and retrieves data for many different database products including, but not limited to:

- **Oracle** (by Oracle Corporation) and **DB2** (by IBM) for large systems—these are competing relational database management systems.
- **Microsoft SQL Server** for mid-sized systems and larger scalable systems.
- **Microsoft Access** and other small-sized, individual user or small group systems.

Entities and Relationships – a database stores data about individual ***entities*** in separate tables – example university database entities include **Students**, **Courses**, **Enrollment** (the enrollment of students in courses), and **States**.

- This figure shows an ***entity-relationship*** diagram for tables in a Microsoft SQL Server version of the **VB University** database.



- **Relationship** – the lines connecting entities represent relationships between the rows in one table and rows in another table.

- **One-to-many relationship** – this is the relationship from rows in the **Student** table to rows in the **Enrollment** table (there can be multiple enrollments by students in a course).
    - The **key symbol** represents the **one** side of the relationship; the **infinity symbol** represents the **many** side of the relationship.
    - A student can have many enrollments, but an enrollment row belongs (is associated) to only one student row.
    - The relationship from **Course** to **Enrollment** is also **one-to-many** – a course can have many enrollments.
    - There are other kinds of relationships that you will study in your course on database modeling and design.

Primary Key Columns –table rows are uniquely identified by one or more **primary key columns**. Each table has a different primary key. In the above figure the primary key column for the various tables are:

- **Student** table key = **StudentSSN** (social security number).
- **States** table key = **StateCode** (2-character abbreviation for the state name).
- **Course** table key = **CourseID** (up to 7-characters identifying a course).
- **Enrollment** table key = **CourseID + StudentSSN + TermCode + YearEnrolled** (all four columns are required to uniquely identify a row in this table since students enroll in a course more than once. Terms are coded values: **SU**=Summer; **FA**=Fall; **SP**=Spring.

Table Types

- **Base table** – a base table is one that stores basic information about entities – examples above are the **Course** and **Student** tables.
- **Association table** – an association table stores information about associations between entities or information about business transactions – the **Enrollment** table associates **Course** and **Student** records and the act of enrolling in a course is a business transaction for the university.
- **Validation table** – a validation table validates data entered into another table. The **States** table is used to validate the value of the **StateCode** column information entered for a student row in the **Student** table.

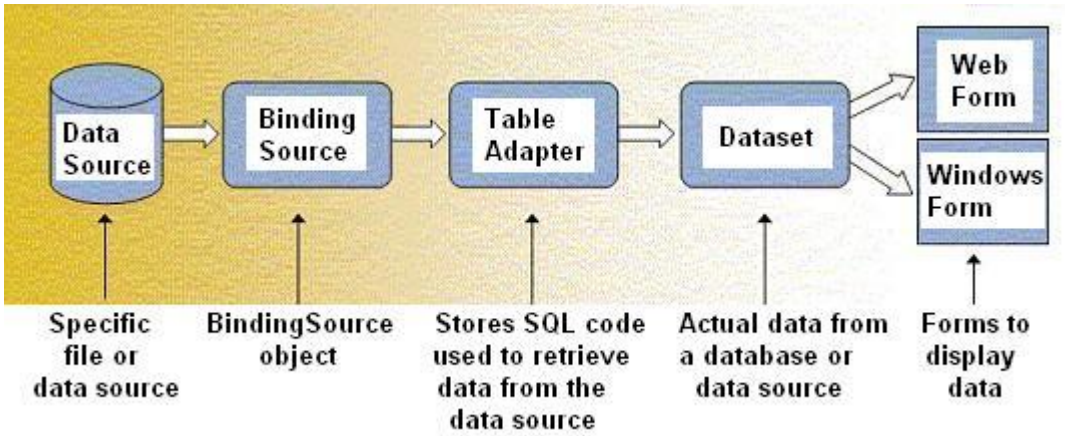
ADO.NET

VB.NET uses **ADO.NET** (Active-X Data Objects with .NET technology), a database technology that supports connection to different database products. ADO.NET:

- Provides the application programming interface between the program application and the Database Management System that manages the database. The current API is **ADO.NET 4**.
- Stores and transfers data using the **Extensible Markup Language** (**XML**).
- Provides four different types of connections to databases across networks:
  - **SQLClient** – used to connect to Microsoft's SQL Server DBMS.
  - **OracleClient** – used to connect to Oracle Corporation's Oracle DBMS.
  - **OLEDB** (Object Linking and Embedding Database) – used to connect to all other database formats – this includes Microsoft Access DBMS.
  - **ODBC** (Open Database Connectivity) – used to connect to miscellaneous data sources.
- ADO.NET supports database access using forms developed for either a **Windows** or **Web Form** environment.
- ADO.NET provides **controls** that you add to a form that are used to connect to and manage data in a database table. Columns from a database table are ***bound*** to database controls.
- Controls you can bind include: Label, TextBox, and ComboBox controls as well as some new controls such as the DataGridView control.

Connecting to a Database or Data Source with VB

This figure shows the steps in setting up a connection to a database or other type of data source.



- Configure a **binding source**. The binding source links to a **data source** – a data source is usually a specific database file, but can be another data source such as an array or text file. *Note: Your text erroneously states that the data source objects replaces the connection object used in earlier versions of VB – actually the data source object is used to configure the connection object – they are two different objects.*
- Configure a **table adapter**. A table adapter handles data retrieval and updating. The table adapter creates a dataset.
- Create a **dataset**. A dataset stores data rows that have been retrieved.
- Add controls to the form and set properties of the controls to **bind** the controls to the columns of a specific table in the dataset.
- VB will automatically write the code needed to fill a dataset.

Databases Used for Class Projects

For our projects, you may work at home or in the computer laboratory. We will use **Microsoft Access** – there are copies of the databases saved as Microsoft SQL Server version files for those of you who wish to experiment in using SQL Server.

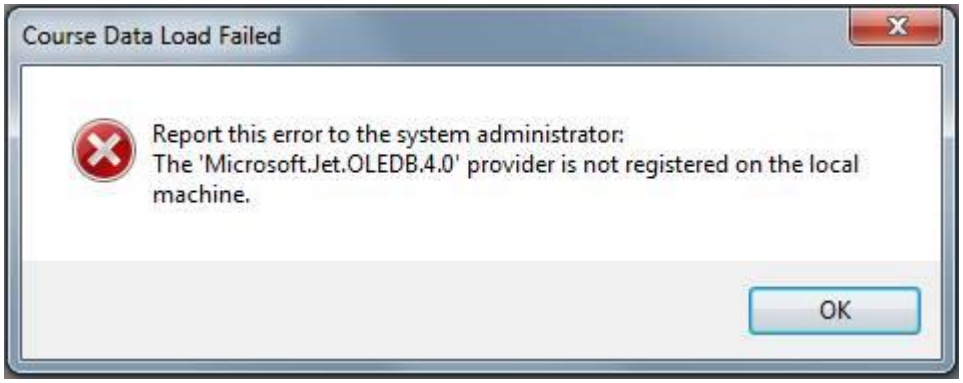
- **MS Access databases** are stored in a single file.
  - A MS Access database file is much smaller than the same data stored in MS SQL Server and is very portable.
  - You will learn to access data in the file through Visual Basic so you need not have Microsoft Office installed on your home computer.
  - The database files are named with a **.mdb** file name extension.

- **MS SQL Server Express** is a free version of this Microsoft DBMS.
    - The free Express version is downloadable from Microsoft.
    - SQL Server and VB make it easy to move small SQL Server databases along with your projects for testing. You will learn how to do this as part of the in-class exercise.
    - The database has at least two files – one is named with a **.mdf** filename extension and the accompany log file used for data recovery has a **.ldf** filename extension.

Microsoft Access and Windows 7 Problems

The Windows 7 operating system is generally purchased as a **64-bit** version. This creates a problem:

- The Microsoft Access database provider (the Microsoft.Jet.OLEDB.4.0 provider) is not available in a **64-bit** version -- it is only available as **32-bit**.
- Generates the **"Microsoft.Jet.OLEDB.4.0 provider is not registered on the local machine**" error when attempting to load a form that connects to a database.
- This figure shows a data load failure exception for the **Course Grid** form that you will build later in this note set. Also, the form will not display any data after you click the OK button.

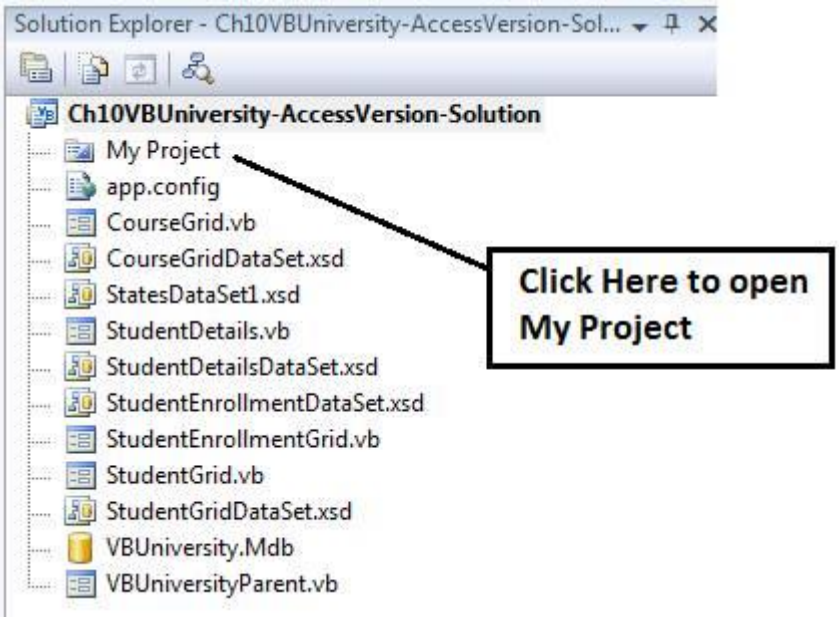


The approach you take to fixing this problem depends on whether you are running Visual Studio Ultimate Edition or the Visual Basic Express Edition.

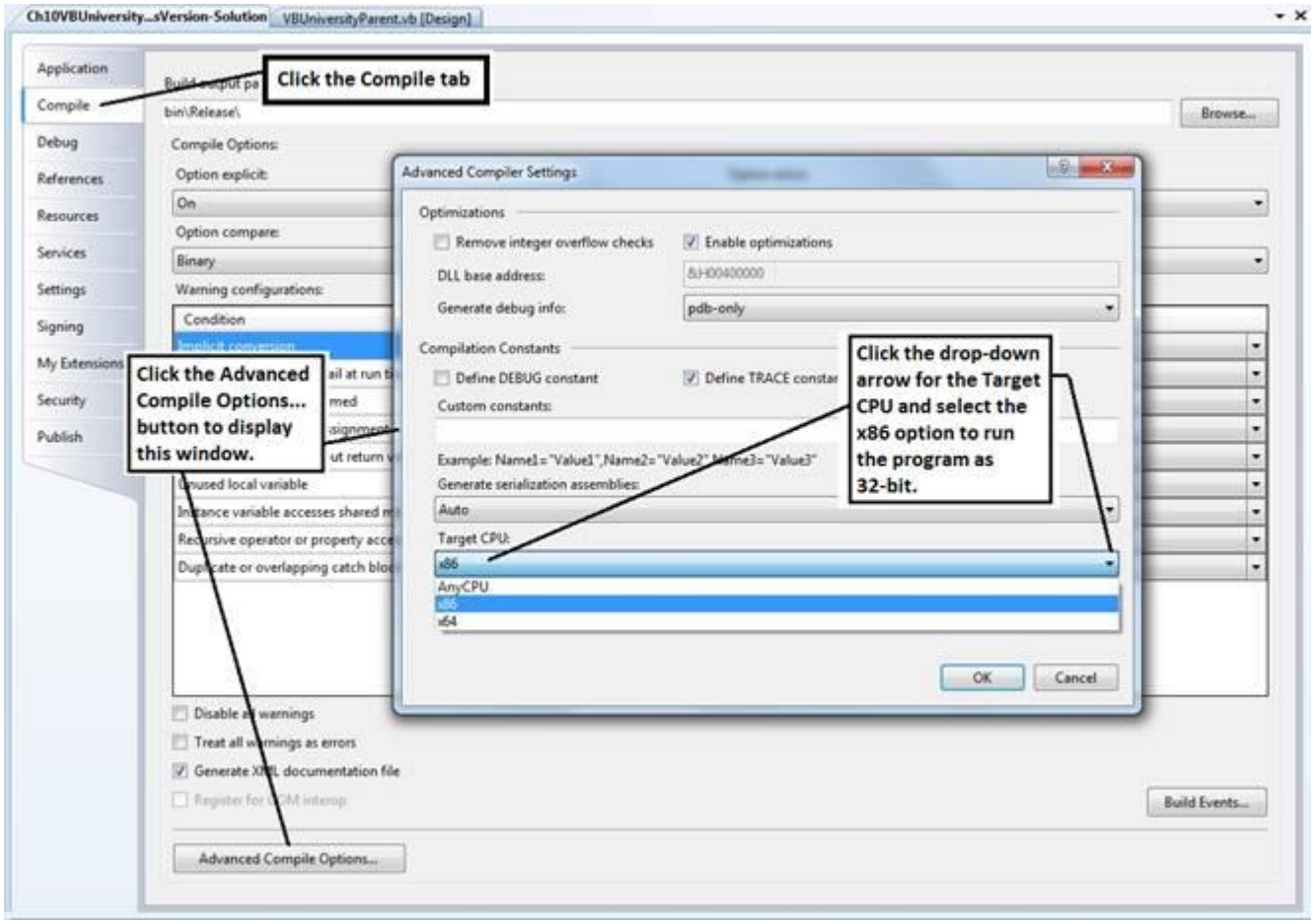
Visual Studio Ultimate Edition Fix

Use this procedure if you are running Visual Studio Professional Edition. In order to connect to an Access database when running Visual Studio on a 64-bit machine, you need to specify for Visual Studio to use 32-bit mode. Follow these steps to eliminate this error:

1. Open Solution Explorer -- click on the **My Project** node to open the **My Project** window.



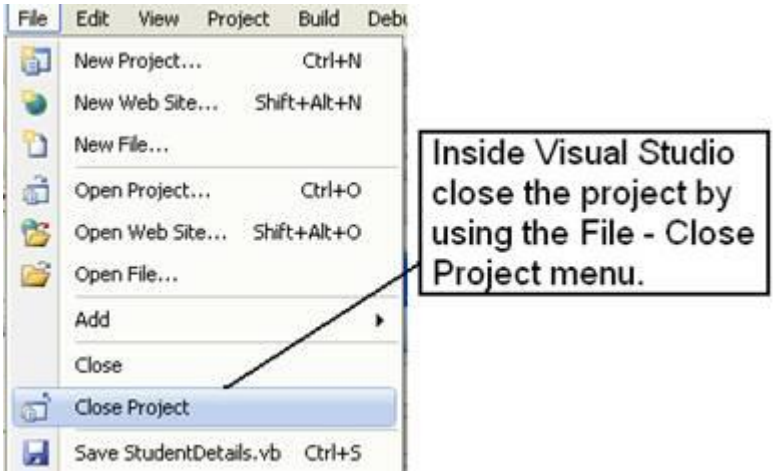
2. Click the **Compile** tab along the left side of **My Project** -- the figure shown below will display.
3. Click the **Advanced Compile Options...** button to display the **Advanced Compiler Settings** dialog box.
4. Click the **Target CPU** drop-down arrow and select the **x86** target -- this will allow your program to run in 32-bit mode.



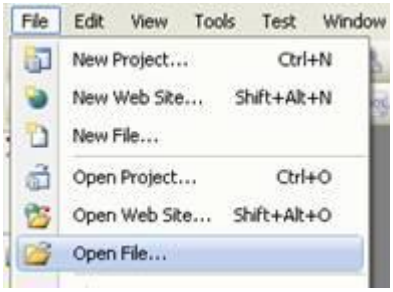
Visual Basic Express Edition Fix

Use this procedure if you are running Visual Basic Express Edition. In order to connect to an Access database when running Visual Studio on a 64-bit machine, you need to specify for Visual Studio to use 32-bit mode; however, the above procedure will not work because the Advanced Compiler Settings window will not display a Target CPU drop-down box. Instead, you will need to carefully modify the project file using a text or XML editor.

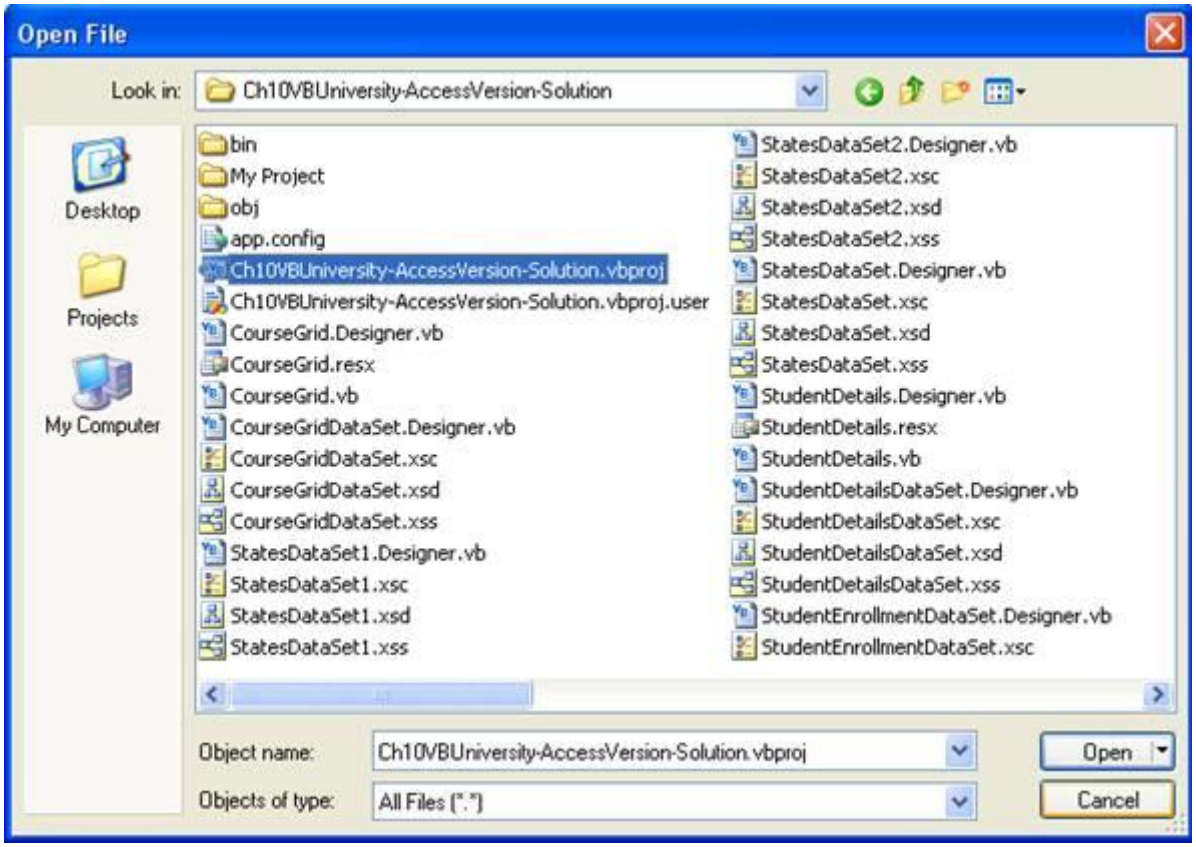
1. With your VB application program open, select the **File menu**, **Close Project** option to close the project and/or solution as shown in this figure.



2. Select **File menu**, **Open File** option as shown in this figure.



3. Navigate (browse) to the project directory, and highlight the project file (the file is inside the solution folder for your project and ends in the filename extension of **.vbproj** as shown in this figure). After highlighting the project file, click the **Open** button.



4. After opening the file, it should display (open) in an XML editor window.
5. Locate the first <PropertyGroup> section and add the following line:

   <PlatformTarget>x86</PlatformTarget>

6. Save the project (**.vbproj)** file. You can now reopen the project—use the **File menu**, **Open Project** option to browse for the project folder

and click the solution file (**.sln**) to open the project, and continue with your debugging and testing.

About SQL Server Express and Potential Problems

SQL Server Express (SSE) is a limited edition version of Microsoft SQL Server database product.

- Designed for use as a single-user, desktop database.
- Operates very much like the full SQL Server database engine.
- Microsoft has recommended that application users move from the use of Access files (.mdb) to SQL Server files (.mdf) for Visual Studio.NET development -- we do not do this in our classrooms because SSE requires you to logon as an administrator, and this is not allowed for students in an SIUE classroom.

You may wish to experiment on your home PC with SSE. There are potential SSE problems. Although the problems are rare, they usually arise because SSE uses a logging file to track of transaction modifications to the database (the .**ldf** file). If you receive error messages while trying to access a SSE database file or while executing code that should connect to an SSE database, you can try to resolve the problem by restarting the SSE Windows service that runs the database engine as follows:

- *Start*
- *Control Panel*
- If you are using Category View, select *Performance and Maintenance*
- *Administrative Tools*
- *Services*
- Click on *SQL Server (SQLEXPRESS)*
- Click on *Restart*
- Try again to access the file in Visual Studio

VB University Project – DataGridView Control

Copying the Database and Starting the Project

For this exercise you can use either the:

- Sql Server Express **VBUniversity.mdf** and accompanying **VBUniversity_log.ldf** files available on the drive **Y:** network server, or
- Microsoft Access **VBUniversity.mdb** file available on the drive **Y:** network server.

1. Copy the one of these databases from the drive **Y:** network server to the **My Documents** folder.
2. Begin a new project – name it **Ch10VBUniversity**.
3. Set the first form's properties as follows:
   - **FileName** = **CourseGrid.vb**
   - **Text** = **Course Grid**
   - **Font – Size** = **9**
   - **Font – Bold** = **True**

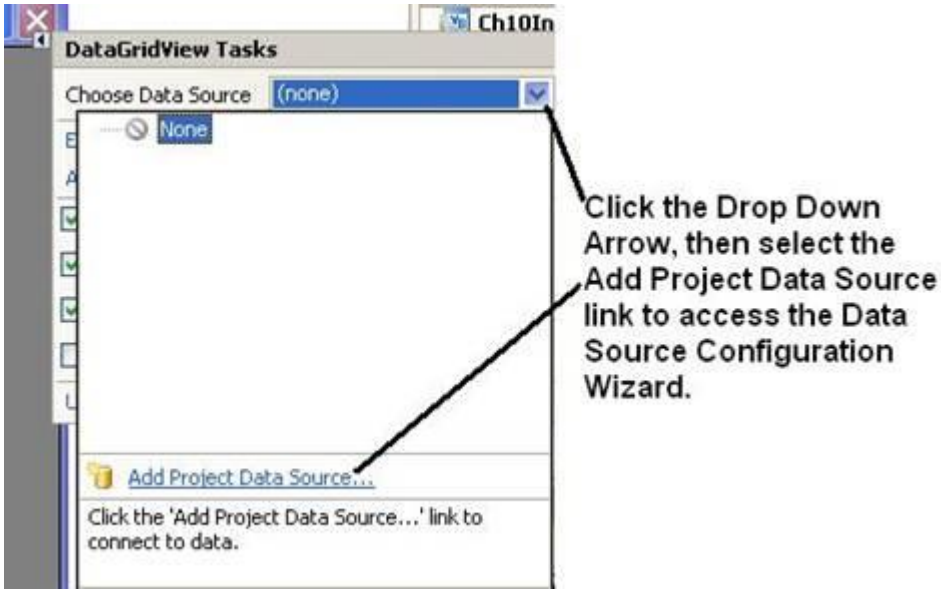
Displaying Data

4. Access the Data section of the Toolbox in design view. Add a **DataGridView** control from the toolbox as shown here.

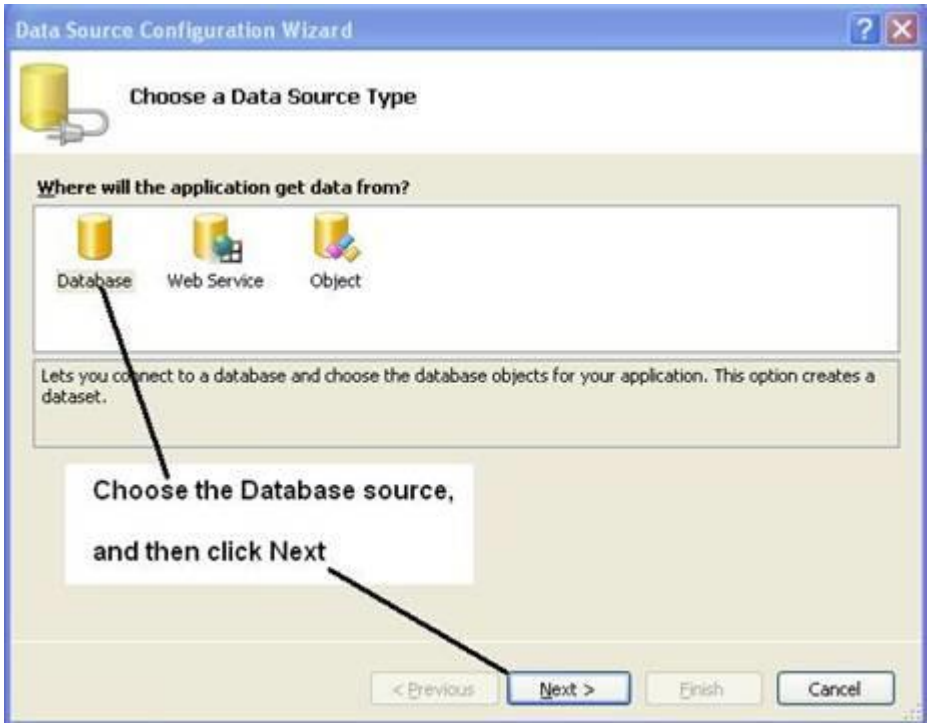
5. Click the **smart arrow tag** in the upper right-corner of the DataGridView control – this displays the **DataGridView Tasks** window shown in the figure given above.

6. Modify the DataGridView Tasks window as follows:

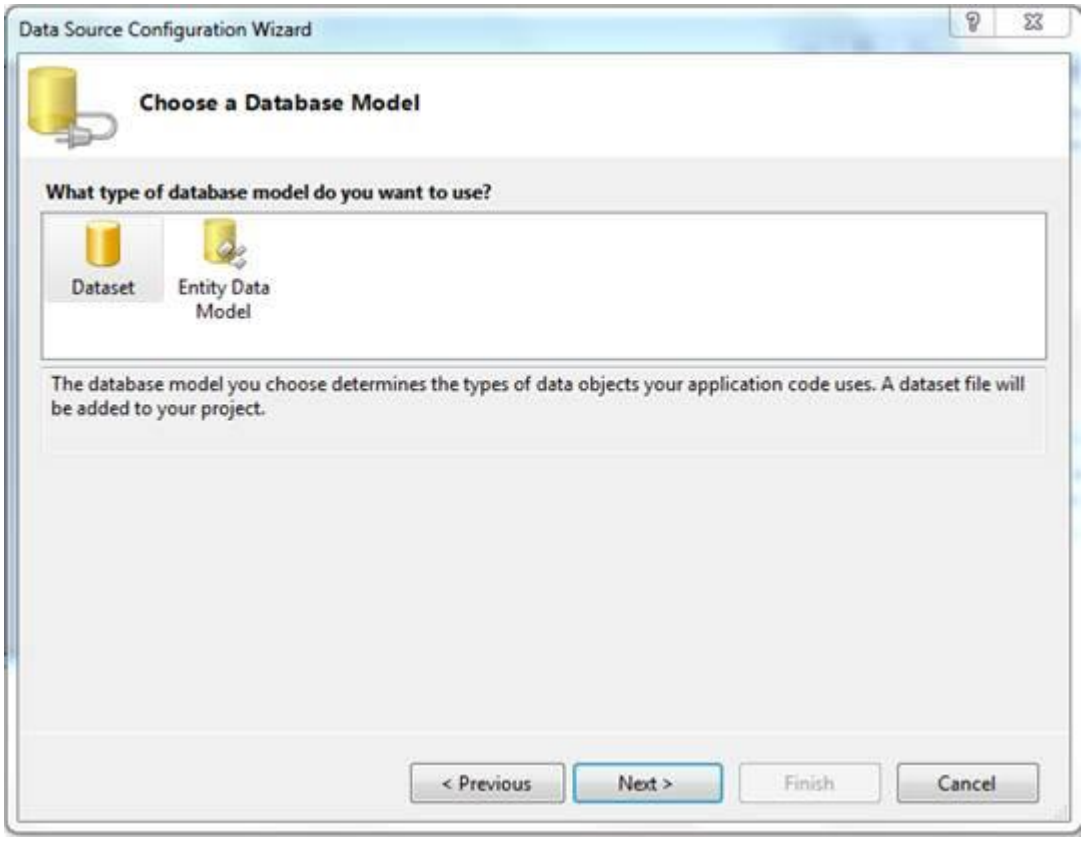
- **Dock** property – you can select either the value **Fill** in the **Properties** window or click the **Dock in parent container** link shown in the figure above – this will cause the control to fill the form.
- Check the **Enable Column Reordering** CheckBox.
- Uncheck the **Enable Adding, Enable Editing, Enable Deleting** – these check boxes that are checked by default, but we want this form to be read-only.
- **Choose Data Source** – click the dropdown and select the **Add Project Data Source…** hyperlink shown in the figure below – data sources can also be added with the **Data**menu.



