The DataGrid control is used with database recordsets created with the ADO Data control. Simply setting the DataGrid’s DataSource property to an ADO Data control causes the DataGrid to automatically process the associated recordset and display the records in a spreadsheet-like fashion. This includes automatic placement of the field names into the first row of the grid.

Unlike the FlexGrid, where the user is restricted to browsing through the records of a recordset, the DataGrid allows the user to edit fields, and to add or delete records directly within the grid. These additional capabilities are programmable, so the programmer can control whether they are available to the user. However, recordsets created as the result of executing an SQL query may not permit some of the actions.

The ADO Data control is new to VB6 and is not the same as the Data control discussed in Section 8.2. Microsoft now calls the Data control introduced in Section 8.2 the “intrinsic” Data control. The intrinsic Data control is based on DAO (Data Access Object) technology and implements data access using the Microsoft Access Jet Database engine – the same database engine that powers Microsoft Access. Microsoft states that “DAO is suited best for either single-system applications or for small, local deployments.” The DataGrid control is not compatible with the older intrinsic Data control.

The ADO Data control is based on ADO (ActiveX Data Object) technology which is a newer, more flexible technology that includes the capabilities of the DAO technology as well as other capabilities. Microsoft suggests that “because of the flexibility of ADO, it is recommended that new database applications be created using the ADO Data control.” However, with this added flexibility, we also get added complexity in terms of use. In this section we do not attempt to cover all the features of the ADO Data control. Rather, we will introduce the main features that are essential in using the ADO Data control with a Microsoft Access database and the DataGrid control.

Be aware that both the DataGrid and the ADO Data control must be added to the toolbox using the Components… option on the Project menu. The DataGrid tool is called the Microsoft DataGrid Control 6.0 (OLEDB) in the Components Dialog box and the ADO Data control is called the Microsoft ADO Data Control 6.0 (OLEDB).

Appearance and Use of the ADO Data Control

Visually the ADO Data control (Adodc) looks just like the intrinsic Data control. Figure 8.42 identifies the Adodc tool and shows an ADO Data control on the form at design time.

Like the intrinsic Data control, the ADO Data control includes buttons that allow the user to navigate through the records of the recordset associated with the control. Alternatively, the programmer could create separate navigation buttons to perform the tasks of moving through the recordset via the Move or Find methods discussed earlier. In fact, with the exception of some different properties, the ADO Data control supports all the functions of the intrinsic Data control discussed earlier in the chapter.
Properties of the ADO Data Control

Three properties are used to connect an ADO Data control to a MS Access database. These are the CommandType, ConnectionString, and RecordSource properties. The RecordSource property of the ADO Data control is just like the RecordSource property of the intrinsic Data control. That is, it can either specify a table within the database or an SQL query that is processed and produces a recordset as a result of this processing. The CommandType property is used to indicate how the recordset will be created. Table 8.8 shows the possible values of the CommandType property.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 – adCmdText</td>
<td>The ADO Data control expects that processing SQL text associated with the RecordSource property will create the recordset.</td>
</tr>
<tr>
<td>2 – adCmdTable</td>
<td>The ADO Data control expects that the recordset will be created from a table associated with the RecordSource property.</td>
</tr>
<tr>
<td>3 – adCmdStoredProc</td>
<td>The ADO Data control expects that processing an SQL stored procedure associated with the RecordSource property will create the recordset. SQL stored procedures are beyond the scope of this text.</td>
</tr>
<tr>
<td>8 – adCmdUnknown</td>
<td>This value indicates that at the present time, how the recordset will be created cannot be determined. At the time the recordset is created, VB will attempt to determine the source. This option is very general but not as efficient as the other options.</td>
</tr>
</tbody>
</table>

For the purposes of this discussion, we will be creating our recordsets either from a single table within the database (2 – adCmdTable) or by executing an SQL query (1 – adCmdText). The CommandType property must be set at design time although it can be changed later during execution.

