

1

Migrating from ADO to ADO.NET

This chapter is an introduction to ADO.NET 2.0 for Visual Basic 6 developers who've decided to bite the bullet and move to Microsoft .NET Framework 2.0, Visual Studio 2005 (VS 2005) or Visual Basic Express (VBX), and Visual Basic 2005 (VB 2005) as their programming language. The ADO.NET 2.0 code examples and sample projects described in the chapter have the following prerequisites:

- ❑ Experience with VB6 database programming, using the Data Environment Designer, and writing code to create and manipulate ADODB Connection, Command, and Recordset objects, including disconnected Recordsets and databound controls.
- ❑ A basic understanding of the organization and use of .NET Framework namespaces and classes.
- ❑ Sufficient familiarity with using the VS 2005 IDE and writing VB 2005 code to create simple Windows Form projects.
- ❑ Microsoft SQL Server 2000 or 2005 Developer edition or higher, MSDE 2000, or SQL Server Express (SQLX) installed on your development computer or accessible from a network location. Access 2000 or later for Jet 4.0 examples is optional.
- ❑ The Northwind sample database installed on an accessible SQL Server instance.
- ❑ A working knowledge of XML document standards, including some familiarity with XML schemas.

If you have experience with ADO.NET 1.x, consider scanning this chapter for new ADO.NET 2.0 features and then continue with Chapter 2, "Introducing New ADO.NET 2.0 Features," for more detailed coverage.

One of Microsoft's objectives for VS 2005 is to minimize the trauma that developers experience when moving from VB6 and VBA to the .NET Framework 2.0 and VB 2005. Whether VS .NET 2005's VB-specific `My` namespace and its accouterments will increase the rate of VB6 developer migration to VB 2005 remains to be seen. What's needed to bring professional VB6 database developers to the third iteration of the .NET Framework and Visual Studio's .NET implementation is increased programming productivity, application or component scalability and performance, and code reusability.

This chapter begins by demonstrating the similarities of VB6 and VBA code to create ADODB objects and VB 2005 code to generate basic ADO.NET 2.0 objects—database connections, commands, and read-only resultsets for Windows form projects. Native ADO.NET data provider classes—especially `SqlConnection` for SQL Server—provide substantially better data access performance than ADODB and its OLE DB data providers. The remaining sections show you multiple approaches for creating ADO.NET DataSets by using new VS 2005 features and wizards to generate the underlying read-write data objects for you automatically. DataSets demonstrate VS 2005's improved data access programming productivity and ADO.NET 2.0's contribution to application scalability.

A New Approach to Data Access

Microsoft designed ADO.NET to maximize the scalability of data-intensive Windows and Web form applications and .NET components. Scalability isn't a critical factor when your project involves a few Windows form clients retrieving and updating tables in a single database. High-traffic Web sites, however, require the ability to *scale up* by adding more processors and RAM to a single server or to *scale out* by adding more application servers to handle the data processing load. Managed ADO.NET code that minimizes the duration and number of concurrent database server connections and uses optimistic concurrency tests for updating tables is the key to achieving a scalable data-intensive .NET project.

The sections that follow explain the role of ADO.NET 2.0 namespaces and managed data providers, which form the foundation of .NET 2.0 data access operations.

The System.Data Namespace

The .NET Framework 2.0 `System.Data` namespace contains all ADO.NET 2.0 namespaces, classes, interfaces, enumerations, and delegates. Figure 1-1 shows Object Browser displaying the `System.Data` namespaces.

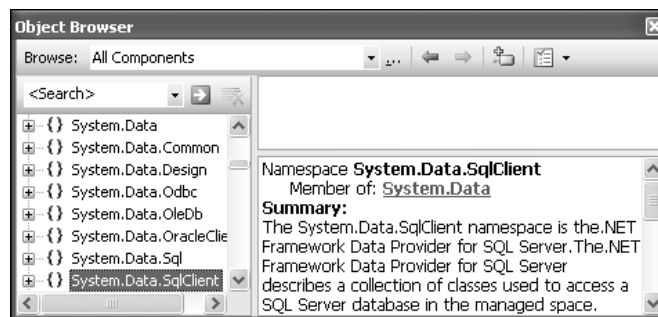


Figure 1-1

VS 2005 doesn't add a reference to the System.Data.dll assembly automatically when you start a new Windows form project. Creating a new data source with the Data Source Configuration Wizard adds references to the System.Data and System.Xml namespaces. The section "Add a Typed DataSet from an SQL Server Data Source," later in this chapter, describes how to use the Data Source Configuration Wizard.

ADO.NET SqlConnection and SqlCommand objects correspond to ADODB.Connection and ADODB.Command objects, but are restricted to use with SQL Server databases. Following are the ADO.NET namespace hierarchies for SqlConnection- and SqlCommand-managed data provider objects; namespaces new in ADO.NET 2.0 are emphasized:

```

System.Object
  System.MarshalByRefObject
    System.ComponentModel.Component
      System.Data.Common.DbConnection
        System.Data.SqlClient.SqlConnection

System.Object
  System.MarshalByRefObject
    System.ComponentModel.Component
      System.Data.Common.DbCommand
        System.Data.SqlClient.SqlCommand
    
```

The following table provides brief descriptions of the System.Data namespaces shown in Figure 1-1 with the namespaces in the preceding hierarchy listed in order.

Namespace	Description
System.Object	The root of the .NET Framework 2.0 type hierarchy (member of System).
System.MarshalByRefObject	Enables remoting of data objects across application domain boundaries (member of System).
System.ComponentModel	Supports object sharing between components and enables runtime and design-time implementations of components.
System.Data	Provides the base classes, interfaces, enumerations, and event handlers for all supported data sources — primarily relational data and XML files or streams.
System.Data.Common	Provides classes that all managed data providers share, such as DbConnection and DbCommand in the preceding hierarchy list.
System.Data.Common.DbConnection	Provides inheritable classes for technology-specific and vendor-specific data providers (new in ADO.NET 2.0).

Table continued on following page

Namespace	Description
<code>System.Data.Odbc</code> , <code>System.Data.OleDb</code> , <code>System.Data.OracleClient</code> , <code>System.Data.SqlClient</code> , and <code>System.Data.SqlCeClient</code>	Namespaces for the five managed data providers included in ADO.NET 2.0; the next section describes these namespaces.
<code>System.Data.SqlTypes</code>	Provides a class for each SQL Server data type, including SQL Server 2005's new <code>xml</code> data type; these classes substitute for the generic <code>DbType</code> enumeration that supports all data providers.
<code>System.XML</code>	Adds the <code>System.Xml.XmlDataDocument</code> class, which supports processing of structured XML documents by <code>DataSet</code> objects.

After you add a project reference to `System.Data.dll`, you can eliminate typing `System.Data` namespace qualifiers and ensure strict type checking by adding the following lines to the top of your class code:

```
Option Explicit On
Option Strict On
Imports System.Data
Imports System.Data.SqlClient
```

Specifying `Option Explicit On` and `Option Strict On` in the Options dialog's Projects and Solutions, VB Defaults page doesn't ensure that other developers who work with your code have these defaults set. Substitute `Imports System.Data.OleDb` for `Imports System.Data.SqlClient` if you're using the `OleDb` data provider.

ADO.NET Data Providers

ADO.NET-managed data providers and their underlying data objects form the backbone of .NET data access. The data providers are an abstraction layer for data services and are similar in concept to ActiveX Data Objects' `ADODB` class, which supports only OLE DB data providers. ADO.NET supports multiple data provider types by the following data provider namespaces:

- ❑ `SqlClient` members provide high performance connectivity to SQL Server 7.0, 2000, and 2005. The performance gain comes from bypassing the OLE DB layer and communicating with SQL Server's native Tabular Data Stream (TDS) protocol. Most of this book's examples use classes in the `SqlClient` namespace.
- ❑ `SqlClientCe` provides features similar to `SqlClient` for SQL Server CE 3.0 and SQL Server 2005 Mobile Edition. This book doesn't cover SQL Server CE or Mobile versions.
- ❑ `OracleClient` members deliver functionality similar to `SqlClient` for Oracle 8i and 9i databases. Oracle offers Oracle Data Provider for .NET (ODP .NET) as a substitute for `OracleClient`; ODP .NET also supports Oracle 10g and later. You can learn more about ODP .NET at <http://otn.oracle.com/tech/windows/odpnet/>.