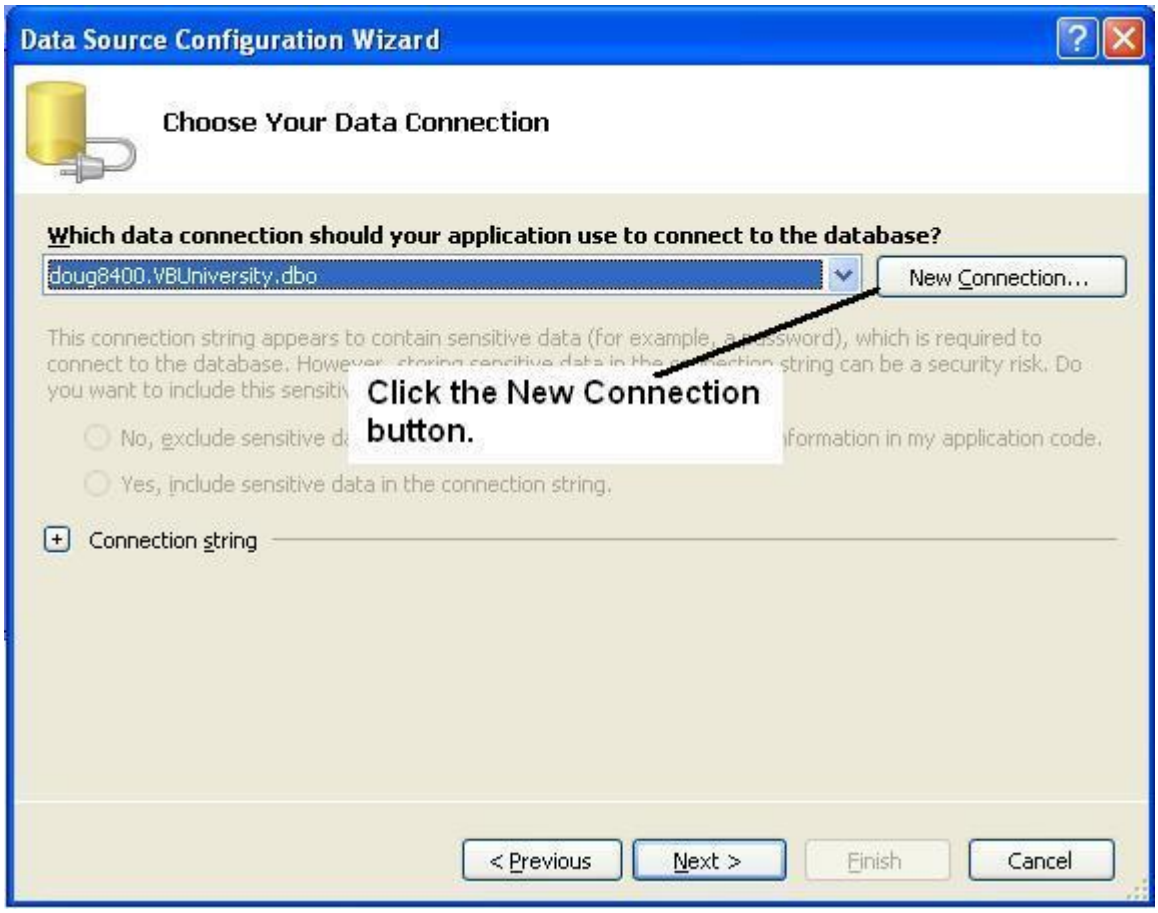
7. **Data Source Configuration Wizard** – the wizard displays the **Choose a Data Source Type** window, select the **Database** as a data source type as shown in this figure, then click the **Next** button.



8. **Choose a Database Model** window – click **Dataset** option and click the Next button.

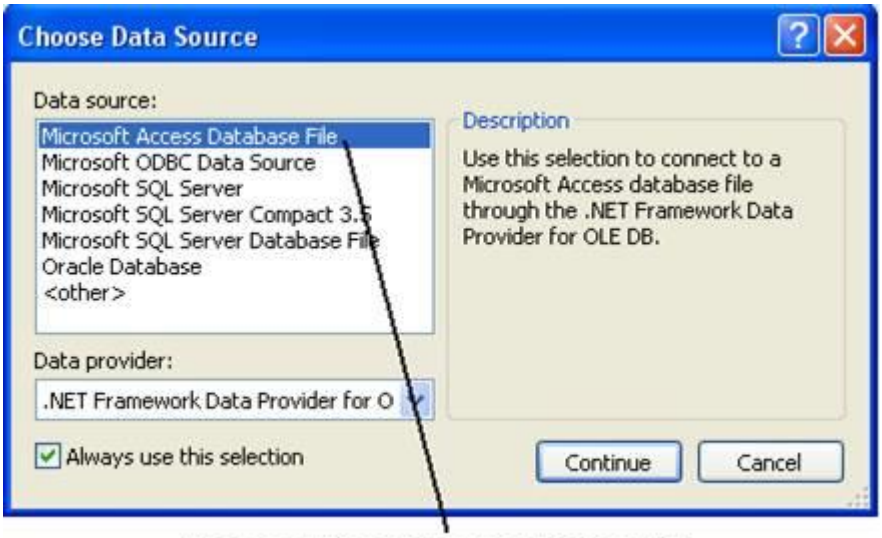


9. **Choose Your Data Connection** window – click the **New Connection** button. Your software may show an existing connection such as that shown in the figure below – it shows a connection to a server named **doug8400** (my home computer) – you want a new connection for this project so ignore any existing connections.



10a**.** Either a **Choose Data Source** or an **Add Connection** window will open.

- If a Choose Data Source window is displayed as shown in this figure, we will usually select the **Microsoft Access Database File** option. Click **Continue**.

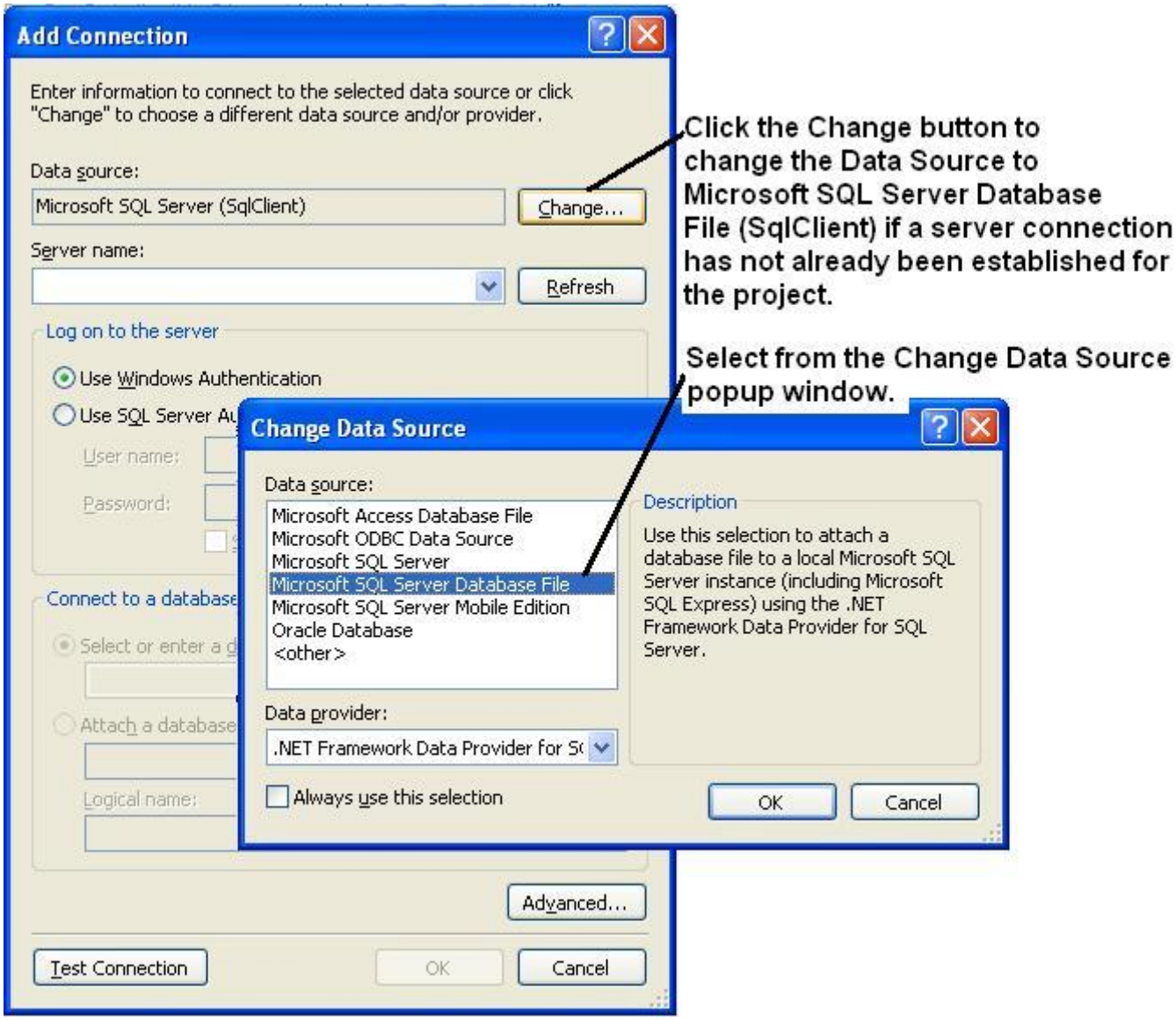


Choose a Microsoft Access Database File.

- If the Add Connection window is open or if you wish to change the type of connection to be created, then click the **Change** button. The default Data Source is **Microsoft SQL Server (SqlClient)** as shown in the figure below – to add either a SQL Server Database file or MS Access database file to your project as the data source, click the **Change**button.

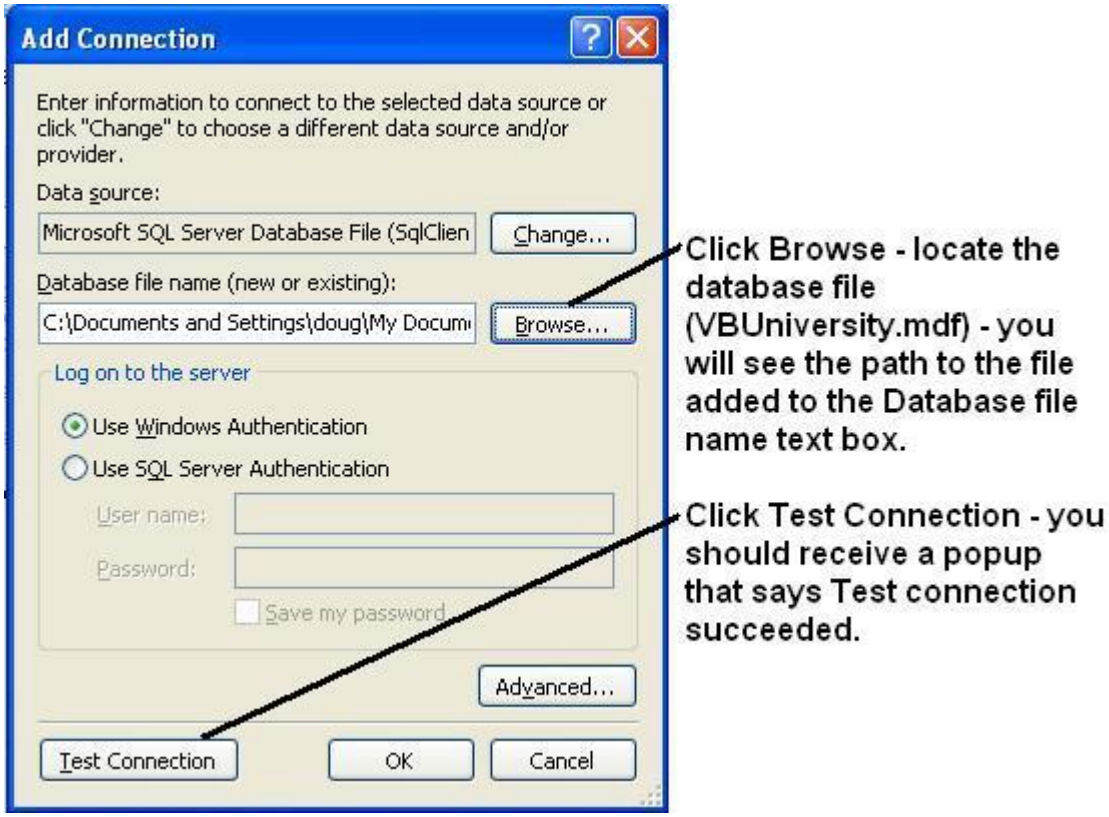
10b. **Change Data Source** window – this window may display next depending on the type of database you copied from drive Y: (either MS SQL Server Express or MS Access). The appearance of the window is different for different data sources. You may want to change the data source by clicking the **Change** button to change to Microsoft Access.

NOTE: Usually in class we will change to a Microsoft Access Database File option. If you are using Microsoft Access, skip to Step 11b.



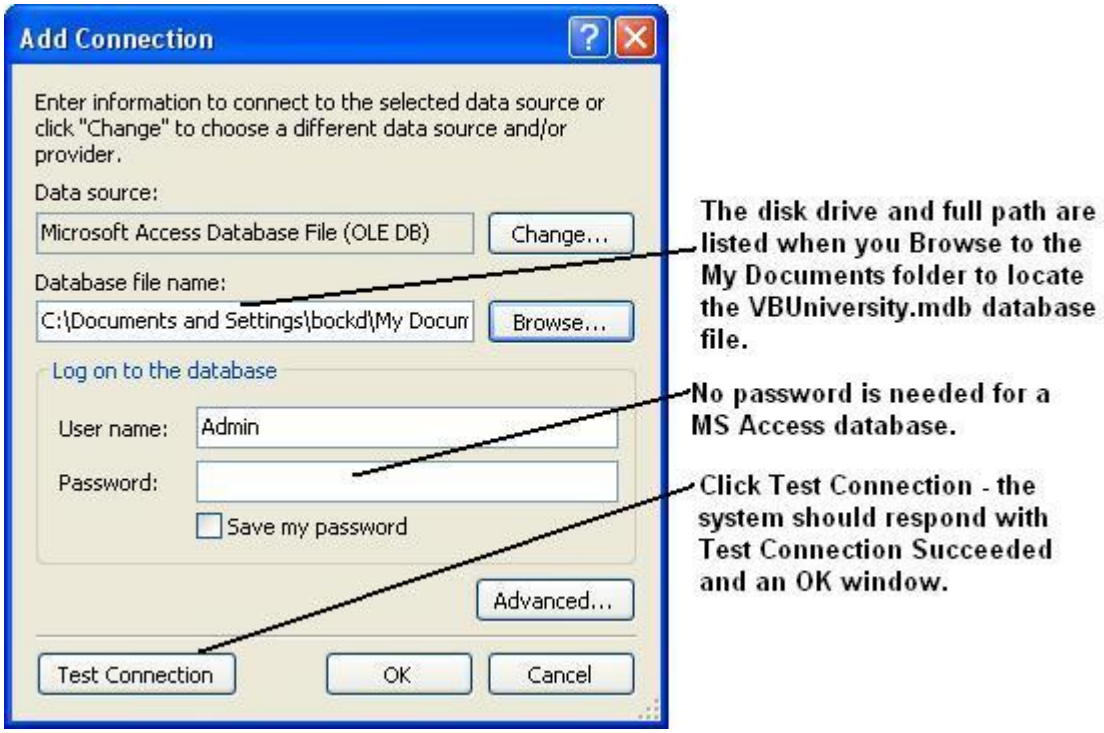
11a. If you are using SQL Server Express:

- The next figure shows the MS SQL Server **Add Connection** dialog box, click **Browse** – locate the **VBUniversity.mdf** file you earlier copied to the **My Documents** location – select it and click **Open** in the browse window – you will see the path and database file name added to the **Add Connection** dialog box as shown in the figure below.
- The server logon will use **Windows authentication**.
- Click the **Test Connection** button – if the connection succeeds, you will see a popup window telling you that the **Test connection succeeded**. Click **OK** to close the popup and **OK** to close the **Add Connection** dialog box.
- Skip to step 12.

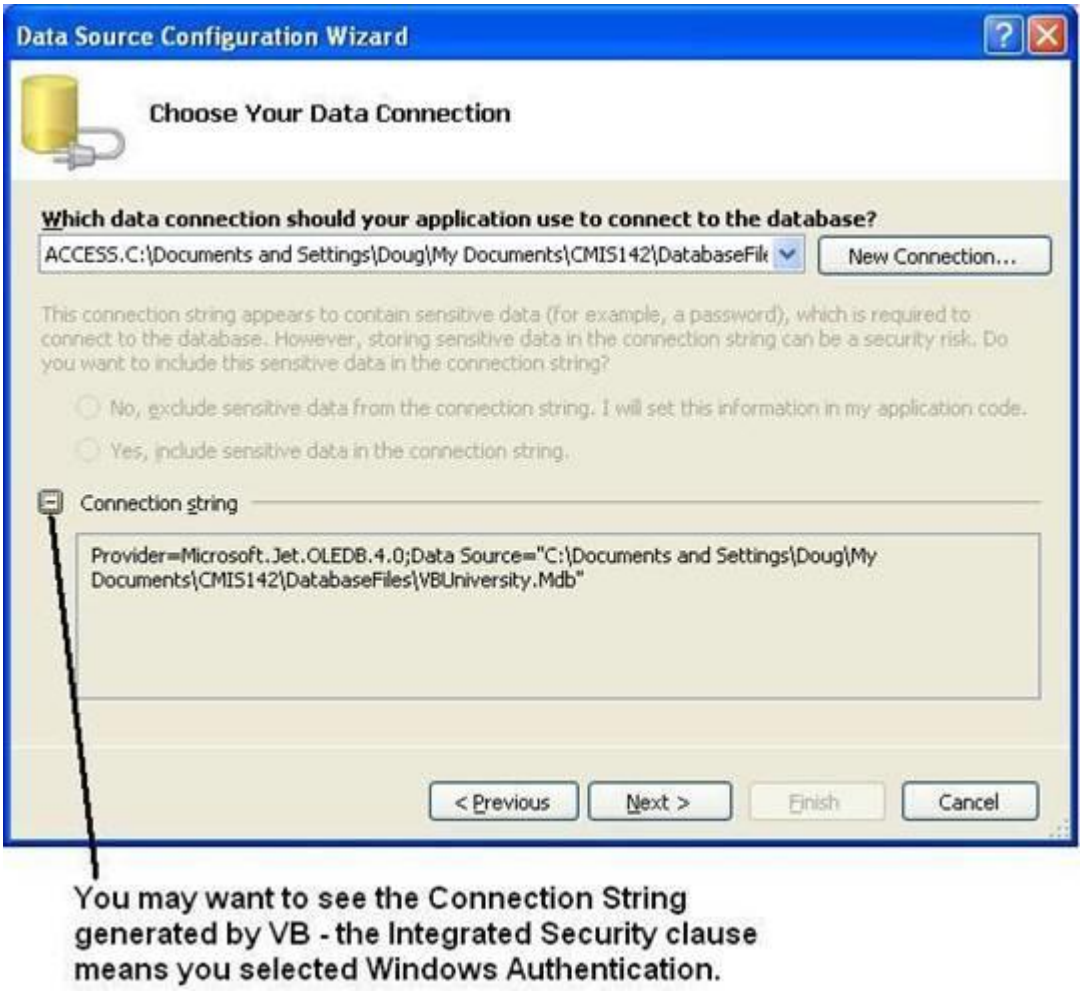


11b. If you are using Microsoft Access:

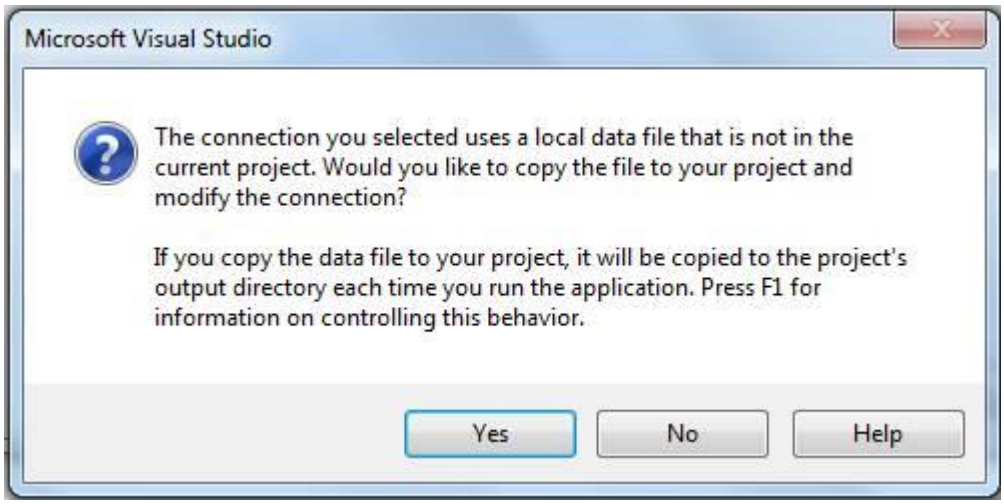
- The next figure shows the Microsoft Access **Add Connection** dialog box, click **Browse** – locate the **VBUniversity.mdb** file you earlier copied to the **My Documents** location – select it and click **Open** in the browse window – you will see the path and database file name added to the **Add Connection** dialog box as shown in the figure below.
- The database logon will use the **Admin** user – no password is necessary for a MS Access database.
- Click the **Test Connection** button – if the connection succeeds, you will see a popup window telling you that the **Test connection succeeded**. Click **OK** to close the popup and **OK** to close the **Add Connection** dialog box.



12. **Choose Your Data Connection** dialog box – you are returned to this window after selecting the database. Note that the name of the database file has been added to the window. This window shows an Access database file. Click **Next**.



13. Visual Studio now asks if you would like the database file (local data file) added to the project as shown in the Figure 10.13 below. Click **Yes**.

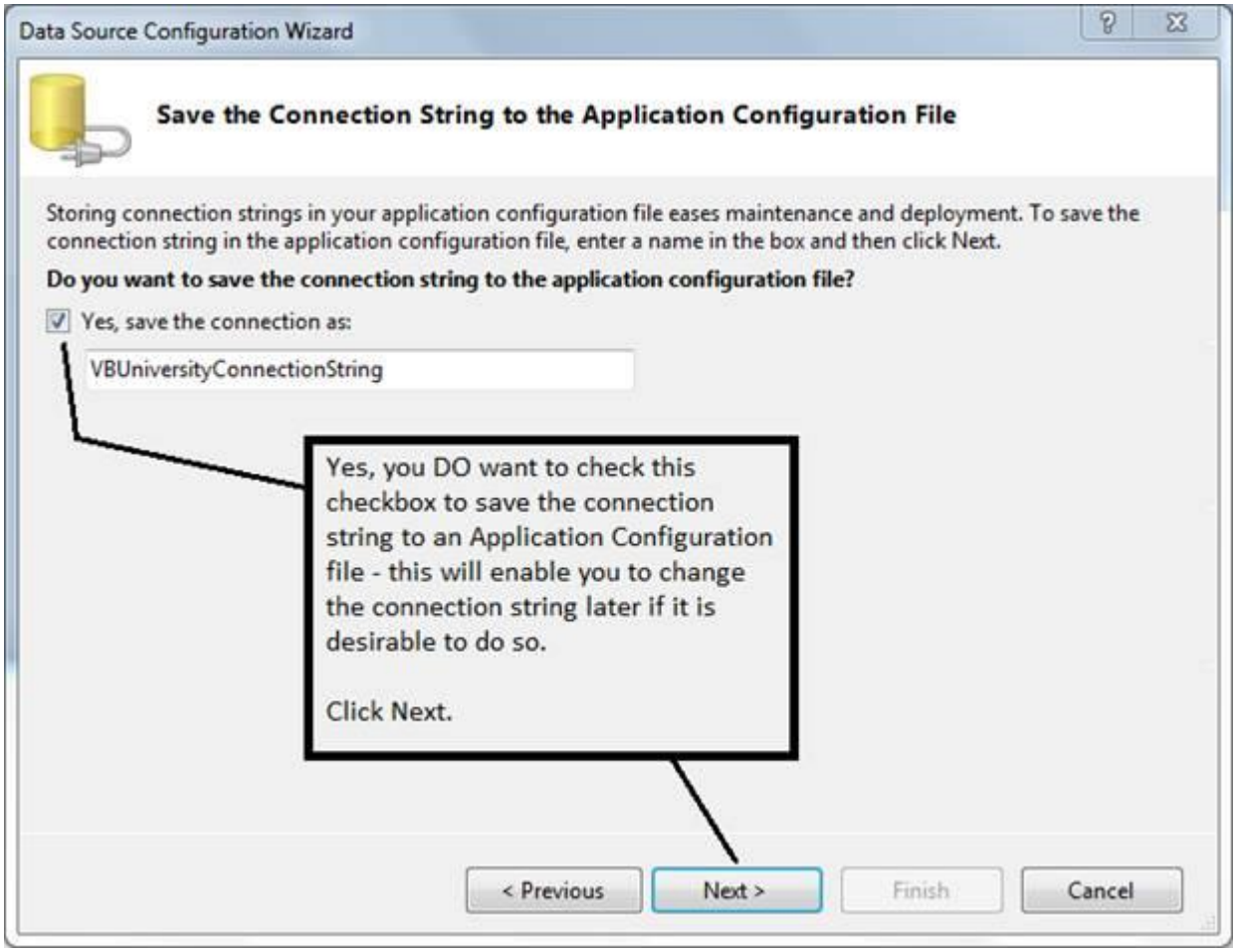


- Clicking **Yes** causes the database file to copy to the project folder's root directory making the project portable so you can take it back and forth to/from work/home/school – however, the

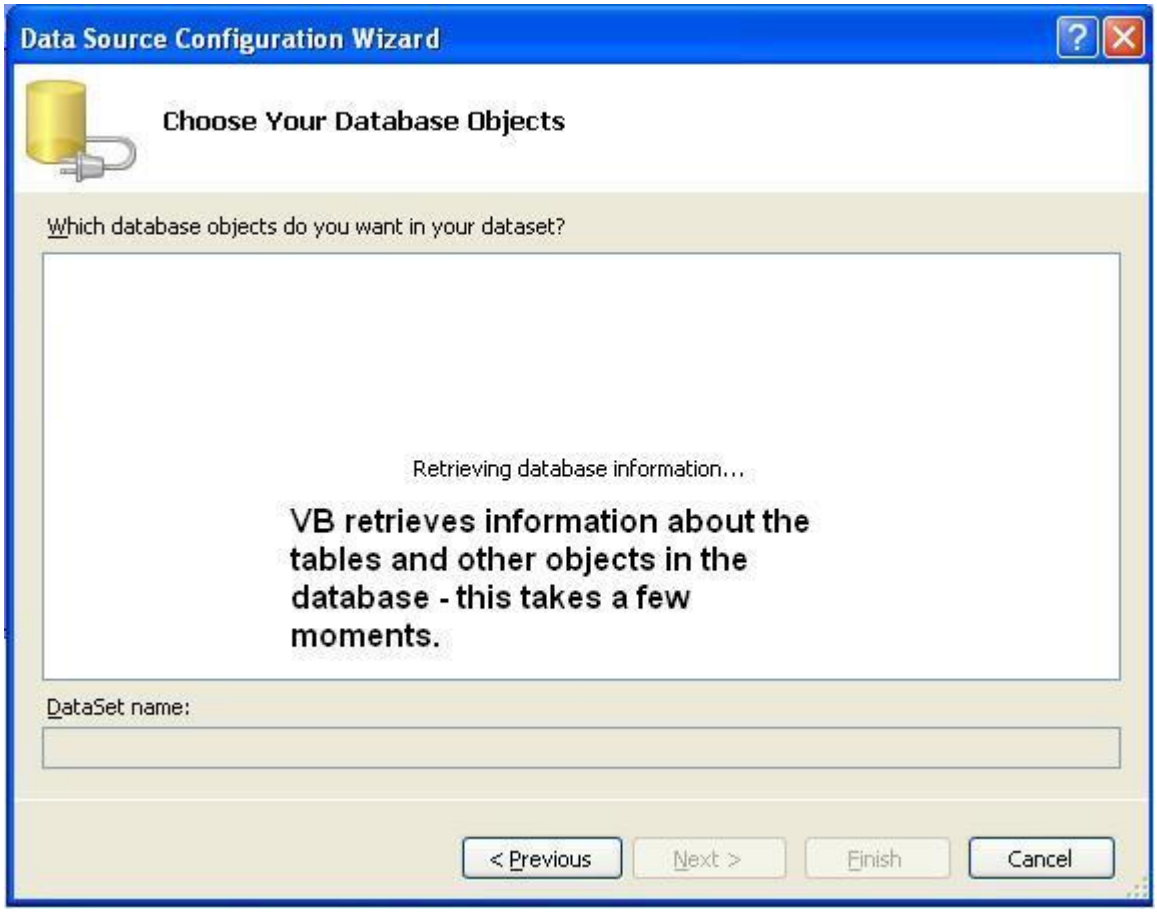
file is copied every time you run the project so any changes you make in terms of adding new rows, modifying existing rows, or deleting rows will not be made permanently to the copy of the file without making additional modifications to the properties of the database file – we will make those changes after configuring the data source.