The ConnectionString property is the most complicated of the three we are discussing. It is basically a command that tells the ADO Data control where and how to connect to a specific database. For example, if the database is a Microsoft Access database (named EmpDept.mdb) and you want to process it using the Microsoft Access Jet technology, the ConnectionString property might be:

```
Provider=Microsoft.Jet.OLEDB.3.51; Persist Security Info=False;
Data Source=C:\Courses\IS320\Databases\EmpDept.mdb
```

Alternatively, if the database is a database that has been registered as an ODBC (Open Database Connectivity) object and named “employees” in that registry, then the ConnectionString property might be:

```
Provider=MSDASQL.1; Persist Security Info=False;
Data Source=employees
```

While these connection strings look complicated, VB can create them for you if you want to set them at design time. To do so, select the ADO Data control and then click on the ConnectionString property in the Properties window. If you then click on the ellipses in the ConnectionString Property setting, you will see the General tab for the ADO Data control’s Property Pages dialog box (shown in Figure 8.43).
Select the “Use Connection String” option and then click on the Build... button. Figure 8.44 shows the Provider tab for the Data Link Properties dialog box. This is where you select the appropriate provider (the type of the database). If you are going to connect to a Microsoft Access database, you would select the first option (Microsoft Jet 3.51 OLE DB Provider). Note that the specific version number (3.51) may be different than shown in the figure if you have a newer (or older) version installed on your computer.

After selecting the provider, you should click on the Next >> button to move to the Connection tab. At this point either enter the path and name of the actual Microsoft Access database or click on the ellipses button so you can browse your system to find the file. Figure 8.45 shows this dialog box after specifying the database name (including the path).
CHAPTER 8 Accessing Business Data

FIGURE 8.45 Specifying the actual database for the connection

You can click on the Test Connection button at this point to verify that the connection works. When you are done, click Ok to dismiss the Data Link Properties dialog box. Then click on OK again to dismiss the Property Pages dialog box. At this point, the connection string will be placed in the ConnectionString property of the Properties window. Be aware that even if you do not plan to specify the ConnectionString value at design time, you can still go through these steps so that VB shows you how to build the connection string. For example, assume that the connection string created by VB was:

\begin{verbatim}
Provider=Microsoft.Jet.OLEDB.3.51;Persist Security Info=False;
Data Source=C:\Courses\IS320\C08 Additional Info\EmpDept.mdb
\end{verbatim}

Also assume that you did not want to specify this at design time because you wanted to use App.Path to specify the current application directory at run time. If this were the case, you could first have VB build the connection string for you, then you could write it down (or copy it from the Properties window). Following this, you could add the following code to your Form_Load() event:

```vbnet
Dim cs As String
cs = "Provider=Microsoft.Jet.OLEDB.3.51;Persist Security Info=False;" & _
"Data Source=\" & App.Path & \"EmpDept.mdb"
datEmployees.ConnectionString = cs
```

You would then need to clear the value in the ConnectionString property in the Properties window so it is empty at design time. When the Form_Load() event is executed, the connection is established with the assumption that the database (EmpDept.mdb) is stored in the same directory as the application.

Appearance and Use of the DataGrid Control

Visually the DataGrid looks very similar to the FlexGrid. The main difference is an added column on the left side of the grid that is used to indicate either the “current” record (a right arrow is placed next to the current record) or an “empty” record (an asterisk is placed next to the empty record). Figure 8.46 identifies the DataGrid tool and shows a DataGrid control on the form at design time.
When first created, the DataGrid control has two columns and one empty row. An additional left-hand column is initially empty but is the place where either the current record arrow or the empty record indicator would be shown. At runtime the actual number of rows and columns is determined by the DataGrid control when it is bound to the recordset.

**Properties of the DataGrid Control**

The DataGrid is a custom control. In addition to the usual access to properties through the Properties window, custom controls provide a customized property dialog box. Clicking on the ellipses in the (Custom) Property setting causes a Property Pages dialog box to be displayed as shown in Figure 8.47.