- ❑ `OleDb` members provide a direct connection to COM-based OLE DB data providers for databases and data sources other than SQL Server, SQL Server CE, and Oracle. You can select from 19 built-in OLE DB data providers when creating a new `OleDbConnection` object. A few of this book's examples use the Microsoft Jet 4.0 OLE DB Data Provider with the Access 2000 or later Northwind.mdb file. ADO.NET 2.0 doesn't provide access to the Microsoft OLE DB Provider for ODBC Drivers.
- ❑ `Odbc` members provide connectivity to legacy data sources that don't have OLE DB data providers. The `Odbc` namespace is present in .NET Framework 2.0 for backward compatibility with .NET Framework 1.x applications.

Each data provider namespace has its own set of data object classes. The provider you choose determines the prefix of data object names—such as `SqlConnection`, `SqlCeConnection`, `OracleConnection`, or `OleDbConnection`.

Basic ADO.NET Data Objects

This chapter defines *basic data objects* as runtime data-access types that have ADODB counterparts. ADO.NET 2.0 provides the following basic data objects for data retrieval, updates, or both:

- ❑ `Connection` objects define the data provider, database manager instance, database, security credentials, and other connection-related properties. The VB 2005 code to create a .NET `Connection` is quite similar to the VB6 code to create an `ADODB.Connection` object. You also can create a new, persistent (design-time) `Connection` object by right-clicking Server Explorer's Data Connections node and choosing Add Connection to open the Connection Properties dialog. Alternatively, choose Tools ⇨ Connect to Database to open the dialog.
- ❑ `Command` objects execute SQL batch statements or stored procedures over an open `Connection`. `Command` objects can return one or more resultsets, subsets of a resultset, a single row, a single scalar value, an `XmlDataReader` object, or the `RowsAffected` value for table updates. Unlike opening `ADODB.Recordset` objects from an `ADODB.Connection`, the ADO.NET `Command` object isn't optional. `Command` objects support an optional collection of `Parameter` objects to execute parameterized queries or stored procedures. The relationship of ADODB and ADO.NET parameters to commands is identical.
- ❑ `DataReader` objects retrieve one or more forward-only, read-only resultsets by executing SQL batch statements or stored procedures. VB .NET code for creating and executing a `DataReader` from a `Command` object on a `Connection` object is similar to that for creating the default, cursorless `ADODB.Recordset` object from an `ADODB.Command` object. Unlike the default forward-only `ADODB.Recordset`, you can't save a `DataReader`'s resultset to a local file and reopen it with a client-side cursor by `Save` and `Open` methods.
- ❑ `XmlReader` objects consume streams that contain well-formed XML documents, such as those produced by SQL Server FOR XML AUTO queries or stored procedures, or native `xml` columns of SQL Server 2005. `XmlReaders` are the equivalent of a read-only, forward-only cursor over the XML document. An `XmlReader` object corresponds to the `ADODB.Stream` object returned by the SQLXML 3.0 and later SQLXMLOLEDB provider.

SqlClient doesn't support bidirectional (navigable) cursors. Microsoft added an SqlDataReader object, which emulated an updatable server-side cursor, to an early VS 2005 beta version. The VS 2005 team quickly removed the SqlDataReader object after concluding that it encouraged "bad programming habits," such as holding a connection open during data editing operations. An ExecutePageReader method, which relied on the SqlDataReader object, was removed at the same time and for the same reason.

Figure 1-2 illustrates the relationships between ADO.NET Connection, Command, Parameter, DataReader, and XmlReader objects. Parameters are optional for ADODB and basic ADO.NET commands. The SqlClient types can be replaced by OleDb or Odbc types. Using the OleDb provider to return an XmlDataReader object from SQL Server 2000 requires installing SQLXML 3.0 SP-2 or later; the Odbc provider doesn't support XMLReaders. SQL Server 2005's setup program installs SQLXML 4.0.

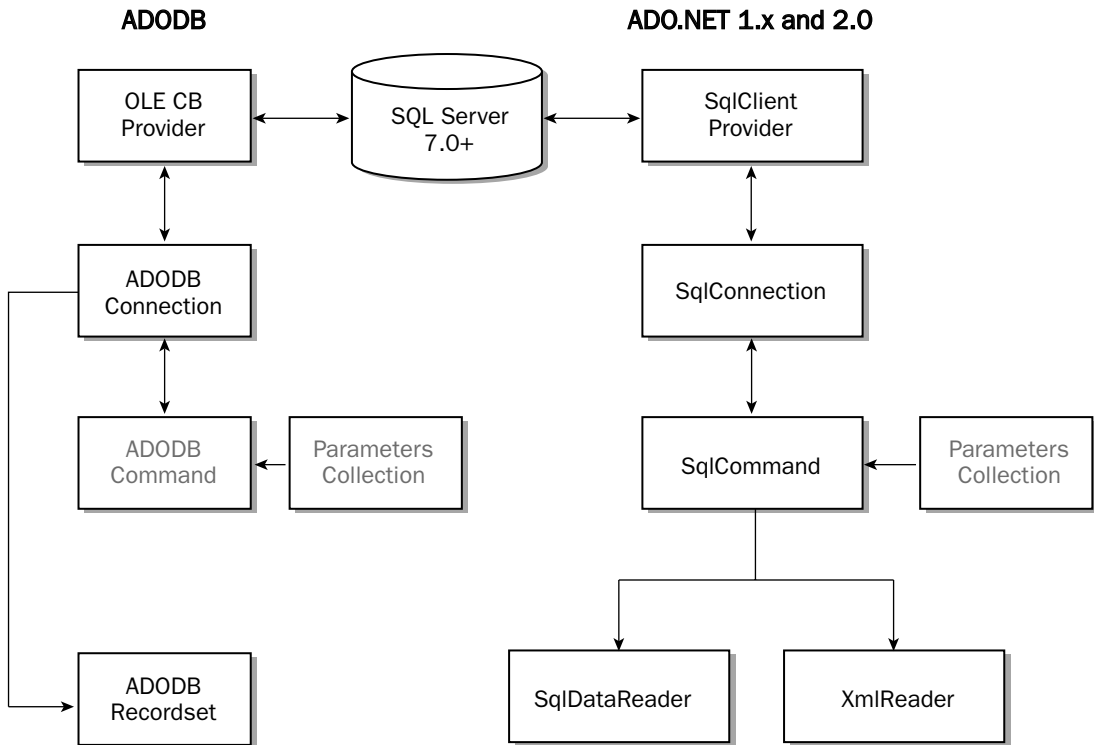


Figure 1-2

Creating Basic ADO.NET Data Objects with SqlClient

The following sections illustrate typical VB 2005 code for defining and opening an SqlConnection object, specifying an SqlCommand object, and invoking the command's ExecuteReader and ExecuteXmlReader methods. The procedures include code to display SqlDataReader column and XmlReader element values. All examples use a local SQL Server 2000 or 2005 Northwind sample database as their data source.

If you're using the default named instance of SQLX on your test machine, change localhost to .\SQLEXPRESS in the strConn connection string. If you're using Access's MSDE 2000 instance as the local server, change Northwind to NorthwindCS. If you're using a remote SQL Server instance, replace localhost with the remote server's network name.

The \VB2005DB\Chapter01\BasicDataObjects folder, which you create by expanding the Chapter01.zip file from the Wrox Web site for the book, contains complete source code for the following procedures. However, you must install the Northwind sample database before running the sample projects. See the Introduction's "Source Code and Sample Databases" section for details.

SqlDataReaders with Multiple Resultsets

One of the most common uses of `SqlDataReader` objects is filling dropdown lists or list boxes with lookup data. You can use multiple resultsets from a single SQL batch query or stored procedure to fill multiple lists in the `FormName_Load` event handler. The following `OpenDataReader` procedure opens a connection to the Northwind sample database, specifies an `SqlCommand` object that returns two resultsets, and invokes its `ExecuteReader` method to generate the `SqlDataReader` instance. The `CommandBehavior.CloseConnection` argument closes the connection when you close the `DataReader`. All basic ADO.NET data objects follow this pattern; only the `ExecuteObject` method and `DataReader` iteration methods differ. The `SqlDataReader.Read` method, which replaces the often-forgotten `RecordSet.MoveNext` instruction, returns `True` while rows remain to be read. Similarly, the `SqlDataReader.NextResult` method is `True` if unprocessed resultsets remain after the initial iteration.