- Clicking **No** causes the project to point (locate) the file based on the **ConnectionString** property setting in its original position – a copy of the database file is not made in the project.

14. **Save the Connection String to the Application Configuration File** – defaults to a selection of **Yes** – saving the connection string to a configuration file will enable you to change the string if necessary at a later point in time. Ensure the check box is **checked**, and then click **Next**.

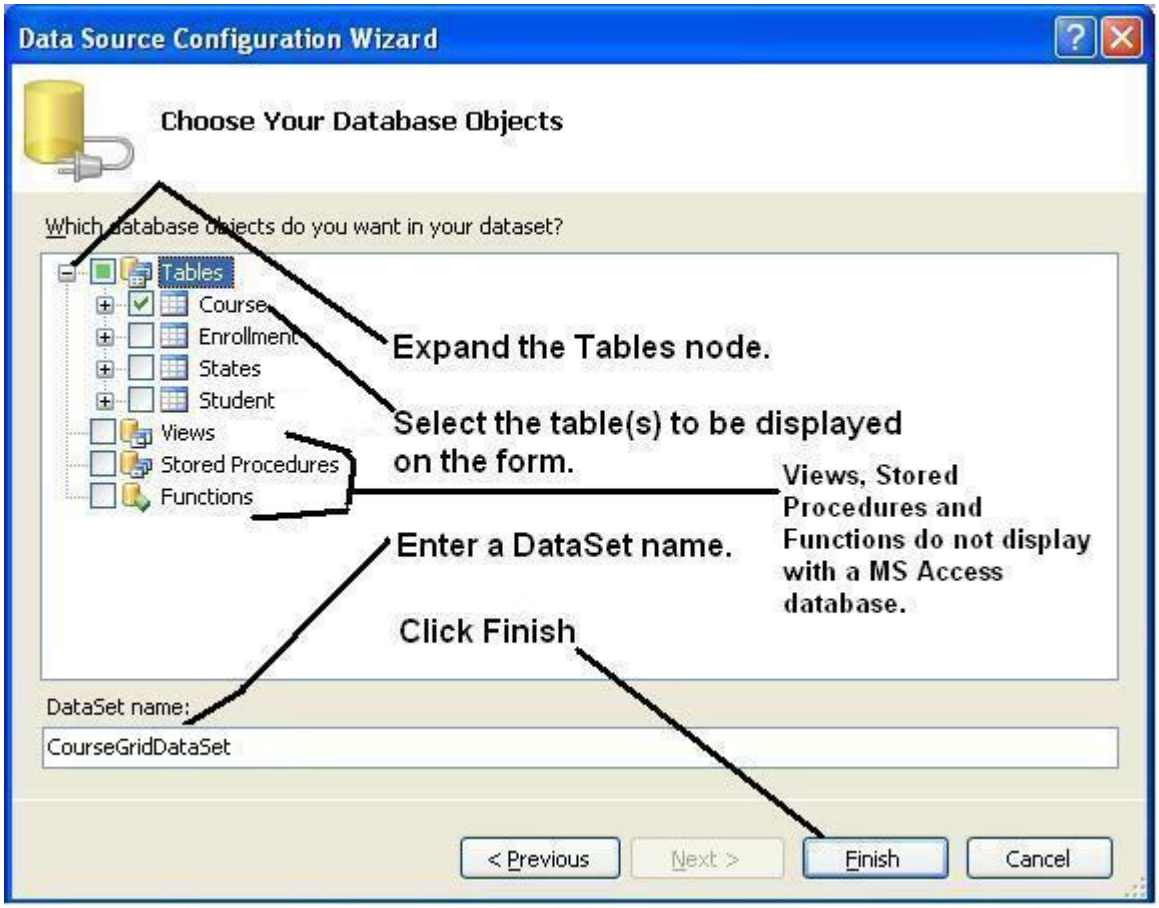


15. **Choose Your Database Objects** – VB retrieves information from the database file about the tables and other database objects available for your use. This takes a few moments as shown in this figure.

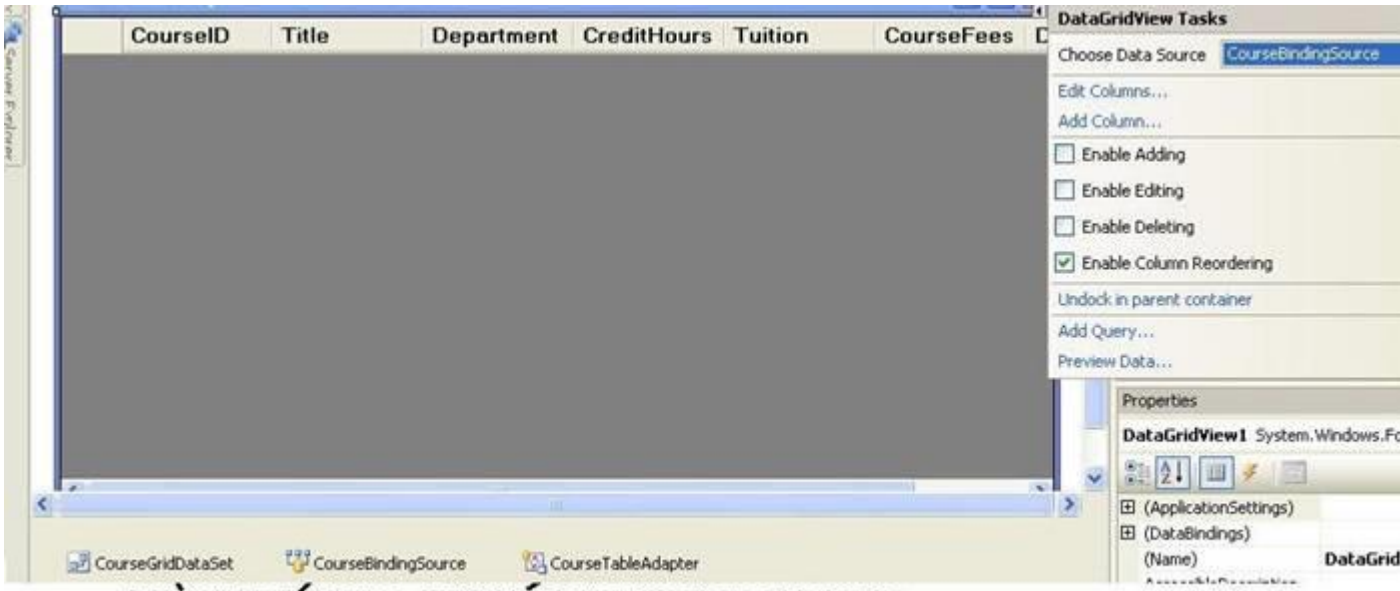


15 (continued). **Choose Your Database Objects** – you must specify the table(s) from which to retrieve the data to be data displayed by the DataGridView control.

- In this figure, the **Tables node** is expanded and the **Course** table is selected.
- Check the **Course** table checkbox – an alternative is to expand the Course table node and check just selected columns—not all columns need be selected if they are not needed by the application user.
- Change the dataset name generated by VB to **CourseGridDataSet** as shown.
- Click **Finish**.



16. After the wizard closes, the DataGridView control now has column names from the **Course** table as column headings. Also the system component tray displays **BindingSource**, **TableAdapter**, and **DataSet** objects with names assigned by VB.

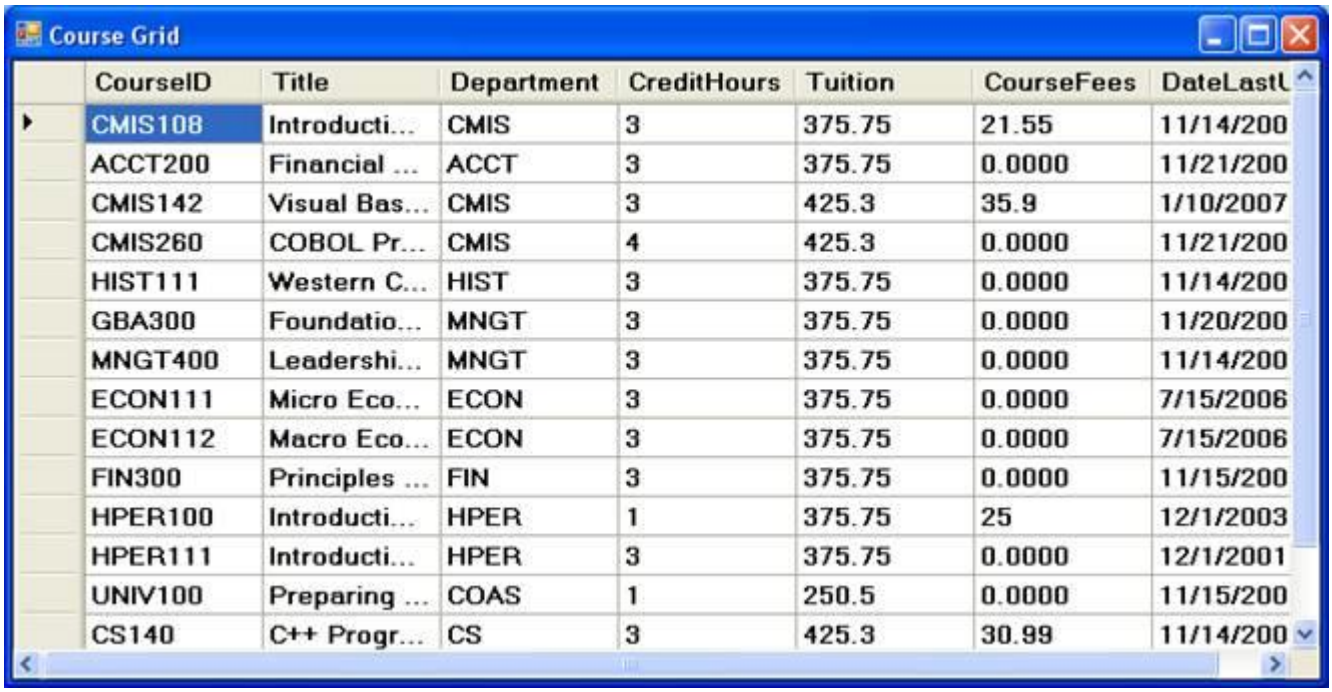


DataSet, BindingSource, and TableAdapter objects are added to the system component tray.

Test the Project

Start Debugging – the **CourseGrid** form will fill with data.

- The form will appear to be similar to the figure shown below.



CourseID	Title	Department	CreditHours	Tuition	CourseFees	DateLastU
CMIS108	Introducti...	CMIS	3	375.75	21.55	11/14/200
ACCT200	Financial ...	ACCT	3	375.75	0.0000	11/21/200
CMIS142	Visual Bas...	CMIS	3	425.3	35.9	1/10/2007
CMIS260	COBOL Pr...	CMIS	4	425.3	0.0000	11/21/200
HIST111	Western C...	HIST	3	375.75	0.0000	11/14/200
GBA300	Foundatio...	MNGT	3	375.75	0.0000	11/20/200
MNGT400	Leadershi...	MNGT	3	375.75	0.0000	11/14/200
ECON111	Micro Eco...	ECON	3	375.75	0.0000	7/15/2006
ECON112	Macro Eco...	ECON	3	375.75	0.0000	7/15/2006
FIN300	Principles ...	FIN	3	375.75	0.0000	11/15/200
HPER100	Introducti...	HPER	1	375.75	25	12/1/2003
HPER111	Introducti...	HPER	3	375.75	0.0000	12/1/2001
UNIV100	Preparing ...	COAS	1	250.5	0.0000	11/15/200
CS140	C++ Progr...	CS	3	425.3	30.99	11/14/200

- You can expand the form to fill the entire computer screen if desired.
- Column sizes can be modified by the application user.

IMPORTANT NOTE: If the form does not display any data and/or you get an error message **The Microsoft.Jet.OLEDB.4.0 provider is not registered on the local machine** shown in the figure below, refer to the section **Microsoft Access and Windows 7 Problems** provided earlier in these notes.

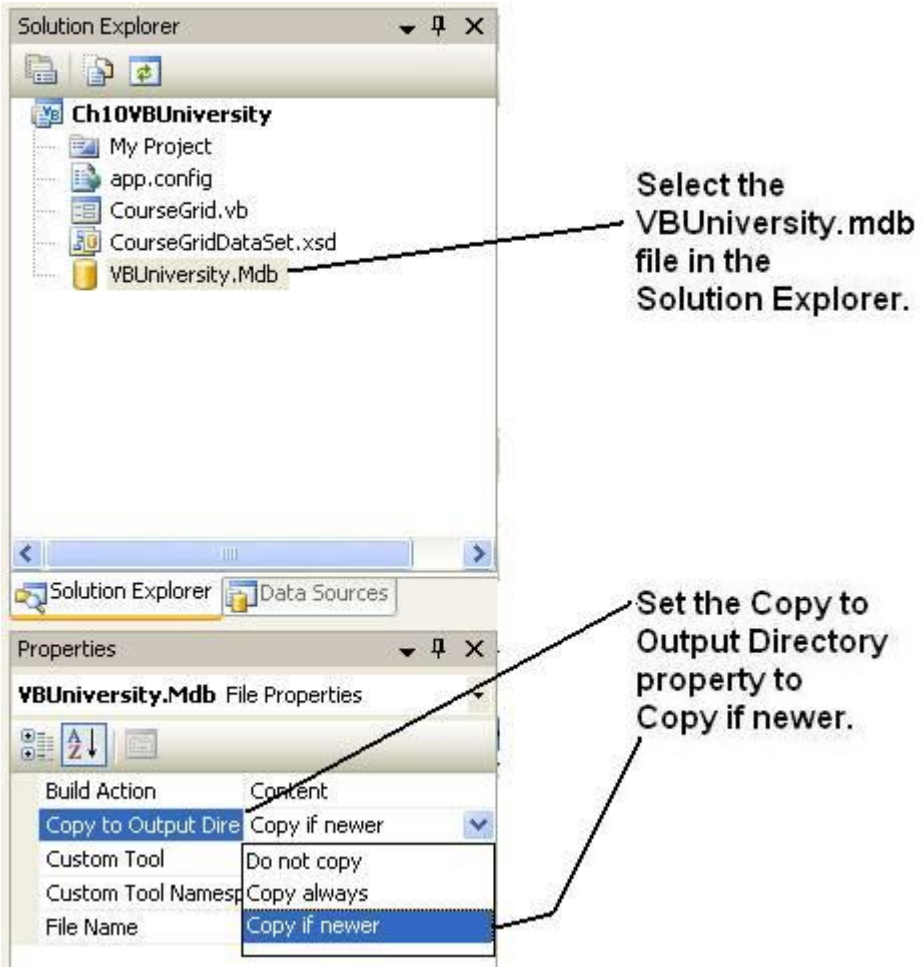
Changing the Database File's Properties

When the project runs, if you make changes to the data in the database, it is NOT saved with the current settings.

The **VBUniversity.mdb** (or **VBUniversity.mdf** file when using SQL Server) in the root folder of the project is copied (overwritten) to the **\bin\Debug** folder – it is this second copy of the database file that is modified during program execution.

Each time the program executes, the original root folder copy of the database is again copied to **\bin\Debug** folder – thus the changes made to the data are lost from any previous execution runs.

- **Copy to Output Directory** property setting – you must change this property of the **VBUniversity.mdb** or **VBUniversity.mdf** file as shown in the figure below.
- Change from **Copy always** to **Copy if newer** – when you make data row changes (inserts, edits, or deletes) during program execution, the changes will now be saved because the copy in the **\bin\Debug** folder will be the newest copy of the database file and the database file copy in the project root directory will not overwrite the database file copy in the **\bin\Debug** folder.



Coding the Course Form

VB automatically generates code for the form's **Load** event. The code is shown here.

- **Fill method** – the table adapter's **Fill** method executes the SQL statement (it is a SQL SELECT statement) that is stored in the table adapter. This fills the data set object with data from the **Course** table.
- VB generates the code automatically, but you can modify the code as necessary.
- Note that the keyword **Me** refers to the current form.

```
    Private Sub CourseGrid_Load(ByVal sender As Syste
m.Object, ByVal e As System.EventArgs) Handles MyBase
.Load
        'TODO: This line of code loads data into the
'CourseGridDataSet.Course' table. You can move, or
remove it, as needed.
```

```
        Me.CourseTableAdapter.Fill(Me.CourseGridDataS
et.Course)

    End Sub
```

- Modify the **Load** event – add a **Try-Catch** block to handle situations where a network connection fails.

```
    Private Sub CourseGrid_Load(ByVal sender As Syste
m.Object, ByVal e As System.EventArgs) Handles MyBase
.Load
        Try
            'Load Course table
            Me.CourseTableAdapter.Fill(Me.CourseGridD
ataSet.Course)
        Catch ex As Exception
            Dim MessageString As String = "Report
this error to the system administrator: " &
ControlChars.NewLine & ex.Message
            Dim TitleString As String = "Course Data
Load Failed"
            MessageBox.Show(MessageString,
TitleString, MessageBoxButtons.OK,
MessageBoxIcon.Error)
        End Try
    End Sub
```

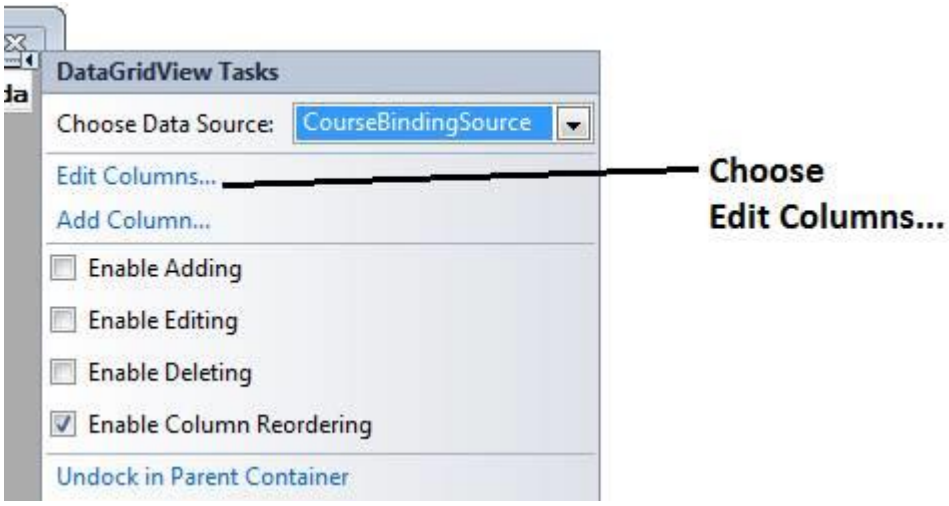
Formatting DataGridView Control Output

You can edit a DataGridView control in order to improve how data is displayed in its data columns, primarily numeric and date/time data columns.

Format Column Headings and Width

Follow these steps to format column headings and column width.

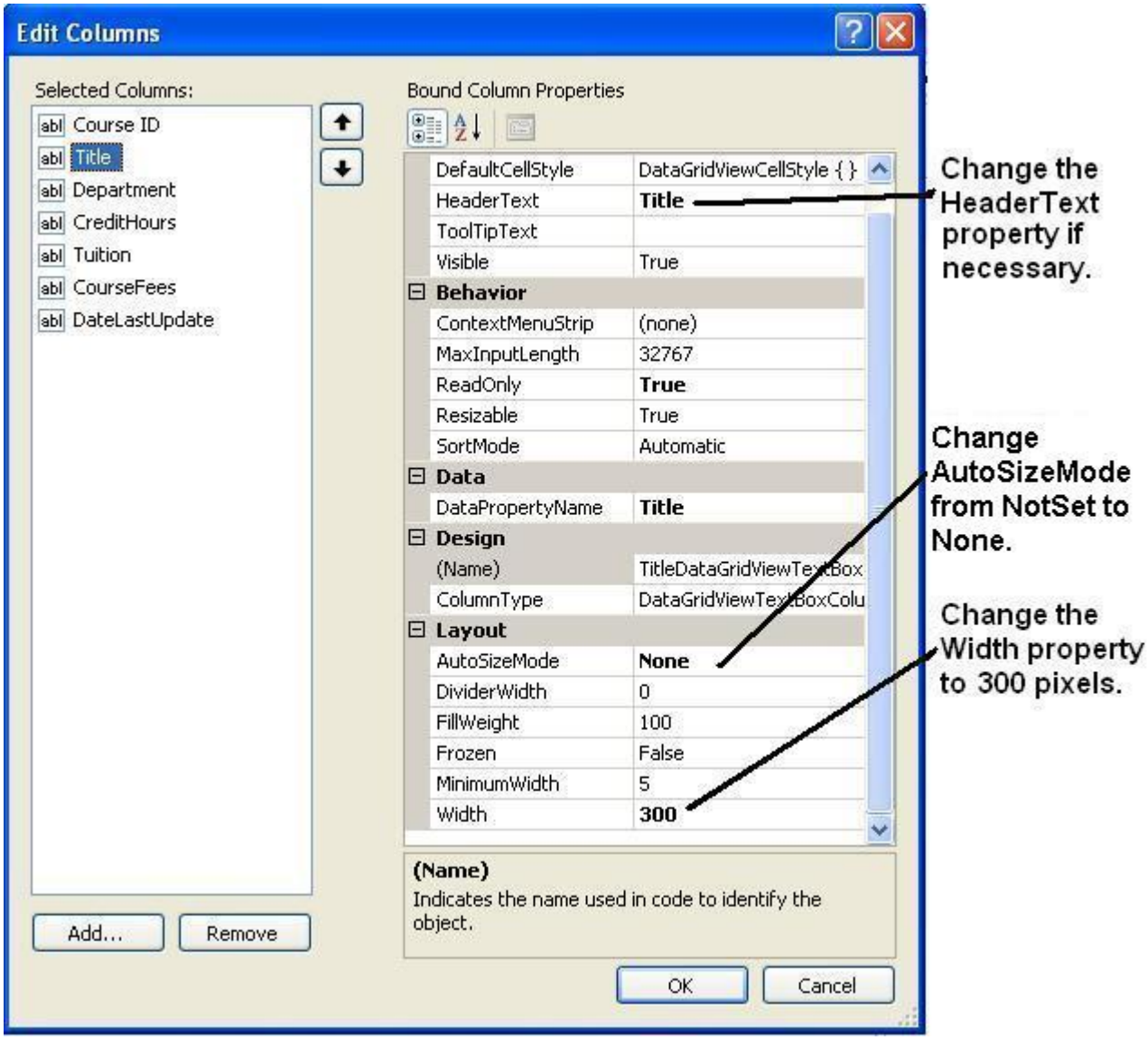
1. **Smart Tag** – click the DataGridView control's **Smart Tag** arrow to display the **DataGridView Tasks** window shown in the figure below – click the **Edit Columns** link as shown in this figure.



2. **Edit Columns** window – edit the properties indicated.

3. **CourseID** Column – set the **HeaderText** property = **Course ID** (added a blank space for readability). DO NOT CHANGE the **DataPropertyName** property by mistake.

4. **Title** Column.
   - **AutoSizeMode** property – change from **Not Set** to **None**.
   - **Width** property – change to **300** pixels – on startup the column will now display the entire **Title** column value.



5. Other Columns – modify the **HeaderText** property of other columns as indicated:

- CreditHours – **Credit Hours**
- CourseFees – **Course Fees**
- DateLastUpdate – **Date Last Updated**

6. Modify the **AutoSizeMode** from **Not Set** to **None** and **Width** property to **75** pixels for the **CreditHours** , **Tuition**, and **CourseFees** columns. Click **OK** to close the Edit Columns dialog box.

7. **AutoSizeColumnsMode** property – an alternative to setting each column's **Width** property is to set the DataGridView control's **AutoSizeColumnsMode** property to a value of **AllCells.** –

- This will cause all column sizes to vary to match the data displayed.
- Individual columns that have the **AutoSizeMode** property changed from **Not Set** to **None** and the **Width** property set to a specific width will retain the specified width – this will override the DataGridView control's AutoSizeColumnsMode property setting.
- Close the Edit Columns window. In the Properties window access the **AutoSizeColumnsMode** property of the **DataGridView** control and set the property to **AllCells**.

Format Numeric Data Display

To format the display of numeric columns such as the **Credit Hours**, **Tuition**, and **CourseFees**, open the **Edit Columns** dialog box again by using the smart arrow tag.

8. Select the **Credit Hours** column.
   - **DefaultCellStyle** property – click the dialog button (… button) for this property to open the **CellStyle Builder** window.
   - **Format** property – click dialog button (… button) for this property to display the **Format String Dialog** window – Select **Numeric** with **2 Decimal places**.
   - **Alignment** property in the **CellStyle Builder** window – choose the **MiddleRight** setting.
   - Click **OK** to close the window.

9. Repeat the steps for this column as you did in step 8 for the **Tuition** and **CourseFees** columns.

10. Click **OK** to close the **Edit Columns** dialog box. Test the project – make any additional changes needed in order to achieve a satisfactory display of data. Stop debugging when you finish testing.

Course Grid

Course ID	Title	Department	Credit Hours	Tuition	Course Fees	Da Up
CMIS108	Introduction to Information Systems	CMIS	3.00	375.75	21.55	11/
ACCT200	Financial Accounting for Nonmajors	ACCT	3.00	375.75	0.00	11/
CMIS142	Visual Basic Programming	CMIS	3.00	399.30	35.90	1/1
CMIS260	COBOL Programming	CMIS	4.00	425.30	0.00	11/
HIST111	Western Civilization	HIST	3.00	375.75	0.00	11/
GBA300	Foundations of Business	MNGT	3.00	375.75	0.00	11/
MNGT400	Leadership and Management	MNGT	3.00	375.75	0.00	11/
ECON111	Micro Economics	ECON	3.00	150.75	0.00	7/1
ECON112	Macro Economics	ECON	3.00	166.75	0.00	7/1
FIN300	Principles of Finance	FIN	3.00	375.75	0.00	11/
HPER100	Introduction to Recreation	HPER	1.00	125.55	25.00	12/
HPER111	Introduction to Physiology	HPER	3.00	375.75	0.00	12/
UNIV100	Preparing for Learning	COAS	1.00	125.55	0.00	11/
CS140	C++ Programming I	CS	3.00	425.30	30.99	11/

VB University Project – StudentGrid Form

This section provides you additional practice using the DataGridView control. Build a form like the one shown in this table to display student information by using a **DataGridView**control.

Student Grid

SSN	Last Name	First Name	Mi	Address	City	State	Zip	Telephone	Email
111-11-1111	Able	Andrea	A	100 S. Andrews St.	Edwardsville	IL	62025	6185551000	Aable1@siu
222-22-2222	Best	Bobby	B	15 Beauford Way	Alton	IL	63000	6185552000	Bbest1@siu
333-33-3333	Charming	Claudia	C	Clover Avenue North	Maryville	IL	610004400		Ccharm1@
888-88-8888	Headstrong	Henrietta	H	S. Howard Road, #8	Maryville	IL	610005600	6185558000	Hheads1@
999-99-9999	Indigo	Inga	I	5203 Icon Way	Alton	IL	63000	6185559000	Iindig1@siu
444-44-4444	Domingo	Donald	D	1804 Drewey Drive	Alton	IL	63000		Ddomin1@
555-55-5555	Easterwing	Edwardo	E	East South Road 18B	Edwardsville	IL	62025	6185555000	Eeast1@siu
666-66-6666	Freewaydrive	Fredericko	F	Four Points South #2	Edwardsville	IL	62025	6185556000	Ffree1@siu
777-77-7777	Greathouse	Greta	G	S. Grand St. #15	Alton	IL	63000	6185557000	Ggreat1@s
111-11-1112	Airhead	Arnold	A	101 S. Andrews	Edwardsville	IL	62025	6185551001	Aairhea1@
555-55-5554	Charming	Claude	C	Northwest Avenue	Alton	Il	63000		CCharm2@
111-11-1113	Headstrong	Harvey	H	105 Northwest St.	Edwardsville	IL	62025	(618) 555-1113	Hheads2@

1. Add a new form to the project. Access the **Project | Add Windows Form…** menu to add a new Windows form.

2. Name the form **StudentGrid.vb**.
   - Set the form's **Font** property = **9 point**.
   - Set the form's **Text** property = **Student Grid**.

3. Drag a **DataGridView** control to the form.

4. Click the **smart arrow tag** of the DataGridView control to display the DataGridView Tasks window.
   - **Dock** the DataGridView control to fill the entire form.
   - **Enable** column reordering, but **disable** adding, editing, and deleting data rows.