The meaning of many of the property settings on the General tab shown in Figure 8.47 are obvious. The three “Allow…” properties enable or disable user editing of the recordset. Placing check marks in these boxes allow the user to add new records, delete existing records, and update (edit) existing records.
The DefColWidth property (default column width) should be set to zero if you want the DataGrid to automatically adjust the column widths according to the data in the recordset. The remaining tabs on the Property Pages dialog box are generally only used in advanced applications. See online help (search phrase “DataGrid control”) for details of the properties on these pages.

You must have an ADO Data control on your form that is bound to a database and a recordset. The DataGrid’s DataSource property must be associated with an ADO Data control in order for it to work properly. If the ADO Data control has its ConnectionString and RecordSource properties set at design time, then you can bind the DataGrid’s DataSource property at design time too. However, if the ConnectionString and RecordSource properties are set at run time, then you must wait until run time to set the DataGrid’s DataSource property. In addition, you must use the Set statement to define the DataSource at run time. For example, if you have a DataGrid named “dbgEmps” and you want to set its DataSource property to an ADO Data control named “datEmpDB”, you would use the following statement:

```vbscript
Set dbgEmps.DataSource = datEmpDB
```

The ADO Data control is an object, not a variable (see Chapter 10 for more information on working with objects).

### Convention

Use the prefix “dbg” for names of DataGrid controls.

There are many events associated with the DataGrid control. Some are familiar, like the Click, KeyPress, and LostFocus events. Others, like the BeforeColEdit, AfterColEdit, and RowColChange events, are designed to detect changes in the contents of the grid and can be used for data validation and correction. These additional events are beyond the scope of this text. If you are interested in investigating them, search online help (search phase “DataGrid control”) for details.

The examples that follow demonstrate how easy it is to use the DataGrid for routine tasks. The database used in the examples is a Microsoft Access database named EmpDept.mdb and is shown in Figure 8.48. The first example uses a single table in the database and the second example combines the tables using an SQL Select statement.

### Example 8.7 Using the DataGrid Control With A Single Table

In this example, we use the DataGrid to display the contents of the Employee table. The form and code are shown in Figure 8.49. The grid’s AllowUpdates property is set to True, so the user will be able to edit existing records, but the AllowAddNew and AllowDelete properties are both set to False so the user will be unable to add or delete records. The only code for the example is the Form_Load event procedure, which defines the ADO Data control’s ConnectionString and RecordSource properties. This event procedure also sets the DataSource property for the DataGrid. It should also be noted that at design time, the ADO Data control’s ConnectionString and RecordSource properties were left empty and the ADO Data control’s CommandType property was set to 2 – adCmdTable.
The project is shown at run time in Figure 8.50. In this figure the user is in the middle of changing employee number 100’s phone number.

Returning to the design mode, we change the AllowAddNew and AllowDelete property settings to True. Figure 8.51 shows the project at run time after this modification. Here, the user is entering a new employee (employee number 160). Notice the asterisk in the last row indicating a new record can be added to it.

To delete an existing record the user clicks on the left-most column of the desired row and presses the DELETE key. Be aware, however, that there is no automatic “undo” of this delete. Providing such an undo feature requires writing code; this is an example of a task for which some of the DataGrid’s advanced events are useful.
Example 8.8 Using the DataGrid Control With An SQL Query

In this example, we combine the records from the two tables using an SQL Select query. The form at design time and code are shown in Figure 8.52. The ADO Data control’s CommandType property was set to 1 – adCmdText. That’s all that is needed to make the project work.

The project at run time is shown in Figure 8.53. In this example, the AllowAddNew, AllowDelete, and AllowUpdate properties should be set with care. In Figure 8.53, all three have been set to False. If the properties are set to True, then certain user actions could cause run-time errors. For example, if AllowUpdate was set to True, then the user could change the values of EmpNo or EmpName fields. However, if the user tries to change the value of the DeptName field, a run-time error is generated stating “Insufficient key field information for updating or refreshing”. This problem occurs because the data displayed in the DataGrid is the result of combining data from two tables. This will be a potential problem whenever the SQL Select statement refers to two or more tables.