Only one resultset is open as you iterate multiple resultsets, which differs from SQL Server 2005's Multiple Active Resultsets (MARS) feature. Chapter 10, "Upgrading from SQL Server 2000 to 2005," describes how to enable the MARS feature.

```
Private Sub OpenDataReader()  
    'Define and open the SqlConnection object  
    Dim strConn As String = "Server=localhost;Database=Northwind;" + _  
        "Integrated Security=SSPI"  
    Dim cnnNwind As SqlConnection = New SqlConnection(strConn)  
    cnnNwind.Open()  
  
    'Define the SqlCommand to return two resultsets  
    Dim strSQL As String = "SELECT * FROM Shippers"  
    strSQL += ";SELECT EmployeeID, FirstName, LastName FROM Employees"  
    Dim cmdReader As SqlCommand = New SqlCommand(strSQL, cnnNwind)  
    cmdReader.CommandType = CommandType.Text  
  
    'Define, create, and traverse the SqlDataReader  
    'Close the connection when closing the SqlDataReader  
    Dim sdrReader As SqlDataReader = _  
        cmdReader.ExecuteReader(CommandBehavior.CloseConnection)  
    sdrReader = cmdReader.ExecuteReader  
    With sdrReader  
        If .HasRows Then  
            While .Read  
                'Fill a Shippers list box
```

```
        lstShippers.Items.Add(.Item(0).ToString + " - " + .Item(1).ToString)
    End While
    While .NextResult
        'Process additional resultset(s)
        While .Read
            'Fill an Employees list box
            lstEmployees.Items.Add(.Item(0).ToString + " - " + _
                .Item(1).ToString + " " + .Item(2).ToString)
        End While
    End While
End If
'Close the SqlDataReader and SqlConnection
.Close()
End With
End Sub
```

Use of the HasRows property is optional because initial invocation of the Read method returns False if the query returns no rows. The SqlDataReader.Item(ColumnIndex) property returns an Object variable that you must convert to a string for concatenation. Structured error handling code is removed for improved readability.

XmlReaders with FOR XML AUTO Queries

Adding a FOR XML AUTO clause to an SQL Server SELECT query or stored procedure returns the resultset as an XML stream. The default XML document format is attribute-centric; add the Elements modifier to return an element-syntax document. Here's the XML document returned by a SELECT * FROM Shippers FOR XML AUTO, Elements query:

```
<?xml version="1.0" encoding="utf-8" ?>
<root>
  <Shippers>
    <ShipperID>1</ShipperID>
    <CompanyName>Speedy Express</CompanyName>
    <Phone>(503) 555-9831</Phone>
  </Shippers>
  <Shippers>
    <ShipperID>2</ShipperID>
    <CompanyName>United Package</CompanyName>
    <Phone>(503) 555-3199</Phone>
  </Shippers>
  <Shippers>
    <ShipperID>3</ShipperID>
    <CompanyName>Federal Shipping</CompanyName>
    <Phone>(503) 555-9931</Phone>
  </Shippers>
</root>
```

ADO.NET 2.0's new SqlCommand.ExecuteXmlReader method loads a System.Xml.XmlReader object with the stream, as shown in the following OpenXmlReader procedure listing. XmlReader is an abstract class with concrete XmlTextReader, XmlNodeReader, and XmlValidatingReader implementations. ADO.NET 2.0's ExecuteXmlReader method returns a concrete implementation.


```

Private Sub OpenXmlReader()
    'Define and open the SqlConnection object
    Dim strConn As String = "Server=localhost;Database=Northwind;" + _
        "Integrated Security=SSPI"
    Dim cnnNwind As SqlConnection = New SqlConnection(strConn)
    Dim xrShippers As System.Xml.XmlReader
    Try
        cnnNwind.Open()

        'Define the SqlCommand
        Dim strSQL As String = "SELECT * FROM Shippers FOR XML AUTO, Elements"
        Dim cmdXml As SqlCommand = New SqlCommand(strSQL, cnnNwind)
        xrShippers = cmdXml.ExecuteReader
        With xrShippers
            .Read()
            Do While .ReadState <> Xml.ReadState.EndOfFile
                txtXML.Text += .ReadOuterXml
            Loop
            'Format the result
            txtXML.Text = Replace(txtXML.Text, "><", ">" + vbCrLf + "<")
        End With
    Catch exc As Exception
        MsgBox(exc.Message + exc.StackTrace)
    Finally
        xrShippers.Close
        cnnNwind.Close()
    End Try
End Sub

```

Substituting `xrShippers.MoveToContent` followed by `xrShippers.ReadOuterXML` (without the loop) returns only the first <Shippers> element group.

You must execute the `XmlReader.Read` method to move to the first element group, followed by a `ReadOuterXml` invocation for each element group, which represents a row of the resultset. The `ExecuteXmlReader` method doesn't support the `CommandBehavior` enumeration, so you must close the `SqlConnection` object explicitly. `OleDbCommand` doesn't support the `ExecuteXmlReader` method; Microsoft wants you to use `SqlClient` classes for *all* SQL Server data access applications, including SQLCLR code running in the SQL Server 2005 process.

Figure 1-3 shows the `BasicDataObjects` project's form after executing from the `frmMain_Load` event handler, which executes the preceding `OpenDataReader` and `OpenXmlReader` procedures, and the following `LoadDataGridView` procedure.

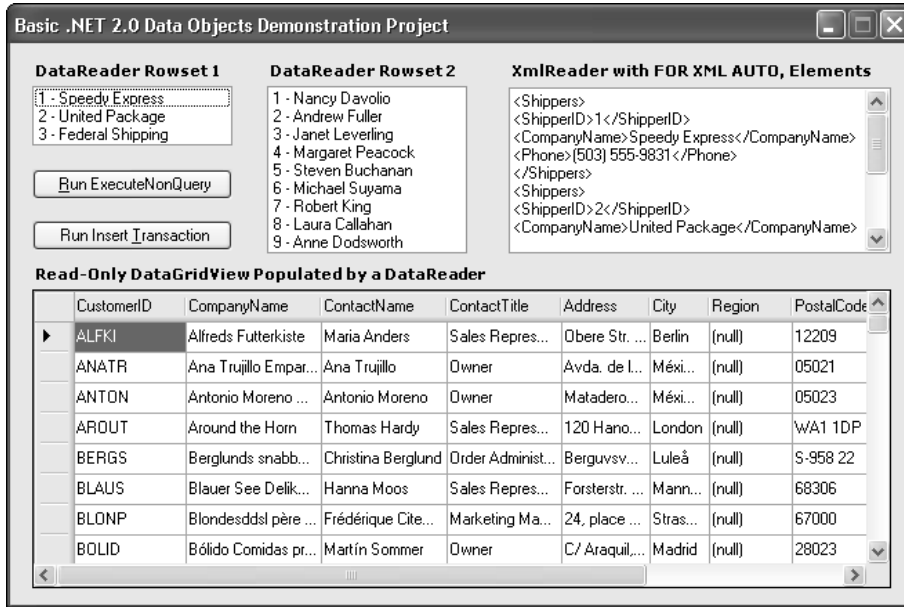


Figure 1-3

FOR XML AUTO queries or stored procedures in production applications cause a substantial performance hit compared with traditional data-access methods. The server must generate the XML stream, many more data bytes travel over the network, and the client or component must transform the XML stream to a usable format.

Fill a DataGridView with a DataReader

If your application needs to display only tabular data, a read-only grid control that's populated by code consumes the fewest resources. The DataGridView control replaces VS 2002 and VS 2003's DataGrid control, and is easy to fill programmatically. A read-only DataGridView populated by a DataReader behaves similarly to VB6's standard (unbound) Grid control, except that DataGridViews have sortable columns by default.