5. Choose a data source – use the **existing data connection** when the wizard displays the **Choose Your Data Connection** dialog box.
   - Work through the wizard. In the **Choose Your Database Objects** dialog box select the **Student** table.
   - Name the dataset **StudentGridDataSet**.
   - Click **Finish** – when the wizard closes, observe the new objects in the system component tray and the columns displayed in the DataGridView control.

6. Resize the form to make it large enough to display most of the data columns.

7. Open the coding window for the form and modify the **Load** event to add a **Try-Catch** block like the one shown here.

```
    Private Sub StudentGrid_Load(ByVal sender As Syst
em.Object, ByVal e As System.EventArgs) Handles MyBas
e.Load
        'Load the Student table
        Try
            Me.StudentTableAdapter.Fill(Me.StudentGri
dDataSet.Student)
        Catch ex As Exception
            Dim MessageString As String = "Report
this error to the system administrator: " &
ControlChars.NewLine & ex.Message
            Dim TitleString As String = "Student Data
Load Failed"
            MessageBox.Show(MessageString,
TitleString, MessageBoxButtons.OK,
MessageBoxIcon.Error)
```

```
        End Try
    End Sub
```

8. Set the DataGridView control's **AutoSizeColumnsMode** property to **AllCells**.

9. Click the smart arrow tag to open the **Edit Columns** link. Use the Edit Columns dialog box to:

- Set the **HeaderText** property – change selected columns to the values indicated:
  - StudentSSN – **SSN**
  - FirstName – **First Name**
  - LastName – **Last Name**
  - MiddleInitial – **Mi**
  - StateCode – **State**
  - ZipCode – **Zip**
  - PhoneNumber – **Telephone**
  - EmailAddress – **Email**
  - AccountBalance – **Account Balance**

- Reorder the first three columns to be SSN, Last Name, and First Name (Select **LastName** and use the Up/Down arrows).
- Set properties for the **AccountBalance** column:
  - **DefaultCellStyle** property – use this to format to display data as **currency** with **2** digits to the right of the decimal.
  - **Alignment** property of this column to **MiddleRight**.
  - **AutoSizeMode** from **Not Set** to **None**.
  - **Width** property – change to **100** pixels.

10. Test the project. In the Solution Explorer window:

- Click **My Project**.
- Select **StudentGrid** as the Startup form from the dropdown selection.
- Close **My Project**.
- Run the project. Check the display of all column data. Adjust any column widths as necessary with the individual column's **AutoSizeMode** and **Width** properties in the Edit Columns window.
- Note that the zip code and telephone number display formatting is not satisfactory – this can be corrected, but only by purchasing a 3rd party add-on program to enable

using **MaskedTextBox** columns that can be edited, or by writing quite a bit of code to edit the column. You will learn to edit telephone numbers for individual control forms later in these notes.

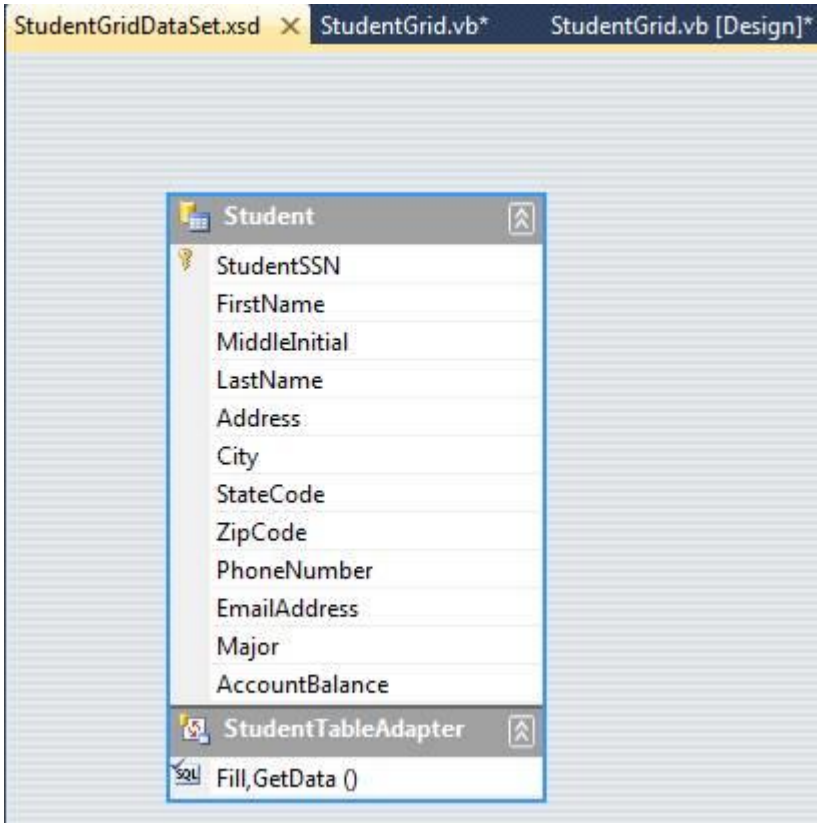
Project Special Files

XML Schema File

Each data source added to a project generates an XML (extensible markup language) schema file with file extension of **.xsd**.

This figure shows the **.xsd** schema file for the **StudentGrid** form in the **Solution Explorer** window – double-clicking the file will display the schema in visual format. The visual format shows:

- the primary key column(s).
- other column names.
- multiple tables and their relationships when more than one table has data displayed.
- the table adapter (**StudentTableAdapter** at the bottom of the Student image) – selecting this displays properties that include the **CommandText** property that stores the SQL statement used to select data from the Student table.



XML is an industry-standard format used to store and transfer data.

You do not need to know how to write XML to use VB to program database applications. VB will automatically generate any necessary XML. It does help to understand basic facts about XML.

Proprietary database formats typically store data in **binary format** – proprietary data cannot be processed by other systems or pass through **Internet firewalls**.

XML data is stored as **plain text** identified by tags like **HTML tags**. You can edit an XML file with a plain text editor such as Notepad, and the data can be transferred through Internet firewalls.

An **XML schema file** can also be used to describe fields (columns), data types, and any constraints such as fields that are required. Each VB.NET project that connects to a database has an XML schema file. The data in the XML data file is validated against the data definitions in the XML schema file.

App.Config File

Recall that earlier you directed the system to generate a configuration file. The code shown below is that generated and stored in the **App.Config** file found in the **Solution Explorer**window.

The code is written by VB in the **XML** (extensible markup language) format. You can modify the application confirmation as necessary by altering this XML code. For example, you might need to modify the connection string information (highlighted below in yellow) to a new data source when you port your application from a testing to production environment.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
    </configSections>
    <connectionStrings>
        <add name="Ch10VBUniversity.My.MySettings.VBU
niversityConnectionString"
            connectionString="Provider=Microsoft.Jet.
OLEDB.4.0;Data
Source=|DataDirectory|\VBUniversity.Mdb"
            providerName="System.Data.OleDb" />
    </connectionStrings>
    <system.diagnostics>
        <sources>
            <!-- This section defines the logging
configuration for My.Application.Log -->
            <source name="DefaultSource" switchName="
DefaultSwitch">
                <listeners>
                    <add name="FileLog"/>
                    <!-- Uncomment the below section
to write to the Application Event Log -->
                    <!--<add name="EventLog"/>-->
                </listeners>
            </source>
        </sources>
        <switches>
            <add name="DefaultSwitch" value="Informat
ion" />
        </switches>
        <sharedListeners>
            <add name="FileLog"
```

```
                    type="Microsoft.VisualBasic.Logging.
FileLogTraceListener, Microsoft.VisualBasic,
Version=8.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a,
processorArchitecture=MSIL"
                    initializeData="FileLogWriter"/>
                <!-- Uncomment the below section and
replace APPLICATION_NAME with the name of your
application to write to the Application Event Log -->
                <!--<add name="EventLog"
type="System.Diagnostics.EventLogTraceListener"
initializeData="APPLICATION_NAME"/> -->
            </sharedListeners>
        </system.diagnostics>
</configuration>
```

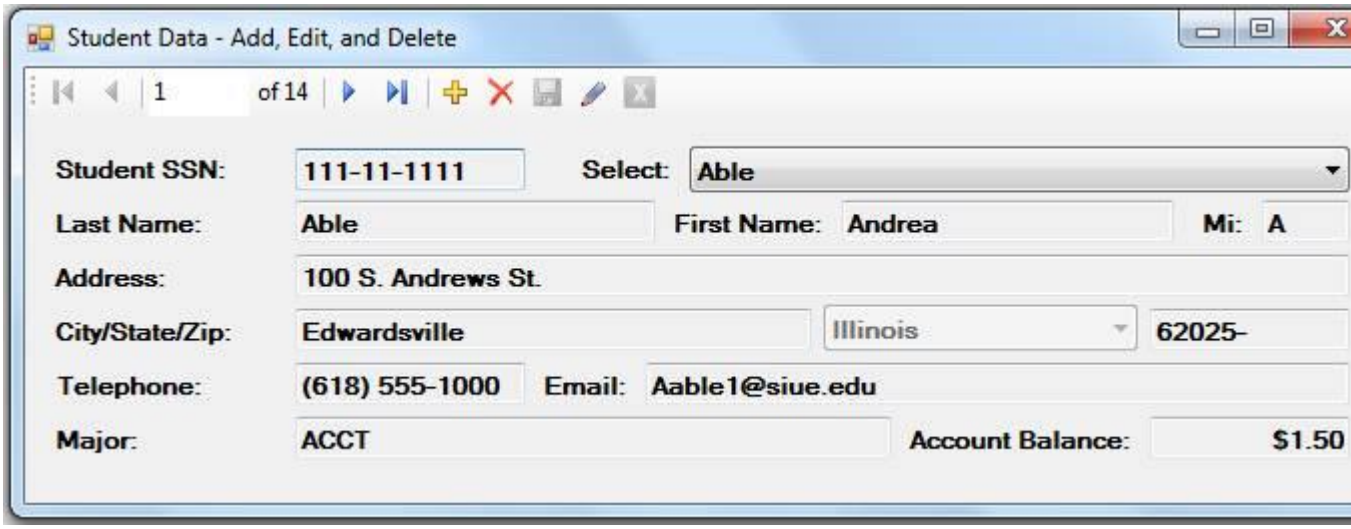
VB University Project – Individual Data Fields – StudentDetails Form

Binding Data to Other Controls

Another way to display data is with controls such as Label, TextBox and ComboBox controls – in this design approach, the controls are termed **bound controls**.

- The form can be designed through use of the **Data Sources** window – set the data source to display and drag the data table to the form.
- **Data Sources** window – access by selecting the **Data** menu, **Show Data Sources** menu item or from the **Data Sources** tab in the **Solution Explorer**.
- Can be used to add a new data source to a project.

You will learn how to build a third form for the project like the one shown here.



Add a New Form

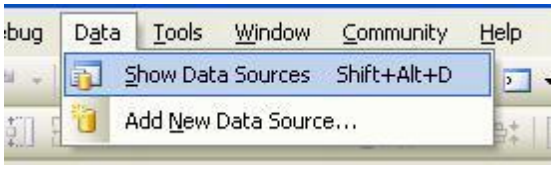
1. **Project menu**, **Add Windows Form** option – click to add a new form named **StudentDetails.vb**.

2. **Font Size** and **Bold** properties – set as desired (the figure has a **9 point** font with **Bold** = **True**).

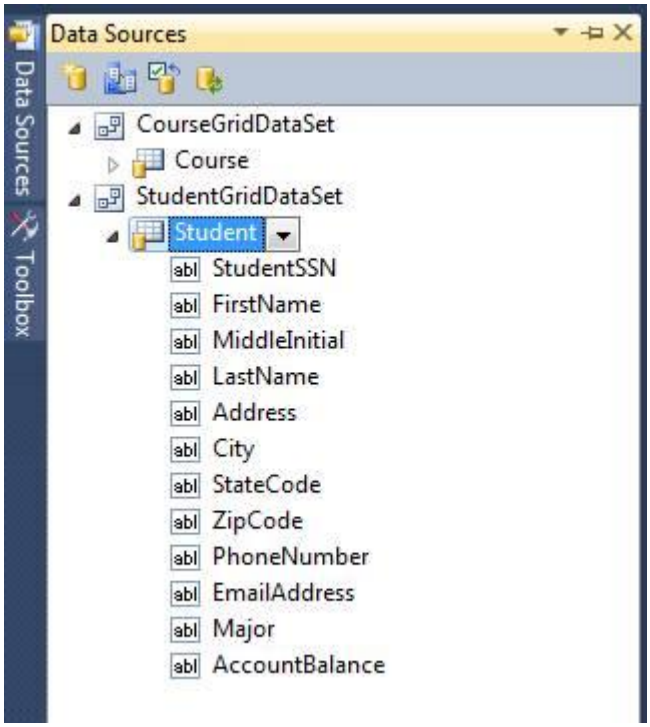
3. **Text** property – set to **Student Data – Add, Edit, and Delete**.

Add Data Source

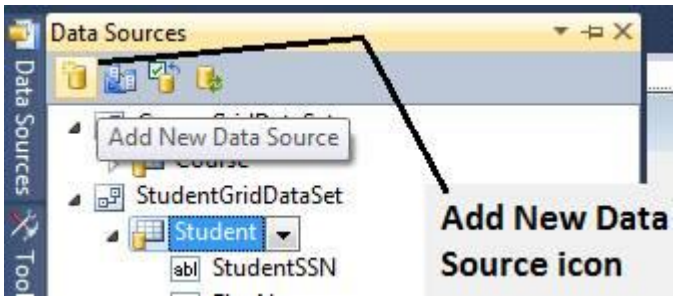
4. Select **Data** menu, **Show Data Sources** menu item as in the figure below (you can also use the Server Explorer window to add a new data source).



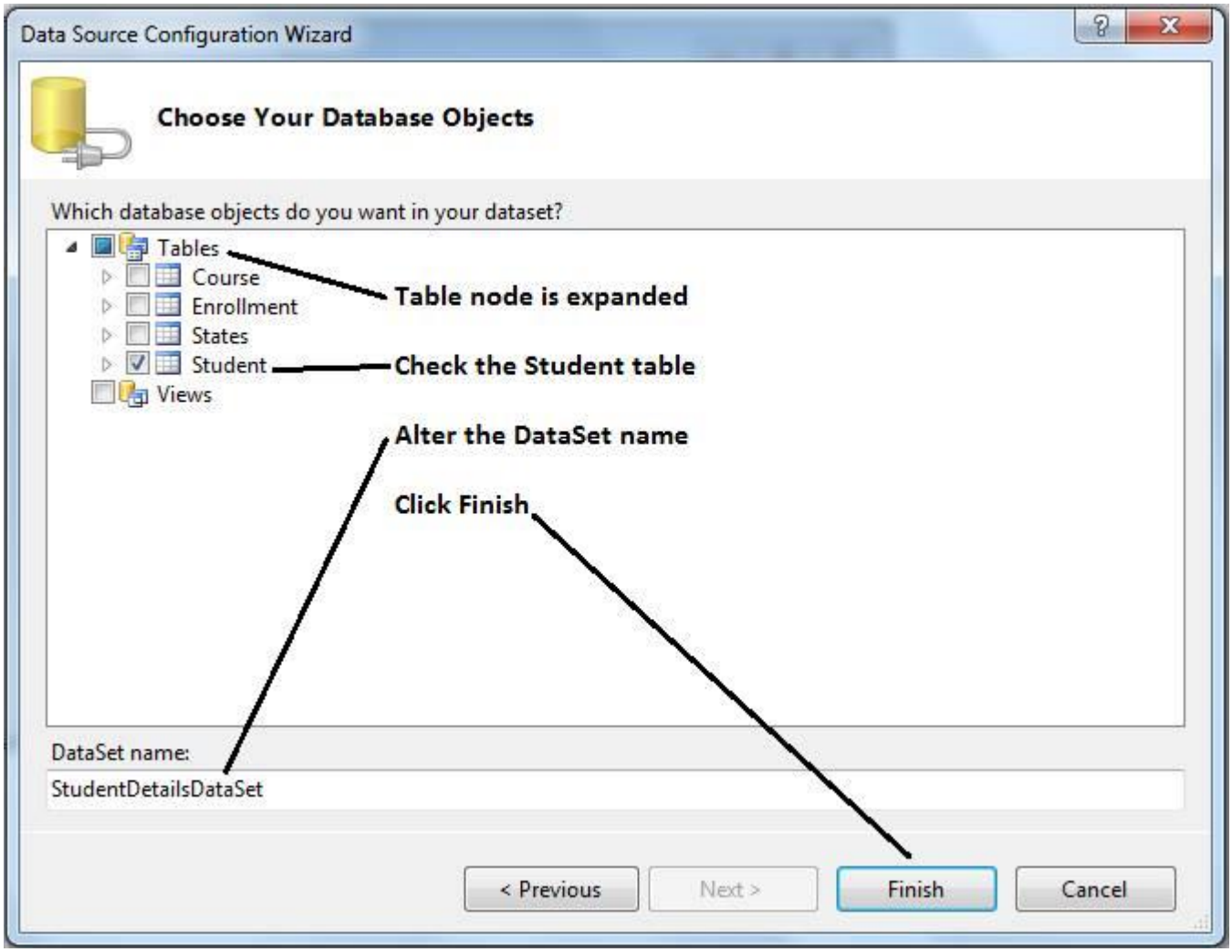
- You should see the existing **CourseGridDataSet** and **StudentGridDataSet** data sources.
- For existing data sources, the dataset(s) will display in the **Data Sources** window as in this figure along with any tables or views stored in the dataset.



5. **Add New Data Source** – add a new data source for this form.
   - Click the icon shown in this figure (or use the **Data** menu, **Add New Data Source** menu option).
   - This causes the **Data Source Configuration** wizard to display.



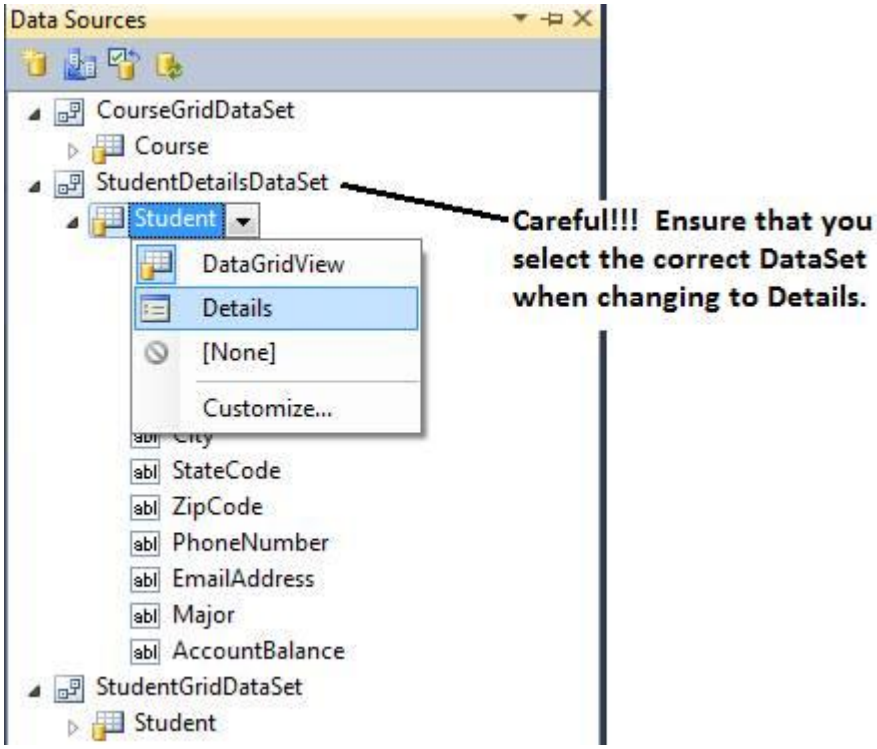
6. **Data Source Configuration** wizard:
   - Select the **Database** icon – click **Next**.
   - Choose a Database Model dialog box – click **Dataset** and then **Next**.
   - Choose Your Data Connection dialog box – click **Next** to use the existing connection.
   - Choose Your Database Objects dialog box – expand the **Tables** node as shown in the figure below – checkmark the **Student** table – name the dataset object **StudentDetailsDataSet** – click **Finish**.



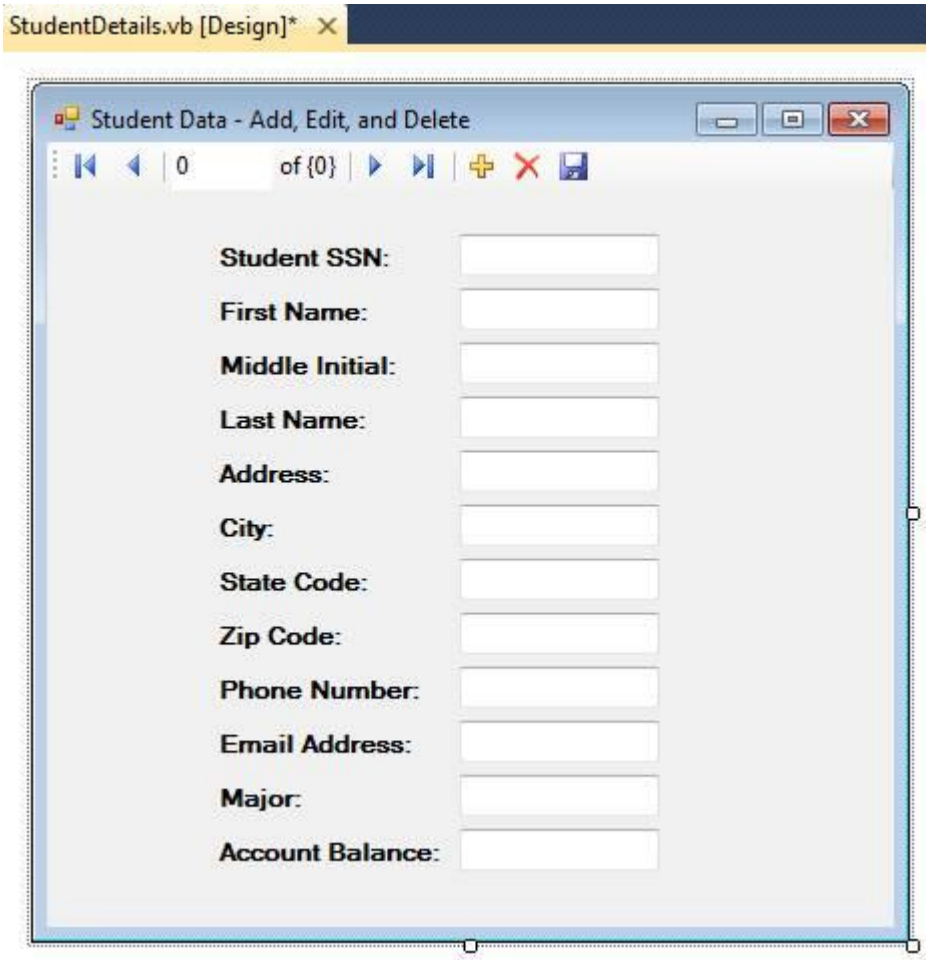
Adding Bound Controls to the Form

You are ready to add bound controls to the form. You will use a drag/drop approach to build the form – this is really fast and easy to use. If you mess it up, just delete the form and start over.

7. **Data Sources** window –
   - Expand the **StudentDetailsDataSet**.
   - Single-click on the **Student** table name as shown in the figure below.
   - Select the drop-down arrow by the **Student** table and select **Details** (if the drop-down arrow does not display, ensure that the form view for design mode is displayed – not the coding view).
   - Note that the **icon** for a **Details** view is different than that for a **DataGridView**.



8. Point at the **Student** table name in the **Data Sources** window – **drag/drop** the table to the **StudentDetails** form to a position about an inch from the top and left margins of the form – the form will automatically have Label and TextBox controls generated and displayed for each column in the **Student** table as shown in this figure.



Note the following:

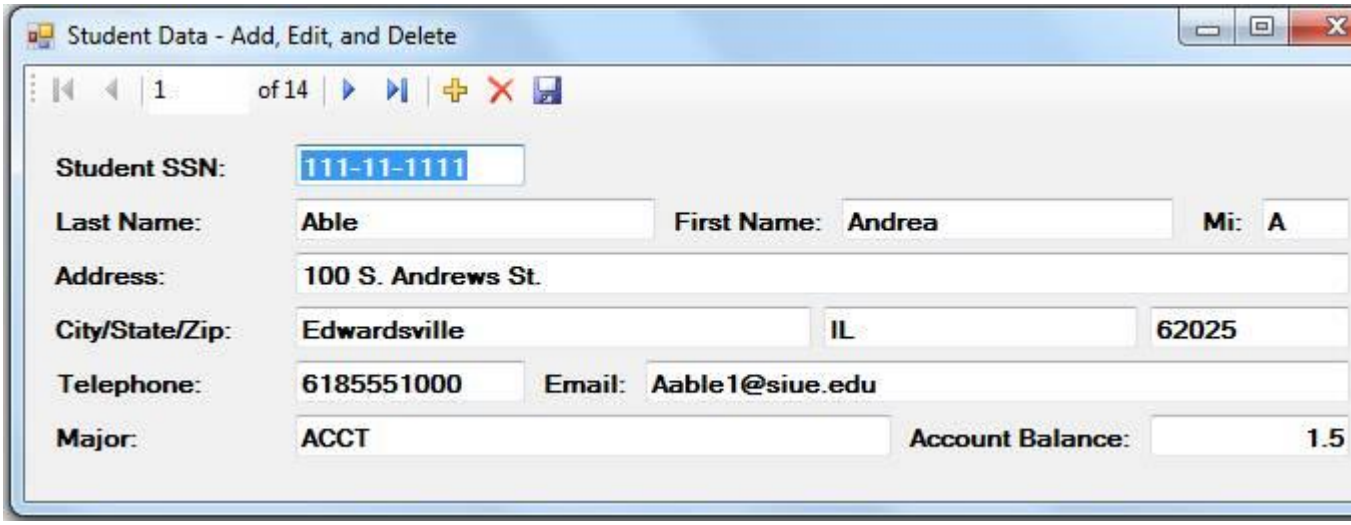
- The labels have the **Text** property setting generated automatically by VB – the column names are automatically divided into prompts appropriate for the form.
  - A column named **StateCode** will generate a label prompt of **State Code:**.
  - A column named **Email_Address** will generate a label prompt of **Email Address:**.
- Note the new components that were automatically added to the system component tray:
  - StudentDetailsDataSet
  - StudentBindingSource
  - StudentTableAdapter
  - TableAdapterManager
  - StudentBindingNavigator – corresponds to the **StudentBindingNavigator** control that is automatically added across the top of the form. It has the following features:

1. **Student record (row) counter** (1 of 14 in the figure).
2. **Move buttons** (Move first and Move previous are grayed out in the figure) such as the Move next button highlighted in the figure next to the Move last button.
3. **Add**, **Delete**, and **Save button icons**.

- The form will run and execute, but the layout is not very "user-friendly."

9. Alter the layout by drag/drop the controls on the form as shown in the figure below.

- Adjust the size of the controls as appropriate as shown here.
- Reorder the name to display Last Name, then First Name, then Middle Initial.
- Change the labels as shown on the figure for the Middle Initial (to M), City, State Code, and Zip Code (to City/State/Zip) and Phone Number (to Phone).
- **Tab order** – reset to reflect the new arrangement to tab from left-to-right, top-to-bottom.
- **AccountBalanceTextBox** control – set **TextAlign** property to **Right**.



Set the Startup Form/Test the Project

Change the startup form to the **StudentDetails** form.

- Open the **My Project** object in the **Solution Explorer** window.

- In the **Application** tab change the **Startup Form** property setting to the **StudentDetails** form.

Test the project.

- Note the telephone textbox and account balance textbox controls do not display formatted.
- Navigation buttons – test the ability to navigate from row to row. Note the display of data as you navigate from row to row
- The form enables you to alter the dataset in memory, and will save changes to the database.
  - Practice saving a row of data – click the **Add** button, enter the data, and click the **Save** button.
  - Shutdown and then run the project again – locate the new data row.
- The BindingNavigator also has some potential problems.
  - Try adding a duplicate row (with **Student SSN** = **111-11-1111**) – note that the system generates an **ConstraintException was unhandled** error message when the **Save**button is clicked.
  - Try to delete a data row (with **Student SSN** = **111-11-1111**) – note that the system generates an **OleDBException was unhandled** error message when the **Save** button is clicked – this is because the **Student** table data row is related to data row(s) in the **Enrollment** table (the exception generated depends on whether you are using a Microsoft Access or Sql Server database).
- These problems are addressed later in this note set by writing additional program code.

VB University Project – Individual Data Fields Coding

Form Load Event

The code generated by VB for the StudentGrid form's **Load** event needs to be modified by adding a Try-Catch block to catch exceptions as was done earlier for the Student form.

