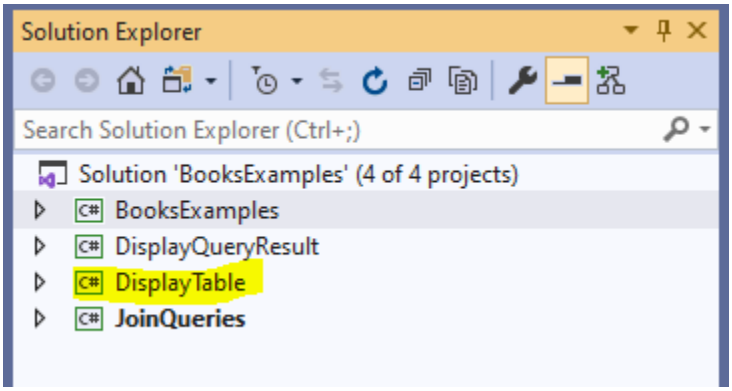
Note: This example exercise is based on using **BooksExamples** library project (created in earlier tutorial- Refer file - *Creating-EF-DataModel-Library Project.pdf* posted on D2L/Blackboard) to the project – **DisplayTable** as highlighted below.

Note - Pre-requisite: BooksExamples Library project must be created.



Exercise 01:

>> Creating a Win Form Project and confirm it to use the Entity Data Model.

Step 01: Creating a project and add it to the existing solution.

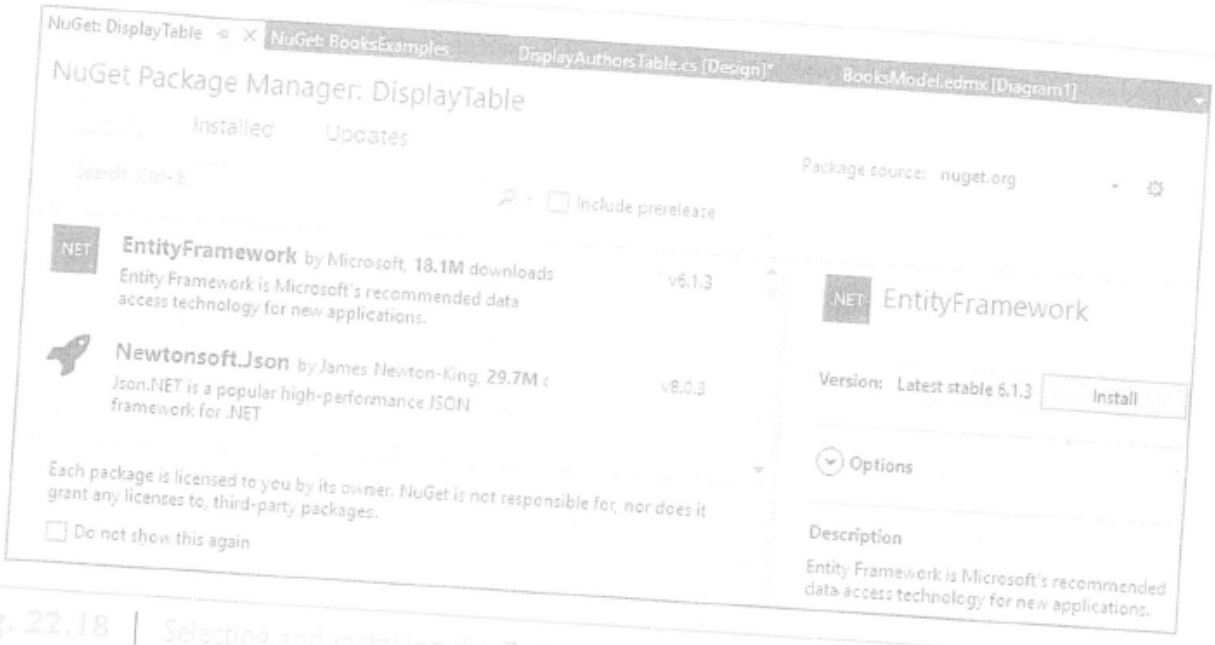
1. Right click Solution 'BooksExamples' (the solution name) in Solution Explorer and select Add > New Project... to display the Add New Project dialog.
2. Select Windows Forms Application from the Visual C# > Windows > Classic Desktop category, name the project `DisplayTable` and click OK.
3. Change the name of the `Form1.cs` source file to `DisplayAuthorsTable.cs`. The IDE updates the `Form`'s class name to match the source file. Set the `Form`'s Text property to `Display Authors Table`.
4. Right click the `DisplayTable` project's name in the Solution Explorer, then select Set as Startup Project to configure the solution so that project `DisplayTable` will execute when you select Debug > Start Debugging (or press *F5*).