The following code defines the dgvCusts DataGridView control's columns and then populates each row with an instance of an objCells() Object array that contains cell values:

```
Private Sub LoadDataGridView()
    'Populate a read-only DataGridView control with an SqlDataReader
    Dim cnnNwind As SqlConnection = New SqlConnection(strConn)
    Try
        Dim strSql As String = "SELECT * FROM Customers"
        Dim cmdGrid As New SqlCommand(strSql, cnnNwind)
        cmdGrid.CommandType = CommandType.Text
        cnnNwind.Open()
        Dim sdrGrid As SqlDataReader = cmdGrid.ExecuteReader
        Dim intCol As Integer
```

```

With sdrGrid
  If .HasRows Then
    dgvCusts.Rows.Clear()
    'Add column definition: FieldName, and ColumnName
    For intCol = 0 To .FieldCount - 1
      dgvCusts.Columns.Add(.GetName(intCol), .GetName(intCol))
    Next
    'Base column width on header text width
    dgvCusts.AutoSizeColumnsMode = _
      DataGridViewAutoSizeColumnsMode.ColumnHeader
    While .Read
      'Get row data as an Object array
      Dim objCells(intCol) As Object
      .GetValues(objCells)
      'Add an entire row at a time
      dgvCusts.Rows.Add(objCells)
    End While
    .Close()
  End If
End With
Catch exc As Exception
  MsgBox(exc.Message)
Finally
  cnnNwind.Close()
End Try
End Sub

```

To sort the DataGridView control on column values, click the column header. Alternate clicks perform ascending and descending sorts.

Return a Single Data Row

Adding a `CommandBehavior.SingleRow` flag to the `SqlDataReader` object returns the first row of a resultset specified by an SQL query or stored procedure. The following code returns the first row of Northwind's Customers table, if you don't specify a WHERE clause. Otherwise the code returns the first row specified by WHERE criteria. Adding a `CommandBehavior.CloseConnection` flag closes the connection automatically when you close the `SqlDataReader` object.

```

Private Sub OpenExecuteRow()
  Dim cnnNwind As SqlConnection = New SqlConnection(strConn)
  Try
    cnnNwind.Open()
    'Define the SqlCommand
    Dim strSQL As String = "SELECT * FROM Customers"
    'Following is optional for the first record
    'strSQL += " WHERE CustomerID = 'ALFKI'"
    Dim cmdRow As SqlCommand = New SqlCommand(strSQL, cnnNwind)
    cmdRow.CommandType = CommandType.Text
    Dim sdrRow As SqlDataReader = _
      cmdRow.ExecuteReader(CommandBehavior.SingleRow Or _
        CommandBehavior.CloseConnection)
    With sdrRow
      If .HasRows Then
        .Read()
      End If
    End With
  End Try
End Sub

```

```
        Dim intFields As Integer = .FieldCount
        Dim strCustID As String = .GetString(0)
        Dim strCompany As String = .GetString(1)
    End If
    'Closes the DataReader and Connection
    .Close()
End With
Catch exc As Exception
    MsgBox(exc.Message + exc.StackTrace)
Finally
    'Close the SqlConnection, if still open
    cnnNwind.Close()
End Try
End Sub
```

Return a Scalar Value

The `SqlCommand.ExecuteScalar` method returns the value of the first column of the first row of a resultset. The most common use of `ExecuteScalar` is to return a single SQL aggregate value, such as `COUNT`, `MIN`, or `MAX`. The following `OpenExecuteScalar` procedure listing returns the number of Customers table records:

```
Private Sub OpenExecuteScalar()
    'Return a single SQL aggregate value
    Dim strConn As String = "Server=localhost;Database=Northwind;" + _
        "Integrated Security=SSPI"
    Dim cnnNwind As SqlConnection = New SqlConnection(strConn)
    cnnNwind.Open()

    'Define the SqlCommand
    Dim strSQL As String = "SELECT COUNT(*) FROM Customers"
    Dim cmdScalar As SqlCommand = New SqlCommand(strSQL, cnnNwind)
    cmdScalar.CommandType = CommandType.Text
    Dim intCount As Integer = CInt(cmdScalar.ExecuteScalar)
    'Close the SqlConnection
    cnnNwind.Close()
End Sub
```

Execute Queries That Don't Return Data

You use the `SqlCommand.ExecuteNonQuery` method to execute SQL queries or stored procedures that update base table data—`INSERT`, `UPDATE`, and `DELETE` operations. As the following `OpenExecuteNonQuery` code demonstrates, `ExecuteNonQuery` rivals the simplicity of `ExecuteScalar`:

```
Private Sub RunExecuteNonQuery()
    'Add and delete a bogus Customers record
    Dim strConn As String = "Server=localhost;Database=Northwind;" + _
        "Integrated Security=SSPI"
    Dim cnnNwind As SqlConnection = New SqlConnection(strConn)
    Dim intRecordsAffected As Integer
    Try
        cnnNwind.Open()

        'Define and execute the INSERT SqlCommand
```

```

Dim strSQL As String = "INSERT Customers (CustomerID, CompanyName) " + _
    "VALUES ('BOGUS', 'Bogus Company')"
Dim cmdUpdates As SqlCommand = New SqlCommand(strSQL, cnnNwind)
cmdUpdates.CommandType = CommandType.Text
intRecordsAffected = cmdUpdates.ExecuteNonQuery

'Update and execute the UPDATE SqlCommand
strSQL = "UPDATE Customers SET CompanyName = 'Wrong Company' " + _
    "WHERE CustomerID = 'BOGUS'"
cmdUpdates.CommandText = strSQL
intRecordsAffected += cmdUpdates.ExecuteNonQuery

'Define and execute the DELETE SqlCommand
strSQL = "DELETE FROM Customers WHERE CustomerID = 'BOGUS'"
cmdUpdates.CommandText = strSQL
intRecordsAffected += cmdUpdates.ExecuteNonQuery
Catch exc As Exception
    MsgBox(exc.Message + exc.StackTrace)
Finally
    'Close the SqlConnection
    cnnNwind.Close()
    If intRecordsAffected <> 3 Then
        MsgBox("INSERT, UPDATE, DELETE, or all failed. " + _
            "Check your Customers table.")
    End If
End Try
End Sub

```

Executing SQL update queries against production databases isn't a recommended practice and most DBAs won't permit direct updates to server base tables. The purpose of the preceding example is to provide a simple illustration of how the `ExecuteNonQuery` method works. In the real world, parameterized stored procedures usually perform table updates.

Applying Transactions to Multi-Table Updates

All updates within a single procedure to more than one table should run under the control of a transaction. The `SqlTransaction` object provides clients with the ability to commit or, in the event of an exception, roll back updates to SQL Server base tables. Managing transactions in ADO.NET is similar to that for `ADODB.Connection` objects, which have `BeginTrans`, `CommitTrans`, and `RollbackTrans` methods. `SqlTransaction` objects have corresponding `BeginTransaction`, `CommitTransaction`, and `RollbackTransaction` methods. Unlike `ADODB` connections, ADO.NET lets you selectively enlist commands in an active transaction.

Following are the steps to execute ADO.NET transacted updates:

- ❑ Define a local transaction as an `SqlTransaction`, `OleDbTransaction`, or `OdbcTransaction` object.
- ❑ Invoke the transaction's `BeginTransaction` method with an optional `IsolationLevel` enumeration argument. The default `IsolationLevel` property value is `ReadCommitted`.
- ❑ Enlist commands in the transaction by their `Transaction` property.

Chapter 1

- ❑ Invoke the `ExecuteNonQuery` method for each command.
- ❑ Invoke the transaction's `Commit` method.
- ❑ If an exception occurs, invoke the transaction's `Rollback` method.

ADO.NET's `IsolationLevel` and ADODB's `IsolationLevelEnum` enumerations share many common members, as shown in the following table.