```
Private Sub StudentDetails_Load(ByVal sender As Syste
m.Object, ByVal e As System.EventArgs) Handles MyBase
.Load
    'Trap exceptions that occur during data load
```

```
    Try
        Me.StudentTableAdapter.Fill(Me.StudentDetails
DataSet.Student)
    Catch ex As Exception
        Dim MessageString As String = "Report this
error to the system administrator: " &
ControlChars.NewLine & ex.Message
        Dim TitleString As String = "Student Details
Data Load Failed"
        MessageBox.Show(MessageString, TitleString,
MessageBoxButtons.OK, MessageBoxIcon.Error)
    End Try
End Sub
```

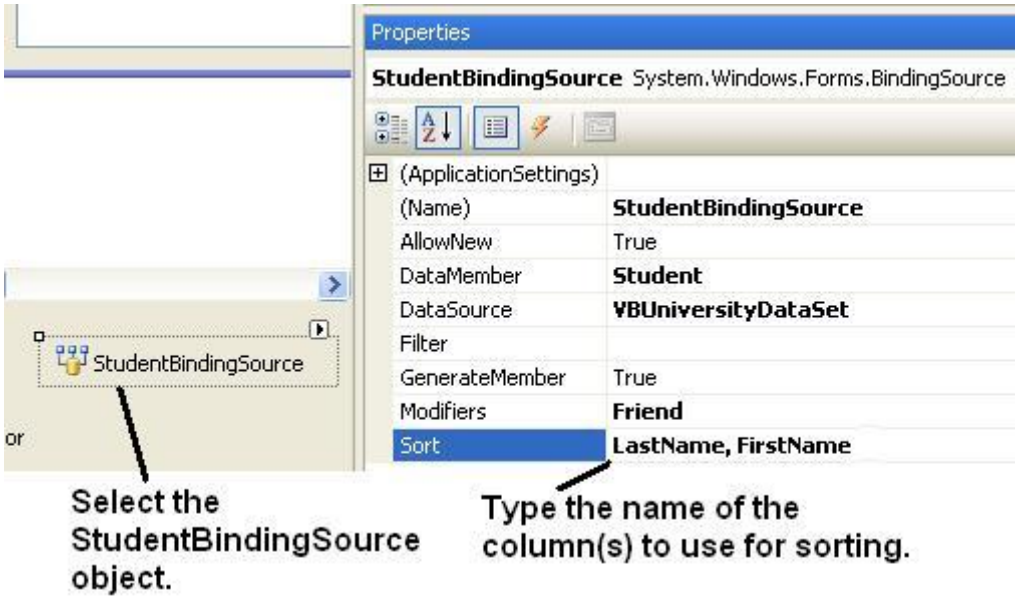
Sorting Data

The data displayed in the form is not sorted.

There are several ways to change the sorted display of data. This describes one approach.

1. Select the **StudentBindingSource** object.

2. In the **Properties** window select the **Sort** property – type the name of the column or columns to be used to sort data in the dataset – the column name(s) entered must **EXACTLY**match those in the dataset.
   - The figure below shows sorting by **LastName** and then **FirstName** column name values.
   - Enter the property value as **LastName, FirstName** – do not forget the comma between the column names.



3. Run the project and confirm that the data is sorted by last name and then first name within last name.

Formatting String Data – Using a MaskedTextBox – Manually Setting Data Binding

You can use a **MaskedTextBox** control to format string output such as a telephone number, zip code, and social security number values.

Format Telephone Number

1. Delete the **PhoneNumberTextBox** control and replace it with a **MaskedTextBox** control.

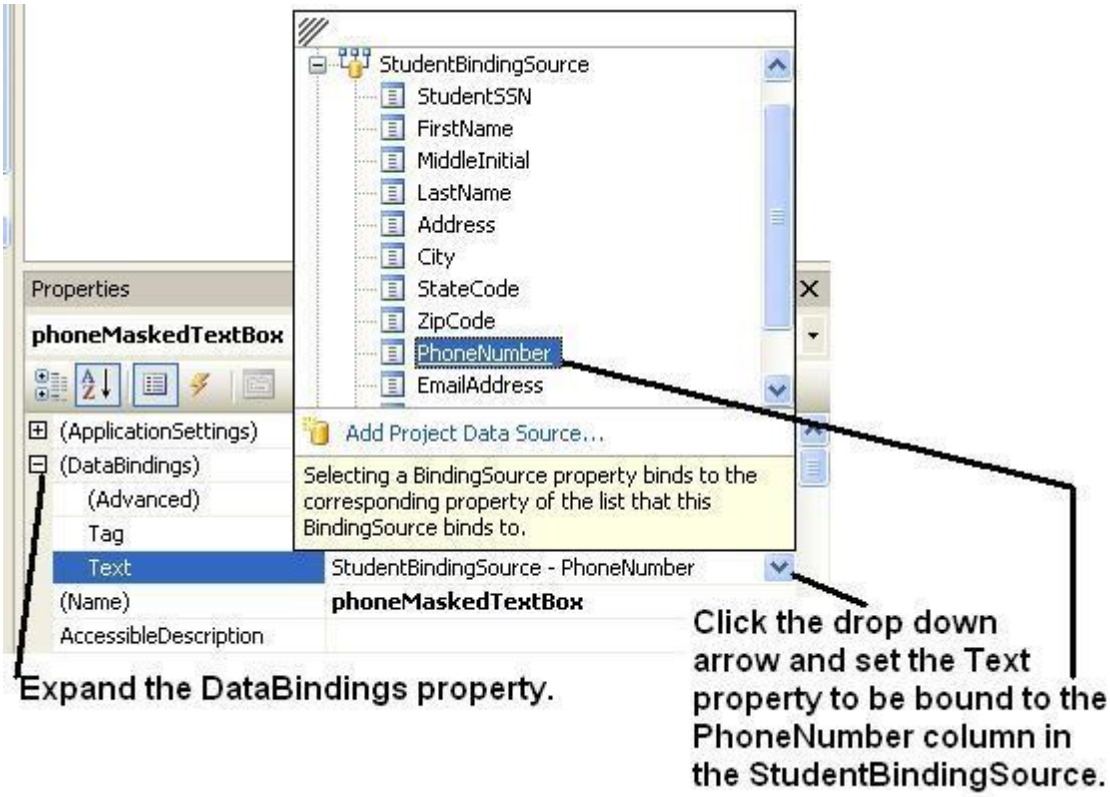
2. Name the control **PhoneMaskedTextBox**.

3. Set the **Mask** property by opening the **Input Mask** window (click the **…** dialog box button) and select **Phone number** for a mask.

   The new MaskedTextBox will not display any data because the control is not bound to the DataSource.

4. Select the **MaskedTextBox** control –
   - Expand the **DataBindings** property.
   - Select the **Text** property – Use the drop-down arrow for this property to select the **StudentBindingSource**, and then select

the **PhoneNumber** column – this will bind the **Text**property so that it will display data from the **PhoneNumber** column of the **StudentBindingSource** object to the MaskedTextBox control as shown in the figure below.



5. Run the project.
   - The telephone numbers should display with the mask.
   - Navigate from row to row with the buttons and note how the form appears when the student has no telephone number recorded in the database.

Format Student SSN and Zip Code

1. Replace the **student SSN** and the **zip code** TextBox controls with MaskedTextBox controls.
   - Name the controls
     - **SSNMaskedTextBox.**
     - **ZipMaskedTextBox**.

2. Set the **Mask** property option for each control.

3. Bind the **DataBindings-Text** property of the controls to the appropriate fields from the **StudentBindingSource** object.

4. Reset the **tab order** for the form.

5. Test the application.

Formatting Numeric or Currency (Money) Display – Format Account Balance

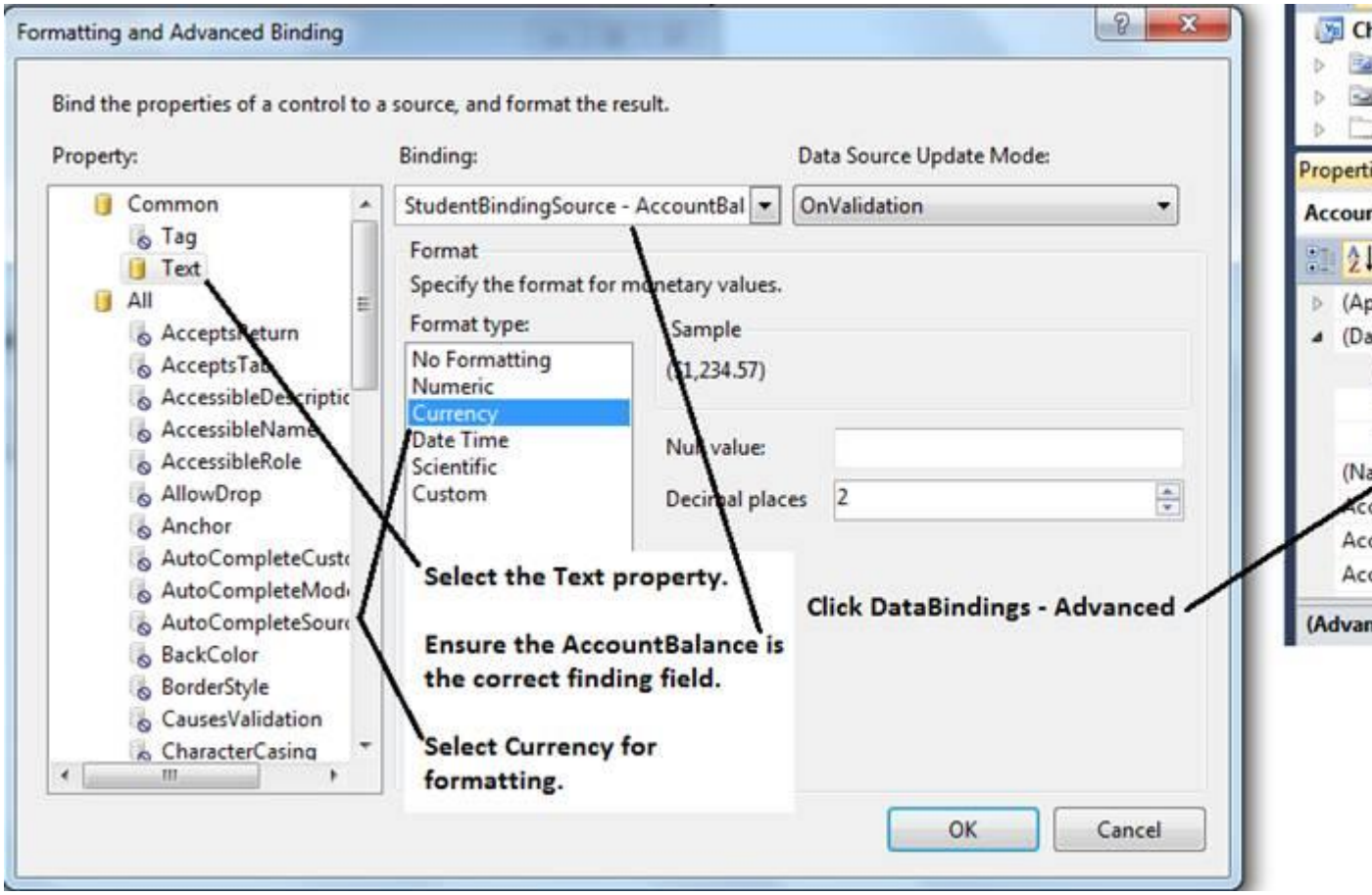
Some tables have data columns that represent numeric output such as the student **AccountBalance** column.

- SQL Server stores the data with four digits to the right of the decimal point so simply displaying the data to a TextBox results in a display that application users may find confusing.
- Neither SQL Server nor MS Access displays numeric data formatted as currency or numeric with appropriate currency symbols, commas, and the correct number of digits to the right of the decimal point.

1. Select **AccountBalanceTextBox** control.

2. **DataBindings-Advanced** property – expand and select this property of the **AccountBalanceTextBox** control to displays numeric data to open the **Formatting and Advanced Binding** dialog box shown in the figure below.

- **Text** property – select this property.
- **Binding** – ensure the correct database table field is bound – here the **AccountBalance** is bound.
- **Format type** – select either currency, numeric, or customize the formatting.
- **Decimal places** – usually select 2 decimal places.
- Click **OK** to save the formatting and advanced binding.



Data Row Edit Operations

This section explains **Data Row Edit Operations** with a **BindingNavigator** control.

- The default configuration enables an application user to modify any data row, but does not provide a means to enforce saving changes when they are made.
- The **Save** button provided by default will save changes, but there is no way to ensure the application user clicks **Save** other than through the Form's **Closing** event.
- Using the Form's **Closing** event to save changes can result in an attempt to save multiple changes at the same time, but some of these may violate **database integrity rules**, such as the length of a data value or the type of data to be saved from a TextBox or other bound control. This can cause the save event code to ABEND.

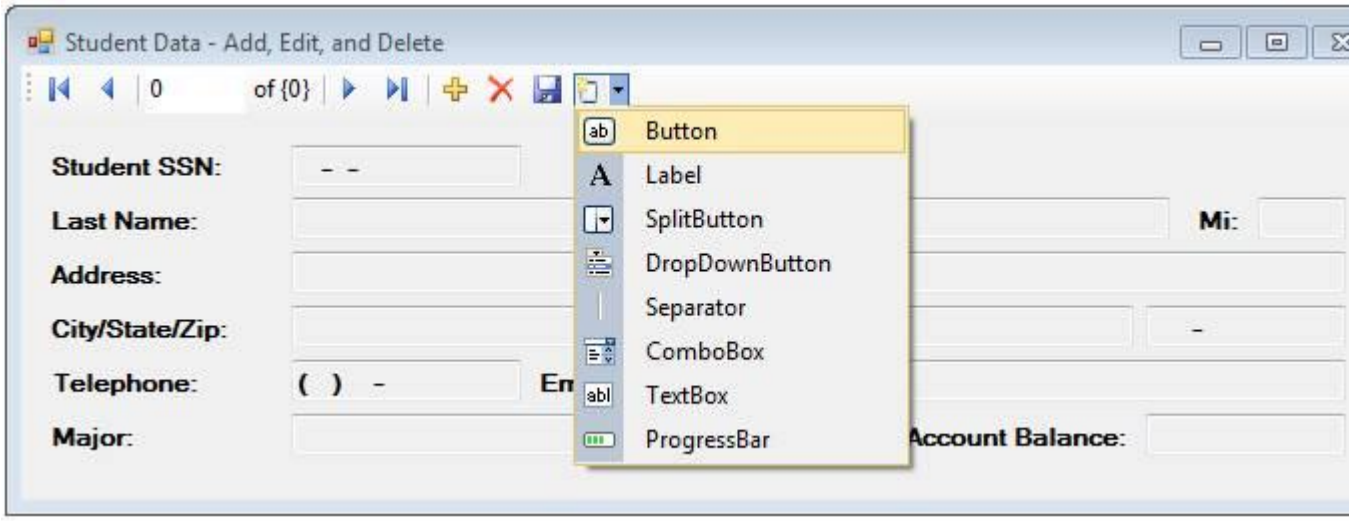
The approach you will learn here changes the default configuration as follows:

- **Make Bound Controls ReadOnly** – each bound control on the form will have **ReadOnly** = **True** – this will prevent changing any data row values.
- **Add an Edit Button** – edit operations will begin by having the application user click an **Edit** Button on the BindingNavigator control – since there is no **Edit** Button on the control, one will need to be added.
- **Add a Cancel Button** – a **Cancel** Button will need to be added to the BindingNavigator control in the event that the application user decides not to edit a data row or clicks the Edit Button by accident.
- **Disable Cancel and Save Buttons** – both of these buttons needs to be disabled while viewing records – their **Enabled** property = **False**; during Add and Edit operations set **Enabled** = **True** for both of these buttons.
- **Alter the BindingNavigator Control Interface** – during Edit operations, all items on the BindingNavigator control except the **Save** and **Cancel** Buttons will be disabled by making them invisible – set **Visible** = **False**.
- When the operation is either saved or canceled, the values of **ReadOnly** and **Visible** will be reversed.

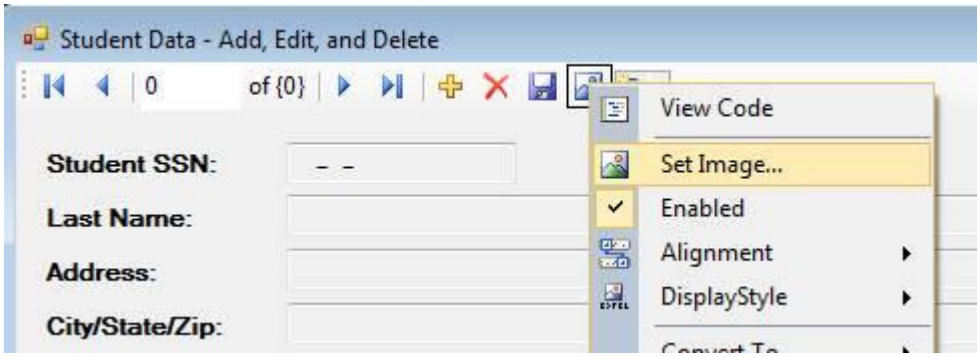
Adding BindingNavigator Buttons

1. Change the **ReadOnly** Property for Bound Controls.
   - Set **ReadOnly** = **True** for all **TextBox** and **MaskedTextBox** controls that display data.
   - Notice that the background color of the controls changes from white to light blue or gray.

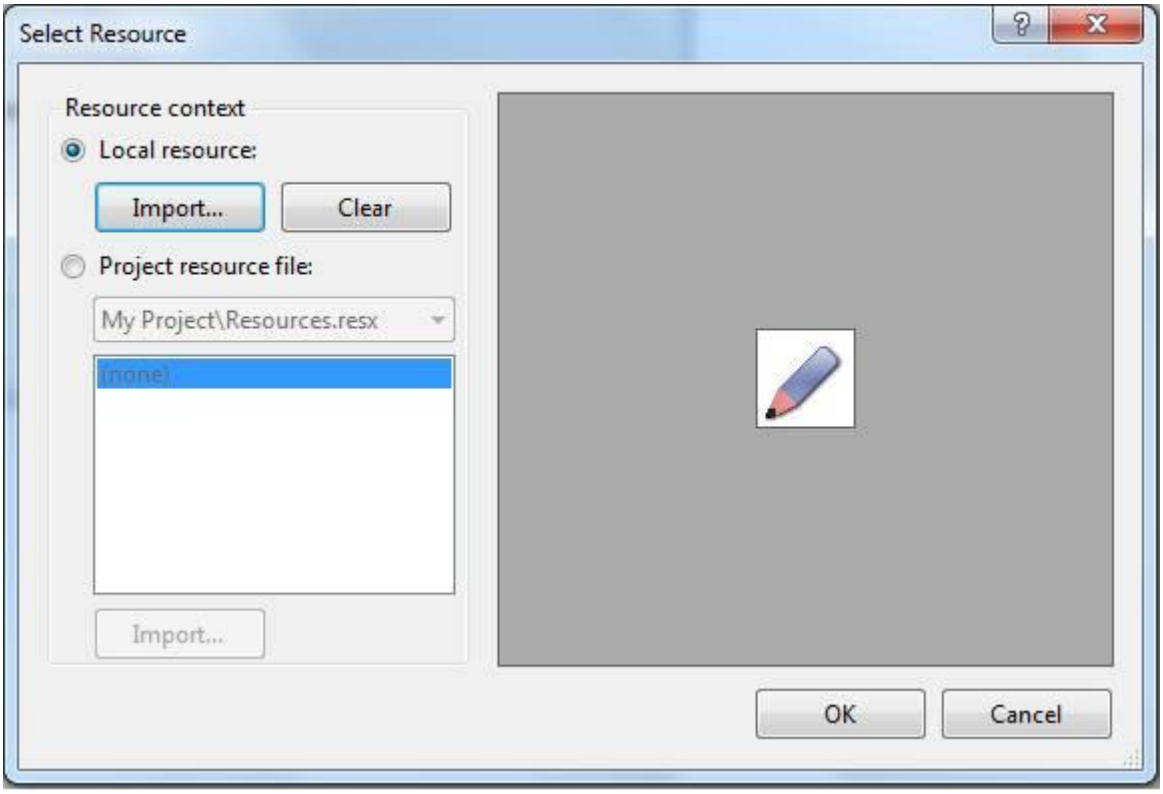
2. Add **EditToolStripButton** and **CancelToolStripButton** Controls.
   - Select the **BindingNavigator** control.
   - Click the **Add ToolStripButton** drop-down on the control and select a **Button** to be added as shown in this figure. This adds a **ToolStripButton**.



- Select the new **ToolStripButton** – set the **Name** property = **EditToolStripButton**.
- Right-click the **EditToolStripButton** button (the default image is a mountain with the sun shining) and either set the **Image** property to an appropriate image or switch the **DisplayStyle** property to **Text** – we will use an image.

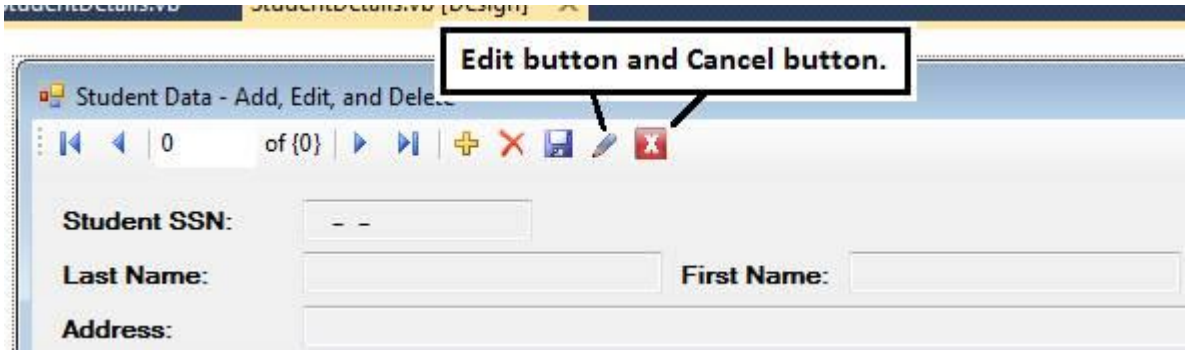


- Images are available on drive Y: for the Chapter 10 project.
  - Select the **Image** property.
  - Click the dialog button (**…**) to open the **Select Resource** dialog box shown in the figure below – click the Import button and navigate to the directory that stores class image files – select the image file named **edit-48x48.png** (as shown in the figure).
  - Click **OK** to close the Select Resource dialog box.



- If you use Text instead of an image, set the **Text** property = **Edit**.
- Repeat these steps to add a second **Button** with **Name** property = **CancelToolStripButton**.
- Set the **DisplayStyle** property = **Image** and import the image file named **x-48x48.png**.

The new buttons are shown in this figure.



3. Change the **Enabled** property for the button controls.
   - Set **Enabled** = **False** for the **Save (StudentBindingNavigatorSaveItem)** and **Cancel (CancelToolStripButton) Buttons** on the BindingNavigator control.

4. Change the **Text** property for the new button controls

- Set the **Text** = **Edit** for the EditToolStripButton.
- Set the **Text** = **Cancel** for the CancelToolStripButton.

SetControls Sub Procedure

We need to code a separate sub procedure that can be called to alter the **BindingNavigator** interface to make BindingNavigator controls invisible except for the **Save** and **Cancel**Buttons.

- This sub procedure will set **ReadOnly** = **False** for the bound controls that display data row values.
- If the parameter **ValueBoolean** = **True**, then the controls are enabled.
- If the parameter **ValueBoolean** = **False**, then the controls are disabled

```
   Private Sub SetControls(ByVal ValueBoolean As Bool
ean)
        'This sub procedure sets the user interface
for the
        'BindingNavigator control and bound controls
for Edit/Add
        'operations

        'ReadOnly/Not ReadOnly the bound controls
        SSNMaskedTextBox.ReadOnly = ValueBoolean
        LastNameTextBox.ReadOnly = ValueBoolean
        FirstNameTextBox.ReadOnly = ValueBoolean
        MiddleInitialTextBox.ReadOnly = ValueBoolean
        AddressTextBox.ReadOnly = ValueBoolean
        CityTextBox.ReadOnly = ValueBoolean
        StateCodeTextBox.ReadOnly = ValueBoolean
        ZipMaskedTextBox.ReadOnly = ValueBoolean
        PhoneMaskedTextBox.ReadOnly = ValueBoolean
        EmailAddressTextBox.ReadOnly = ValueBoolean
        MajorTextBox.ReadOnly = ValueBoolean
        AccountBalanceTextBox.ReadOnly = ValueBoolean

        'Make the Move, Position, and Buttons
        '(except Save and Cancel) Invisible
        BindingNavigatorMoveFirstItem.Visible =
ValueBoolean
```

```
        BindingNavigatorMoveLastItem.Visible =
ValueBoolean
        BindingNavigatorMoveNextItem.Visible =
ValueBoolean
        BindingNavigatorMovePreviousItem.Visible =
ValueBoolean
        BindingNavigatorPositionItem.Visible =
ValueBoolean
        BindingNavigatorCountItem.Visible =
ValueBoolean
        BindingNavigatorAddNewItem.Visible =
ValueBoolean
        BindingNavigatorDeleteItem.Visible =
ValueBoolean
        EditToolStripButton.Visible = ValueBoolean

        'Enable/disable the Save and Cancel Buttons
        StudentBindingNavigatorSaveItem.Enabled
= Not ValueBoolean
        CancelToolStripButton.Enabled
= Not ValueBoolean
    End Sub
```

Add this sub procedure to your program. Check that the control names on your form match the names used in the sub procedure.

EditToolStripButton Click Event

You must add a click event sub procedure for the new **EditToolStripButton** control. The event is very simple. You are altering the controls on the form to allow editing by calling the **SetControls** sub procedure with a value of **False**.

- Makes the TextBox and MaskedTextBox controls ReadOnly = False so the data can be altered.
- Makes the BindingNavigator control's Move, Position and Buttons (except Cancel and Save) invisible.
- Enables the Save and Cancel buttons.

```
   Private Sub EditToolStripButton_Click(ByVal sender
 As System.Object, ByVal e As System.EventArgs) Handl
es EditToolStripButton.Click
```

```
        'Call SetControls with False to alter the
form to
        'allow editing a data row
        SetControls(False)
    End Sub
```

Add this sub procedure to your program. Check that the control names on your form match the names used in the sub procedure.

CancelToolStripButton Click Event

You must add a click event sub procedure for the new **CancelToolStripButton** control. The event requires two actions.

1. **CancelEdit** method – use this method of the **BindingSource** control to "throw away" any changes that may have been made to the current data row.
   - Application users may wish to cancel when they accidentally click Edit or Add.
   - They may also wish to click cancel when they change their mind about editing a record.

2. **SetControls** – call this sub procedure with a value of **True** to make the form's bound controls **ReadOnly** again and to make the **BindingNavigator** control's buttons visible.
   - This reverses the status of the TextBox, MaskedTextBox, and BindingNavigator control's buttons.

```
    Private Sub CancelToolStripButton_Click(ByVal sen
der As System.Object, ByVal e As System.EventArgs) Ha
ndlesCancelToolStripButton.Click
        'Cancel the operation
        StudentBindingSource.CancelEdit()

        'Call SetControls with True to alter the form
to
        'make the form ReadOnly
        SetControls(True)
    End Sub
```

Test the project.

- Run the project to ensure that the form starts up with all **BindingNavigator** controls visible and enabled except for the **Save** and **Cancel** Button controls.
- Ensure the bound controls displaying data are **ReadOnly**.
- Click **Edit** – the form should now enable editing with only the **Save** and **Cancel** Button controls visible on the BindingNavigator.
- Change a record and click **Cancel** – the change should be "thrown away" and the form will be restored to its original configuration.
- Before you can save any changes, you need to modify the **Save** Button code—the next section explains **Data Row Save Operations**.

Data Row Save Operations

This section explains **Data Row Save Operations** with a **BindingNavigator** control. It is necessary to have a way to save changes made to a DataSet during both **Edit** and **Add**operations so that the changes are updated across the network to the database.

About the Save Button – Coding the Save Button

Normally when a BindingNavigator control is added to a form it does **not** include a **Save** Button.

- Visual Studio adds a **Save** Button to the **BindingNavigator** when the **TableAdapter** is generated when you drag the table onto the form.
- If you add a **BindingNavigator** control to a form, you will not see a **Save** Button and will need to add one.

The code generated by VB for the **Save** button of the binding navigator control is very simple and does not catch exceptions. The code is shown here.