Step 02: Adding a reference of the **BooksExamples** class library.

1. Right click the `DisplayTable` project's **References** node in the **Solution Explorer** and select **Add Reference...**.
2. In the left column of the **Reference Manager** dialog that appears, select **Projects** to display the other projects in this solution, then in center of the dialog ensure that the checkbox next to `BooksExamples` is checked and click **OK**. `BooksExamples` should now appear in the projects **References** node.

Step 03: Adding a reference to Entity Framework

1. Right click the project's name in the **Solution Explorer** and select **Manage NuGet Packages...** to display the **NuGet** tab in Visual Studio's editors area. NuGet is a tool (known as a *package manager*) that that helps you download and manage libraries (known as *packages*) used by your projects.
2. In the dialog that appears, click **Browse**, then select the **EntityFramework by Microsoft** and click **Install** (Fig. 22.18).
3. The IDE will ask you to review the changes. Click **OK**.
4. The IDE will ask you to accept the `EntityFramework` license. Click **I Accept** to complete the installation.



g. 22.18 | Selecting and installing the EntityFramework in the N

Step 4: Adding the Connection String to the Windows Forms App

Each app that will use the entity data model also requires the *connection string* that tells the Entity Framework how to connect to the database. The connection string is stored in the `BooksExamples` class library's `App.Config` file. In the **Solution Explorer**, open the `Books-Examples` class library's `App.Config` file, then copy the `connectionStrings` element (lines 7–9 in our file), which has the format:

```
<connectionStrings>
   Connection string information appears here
</connectionStrings>
```

Next, open the `App.Config` file in the `DisplayTable` project and paste the connection string information *after* the line containing `</entityFramework>` and *before* the line containing `</configuration>`. Save, then close the `App.Config` file.

22.5.3 Data Binding

>> Databinding between the Controls and Entity Data Model

Step 01: Adding a data source for the Authors Table

To use the entity data model classes for the data binding, you must first add them as a data source. To do so:

#1. Select View > Other Windows > Data Sources to display the **Data Sources** window at the left side of the IDE, then in that Window click the **Add New Data Source…** link to display the **Data Sources Configuration Wizard**

#2. The Entity Data Model classes are used to create objects representing the tables in the Database, so we will use an **Object** data source. And follow the dialog windows.

Step 02: Creating GUI elements.