ADO.NET Member	ADODB Member	ADO.NET <code>IsolationLevel</code> Description
Chaos	<code>adXactChaos</code>	Prevents pending changes from more highly isolated transactions from being overwritten
<code>ReadCommitted</code>	<code>AdXactReadCommitted</code> <code>adXactCursorStability</code>	Avoids dirty reads but permits non-repeatable reads and phantom data (default)
<code>ReadUncommitted</code>	<code>AdXactReadUncommitted</code> <code>adXactBrowse</code>	Allows dirty reads, non-repeatable rows, and phantom rows
<code>RepeatableRead</code>	<code>adXactRepeatableRead</code>	Prevents non-repeatable reads but allows phantom rows
<code>Serializable</code>	<code>AdXactSerializable</code> <code>adXactIsolated</code>	Prevents dirty reads, non-repeatable reads, and phantom rows by placing a range lock on the data being updated
<code>Snapshot</code>	None	Stores a version of SQL Server 2005 data that clients can read while another client modifies the same data
<code>Unspecified</code>	<code>adXactUnspecified</code>	Indicates that the provider is using a different and unknown isolation level

`Snapshot` is a new ADO.NET 2.0 isolation level for SQL Server 2005 only. `Snapshot` isolation eliminates read locks by providing other clients a copy (snapshot) of the unmodified data until the transaction commits. You must enable `Snapshot` isolation in SQL Server Management Studio (SSMS) or by issuing a T-SQL `ALTER DATABASE DatabaseName SET ALLOW_SNAPSHOT_ISOLATION ON` command to take advantage of the transaction scalability improvement that this new isolation level offers.

The following `RunInsertTransaction` listing illustrates reuse of a single `SqlTransaction` and `SqlCommand` object for sets of update transactions on the Northwind Customers and Orders tables. Running this transaction makes non-reversible changes to the `OrderID` column of the Orders table, so it's a good idea to back up the Northwind database before running this type of code. Notice that you must re-enlist the `SqlCommand` object in the `SqlTransaction` after a previous transaction commits.

```
Public Sub RunInsertTransaction()  
    'Add and delete new Customers and Orders records  
    Dim strConn As String = "Server=localhost;Database=Northwind;" + _  
        "Integrated Security=SSPI"  
    Dim cnnNwind As SqlConnection = New SqlConnection(strConn)  
  
    'Specify a local transaction object  
    Dim trnCustOrder As SqlTransaction
```

```

Dim intRecordsAffected As Integer
Dim strTitle As String
Try
    cnnNwind.Open()
    Try
        trnCustOrder = cnnNwind.BeginTransaction(IsolationLevel.RepeatableRead)
        'Define and execute the INSERT SqlCommand for a new customer
        strTitle = "INSERT "
        Dim strSQL As String = "INSERT Customers (CustomerID, CompanyName) " + _
            "VALUES ('BOGUS', 'Bogus Company')"
        Dim cmdTrans As SqlCommand = New SqlCommand(strSQL, cnnNwind)
        cmdTrans.CommandType = CommandType.Text

        'Enlist the command in the transaction
        cmdTrans.Transaction = trnCustOrder
        intRecordsAffected = cmdTrans.ExecuteNonQuery

        'INSERT an Order record for the new customer
        strSQL = "INSERT Orders (CustomerID, EmployeeID, OrderDate, ShipVia) " + _
            "VALUES ('BOGUS', 1, '" + Today.ToShortDateString + "', 1)"
        cmdTrans.CommandText = strSQL
        intRecordsAffected += cmdTrans.ExecuteNonQuery
        'Commit the INSERT transaction
        trnCustOrder.Commit()

        'Delete the Orders and Customers records
        strTitle = "DELETE "
        trnCustOrder = cnnNwind.BeginTransaction(IsolationLevel.RepeatableRead)
        strSQL = "DELETE FROM Orders WHERE CustomerID = 'BOGUS'"
        cmdTrans.CommandText = strSQL

        'The previous transaction has terminated, so re-enlist
        cmdTrans.Transaction = trnCustOrder
        intRecordsAffected += cmdTrans.ExecuteNonQuery

        strSQL = "DELETE FROM Customers WHERE CustomerID = 'BOGUS'"
        cmdTrans.CommandText = strSQL
        intRecordsAffected += cmdTrans.ExecuteNonQuery

        'Commit the DELETE transaction
        trnCustOrder.Commit()

    Catch excTrans As SqlException
        MsgBox(excTrans.Message + excTrans.StackTrace, , _
            strTitle + "Transaction Failed")
    Try
        trnCustOrder.Rollback()
    Catch excRollback As SqlException
        MsgBox(excTrans.Message + excTrans.StackTrace, , _
            strTitle + "Rollback Failed")
    End Try
    End Try
Catch exc As Exception
    MsgBox(exc.Message + exc.StackTrace)
Finally
    'Close the SqlConnection

```

```
    cnnNwind.Close()
    Dim strMsg As String
    If intRecordsAffected = 4 Then
        strMsg = "INSERT and DELETE transactions succeeded."
    Else
        strMsg = "INSERT, DELETE, or both transactions failed. " + _
            "Check your Customers and Orders tables."
    End If
    MsgBox(strMsg, , "RunInsertTransaction")
End Try
End Sub
```

This is another example of client operations that most DBAs won't permit. In production applications, stored procedures with T-SQL BEGIN TRAN[SACTION], COMMIT TRAN[SACTION], and ROLLBACK TRAN[SACTION] statements handle multi-table updates.

Using OleDb, SqlXml, and Odbc Member Classes

Most data-centric VB 2005 demonstration projects connect to an SQL Server instance with `SqlClient` objects while developers gain familiarity with .NET's panoply of `System.Data` classes. Thus, the preceding examples use the `SqlClient` data provider. You should, however, give the other managed providers — `System.Data.OleDb`, `System.Data.Odbc`, and `Microsoft.Data.SqlXml` — a test drive with the `OleDbDataProjects.sln` project in your `\VB2005DB\Chapter01\OleDbDataProjects` folder. Figure 1-4 shows `OleDbDataProject`'s form with list boxes and a text box that display data generated by each of the three providers. Marking the `Use OdbcDataReader` checkbox substitutes the `Odbc` for the `OleDb` data provider to fill the `Rowset 1` (Shippers) list box.

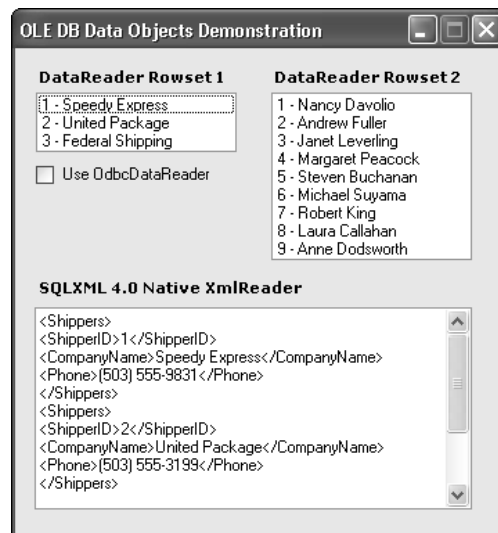


Figure 1-4

You can take advantage of ADO.NET 2.0's new `DbProviderFactories.GetFactory` ("System.Data.Provider") method and the `DbProviderFactory.CreateConnection` and `CreateCommand` methods to generate a connection to and commands for any available managed data provider. Chapter 2's "Use the `DbProviderFactories` to Create Database-Agnostic Projects" section shows you how to write applications that accommodate multiple relational database management systems.

Each sample procedure has its own connection string. You must modify each connection string to point to your Microsoft Access, SQL Server, or SQL Express instance.

The SQLXML Managed Classes (Microsoft.Data.SqlXml) native data provider for SQL Server 2000 isn't a member of the .NET Framework 2.0. It's a component of Microsoft SQLXML 4.0, which VS 2005 and VB Express install as Microsoft.Data.SqlXml.dll.