```
    Private Sub StudentBindingNavigatorSaveItem_Click
(ByVal sender As System.Object, ByVal e As System.Eve
ntArgs) HandlesStudentBindingNavigatorSaveItem.Click
        Me.Validate()
        Me.StudentBindingSource.EndEdit()
```

```
        Me.TableAdapterManager.UpdateAll(Me.StudentDetailsDataSet)
    End Sub
```

- The **Validate** method for the form confirms that any **Validating** event sub procedures have validated data on the form – on this project you will not use any **Validating** event sub procedures so this line of code has no purpose for your program – this line of code can be deleted or converted to a remark in case it is needed later.
- The **EndEdit** method of the **StudentBindingSource** object ends all edit operations. This applies to both updates of existing data rows and the addition of new data rows.
- The **UpdateAll** method of the **TableAdapterManager** control causes a reconnection to the database and updates any modifications of **StudentDetailsDataSet**.
- If an exception occurs, it will be caused by the **UpdateAll** method – possible exceptions include:
  - Trying to save a new row that duplicates an existing database row.
  - Trying to update a column value with an illegal value.
- It is also necessary to call the **SetControls** sub procedure to reestablish the user interface on the form following a successful save.

Using a Try-Catch Block

Modify the code by using a **Try-Catch** block to catch exceptions. The revised sub procedure:

```
   Private Sub StudentBindingNavigatorSaveItem_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) HandlesStudentBindingNavigatorSaveItem.Click
        'Trap any exceptions during student table update
        Try
            'Me.Validate()
            Me.StudentBindingSource.EndEdit()
            Me.TableAdapterManager.UpdateAll(Me.StudentDetailsDataSet)
```

```
            'Call sub procedure to enable
BindingNavigator controls
            'by sending a parameter value True
            Me.SetControls(True)

        Catch ex As Exception
            Dim MessageString As String = "Report
this error to the system administrator: " &
ControlChars.NewLine & ex.Message
            Dim TitleString As String = "Error During
Save Operation"
            MessageBox.Show(MessageString,
TitleString, MessageBoxButtons.OK,
MessageBoxIcon.Error)
        End Try
    End Sub
```

The **UpdateAll** method is fairly complex and is more useful than the older **Update** method that was previously used to update datasets.

- Updates are performed on a **row-by-row basis** – this means that if multiple **DataSet** modifications (either Insert, Update, or Delete operations), then you would need to control the order of the updates.
- Using the **TableAdapterManager** ensures that updates involving multiple tables are processed in the correct order where the order of the updates is important.
- The code you're learning in this note set only allows a single update operation to take place at a time – if the database rejects the update, then it is easier to handle the **OleDbException** that is thrown by the database back to your application with a Try-Catch block.
- The **Structured Query Language** commands to execute the required **INSERT**, **UPDATE**, or **DELETE** statements on the database are generated automatically by Visual Studio and the **UpdateAll** method automatically selects the correct command to pass to the database.
- When an **UpdateAll** method fires, the **TableAdapterManager** examines each data row's **RowState** property in order to execute the required **INSERT**, **UPDATE**, or **DELETE**statements.
- After a successful update, changes to the data row are accepted within the **DataSet** – this makes it unnecessary to call the **AcceptChanges** method of the **DataSet** object.

Test the project.

- Run the project, select a data row to edit, and click the **Edit** Button.
- Make a change to the data (such as the student's Major field of study) and click the **Save** button.
- Shutdown the application, then start it back up again – you should find that the data row was saved with the new data value.

Data Row Add Operations

This section explains **Data Row Add Operations** with a **BindingNavigator** control. The purpose of **Add** operations is to enable adding a new data row to a **DataSet** and then update a database. New data row values are checked for violations of data integrity rules built into the database.

Controlling Add Operations

An application program may ABEND during an **Add** operation if the application user does any of these things:

- Clicks the **Add New** button to add a second new record before the first **Add operation** is finished.
- Clicks one of the **Move** buttons – this triggers an **Update** of the dataset even though the new record isn't complete.

Calling the **SetControls** sub procedure with a value of **False** will ensure that neither of the above conditions can occur because during an Add Operation the **Add New** button and all **Move** buttons are invisible.

The VB code must accomplish the following:

- Display the form with "empty" controls to allow entry of a new student record – this is accomplished by the application user clicking the **Add New** Button on the **BindingNavigator**control – this requires **NO** coding on your part as the **AddNew** method is automatically called by clicking the **Add New** Button
- Once an **Add** operation begins, the application user must **only** be able to click **Save** or **Cancel**. Calling **SetControls** accomplishes this.

- The initial focus should be set to the **SSNMaskedTextBox** control.
- When an **Add** operation is finished, the application user clicks either **Save** or **Cancel**.

BindingNavigatorAddNewItem Click Event

1. **Add New Item** button – add a **Click** event sub procedure for this button on the **BindingNavigator** control as shown here.
   - **SetControls** – the click event sub procedure calls the **SetControls** sub procedure to disable some of the binding navigator controls.
   - **Focus** method – this method sets focus to the **SSNMaskedTextBox** to make it easy for the application user to begin to enter new row data.

```
    Private Sub BindingNavigatorAddNewItem_Click(ByVa
l sender As System.Object, ByVal e As System.EventArg
s) HandlesBindingNavigatorAddNewItem.Click
        'Call SetControls with False to alter the
form to
        'allow adding a data row
        SetControls(False)

        'Set focus
        SSNMaskedTextBox.Focus()
    End Sub
```

2. Test the project.
   - Click the **Add** Button – add a new data row for yourself as a student.
   - Try to add a second data row that has a duplicate **SSN** value of **111-11-1111** – clicking **Save** should cause the data row to be rejected because it would be a duplicate student row.
   - Clicking **Cancel** should "throw away" the new duplicate data row.

Data Row Delete Operations

This section explains **Data Row Delete Operations** with a **BindingNavigator** control. This operation does not use

the **Save** or **Cancel** Buttons – rather, the code for the **Delete** button click event updates the database.

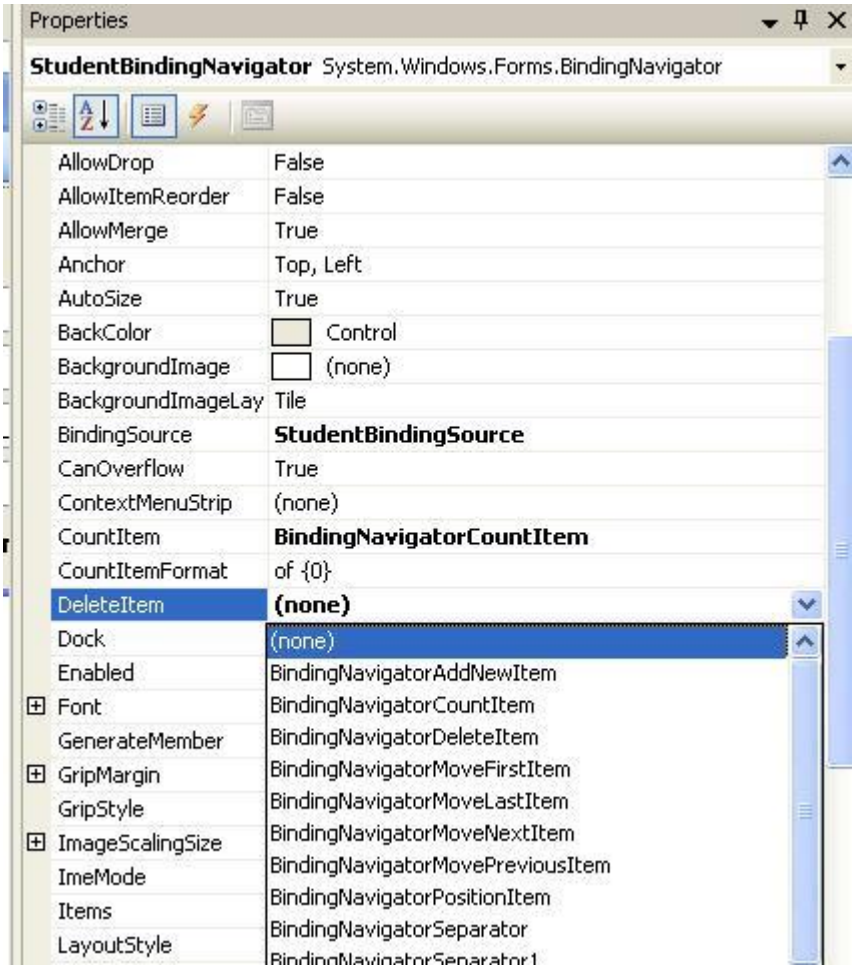
Changing BindingNavigatorDeleteItem Button Behavior

You should change the way that the **Delete** Button on the **BindingNavigator** works in order to ask for application user confirmation a deletion before deleting a data row.

- Sounds simple, but by the time the code that you write for the **Button's Click** event executes, the **BindingNavigator** has already called the **RemoveCurrent** method for the **BindingSource** object and there is no simple way to cancel the action.
- The BindingNavigator has a **DeleteItem** property – when the **ToolStripItem** (**Delete Button**) associated with the property is clicked, the **RemoveCurrent** method is called.
- To change the behavior, clear the **DeleteItem** property to prevent implicitly calling the **BindingSource** object's **RemoveCurrent** method.

1. **StudentBindingNavigator** – select this component.

2. **Properties** window – set the **DeleteItem** property of the **StudentBindingNavigator** = (**none)** at the top of the list of enumerated values as shown in the figure below.



BindingNavigatorDeleteItem Click Event

3. Code a sub procedure to handle removal of data rows.
   - **Delete Button** of the **BindingNavigator** control – double-click to create a **Click** event sub procedure.
   - Add the code shown here.

```
    Private Sub BindingNavigatorDeleteItem_Click(ByVa
l sender As System.Object, ByVal e As System.EventArg
s) HandlesBindingNavigatorDeleteItem.Click
        'Delete the row if there is no relationship
to
        'existing data rows in the ENROLLMENT table

        'Store the current DataSet position in case
the deletion fails
```

```
        Dim RowNumberInteger As Integer =
StudentBindingSource.Position

        Try
            Dim ResponseDialogResult As DialogResult
= MessageBox.Show("Confirm to delete the student
record.", "Delete Y/N?", MessageBoxButtons.YesNo,
MessageBoxIcon.Question,
MessageBoxDefaultButton.Button2)

            If ResponseDialogResult =
Windows.Forms.DialogResult.Yes Then
                'Delete the row by removing the
current record,
                'ending the edit, and calling the
Update method
                StudentBindingSource.RemoveCurrent()
                StudentBindingSource.EndEdit()
                TableAdapterManager.UpdateAll(Student
DetailsDataSet)
            End If

        Catch exOleDb As OleDb.OleDbException
            'The deletion attempt failed due to a
relationship
            'to existing data rows in the ENROLLMENT
table

            'Restore the deleted row with the
RejectChanges method
            StudentDetailsDataSet.RejectChanges()

            'Reposition to the row that was deleted
            StudentBindingSource.Position =
RowNumberInteger

            'Display appropriate error message
            MessageBox.Show("This student cannot be
deleted - the student is enrolled in courses." &
ControlChars.NewLine & exOleDb.Message, "Delete
Operation Error", MessageBoxButtons.OK,
MessageBoxIcon.Error)

        Catch ex As Exception
```

```
            'Some other exception was triggered
            MessageBox.Show("Unexpected error in
delete operation: " & ControlChars.NewLine &
ex.Message, "Delete Operation Error",
MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try

    End Sub
```

- Assume that the delete operation may **fail** – it will be necessary to **restore** the dataset and redisplay the deleted record. The first line of code saves the record number of the data row to be deleted.
- If the application user responds to delete the row, the value of **ResponseDialogResult** is checked. If it is "**Yes**", then:
  - The **RemoveCurrent** method removes the row.
  - The **EndEdit** method ends the edit.
  - The **UpdateAll** method of the **TableAdapterManager** updates the **StudentDetailsDataSet** object (This can also be coded using the Update method of the **StudentTableAdapter**: **StudentTableAdapter.Update(StudentDetailsDataSet.Student)**

- If the deletion fails, the line of code generating the exception will usually be the line of code with the **UpdateAll** method. This triggers an **OleDbException** that is caught by the first of two **Catch** blocks. The **OleDbException** is raised when a deletion fails due to referential integrity constraints:
  - A message box displays an appropriate message.
  - The **DataSet's RejectChanges** method is used to reject the deletion in the DataSet – essentially this **<u>undeletes</u>** the deleted row that the database rejected.
  - The **BindingSource's Position** property is reset to the record number of the data row for which deletion failed.

A DataSet's **RejectChanges** method rolls back all DataSet changes made since the DataSet was created, or since the last execution of **DataSet.AcceptChanges**.

- When a **DataSet's AcceptChanges** method fires, all DataRow objects in edit-mode successfully end their edits.

- Each **DataRow's RowState** property changes – **Added** and **Modified** rows become **Unchanged**, and **Deleted** rows are removed from the DataSet.
- However, it is not necessary to code a call of the **AcceptChanges** method because the **UpdateAll** method automatically does this for you as the programmer.

Data Validation

Additional Exception Handling

A program can be improved through the use of techniques to ensure data values entered during **Edit and Add** operations are valid.

- You have learned to use a **ValidData** function to validate data entry
- This chapter also requires you to write a ValidData function to validate data. A good practice is to use a **ValidData** function to validate data entered for a new record or modified for an existing record prior to executing an **UpdateAll** method to modify the database.
- You can replace the **Me.Validate** command generated for a **Save** button with the **BindingNavigator** control with a call to a **ValidData** function.

ValidData Function

The function shown here enforces three different types of validation rules:

- **Missing data** – data cannot be missing from a bound control if the corresponding field in the database table requires stores of data, i.e., required data.
    - Here we are testing for missing data for the:
        - **LastNameTextBox**
        - **FirstNameTextBox**
        - **AddressTextBox**
        - **CityTextBox**

        - **StateCodeTextBox**
        - **ZipMaskedTextBox** (this control is allowed to be empty by the database, but is required here to demonstrate how to test this type of value/control for valid data).
    - The following **TextBox** and **MaskedTextBox** controls are allowed to be empty:
        - **MiddleInitialTextBox**
        - **PhoneMaskedTextBox**
        - **EmailAddressTextBox**
        - **MajorTextBox**
        - **AccountBalanceTextBox**

- **A control cannot be partially completed** – data fields such as the SSN and Zip Code cannot contain blank spaces. The **MaskCompleted** method can test a MaskedTextBox control to determine if all required values are completed. In the case of the SSN, the control cannot contain any blank spaces. The mask setting of the Zip Code enforces the rule that it cannot contain any blank spaces within the first five characters – the last four characters of a zip code can be blank.
- **Data must be numeric** – the **AccountBalanceTextBox** is allowed to be blank; however, if it contains data, it must be valid numeric data.

Initially the code sets the function name **ValidData = False** – this assumes that some of the data is not valid.

- Each business rule is tested.
- If a rule fails, an error message is displayed and control passes to the **End If** at the bottom of the function and the function exits returning a value of **False**. This approach ensures that only one error message at a time will display.
- If all business rules are satisfied, the **Else** branch executes setting **ValidData = True** and the function exits returning a value of **True**.
- Note the use of **Focus** and **SelectAll** methods to make the program more user-friendly.

1. **ValidData Function** – add the function to your program.

```
Private Function ValidData() As Boolean
    'Initialize function return value
```

```
        ValidData = False
        Dim MessageString As String

        'Test CustomerID is complete
        If SSNMaskedTextBox.MaskCompleted
= False Then
            'Required employee ID is not complete
            MessageBox.Show("Student SSN is not
complete", "Student SSN Error", MessageBoxButtons.OK,
MessageBoxIcon.Error)
            SSNMaskedTextBox.Focus()
            SSNMaskedTextBox.SelectAll()
        ElseIf LastNameTextBox.Text.Trim
= String.Empty Then
            'Validate Last Name
            MessageString = "Last name is required."
            MessageBox.Show(MessageString, "Data
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            LastNameTextBox.Focus()
            LastNameTextBox.SelectAll()
        ElseIf FirstNameTextBox.Text.Trim
= String.Empty Then
            'Validate First Name
            MessageString = "First name is required."
            MessageBox.Show(MessageString, "Data
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            FirstNameTextBox.Focus()
            FirstNameTextBox.SelectAll()
        ElseIf AddressTextBox.Text.Trim
= String.Empty Then
            'Validate Address
            MessageString = "Address is required."
            MessageBox.Show(MessageString, "Data
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            AddressTextBox.Focus()
            AddressTextBox.SelectAll()
        ElseIf CityTextBox.Text.Trim
= String.Empty Then
            'Validate City
            MessageString = "City is required."
            MessageBox.Show(MessageString, "Data
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            CityTextBox.Focus()
            CityTextBox.SelectAll()
```

```
        ElseIf StateCodeTextBox.Text
= String.Empty Then
            'Validate State
            MessageString = "State is required."
            MessageBox.Show(MessageString, "Data
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            StateCodeTextBox.Focus()
        ElseIf ZipMaskedTextBox.MaskCompleted
= False Then
            'Zip code required
            MessageString = "Zip code is incomplete."
            MessageBox.Show(MessageString, "Data
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            ZipMaskedTextBox.Focus()
            ZipMaskedTextBox.SelectAll()
        ElseIf AccountBalanceTextBox.Text.Trim
<> String.Empty AndAlso IsNumeric(AccountBalanceTextB
ox.Text) = False Then
            'Validate Account Balance is numeric if
there is a value stored here
            MessageString = "Account balance must be
a numeric amount."
            MessageBox.Show(MessageString, "Data
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            AccountBalanceTextBox.Focus()
            AccountBalanceTextBox.SelectAll()
        Else
            'All of the data is valid
            ValidData = True
        End If
    End Function
```

The initial **If** statement tests the first validation rule – SSN must be 11 characters.

- The **MaskCompleted** method of the MaskedTextBox determines if the controls is full of numbers – no blanks, none missing.

Later in the ValidData function the **MaskCompleted** method tests the **ZipMaskedTextBox** control – the code will discover whether or not the Zip Code value contains the minimum required 5 digits. Remember, the last four digits of a zip code (digits beyond the dash) are optional.

2. **Save Button Click** event – modify the event to call the **ValidData** function. The new code is highlighted in **yellow**.

```
Private Sub StudentBindingNavigatorSaveItem_Click(ByV
al sender As System.Object, ByVal e As System.EventAr
gs) HandlesStudentBindingNavigatorSaveItem.Click
    'Trap any errors during update of the student
table
    Try
        If ValidData Then
            'Me.Validate()
            Me.StudentBindingSource.EndEdit()
            Me.TableAdapterManager.UpdateAll(Me.Stude
ntDetailsDataSet)

            'Call sub procedure to enable
BindingNavigator controls
            'by sending a parameter value True
            SetControls(True)
        End If
    Catch ex As Exception
. . . More code follows to handle exceptions.
```

MaxLength Property

One of the easiest ways to enforce validation rules regarding the maximum length (size) of data to be stored to a particular column in a table is by setting the **MaxLength** property of a control such as a TextBox.

- **MaxLength** property – limits the number of characters that can be typed into a control.
- Example: the value entered for **StateCodeTextBox** cannot be larger than **2** characters such as CA for California or IL for Illinois. You should set the **MaxLength** property of the **StateCodeTextBox** control to **2**.

MaxLength should be set for each control that stores data to be saved to the database as follows:

- **String data.** If the control (a TextBox or label) displays string data, you can examine the database table in design mode and determine the maximum allowed length for values to be stored/displayed in the control.

- **Numeric data.** For numeric data, you need to use good judgment or determine what rules the business wishes to enforce.
    - **Example #1**: a TextBox control that displays the number of credit hours for which a student can enroll may be limited to a maximum value of 12 hours if the University does not offer courses for more than that number of credit hours (some School of Education student teaching classes may be taken for up to **12** credits per term) – since 12 hours is the maximum credit hours for a course and two characters are sufficient to store the largest value, set the **MaxLength** for this TextBox to **2**.
    - **Example #2**: You need to store a currency value such as **$99,999.00**. You need to allow characters for the display/storage of the dollar sign, comma and decimal point. This particular value is **10** characters at a maximum.
- **Date/Time data.** For date/time data, do not worry about enforcing a maximum length – date/time values are stored in a database as data that are a fixed length in size.

The maximum size of a data value to be stored to the **STUDENT** table can be obtained by using the **View menu, Server Explorer** window option. Drill down to the **STUDENT** table and examine each field – the **Properties** window will display the **Length** property.

Set the **MaxLength** property values as shown here:

- Student SSN – cannot be set, a MaskedTextBox does not have a MaxLength property.
- Last Name = 20.
- First Name = 20.
- Middle Initial = 1.
- Address = 50.
- City = 40.
- State Code = 2.
- Zip Code – cannot be set, a MaskedTextBox does not have a MaxLength property.
- Phone Number – cannot be set, a MaskedTextBox does not have a MaxLength property.
- Email Address = 50.
- Major = 20.
- Account Balance = 11 (to allow a value up to $999,999.99).

Key_Press Event

The **Key_Press** event of a control is used to validate various keys on the keyboard as shown in the example below for the **AccountBalanceTextBox_KeyPress** subroutine.

This event validates the keystrokes for a TextBox or other type of control such as a ComboBox.

- Example: the **AccountBalanceTextBox** control for the **STUDENT** table should only store numbers, a decimal point, a comma, and a currency symbol (we will use a **$** assuming US currency) – example account balance: **$1,542.98**.
- The application user must also be able to use the backspace key to delete erroneous entries.
- The event should restrict keystrokes that are accepted into the **TextBox** to numbers (**Asc** values **48** to **57**), a decimal point (**Asc** value **46**), a comma (**Asc** value **44**), the dollar sign (**Asc** value **36**), and the backspace key (**Asc** value **8**).
- Note that the **Key_Press** event does not trap the keyboard's **delete** key and **arrow** keys.

The value of the **KeyChar** property of "**e**", a **KeyPressEventArgs** value, is used in the **SELECT Case** structure and the **Handled** property is set to **False** to allow the key; otherwise the keystroke is handled (ignored) by this event.

```
Private Sub AccountBalanceTextBox_KeyPress(ByVal send
er As Object, ByVal e As System.Windows.Forms.KeyPres
sEventArgs) HandlesAccountBalanceTextBox.KeyPress
    'Allow Backspace (8), numeric keys (48 to 57),
comma (44),
    'decimal point (46)and dollar sign (36)
    Select Case Asc(e.KeyChar)
        Case 8, 36, 44, 46, 48 To 57
            e.Handled = False  'Allow the key
        Case Else
            e.Handled = True   'Ignore the key
    End Select
End Sub
```

Note that the **AccountBalanceTextBox** must still be tested by the **ValidData** function because an application user could enter a numeric value such as **8.9.$5,332**, however, this numeric value is

obviously invalid and the **IsNumeric** function will catch this error and return a value of **False**.

If you have not already done so, modify your project as follows:

- Code a **ValidData** function and **Key_Press** event sub procedure.
- Set the **MaxLength** property of the appropriate **TextBox** controls.
- Set the **CharCasing** property of the **StateCodeTextBox** to **Upper**.
- Modify the **Save** button's Click event sub procedure
- Test the program by trying to enter invalid data.

Using a ComboBox Control

Earlier in the course you learned to use ComboBox and ListBox controls.

- You stored data to the Items (collection) property at design time thereby creating a static listing.
- Here you will learn to store data to the Items (collection) property at run time to create a dynamic listing.

Making Data Navigation Easier with a LastNameComboBox

You can use a list type of control, such as a ComboBox to make it easier to find student records if the list of students in the dataset is large, such as when there are several hundred or thousand records.

Use a ComboBox to display the student last name or SSN value and then when a value is selected from the ComboBox, display the corresponding values in the other bound controls on the form.

Add and Data Bind a ComboBox

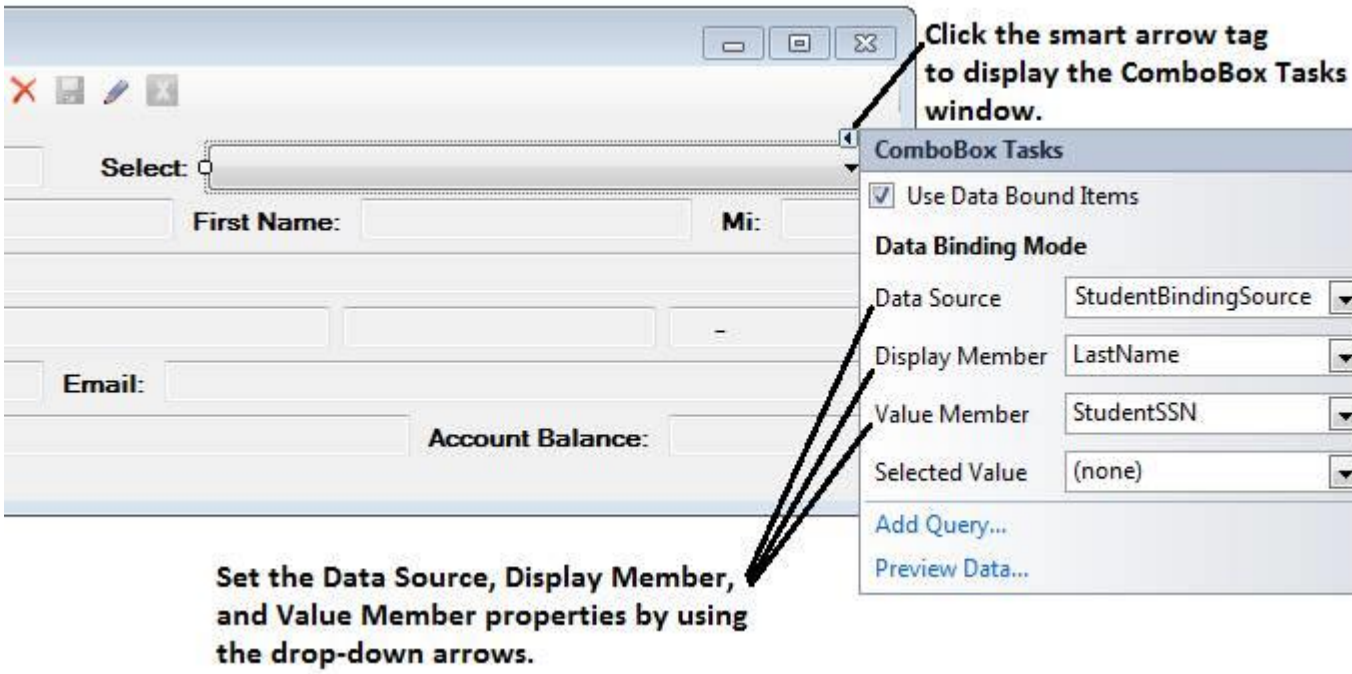
1. Add a label and accompanying ComboBox control next to the existing **SSNMaskedTextBox** control.

- Label **Text** property – **Select:**.
- ComboBox **Name** property – **LastNameComboBox**.
- **DropDownStyle** – **DropDownList**.

2. Bind the ComboBox to the **StudentBindingSource** object by clicking the ComboBox smart tag arrow.

3. Check the **Use data bound items** check box as shown in the figure below – this causes the **Data Binding Mode** properties to display as shown in the figure:
   - **DataSource** – **StudentBindingSource**.
   - **DisplayMember** – **LastName**.
   - **ValueMember** – **StudentSSN**.

4. **Properties window** – set **TabStop** = **False**.



Alter the SetControls Sub Procedure

You do not want the application user to access the **LastNameComboBox** control during an **Edit** or **Add** operation.

5. **SetControls** sub procedure -- add the following line of code to the **SetControls** sub procedure.