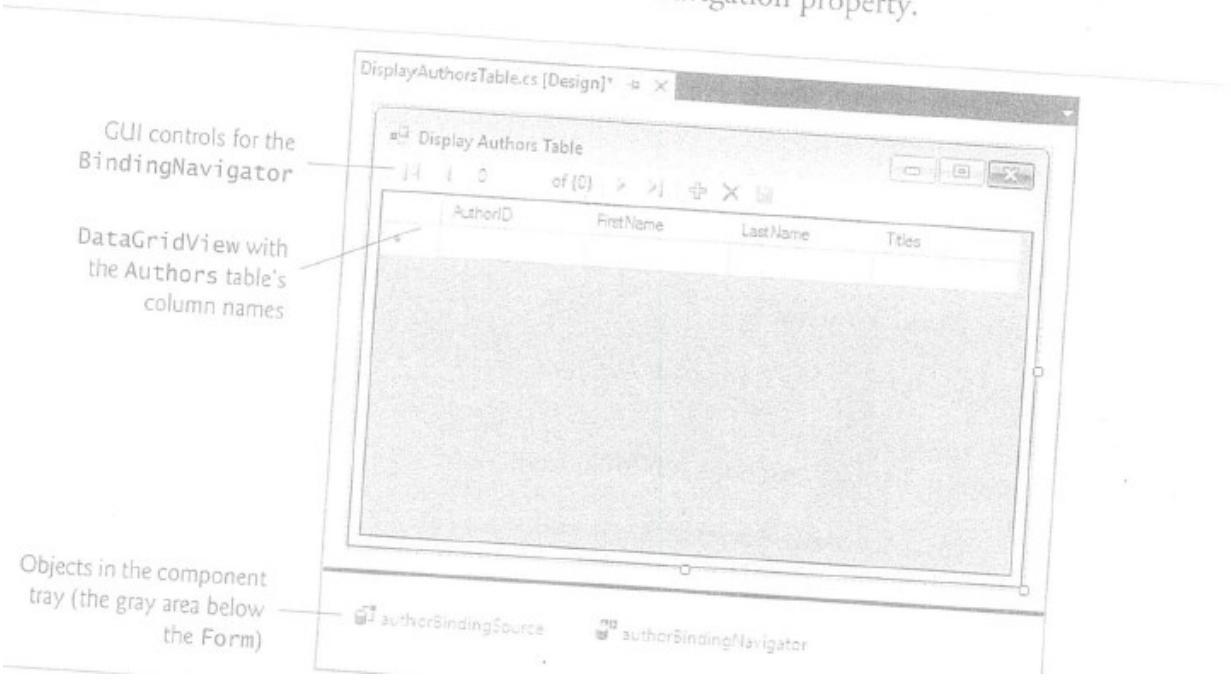
Step 2: Creating GUI Elements

Next, you'll use the **Design** view to create a `DataGridView` control that can display the `Authors` table's data. To do so:

1. Switch to **Design** view for the `DisplayAuthorsTable` class.

2. Click the Author node in the Data Sources window—it should change to a drop-down list. Open the drop-down by clicking the down arrow and ensure that the DataGridView option is selected—this is the GUI control that will be used to display and interact with the data.
3. Drag the Author node from the Data Sources window onto the Form in Design view. You'll need to resize the Form to fit the DataGridView.

The IDE creates a DataGridView (Fig. 22.21) with column names representing all the properties for an Author, including the Titles navigation property.



Step 3: Connecting the Data Source to the `authorBindingSource`

The final step is to connect the data source to the authorBindingSource, so that the app can interact with the database. Figure 22.22 shows the code needed to obtain data from the database and to save any changes that the user makes to the data back into the database.

```
// Displaying data from a database table in a DataGridView.
using System;
using System.Data.Entity;
using System.Data.Entity.Validation;
using System.Linq;
using System.Windows.Forms;

namespace DisplayTable
{
   public partial class DisplayAuthorsTable : Form
   {
      // constructor
      public DisplayAuthorsTable()
      {
         InitializeComponent();
      }

      // Entity Framework DbContext
      private BooksExamples.BooksEntities dbcontext =
         new BooksExamples.BooksEntities();

      // load data from database into DataGridView
      private void DisplayAuthorsTable_Load(object sender, EventArgs e)
      {
         // load Authors table ordered by LastName then FirstName
         dbcontext.Authors
            .OrderBy(author => author.LastName)
            .ThenBy(author => author.FirstName)
            .Load();

         // specify DataSource for authorBindingSource
         authorBindingSource.DataSource = dbcontext.Authors.Local;
      }

      // click event handler for the Save Button in the
      // BindingNavigator saves the changes made to the data
      private void authorBindingNavigatorSaveItem_Click(
         object sender, EventArgs e)
      {
         Validate(); // validate the input fields
         authorBindingSource.EndEdit(); // complete current edit, if any
```

```
43
44          // try to save changes
45          try
46          {
47              dbcontext.SaveChanges(); // write changes to database file
48          }
49          catch(DbEntityValidationException)
50          {
51              MessageBox.Show(
52
53          }                                   );
54      }
55  }
56  }
```