Substitute OleDb for SqlClient Objects

The OleDb data provider is your best bet for connecting to Access (Jet 4.0) database files or database servers for which you don't have a native .NET data provider. The OleDb provider also lets you create applications that *might* work with the user's choice of database servers. In most cases, you can replace `Imports System.Data.SqlClient` with `Imports System.Data.OleDb`, substitute the appropriate OLE DB connection string, and replace the prefix of data objects from `Sql` to `OleDb`. In some cases, you might need to alter the SQL statement for a specific database back end's SQL dialect. For example, the Jet query engine recognizes the semicolon as an SQL statement terminator but won't return additional resultsets from another SQL statement that follows the semicolon. Thus, the code for `Northwind.mdb` in the following `OpenOleDbDataReader` listing reuses the `OleDbCommand` with a second SQL statement:

```
Private Sub OpenOleDbDataReader()  
    'Define and open the OleDbConnection object  
    Dim strConn As String = "Provider=Microsoft.Jet.OLEDB.4.0;" + _  
        "Data Source=C:\Program Files\Microsoft Office\OFFICE11" + _  
        "\SAMPLES\Northwind.mdb;Persist Security Info=False"  
    'Substitute the following if you don't have Northwind.mdb available  
    'Dim strConn As String = "Provider=SQLOLEDB;" + _  
    ' "Data Source=localhost;Initial Catalog=Northwind;Integrated Security=SSPI"  
  
    Dim cnnNwind As OleDbConnection = New OleDbConnection(strConn)  
    cnnNwind.Open()  
  
    'Define the OleDbCommand  
    Dim strSQL As String = "SELECT * FROM Shippers"  
    'strSQL += ";SELECT EmployeeID, FirstName, LastName FROM Employees"  
    Dim cmdReader As OleDbCommand = New OleDbCommand(strSQL, cnnNwind)  
    cmdReader.CommandType = CommandType.Text  
  
    'Define, create, and traverse the OleDbDataReader  
    'Don't close the connection when closing the OleDbDataReader  
    Dim odbReader As OleDbDataReader = _  
        cmdReader.ExecuteReader(CommandBehavior.Default)  
    lstShippers.Items.Clear()  
    With odbReader  
        If .HasRows Then  
            While .Read  
                'Process the rows  
                lstShippers.Items.Add(.Item(0).ToString + _  
                    " - " + .Item(1).ToString)
```

```
        End While
        .Close()
    End If
End With
lstEmployees.Items.Clear()
cmdReader.CommandText = "SELECT EmployeeID, FirstName, LastName FROM Employees"
odbReader = cmdReader.ExecuteReader(CommandBehavior.CloseConnection)
'Process additional resultsets
With odbReader
    If .HasRows Then
        While .Read
            'Process additional rows
            lstEmployees.Items.Add(.Item(0).ToString + " - " + _
                .Item(1).ToString + " " + .Item(2).ToString)
        End While
    End If
    'Close the OleDbDataReader and the OleDbConnection
    .Close()
End With
End Sub
```

You must close the first DataReader before you change the CommandText property to reuse the OleDbCommand object.

Replace SqlConnection and SqlCommand with SqlXmlCommand

Returning XmlReader objects with the OleDb data provider requires adding a project reference to Microsoft.Data.SqlXml. Adding an Imports Microsoft.Data.SqlXml statement to your form's class file simplifies references to its classes. An interesting feature of the SqlXmlCommand object is that it doesn't require a SqlConnection object, as illustrated by the following listing for the OpenSqlXmlReader procedure:

```
Private Sub OpenSqlXmlReader()
    'This procedure requires installing SQLXML 3.0 SP-2 or later
    'and a project reference to Microsoft.Data.SqlXml

    'Define OleDb connection string
    Dim strConn As String = "Provider=SQLOLEDB;Data Source=localhost;" + _
        "Initial Catalog=Northwind;Integrated Security=SSPI"

    'Define the SqlXmlCommand
    Dim strSQL As String = "SELECT * FROM Shippers FOR XML AUTO, Elements"
    Dim cmdXml As SqlXmlCommand = New SqlXmlCommand(strConn)
    cmdXml.CommandText = strSQL
    Dim xrShippers As System.Xml.XmlReader = cmdXml.ExecuteReader
    With xrShippers
        .Read()
        Do While .ReadState <> Xml.ReadState.EndOfFile
            txtXML.Text += .ReadOuterXml
        Loop
        'Format the result
        txtXML.Text = Replace(txtXML.Text, "><", ">" + vbCrLf + "<")
        .Close()
    End With
End Sub
```

Test the Odbc Data Provider

You're not likely to use an Odbc data provider unless you're working with a legacy database server for which an OLE DB data provider isn't available. The following `OpenOdbcDataReader` procedure listing is present for completeness only:

```
Private Sub OpenOdbcDataReader()
    'Define and open the OdbcConnection object
    Dim strConn As String = "DRIVER={SQL Server};SERVER=localhost;" + _
        "Trusted_connection=yes;DATABASE=Northwind;"

    Dim cnnNwind As OdbcConnection = New OdbcConnection(strConn)
    cnnNwind.Open()

    'Define the OdbcCommand
    Dim strSQL As String = "SELECT * FROM Shippers"
    Dim cmdReader As OdbcCommand = New OdbcCommand(strSQL, cnnNwind)
    cmdReader.CommandType = CommandType.Text

    'Define, create, and traverse the OdbcDataReader
    'Close the connection when closing the OdbcDataReader
    Dim sdrReader As OdbcDataReader = _
        cmdReader.ExecuteReader(CommandBehavior.CloseConnection)
    If chkUseOdbc.Checked Then
        lstShippers.Items.Clear()
    End If
    With sdrReader
        If .HasRows Then
            While .Read
                'Process the rows
                Dim intShipperID As Integer = .GetInt32(0)
                Dim strCompany As String = .GetString(1)
                Dim strPhone As String = .GetString(2)
                If chkUseOdbc.Checked Then
                    lstShippers.Items.Add(.Item(0).ToString + _
                        " - " + .Item(1).ToString)
                End If
            End While
        End If
        'Close the OdbcDataReader and the OdbcConnection
        .Close()
    End With
End Sub
```

Working with Typed DataReader and SqlResultSet Data

The preceding code examples use `Reader.Item(ColumnIndex).ToString`, `Reader.GetString(ColumnIndex)`, and `Reader.GetInt32(ColumnIndex)` methods to extract column values to native .NET data types, which the `System` namespace defines. ADO.NET 2.0 provides the following data-specific enumerations:

Chapter 1

- ❑ `System.Data.DbType` is a generic enumeration for setting the data types of OleDb and Odbc parameters, fields, and properties.
- ❑ `System.Data.SqlDbType` is an enumeration for use with `SqlParameter` objects only. VS 2005 automatically adds `SqlParameters` when you create typed `DataSets` from SQL Server tables in the following sections.
- ❑ `System.Data.SqlTypes` is a namespace that contains structures for all SQL Server 2000 and 2005 data types, except timestamp, and related classes and enumerations. Using `SqlTypes` structures improves data-access performance by eliminating conversion to native .NET types, and assures that column values aren't truncated.

VS 2005's online help provides adequate documentation for `DbType` and `SqlDbType` enumerations, and `SqlTypes` structures, so this chapter doesn't provide a table to relate these enumerations and types.

The following `OpenDataReaderSqlTypes` listing shows examples of the use of typical `GetSqlDataType(ColumnIndex)` methods:

```
Private Sub OpenDataReaderSqlTypes()  
    'Define and open the SqlConnection object  
    Dim strConn As String = "Server=localhost;Database=Northwind;" + _  
        "Integrated Security=SSPI"  
    Dim cnnNwind As SqlConnection = New SqlConnection(strConn)  
    Dim sdrReader As SqlDataReader  
    Try  
        cnnNwind.Open()  
  
        'Define the SqlCommand  
        Dim strSQL As String = "SELECT Orders.*, " + _  
            "ProductID, UnitPrice, Quantity, Discount " + _  
            "FROM Orders INNER JOIN [Order Details] ON " + _  
            "Orders.OrderID = [Order Details].OrderID WHERE CustomerID = 'ALFKI'"  
        Dim cmdReader As SqlCommand = New SqlCommand(strSQL, cnnNwind)  
  
        'Create, and traverse the SqlDataReader, assigning SqlTypes to variables  
        sdrReader = cmdReader.ExecuteReader(CommandBehavior.CloseConnection)  
        With sdrReader  
            If .HasRows Then  
                While .Read  
                    'Get typical SqlTypes  
                    Dim s_intOrderID As SqlInt32 = .GetSqlInt32(0)  
                    Dim s_strCustomerID As SqlString = .GetSqlString(1)  
                    Dim s_datOrderDate As SqlDateTime = .GetSqlDateTime(3)  
                    Dim s_curUnitPrice As SqlMoney = .GetSqlMoney(15)  
                    Dim s_sngDiscount As SqlSingle = .GetSqlSingle(17)  
                End While  
            End If  
        End With  
        Catch exc As Exception  
            MsgBox(exc.Message + exc.StackTrace)  
        Finally  
            'Close the SqlDataReader and the SqlConnection  
            sdrReader.Close()  
        End Try  
    End Sub
```

You can update `SqlResultSet` object column values with strongly typed variables by invoking the `SqlResultSet.SetSqlDataType(ColumnIndex)` method. You'll see more examples of strongly typed SQL Server data retrieval and update operations that use these methods in later chapters.