```
'Enable/disable ComboBox
LastNameComboBox.Enabled = ValueBoolean
```

6. Test the project.
   - Select a name from the ComboBox.

- Some names are **duplicated** because the ComboBox displays the last name column (due to the **DisplayMember** property setting) and the **LastName** column is **not** unique – more than one student can have the same last name.
- The data is actually found by the setting of the **ValueMember** property since **LastName** is not unique, but **StudentSSN** is unique.
- This technique may not get you to the exact student record desired, but it does get you **close** to the record – use the **Move** navigation buttons of the **BindingNavigator** control to navigate to the exact student record.
- Click the Edit or Add Button – confirm the ComboBox is not accessible (disabled). Click the Cancel Button – confirm the ComboBox is now accessible again.

If the column displayed by the **DisplayMember** property is the **primary key** column of the table, then you need not set the **ValueMember** property; otherwise, set **ValueMember** to the **primary key** column.

A State ComboBox

The **States** table of the VBUniversity database is a **validation table**. It has two columns:

- **StateCode** – stores the two character abbreviation code.
- **StateName** – stores the full name of a state.

It may be desirable to display the entire state name on the **StudentDetails** form because some application users will not know all of the two-character state code abbreviations.

- Even though the full state name will be displayed, you must still save just the two-character state code to the **Student** table when editing existing records or when adding a new record.
- The state table contents can be displayed to the form by use of a **ComboBox** control with **DropDownStyle** = **DropDownList** so that new states cannot be added by the application user.

Add a ComboBox

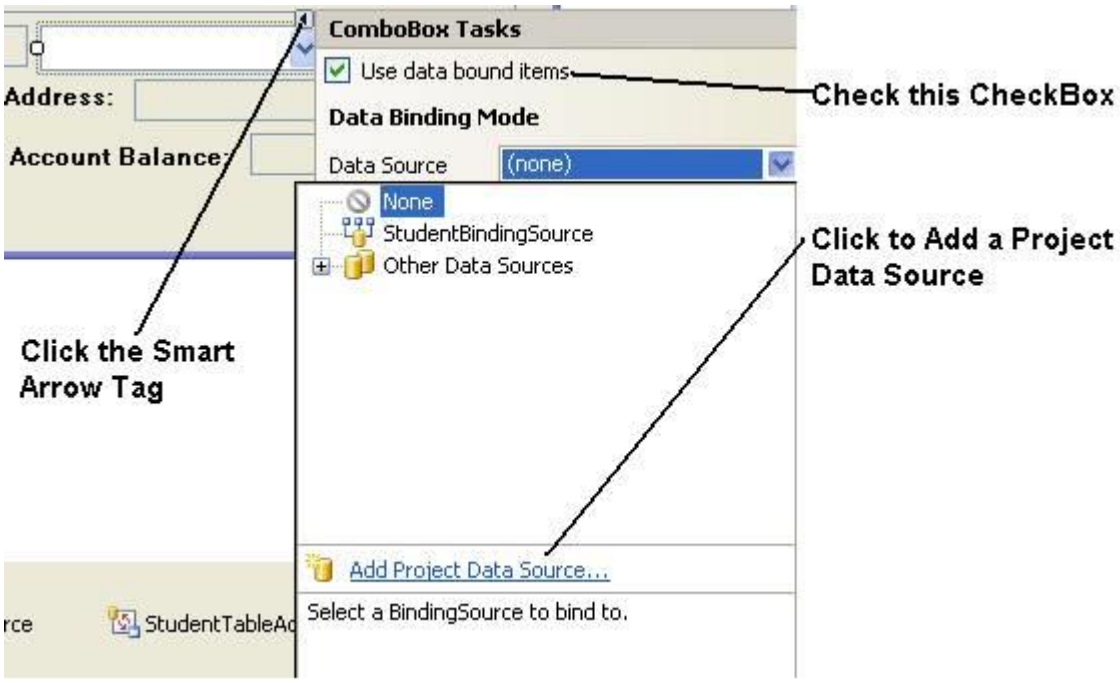
1. Delete the existing **StateCodeTextBox** control.

2. Add a **ComboBox** control – set properties as follows
   - **Name** = **StateComboBox**.
   - **DropDownStyle** = **DropDownList**.
   - **Enabled** = **False**.

3. Reset the form's tab order.

Add a DataSource

4. Click the StateComboBox **Smart Arrow Tag** as shown in the figure below.
5. Check the **Use data bound items** CheckBox as shown in the figure blow.
6. Select the **Data Source drop-down arrow** and then click the **Add a Project Data Source** link.

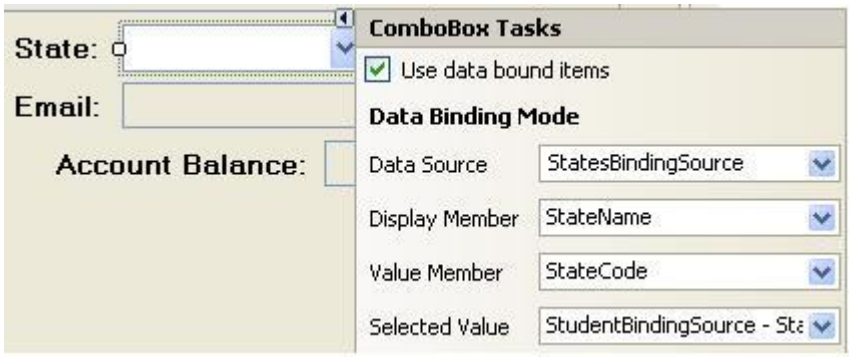


7. Data Source Configuration Wizard
   - **Choose a Data Source Type** – choose **Database** and click **Next**.
   - **Choose a Data Model** – select **Dataset** and click **Next**.
   - **Choose Your Data Connection** – use the existing connection to the VBUniversity database.
   - **Choose Your Database Objects** – expand the **Tables** node and check the **States** table.

- Name the dataset **StatesDataSet** and click **Finish**.
- Note the new **StatesBindingSource**, **StatesTableAdapter**, and **StatesDataSet** components that display to the system component tray.

Bind the StateComboBox

8. Bind the **StateComboBox** – click the smart arrow tag for the **StateComboBox** and set the four properties shown in the figure below.



- **DataSource** = **StatesBindingSource** – this binds the ComboBox control to this data source.
- **DisplayMember** = **StateName** – this is the **StateName** column of the **States** table – this value displays in the ComboBox's Text property.
- **ValueMember** = **StateCode** – this is the **StateCode** column of the **States** table – this is the name of the data column of the **States** table whose value is stored in the list and is the value returned by the **SelectedValue** property of the ComboBox.
- **SelectedValue** = **StudentBindingSource – StateCode** – this property gets the value of the currently selected item and binds the **ValueMember** property to the **StateCode**column of the **Student** table.

- These four property settings enable you to display data from one column of a validation table (here this is the **States** table), and save a value from another column of a validation table to a column of a table in another data source (here this is the **Student** table, **StateCode** column).
- The value actually saved to an edited or new **Student** table row is the two-character state code.

- You must ensure that the **(DataBindings)-Text** property of the **ComboBox** = **None** – if this property is not cleared, the **ComboBox** will not work correctly.

9. Sort the new data source – select the **StatesBindingSource** – in the Properties window, set the **Sort** property = **StateName**.

Modify Form Load, SetControls, and ValidData Procedures

10. **Form's Load Event** – adding the new **StatesDataSource** will cause VB to generate a new line of code in the **StudentDetails_Load** event – you need to modify this new code by moving the line of code into the Try-Catch block that is highlighted in yellow.

```
    Private Sub StudentDetails_Load(ByVal sender As S
ystem.Object, ByVal e As System.EventArgs) Handles My
Base.Load
        'Trap exceptions that occur during data load
        Try
            Me.StatesTableAdapter.Fill(Me.StatesDataS
et.States)
            Me.StudentTableAdapter.Fill(Me.StudentDet
ailsDataSet.Student)
        Catch ex As Exception
            Dim MessageString As String = "Report
this error to the system administrator: " &
ControlChars.NewLine & ex.Message
            Dim TitleString As String = "Student
Details Data Load Failed"
            MessageBox.Show(MessageString,
TitleString, MessageBoxButtons.OK,
MessageBoxIcon.Error)
        End Try
    End Sub
```

11. The **StateCodeTextBox** control has been deleted – you must remove all references to this control in the form's code and replace the lines of code with appropriate code for the new**StateComboBox** control.

- **SetControls** sub procedure. Delete the line of code that sets the **ReadOnly** property of the **StateCodeTextBox**.

- Add a line of code to change **Enabled** for the **StateComboBox**.

```
'Remark out next line when StateComboBox replaces
'the StateCodeTextBox and add the line of
'code to enable the StateComboBox
'StateCodeTextBox.ReadOnly = ValueBoolean
StateComboBox.Enabled = Not ValueBoolean
```

- **ValidData** function.
  - Delete the code that validates the **StateCodeTextBox**.
  - Add new code to validate the **StateComboBox** as shown here.

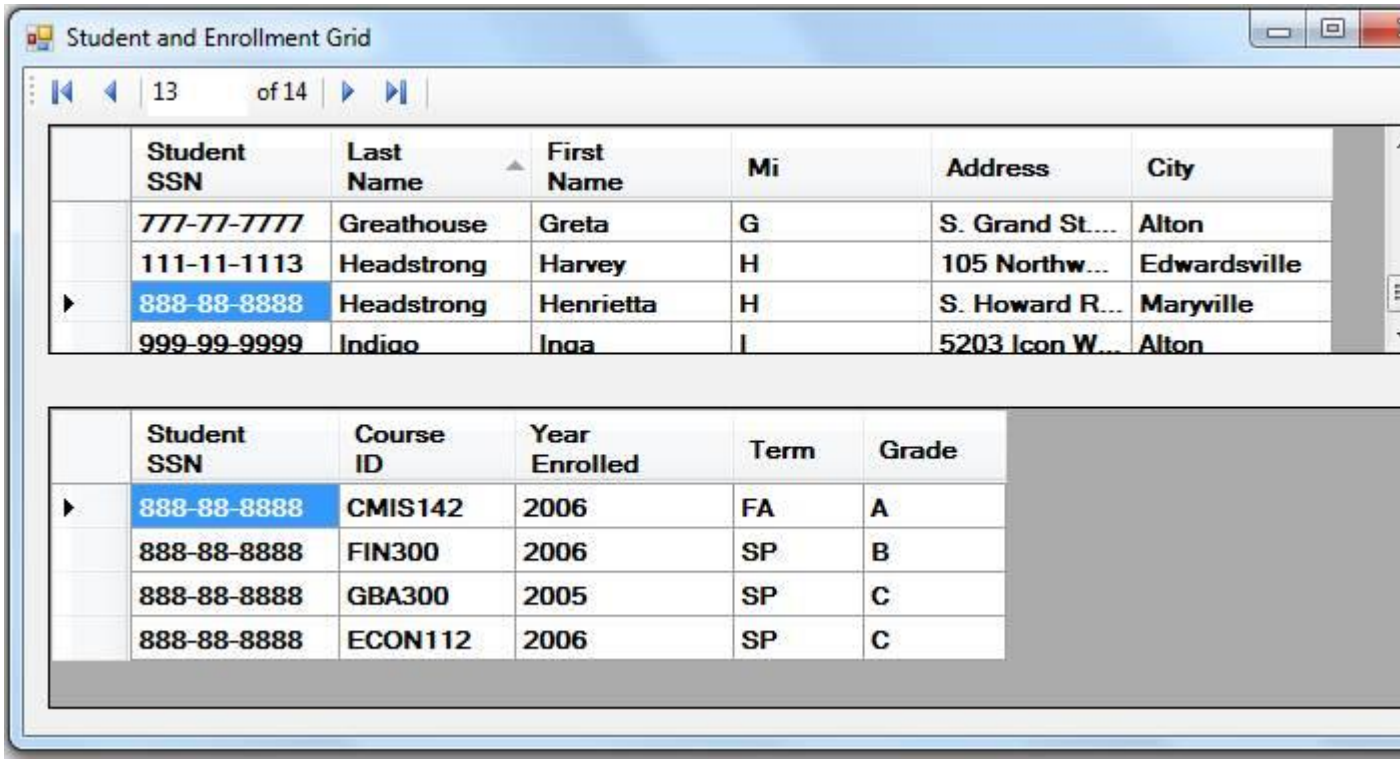
```
ElseIf StateComboBox.SelectedIndex = -1 Then
        'Validate State
        MessageString = "State is required."
        MessageBox.Show(MessageString, "Data Error",
MessageBoxButtons.OK, MessageBoxIcon.Error)
        StateComboBox.Focus()
```

12. Test the form by both editing and adding rows.

VB University Project – Two DataGridView Controls – StudentEnrollmentGrid Form

This section demonstrates how to create a **Master-Detail Form** that uses two **DataGridView** controls like that shown in the figure below.

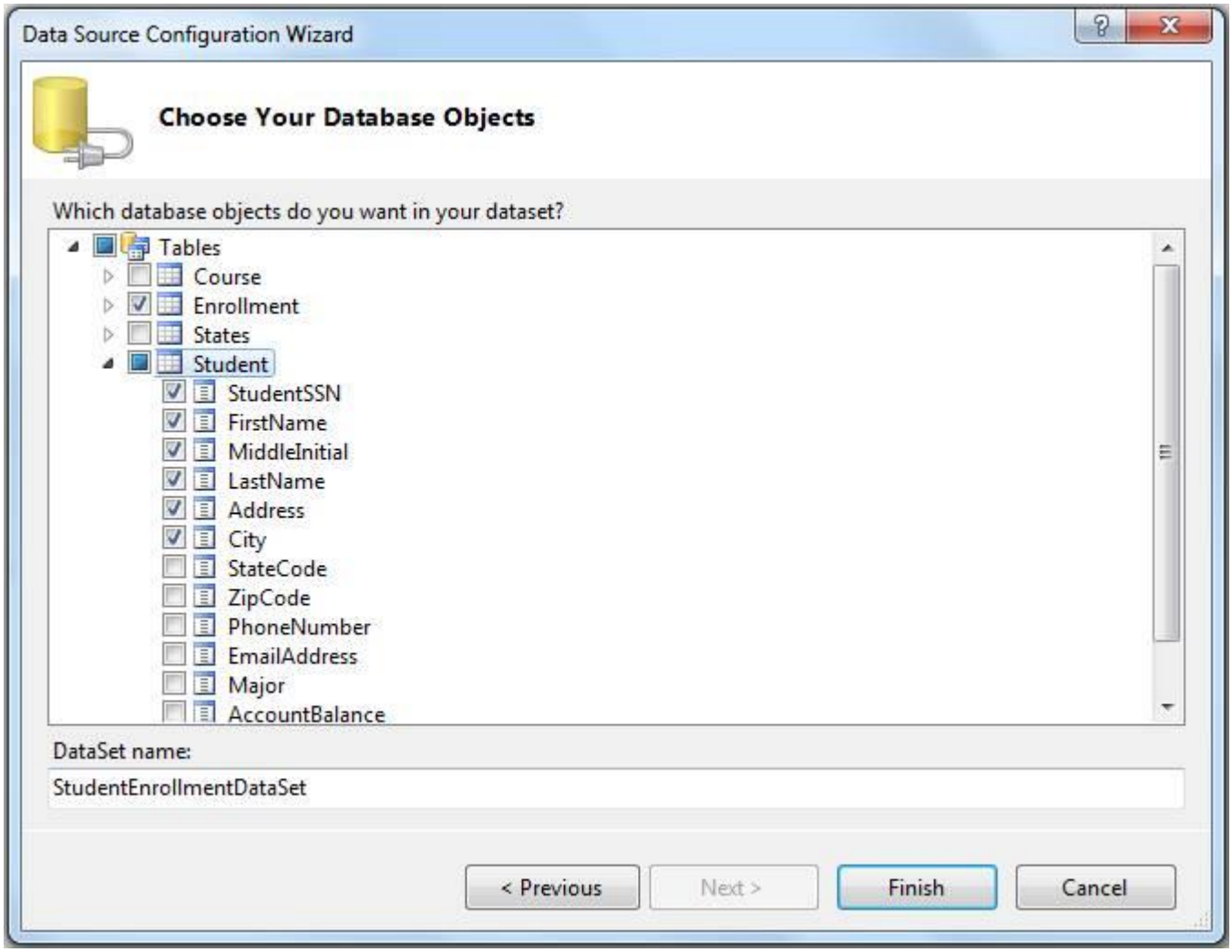
- **Student Table Data** – the first DataGridView control (the **Master**) displays information from the **Student** table.
- **Enrollment Table Data** – the second DataGridView control (the **Detail**) will display detail **Enrollment** table information for a selected Student.



Add a Data Source

1. Add a new data source for the **Student** and **Enrollment** Tables – select the **Data | Add New Data Source** menu.

2. Data Source Configuration Wizard
   - **Choose a Data Source Type** –choose **Database** and click **Next**.
   - **Choose Your Data Connection** – use the existing connection to the VBUniversity database.
   - **Choose Your Database Objects** – expand the **Tables** node and check both the **Student** and **Enrollment** tables as shown in the figure below. Expand the **Student** table node and only select the columns indicated in the figure.



- Name the dataset **StudentEnrollmentDataSet** and click **Finish**.

Add a New Form

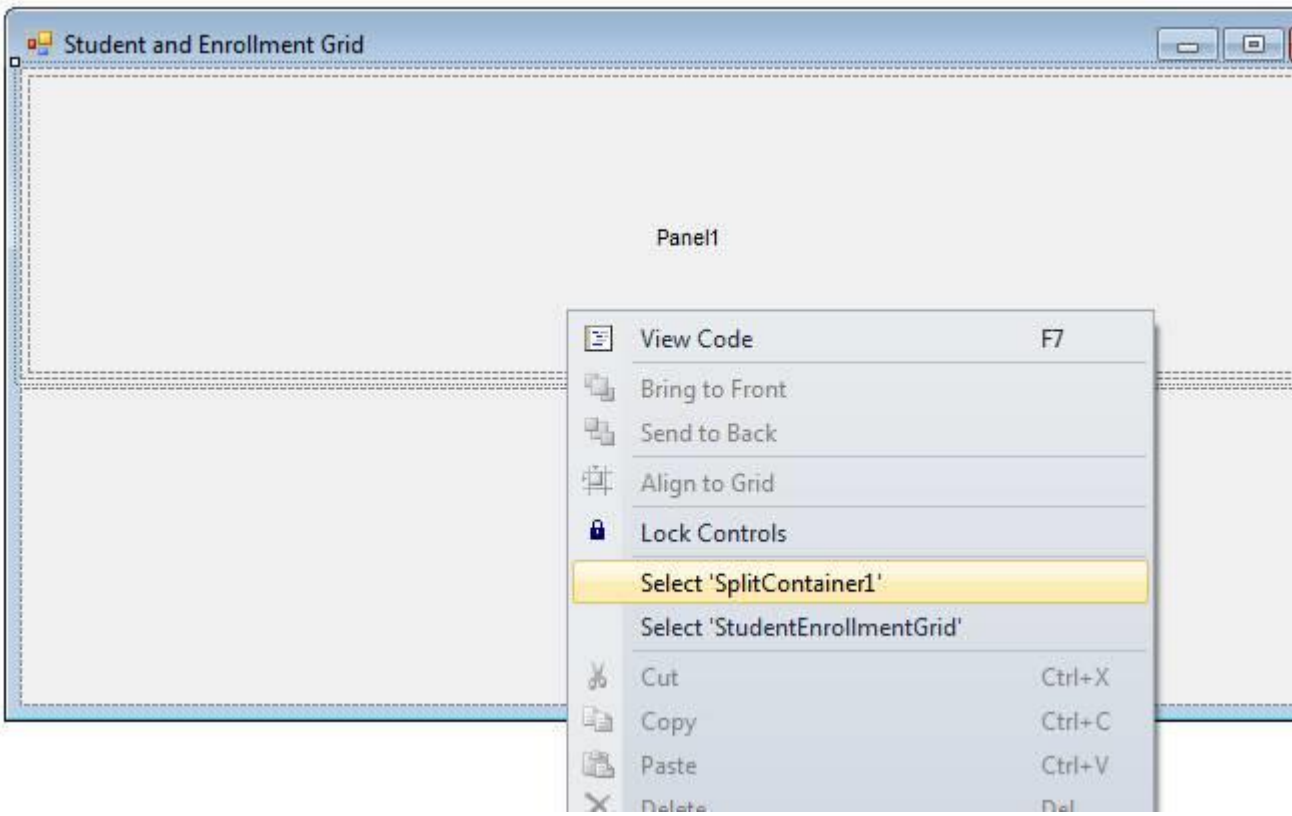
3. Add a new form to the project. Access the **Project | Add Windows Form…** menu to add a new Windows form.

4. Name the form **StudentEnrollmentGrid.vb**.
   - Set the form's **Font** property = **9 point**.
   - Set the form's **Text** property = **Student and Enrollment Grid**.

5. Access the Toolbox | Containers section. Drag a **SplitContainer** control to the form. A SplitContainer control tends to fill the form. It is a container type of control. You will split the form horizontally into top and bottom halves.

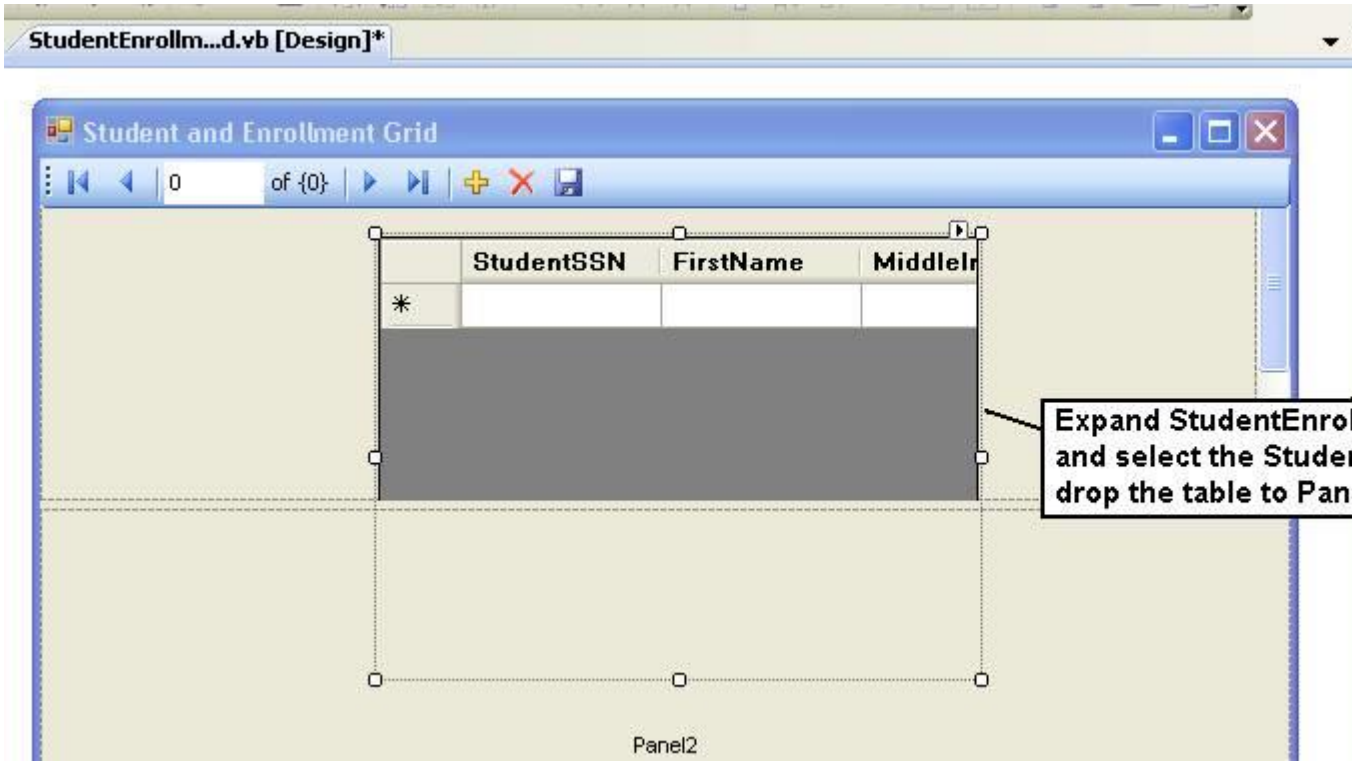
- Right-click the design surface as shown in the figure below and choose the **Select 'SplitContainer1'** menu option.
- In the Properties window set the control's properties as follows:
  - **Dock** = **Fill**.
  - **Orientation** =**Horizontal**.

- Use the mouse to drag/drop the divider line between **Panel1** and **Panel2** and adjust the panel sizes so the form is about evenly split.



Add a StudentDataGridView Control

6. Data Sources window.
   - Expand the **Data Sources** window as shown in the figure below.
   - Expand the **StudentEnrollmentDataSet** and select the **Student** table.
   - Drag/drop the **Student** table to **Panel1** of the new form.

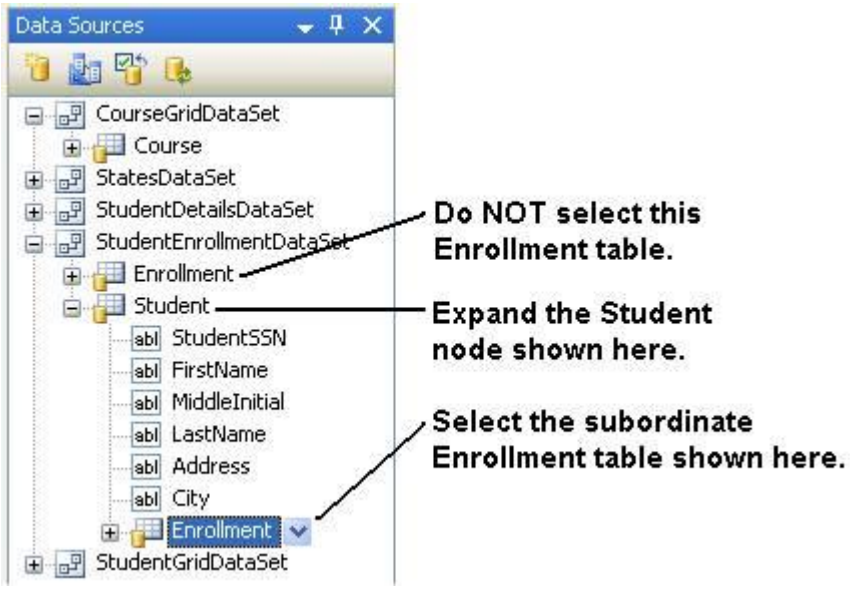


7. Modify the **StudentDataGridView** control.
   - Name the control **StudentDataGridView**.
   - Position the **StudentDataGridView** control within **Panel1** – resize the form horizontally to enable display of all **Student** table column data at the same time.
   - Edit the **StudentDataGridView** control.
     - Disable Adding, Editing, and Deleting.
     - Enable Column Reordering.
     - Edit the columns and order columns: StudentSSN, LastName, FirstName, MiddleInitial, Address, and City.
     - Set **AutoSizeColumnsMode** property = **AllCells** for the DataGridView.

8. Modify the **StudentBindingNavigator** control and **StudentBindingSource**.
   - Delete the **Add**, **Delete**, and **Save** buttons from the **StudentBindingNavigator** control.
   - Select the **StudentBindingSource** control – in the **Properties Window** sort the data by setting the **Sort** property = **LastName, FirstName.**

Add an EnrollmentDataGridView Control

9. Data Sources window.
   - Expand the **Data Sources** window as shown in the figure below.
   - Expand the **StudentEnrollmentDataSet** and then expand the **Student** table node.
   - Select the **subordinate Enrollment** table shown in the figure below, then drag/drop this table to **Panel2** of the new form.



10. Modify the **EnrollmentDataGridView** control.
   - Name the control **EnrollmentDataGridView**.
   - Position the **EnrollmentDataGridView** control within **Panel2** – resize the form horizontally to enable display of all **Enrollment** table column data at the same time.
   - Edit the **EnrollmentDataGridView** control.
     - Disable Adding, Editing, and Deleting.
     - Enable Column Reordering.
     - Edit the columns and order columns: StudentSSN, CourseID, TermCode, YearEnrolled, and GradeCode.
     - Set **AutoSizeColumnsMode** property = **AllCells** for the DataGridView.

Modify Programming Code for the Form

11. Modify the programming code for the form's Load event as shown here by adding a **Try-Catch** block.

```
    Private Sub StudentEnrollmentGrid_Load(ByVal send
er As System.Object, ByVal e As System.EventArgs) Han
dles MyBase.Load
        'Fill the StudentEnrollmentDataSet
        Try
            Me.StudentTableAdapter.Fill(Me.StudentEnr
ollmentDataSet.Student)
            Me.EnrollmentTableAdapter.Fill(Me.Student
EnrollmentDataSet.Enrollment)
        Catch ex As Exception
            Dim MessageString As String = "Report
this error to the system administrator: " &
ControlChars.NewLine & ex.Message
            Dim TitleString As String = "Student or
Enrollment Data Load Failed"
            MessageBox.Show(MessageString,
TitleString, MessageBoxButtons.OK,
MessageBoxIcon.Error)
        End Try
    End Sub
```

12. Delete the **StudentBindingNavigatorSaveItem_Click** sub procedure.

13. Test the project:
    - In the Solution Explorer window click the **My Project** node.
    - Select **StudentEnrollment** as the startup form from the dropdown selection.
    - Close My Project.
    - Run the project.
      - A form similar to that shown earlier in this note set should display. Check the display of all column data.
      - Change student records and note that the correct enrollment records display for each student.
      - You can adjust the size of controls, modify column headings, and adjust column widths as necessary in order to provide a more professional looking interface.

Question: Can you determine how the program automatically modifies the display of enrollment records to match each student record?

Multiple Document Interface

The project now has multiple forms. The **multiple document interface** (**MDI**) allows the creation of a parent form within which child forms can be opened.

Advantages of the MDI include:

- The parent form can have a menu that can be used to manage the parent and child forms.
- Child forms can be minimized, maximized, and restored within the parent form.
- When the parent form closes, all child forms close automatically.
- Child forms cannot move outside of the parent form.
- Forms such as Splash forms can continue to operate independently of the parent form.
- The parent form can have a Window menu item that enables the display of a list of open windows (forms) and can help you move from one active form to another.

Add a MDIParent Form

Visual Studio has a form template for a **MDIParent** form; however, we will not use it as most of the features built into the template are not very useful for business applications. Instead we will use a regular Windows Form and convert it to a **MDIParent** form.

1. Select the **Project** menu, **Add Windows Form** option and add a new form named **VBUniversityParent.vb**.

2. In the **Properties** window for the new form set the **IsMDIContainer** property to **True** – this causes the new form to be a parent form. Note that the **BackColor** property of the form is now a dark gray.

3. You may want to size the form either:
   - Size the form manually through trial and error until you achieve an acceptable size and set **StartPosition** = **CenterScreen**, or
   - Set the **WindowState** property to a value of **Maximized** – this will cause the parent form to fill the entire display screen on startup.

4. Open **My Project** – set the **StartupForm** property to **VBUniversityParent** – close **My Project**.
Set the form's **Text** property to **VB University**.

Modify Each Child Form

Child forms should open within the parent form such that multiple child forms tile across one another.

5. Open each child form (**CourseGrid**, **StudentGrid**, **StudentEnrollmentGrid**, and **StudentDetails**) – check that the **StartPosition** property = **WindowsDefaultLocation**.

Add a Menu to the Parent Form

6. Add a **menu strip** control to the parent form with the following menu items.

&File &Close E&xit	&Display &Course Grid &Student Grid Student-&Enrollment Grid Student &Details	&Window &Cascade Tile &Horizontally Tile &Vertically

7. Access the **Properties** window for the menu strip control.

8. Select the **MdiWindowListItem** property, set it to a value of **WindowToolStripMenuItem**. This causes the Window top-level menu to display a list of all open forms.

Code the Parent Form

Displaying Forms. Clicking the **Course Grid** menu item will open and display an instance of the **Course** form. This is accomplished in the click event for this menu item by:

1. Declaring an instance of the form. In the code below a generic name for the form is used (**aForm**) – any form name will due.

2. Set the form instance **MdiParent** property to **Me** (the parent form).
3. Execute the **Show** method to show the form – using the **Show** method instead of **ShowDialog** will enable the application user to move between multiple open forms.

9. Add this sub procedure to your program.

```
Private Sub CourseGridToolStripMenuItem_Click(ByVal s
ender As System.Object, ByVal e As System.EventArgs)
HandlesCourseGridToolStripMenuItem.Click
    'Display the form
    'Declare an instance of the CourseGrid form
    Dim aForm As New CourseGrid

    'Assign the MdiParent property the name of the
parent form
    aForm.MdiParent = Me

    'Show the form so that the application user can
switch
    'between any forms that are open
    aForm.Show()
End Sub
```

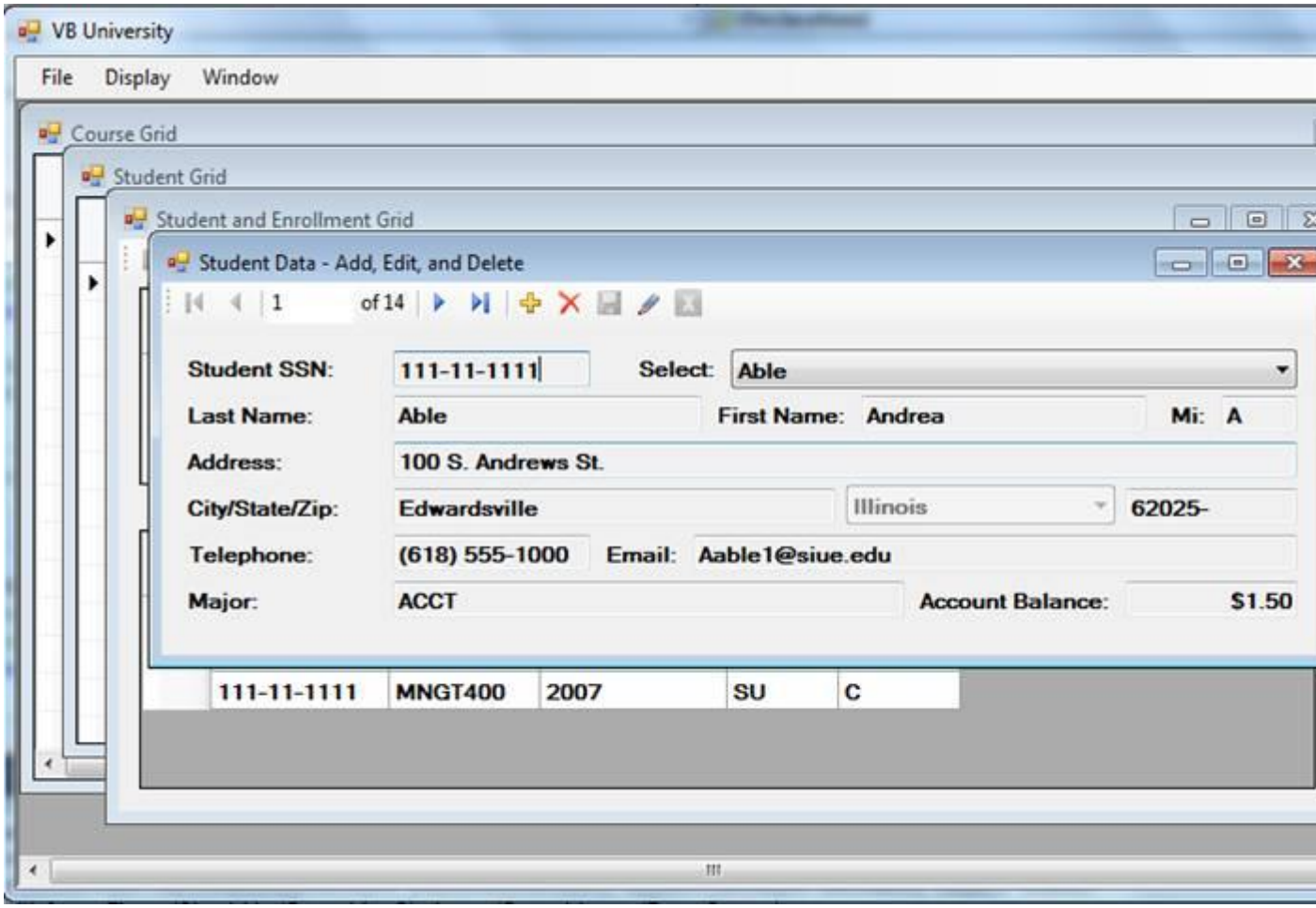
You can use this same approach to code the **Student Grid**, **Student-Enrollment Grid**, and **Student Details** menu items. Note that the same generic form name is used, thus making it easier to copy/past/modify the code.

10. Add these three sub procedures to your program.

```
    Private Sub StudentGridToolStripMenuItem_Click(By
Val sender As System.Object, ByVal e As System.EventA
rgs) HandlesStudentGridToolStripMenuItem.Click
        'Display the form
        'Declare an instance of the StudentGrid form
        Dim aForm As New StudentGrid

        'Assign the MdiParent property the name of
the parent form
        aForm.MdiParent = Me
```

```
        'Show the form so that the application user
can switch
        'between any forms that are open
        aForm.Show()
    End Sub

    Private Sub StudentEnrollmentGridToolStripMenuIte
m_Click(ByVal sender As System.Object, ByVal e As Sys
tem.EventArgs) HandlesStudentEnrollmentGridToolStripM
enuItem.Click
        'Display the form
        'Declare an instance of the
StudentEnrollmentGrid form
        Dim aForm As New StudentEnrollmentGrid

        'Assign the MdiParent property the name of
the parent form
        aForm.MdiParent = Me

        'Show the form so that the application user
can switch
        'between any forms that are open
        aForm.Show()
    End Sub

    Private Sub StudentDetailsToolStripMenuItem_Click
(ByVal sender As System.Object, ByVal e As System.Eve
ntArgs) HandlesStudentDetailsToolStripMenuItem.Click
        'Display the form
        'Declare an instance of the StudentDetails
form
        Dim aForm As New StudentDetails

        'Assign the MdiParent property the name of
the parent form
        aForm.MdiParent = Me

        'Show the form so that the application user
can switch
        'between any forms that are open
        aForm.Show()
    End Sub
```

11. Run the project. Note that this approach enables opening multiple instances of a form. This figure shows one of each of the forms open, but it is easy to also display multiple instances of the **CourseGrid** form and other forms—try it!



Code Windows Layout Options

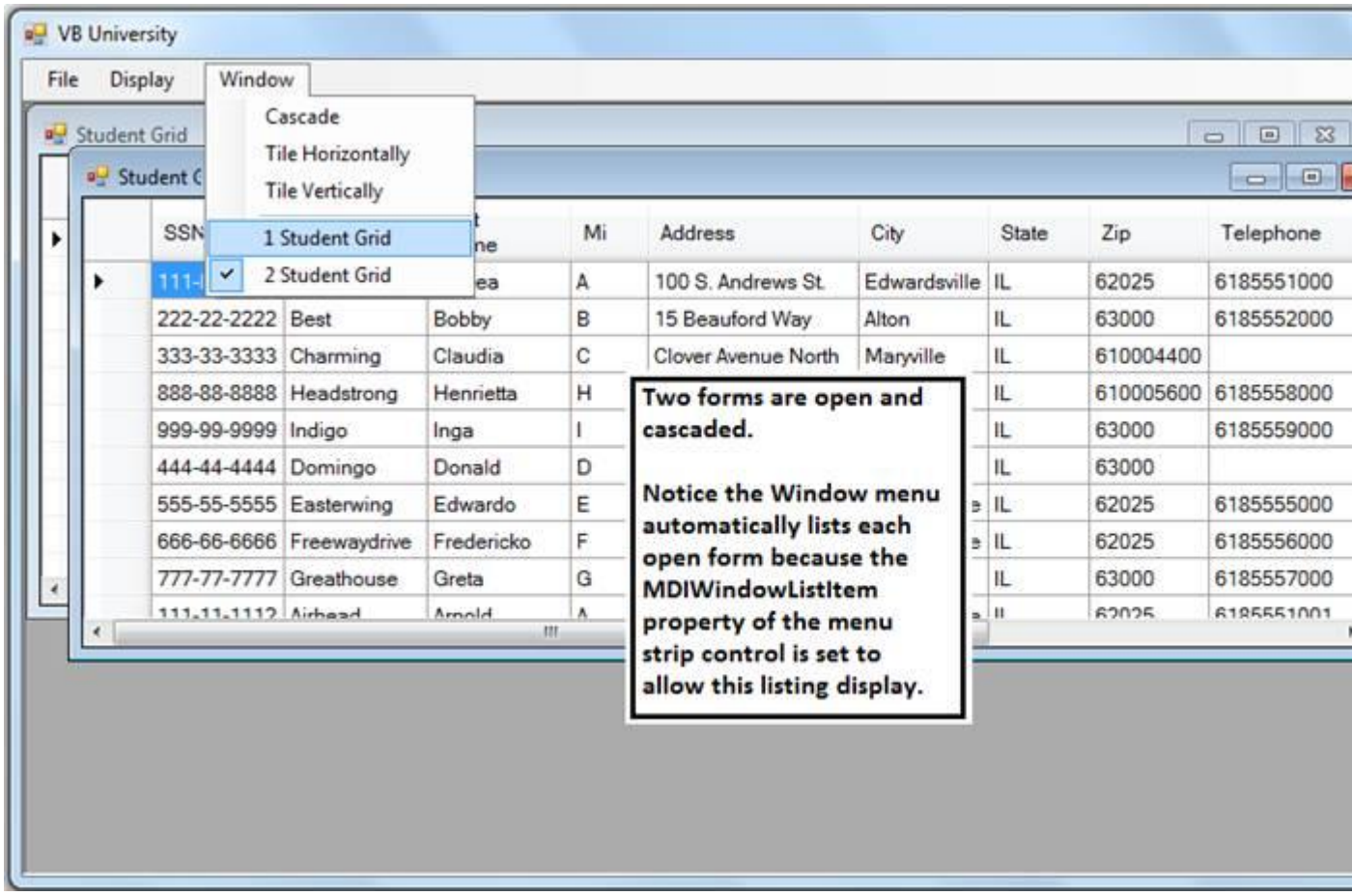
The code for the menu items within the **Window** top-level menu item is shown here. The **LayoutMDI** method is used along with the **MdiLayout** enumeration and its enumerated values to arrange the display of open forms (windows).

12. Add the code for these three sub procedures to the program.

```
Private Sub CascadeToolStripMenuItem_Click(ByVal send
er As System.Object, ByVal e As System.EventArgs) Han
dlesCascadeToolStripMenuItem.Click
    'Arrange open forms by cascading
    Me.LayoutMdi(MdiLayout.Cascade)
```

```
End Sub

Private Sub TileHorizontallyToolStripMenuItem_Click(B
yVal sender As System.Object, ByVal e As System.Event
Args) HandlesTileHorizontallyToolStripMenuItem.Click
    'Arrange open forms by tiling horizontally
    Me.LayoutMdi(MdiLayout.TileHorizontal)
End Sub

Private Sub TileVerticallyToolStripMenuItem_Click(ByV
al sender As System.Object, ByVal e As System.EventAr
gs) HandlesTileVerticallyToolStripMenuItem.Click
    'Arrange open forms by tiling vertically
    Me.LayoutMdi(MdiLayout.TileVertical)
End Sub
```

This figure shows two forms open in windows arranged for display by cascading. Note the **Window** menu displays a listing of each form automatically due to setting the **MdiWindowListItem** property of the menu strip control to the **Window** menu item.



Code File Close and Exit Menu Options

The code for the **Close** menu item under the **File** menu is shown here.

- The **Me.ActiveMdiChild** property is checked to determine if there is an active child form with the **If** statement. This is done by comparing the value of the property to the keyword **Is Nothing**.
- The **Me.ActiveMdiChild.Close** method closes the currently selected (active) child form.

```
Private Sub CloseToolStripMenuItem_Click(ByVal sender
 As System.Object, ByVal e As System.EventArgs) Handl
esCloseToolStripMenuItem.Click
    'Close the currently selected active child
form.  Check to
    'determine if a child form is active.
    If Not Me.ActiveMdiChild Is Nothing Then
        Me.ActiveMdiChild.Close()
    End If
End Sub
```

The code for the **Exit** menu item under the **File** menu is straight-forward – all open child forms will automatically close.

```
Private Sub ExitToolStripMenuItem_Click(ByVal sender
As System.Object, ByVal e As System.EventArgs) Handle
s ExitToolStripMenuItem.Click
    'Close the main form - this automatically closes
all child forms.
    Me.Close()
End Sub
```

13. Add the code for the **Close** and **Exit ToolStripMenuItem** click event sub procedures to the program.

You can provide a “polished” touch to the program by adding program comments for each form. Each form must have comments identifying the project, programmer, date the form was developed, form identifier, and **Option Strict On**.

You should also organize the code for each form by using **Region** statements.

You can continue to develop the MDI layout by adding additional tools to the parent form including toolbars with the tool strip control and status bars with the status strip control, among other controls. Many of these features are covered in our advanced VB course.

This completes the Chapter 10 project.

Solution to In-Class Exercise

This solution shows the Ch10VBUniversity exercise code organized into regions.

Parent Form

```
'Project: Ch10VBUniversity-AccessVersion
'D. Bock
'Today's Date

Option Strict On

Public Class VBUniversityParent

#Region " File Menu Events "

    Private Sub CloseToolStripMenuItem_Click(ByVal sender As System.Object,
 ByVal e As System.EventArgs) Handles CloseToolStripMenuItem.Click
        'Close the currently selected active child form.  Check to
        'determine if a child form is active.
        If Not Me.ActiveMdiChild Is Nothing Then
            Me.ActiveMdiChild.Close()
        End If
    End Sub

    Private Sub ExitToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles ExitToolStripMenuItem.Click
        'Close the main form - this automatically closes all child forms.
        Me.Close()
    End Sub

#End Region

#Region " Display Menu Events "

    Private Sub CourseGridToolStripMenuItem_Click(ByVal sender As System.Ob
ject, ByVal e As System.EventArgs) Handles CourseGridToolStripMenuItem.Clic
k
        'Display the form
        'Declare an instance of the CourseGrid form
        Dim aForm As New CourseGrid
```

```
        'Assign the MdiParent property the name of the parent form
        aForm.MdiParent = Me

        'Show the form so that the application user can switch
        'between any forms that are open
        aForm.Show()
    End Sub

    Private Sub StudentGridToolStripMenuItem_Click(ByVal sender As System.O
bject, ByVal e As System.EventArgs) Handles StudentGridToolStripMenuItem.Cl
ick
        'Display the form
        'Declare an instance of the StudentGrid form
        Dim aForm As New StudentGrid

        'Assign the MdiParent property the name of the parent form
        aForm.MdiParent = Me

        'Show the form so that the application user can switch
        'between any forms that are open
        aForm.Show()
    End Sub

    Private Sub StudentEnrollmentGridToolStripMenuItem_Click(ByVal sender A
s System.Object, ByVal e As System.EventArgs) Handles StudentEnrollmentGrid
ToolStripMenuItem.Click
        'Display the form
        'Declare an instance of the StudentEnrollmentGrid form
        Dim aForm As New StudentEnrollmentGrid

        'Assign the MdiParent property the name of the parent form
        aForm.MdiParent = Me

        'Show the form so that the application user can switch
        'between any forms that are open
        aForm.Show()
    End Sub

    Private Sub StudentDetailsToolStripMenuItem_Click(ByVal sender As Syste
m.Object, ByVal e As System.EventArgs) Handles StudentDetailsToolStripMenuI
tem.Click
        'Display the form
        'Declare an instance of the StudentDetails form
        Dim aForm As New StudentDetails

        'Assign the MdiParent property the name of the parent form
        aForm.MdiParent = Me

        'Show the form so that the application user can switch
        'between any forms that are open
        aForm.Show()
    End Sub

#End Region

#Region " Window Menu Events "

    Private Sub CascadeToolStripMenuItem_Click(ByVal sender As System.Objec
t, ByVal e As System.EventArgs) Handles CascadeToolStripMenuItem.Click
        'Arrange open forms by cascading
        Me.LayoutMdi(MdiLayout.Cascade)
```

```
    End Sub

    Private Sub TileHorizontallyToolStripMenuItem_Click(ByVal sender As Sys
tem.Object, ByVal e As System.EventArgs) Handles TileHorizontallyToolStripM
enuItem.Click
        'Arrange open forms by tiling horizontally
        Me.LayoutMdi(MdiLayout.TileHorizontal)
    End Sub

    Private Sub TileVerticallyToolStripMenuItem_Click(ByVal sender As Syste
m.Object, ByVal e As System.EventArgs) Handles TileVerticallyToolStripMenuI
tem.Click
        'Arrange open forms by tiling vertically
        Me.LayoutMdi(MdiLayout.TileVertical)
    End Sub

#End Region

End Class
```

CourseGrid form

```
'Project: Ch10VBUniversity-AccessVersion
'D. Bock
'Today's Date

Option Strict On

Public Class CourseGrid

    Private Sub CourseGrid_Load(ByVal sender As System.Object, ByVal e As S
ystem.EventArgs) Handles MyBase.Load
        Try
            'Load Course table
            Me.CourseTableAdapter.Fill(Me.CourseGridDataSet.Course)
        Catch ex As Exception
            Dim MessageString As String = "Report this error to the system
administrator: " & ControlChars.NewLine & ex.Message
            Dim TitleString As String = "Course Data Load Failed"
            MessageBox.Show(MessageString, TitleString,
MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End Sub

End Class
```

Student Grid Form

```
'Project: Ch10VBUniversity-AccessVersion
'D. Bock
'Today's Date

Option Strict On

Public Class StudentGrid
```

```
    Private Sub StudentGrid_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'Load the Student table
        Try
            Me.StudentTableAdapter.Fill(Me.StudentGridDataSet.Student)
        Catch ex As Exception
            Dim MessageString As String = "Report this error to the system
administrator: " & ControlChars.NewLine & ex.Message
            Dim TitleString As String = "Student Data Load Failed"
            MessageBox.Show(MessageString, TitleString,
MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End Sub

End Class
```

StudentEnrollment Grid Form

```
'Project: Ch10VBUniversity-AccessVersion
'D. Bock
'Today's Date

Option Strict On

Public Class StudentEnrollmentGrid

    Private Sub StudentEnrollmentGrid_Load(ByVal sender As System.Object, B
yVal e As System.EventArgs) Handles MyBase.Load
        'Fill the StudentEnrollmentDataSet
        Try
            Me.StudentTableAdapter.Fill(Me.StudentEnrollmentDataSet.Student
)
            Me.EnrollmentTableAdapter.Fill(Me.StudentEnrollmentDataSet.Enro
llment)
        Catch ex As Exception
            Dim MessageString As String = "Report this error to the system
administrator: " & ControlChars.NewLine & ex.Message
            Dim TitleString As String = "Student or Enrollment Data Load
Failed"
            MessageBox.Show(MessageString, TitleString,
MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End Sub

End Class
```

Student Details Form

```
'Ch10VBUniverrsity - StudentDetails Form
'D. Bock
'Today's Date

Option Strict On

Public Class StudentDetails
```

```
    Private Sub StudentBindingNavigatorSaveItem_Click(ByVal sender As System.Object, B
yVal e As System.EventArgs) Handles StudentBindingNavigatorSaveItem.Click
        'Trap any exceptions during student table update
        Try
            If ValidData() Then
                'Me.Validate()
                Me.StudentBindingSource.EndEdit()
                Me.TableAdapterManager.UpdateAll(Me.StudentDetailsDataSet)

                'Call sub procedure to enable BindingNavigator controls
                'by sending a parameter value True
                Me.SetControls(True)
            End If
        Catch ex As Exception
            Dim MessageString As String = "Report this error to the system
administrator: " & ControlChars.NewLine & ex.Message
            Dim TitleString As String = "Error During Save Operation"
            MessageBox.Show(MessageString,
TitleString, MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End Sub

    Private Sub StudentDetails_Load(ByVal sender As System.Object, ByVal e As System.E
ventArgs) Handles MyBase.Load
        'Trap exceptions that occur during data load
        Try
            Me.StatesTableAdapter.Fill(Me.StatesDataSet.States)
            Me.StudentTableAdapter.Fill(Me.StudentDetailsDataSet.Student)
        Catch ex As Exception
            Dim MessageString As String = "Report this error to the system
administrator: " & ControlChars.NewLine & ex.Message
            Dim TitleString As String = "Student Details Data Load Failed"
            MessageBox.Show(MessageString,
TitleString, MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End Sub

    Private Sub SetControls(ByVal ValueBoolean As Boolean)
        'This sub procedure sets the user interface for the
        'BindingNavigator control and bound controls for Edit/Add
        'operations

        'ReadOnly/Not ReadOnly the bound controls
        SSNMaskedTextBox.ReadOnly = ValueBoolean
        LastNameTextBox.ReadOnly = ValueBoolean
        FirstNameTextBox.ReadOnly = ValueBoolean
        MiddleInitialTextBox.ReadOnly = ValueBoolean
        AddressTextBox.ReadOnly = ValueBoolean
        CityTextBox.ReadOnly = ValueBoolean
        ZipMaskedTextBox.ReadOnly = ValueBoolean
        PhoneMaskedTextBox.ReadOnly = ValueBoolean
        EmailAddressTextBox.ReadOnly = ValueBoolean
        MajorTextBox.ReadOnly = ValueBoolean
        AccountBalanceTextBox.ReadOnly = ValueBoolean

        'Remark out next line when StateComboBox replaces
        'the StateCodeTextBox and add the line of
        'code to enable the StateComboBox
        'StateCodeTextBox.ReadOnly = ValueBoolean
        StateComboBox.Enabled = Not ValueBoolean

        'Make the Move, Position, and Buttons
```

```
        '(except Save and Cancel) Invisible
        BindingNavigatorMoveFirstItem.Visible = ValueBoolean
        BindingNavigatorMoveLastItem.Visible = ValueBoolean
        BindingNavigatorMoveNextItem.Visible = ValueBoolean
        BindingNavigatorMovePreviousItem.Visible = ValueBoolean
        BindingNavigatorPositionItem.Visible = ValueBoolean
        BindingNavigatorCountItem.Visible = ValueBoolean
        BindingNavigatorAddNewItem.Visible = ValueBoolean
        BindingNavigatorDeleteItem.Visible = ValueBoolean
        EditToolStripButton.Visible = ValueBoolean

        'Enable/disable the Save and Cancel Buttons
        StudentBindingNavigatorSaveItem.Enabled = Not ValueBoolean
        CancelToolStripButton.Enabled = Not ValueBoolean

        'Enable/disable ComboBox
        LastNameComboBox.Enabled = ValueBoolean
    End Sub

    Private Sub EditToolStripButton_Click(ByVal sender As System.Object, ByVal e As Sy
stem.EventArgs) Handles EditToolStripButton.Click
        'Call SetControls with False to alter the form to
        'allow editing a data row
        SetControls(False)
    End Sub

    Private Sub CancelToolStripButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles CancelToolStripButton.Click
        'Cancel the operation
        StudentBindingSource.CancelEdit()

        'Call SetControls with True to alter the form to
        'make the form ReadOnly
        SetControls(True)
    End Sub

    Private Sub BindingNavigatorAddNewItem_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles BindingNavigatorAddNewItem.Click
        'Call SetControls with False to alter the form to
        'allow adding a data row
        SetControls(False)

        'Set focus
        SSNMaskedTextBox.Focus()
    End Sub

    Private Sub BindingNavigatorDeleteItem_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles BindingNavigatorDeleteItem.Click
        'Delete the row if there is no relationship to
        'existing data rows in the ENROLLMENT table

        'Store the current DataSet position in case the deletion fails
        Dim RowNumberInteger As Integer = StudentBindingSource.Position

        Try
            Dim ResponseDialogResult As DialogResult = MessageBox.Show("Confirm to
delete the student record.", "Delete
Y/N?", MessageBoxButtons.YesNo, MessageBoxIcon.Question, MessageBoxDefaultButton.Butto
n2)

            If ResponseDialogResult = Windows.Forms.DialogResult.Yes Then
                'Delete the row by removing the current record,
```

```
                'ending the edit, and calling the Update method
                StudentBindingSource.RemoveCurrent()
                StudentBindingSource.EndEdit()
                TableAdapterManager.UpdateAll(StudentDetailsDataSet)
            End If

        Catch exOleDb As OleDb.OleDbException
            'The deletion attempt failed due to a relationship
            'to existing data rows in the ENROLLMENT table

            'Restore the deleted row with the RejectChanges method
            StudentDetailsDataSet.RejectChanges()

            'Reposition to the row that was deleted
            StudentBindingSource.Position = RowNumberInteger

            'Display appropriate error message
            MessageBox.Show("This student cannot be deleted - the student is enrolled
in courses." & ControlChars.NewLine & exOleDb.Message, "Delete Operation
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)

        Catch ex As Exception
            'Some other exception was triggered
            MessageBox.Show("Unexpected error in delete operation:
" & ControlChars.NewLine & ex.Message, "Delete Operation
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End Sub

    Private Function ValidData() As Boolean
        'Initialize function return value
        ValidData = False
        Dim MessageString As String

        'Test SSN is correct length
        'Test existence of values in each textbox
        If SSNMaskedTextBox.MaskCompleted = False Then
            'The SSN value is not completes
            MessageString = "SSN value is not complete."
            MessageBox.Show(MessageString, "Data
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            SSNMaskedTextBox.Focus()
            SSNMaskedTextBox.SelectAll()
        ElseIf LastNameTextBox.Text.Trim = String.Empty Then
            'Validate Last Name
            MessageString = "Last name is required."
            MessageBox.Show(MessageString, "Data
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            LastNameTextBox.Focus()
            LastNameTextBox.SelectAll()
        ElseIf FirstNameTextBox.Text.Trim = String.Empty Then
            'Validate First Name
            MessageString = "First name is required."
            MessageBox.Show(MessageString, "Data
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            FirstNameTextBox.Focus()
            FirstNameTextBox.SelectAll()
        ElseIf AddressTextBox.Text.Trim = String.Empty Then
            'Validate Address
            MessageString = "Address is required."
            MessageBox.Show(MessageString, "Data
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
```

```
            AddressTextBox.Focus()
            AddressTextBox.SelectAll()
        ElseIf CityTextBox.Text.Trim = String.Empty Then
            'Validate City
            MessageString = "City is required."
            MessageBox.Show(MessageString, "Data
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            CityTextBox.Focus()
            CityTextBox.SelectAll()
        ElseIf StateComboBox.SelectedIndex = -1 Then
            'Validate State
            MessageString = "State is required."
            MessageBox.Show(MessageString, "Data
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            StateComboBox.Focus()
        ElseIf ZipMaskedTextBox.MaskCompleted = False Then
            'Zip code required
            MessageString = "Zip code is incomplete."
            MessageBox.Show(MessageString, "Data
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            ZipMaskedTextBox.Focus()
            ZipMaskedTextBox.SelectAll()
        ElseIf AccountBalanceTextBox.Text.Trim
<> String.Empty AndAlso IsNumeric(AccountBalanceTextBox.Text) = False Then
            'Validate Account Balance is numeric if there is a value stored here
            MessageString = "Account balance must be a numeric amount."
            MessageBox.Show(MessageString, "Data
Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
            AccountBalanceTextBox.Focus()
            AccountBalanceTextBox.SelectAll()
        Else
            'All of the data is valid
            ValidData = True
        End If
    End Function

    Private Sub AccountBalanceTextBox_KeyPress(ByVal sender As Object, ByVal e As Syst
em.Windows.Forms.KeyPressEventArgs) Handles AccountBalanceTextBox.KeyPress
        'Allow Backspace (8), numeric keys (48 to 57), comma (44),
        'decimal point (46)and dollar sign (36)
        Select Case Asc(e.KeyChar)
            Case 8, 36, 44, 46, 48 To 57
                e.Handled = False  'Allow the key
            Case Else
                e.Handled = True   'Ignore the key
        End Select
    End Sub

End Class
```

END OF NOTES