ADO.NET Typed DataSet Objects

The `DataSet` object is unique to ADO.NET and typed `DataSets` are the preferred method for retrieving and updating relational tables, although `DataSets` aren't limited to processing relational data. You create typed `DataSets`, which are defined by an XML schema and implemented by a very large amount of auto-generated VB 2005 code, with VS 2005 designers. Untyped `DataSets` are runtime objects that you create with code. `DataSets` have no corresponding ADODB object, but both classes of `DataSets` *behave* similarly to disconnected `Recordsets` in the following ways:

- ❑ They open a connection, retrieve and cache the data to edit, and then close the connection.
- ❑ They bind to simple and complex Windows form controls for editing.
- ❑ They permit editing locally cached data while the connection is closed.
- ❑ They can be saved to local files and reopened for editing.
- ❑ They let you reopen the connection and apply updates to base tables in batches.
- ❑ They implement optimistic concurrency for base table updates. You must write code to handle concurrency violations gracefully.

Following are the most important differences between `DataSets` and disconnected `Recordsets`:

- ❑ A `DataSet` consists of cached copies of one or more sets of records — called `DataTable` objects — selected from one or more individual base tables. A `Recordset` is a single set of records that can represent a view of one or two or more related tables.
- ❑ Persisting a `DataSet` serializes the `DataTables'` records to a hierarchical, element-centric XML Infoset document and saves it to the local file system. Disconnected `Recordsets` store data locally as a flat, attribute-centric XML file.
- ❑ `DataTables` usually are — but need not be — related by primary-key/foreign-key relationships.
- ❑ Primary-key and foreign-key constraints, and table relationships, must be manually defined, unless you create the `DataSet` automatically with VS 2005's Data Source Configuration Wizard.
- ❑ You can create `DataTables` from base tables of any accessible database server instance.
- ❑ You can create `DataTables` from structured (tabular) XML Infoset documents.
- ❑ `TableAdapters` fill and update `DataTables` through a managed connection. `TableAdapters` are wrappers over `DataAdapter` objects.
- ❑ The Data Source Configuration Wizard lets you choose an existing data connection that's defined in the Server Explorer, or create a new connection object. The wizard then generates parameterized SQL queries or stored procedures for performing `UPDATE`, `INSERT`, and `DELETE` operations. These queries are based on the `SELECT` query or stored procedure that you specify for filling each `DataTable`.

- ❑ DataSets cache copies of original and modified table data in XML format. Thus, DataSets that have a large number of rows consume much more client RAM resources than Recordsets that have the same number of rows.
- ❑ You can write code to create runtime data connections, DataAdapters, and basic DataSets, but it's much easier to take advantage of VS 2005 automated processes for generating the code to create typed DataSets, which are defined by an XML schema.
- ❑ DataSet updates occur row-by-row if you don't specify a value greater than 1 for the new `DataAdapter.BatchSize` property, which sets the maximum number of updated rows per batch.

Figure 1-5 compares the objects required by updatable ADODB Recordsets and ADO.NET 1.x and 2.0 typed DataSets. Components that are new in ADO.NET 2.0 are shaded. Parameters are optional for ADODB commands, but not for updatable TableAdapters, which have four standard commands — `SelectCommand`, `InsertCommand`, `UpdateCommand`, and `DeleteCommand`. Use of the new ADO.NET 2.0 `BindingNavigator` components is optional. The section “Add a `DataGridView` and `DataNavigator` Controls,” later in this chapter, describes how the `BindingSource` fits into ADO.NET 2.0's data access architecture.

The following sections show you alternative methods for generating Figure 1-5's ADO.NET objects with VS 2005 and SQL Server 2000 or 2005.

VS 2005 materializes TableAdapters, DataSets, BindingSources, and BindingNavigators as named objects in the form design tray. TableAdapters and DataSets also appear in the Toolbox's ProjectName Components section; the Data section has DataSet, BindingSource, and BindingNavigator controls. During the early part of VS 2005's long gestation period, these design-time objects collectively were called Data Components, BindingSource was called a DataConnector, and BindingNavigator was DataNavigator. This book uses the term data component to refer to named design-time data objects that reside in the form design tray.

Add a Typed DataSet from an SQL Server Data Source

ADO.NET uses the term *data source* as a synonym for a typed DataSet with a predefined, persistent database connection. The process of creating an ADO.NET data source is similar to using VB6's Data Environment Designer to specify an OLE DB data provider from one or more tables. Unlike the Data Environment Designer, multi-table DataSets don't have the hierarchical structure that the OLE DB Shape provider creates for display in VB6's Hierarchical FlexGrid control.

Web services and object instances also can act as ADO.NET data sources, as you'll see in later chapters.

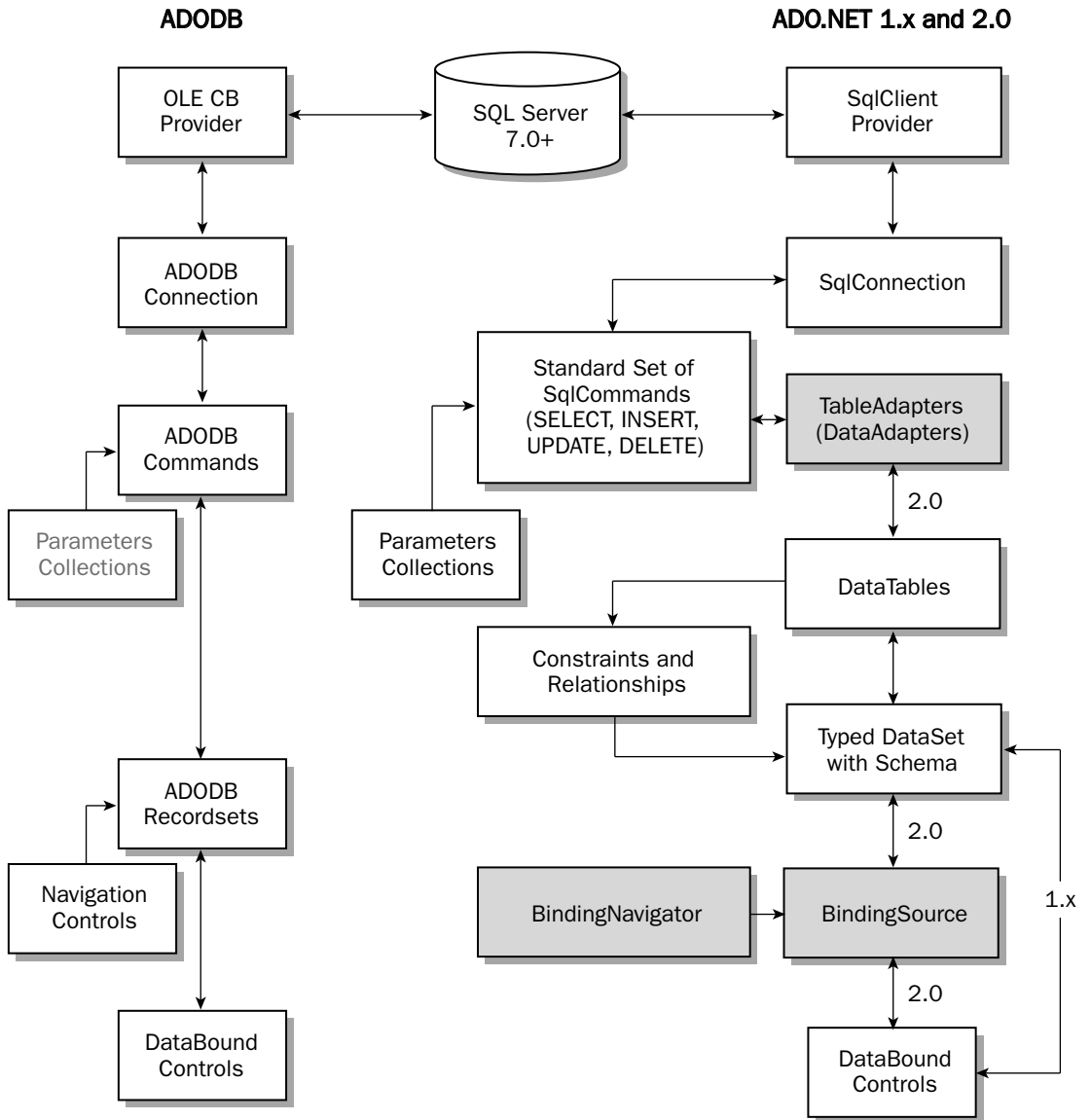


Figure 1-5

Chapter 1

Here's how to add a new SQL Server Northwind data source for a new Windows form project and automatically generate a typed DataSet and its components from the Customers table:

1. Choose Data ⇨ Show Data Sources to open the Data Sources window, if necessary, and click Add New Data Source to start the Data Source Configuration Wizard.
2. On the Choose a Data Source Type page, accept the default Database type, and click Next to open the Choose Your Database Connection page, which displays existing data connections, if any, in a dropdown list.
3. Click the New Connection button to open a simplified Add Connection dialog, which usually defaults to Microsoft SQL Server Database File. This option requires attaching a copy of northwnd.mdb to your SQL Server or SQLX instance, so click the Change button to open the Change Data Source dialog, select Microsoft SQL Server in the Data Source list, and click Continue to open the full version of the Add Connection dialog.
4. Type `localhost` or `.\SQLEXPRESS` in the Select or Enter a Server Name combo box. Alternatively, select a local or networked SQL Server or MSDE instance that has a Northwind or NorthwindCS database.
5. Accept the default Use Windows NT Integrated Security option, and open the Select or Enter a Database Name list and select Northwind. Click Test Connection to verify the SqlConnection object, as shown in Figure 1-6.

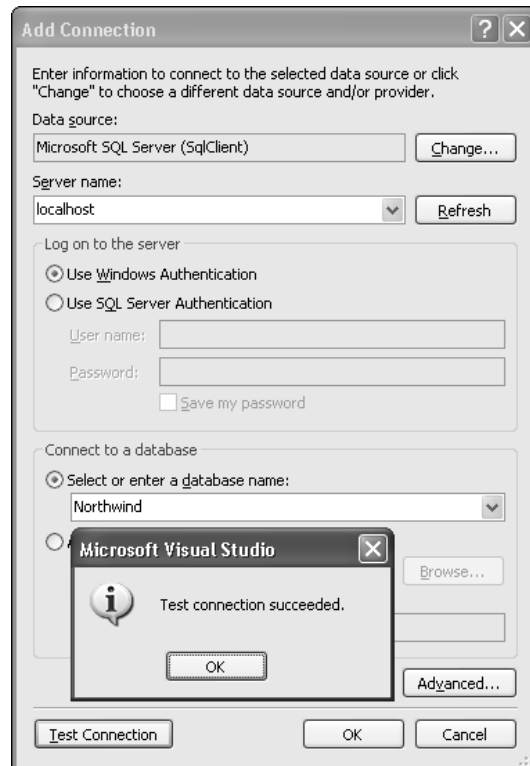


Figure 1-6

- Click OK to close the dialog and return to the Choose Your Data Connection page, which displays `ServerName.Northwind.dbo` as the new connection name, `System.Data.SqlClient` as the Provider, and `Data Source=localhost; Integrated Security=True; Database=Northwind` as the Connection String.
- Click Next to display the Save the Connection String to the Application Configuration File page. Mark the Yes, Save the Connection As checkbox and accept the default `NorthwindConnectionString` as the connection string name.
- Click Next to open the Choose Your Database Objects page, which displays treeview Tables, Views, Stored Procedures, and table-returning Functions. Expand the Tables node and mark the Customers table. Accept `NorthwindDataSet` as the DataSet Name, as shown in Figure 1-7.

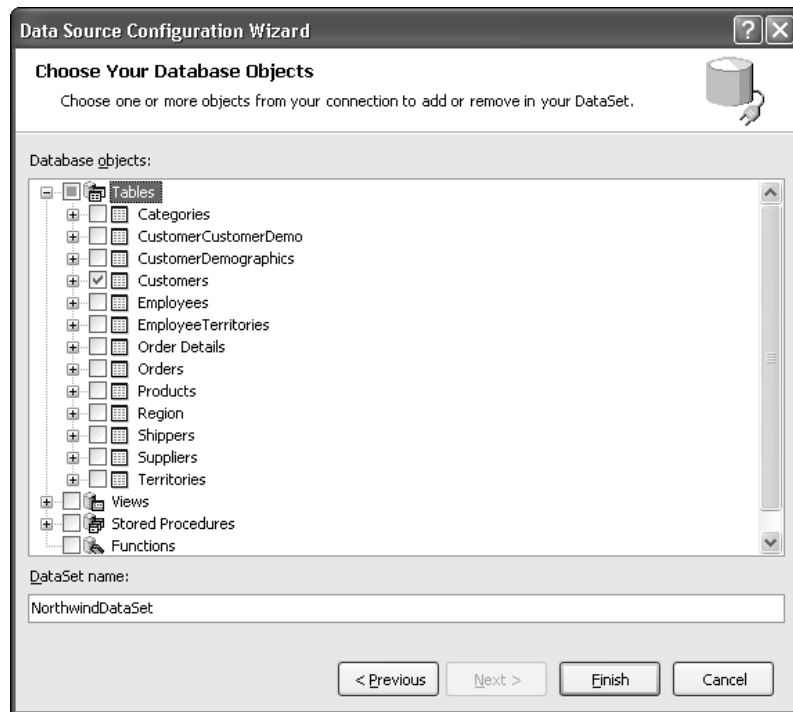


Figure 1-7

Selecting a table automatically generates the `SelectCommand` that retrieves all table rows, and an `UpdateCommand`, `InsertCommand`, and `DeleteCommand` for base table updates.

- Click Finish to generate the `NorthwindDataSet` typed DataSet and display it in the Data Sources window. Expand the Customers node to display the Customers table's columns, as shown in Figure 1-8.

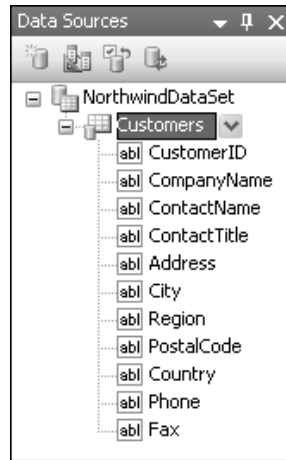


Figure 1-8

The new `SqlConnection` object you created in preceding Steps 3 through 5 appears under Server Explorer's DataConnections node as `ServerName.Northwind.dbo`. You can rename the node in Server Explorer to a simpler name, such as **localhost.Northwind**; doing this doesn't affect dependent objects in your project.

Adding a typed DataSet generates an XSD schema, `NorthwindDataSet.xsd` for this example, and adds 1,197 lines of VB 2005 code to the `NorthwindDataSet.Designer.vb` partial class file, which weighs in at 73KB. Partial classes are a new VB 2005 and C# feature that enable extending a class, such as `NorthwindDataSet`, with additional class files. VB 2005 uses the `Public Partial Class className` statement to identify a partial class file. You must choose Project ⇄ Show All Files to see `NorthwindDataSet.Designer.vb` and two empty `NorthwindDataSet.xsc` and `NorthwindDataSet.xss` files.

Double-click the `NorthwindDataSet.xsd` node in Project Explorer to display the Customers DataTable and its associated Customers TableAdapter, as shown in Figure 1-9, in the Schema Designer window. The VB 2005 code in `DataSetName.Designer.vb` provides IntelliSense for DataSet objects and lets you early-bind DataTable and DataSet objects. The code also provides direct access to named classes, methods, and events for the DataSet and its TableAdapter(s)—Customers TableAdapter for this example—in the `NorthwindDataSet.Designer.vb` code window's Classes and Methods lists.

Figure 1-10 shows Internet Explorer displaying the first few lines of the 352-line schema .

If you've worked with typed DataSets in VS 2003, you'll notice that the schema for ADO 2.0 DataSets is much more verbose than the ADO 1.x version, which has only 30 lines that define the Customers DataSet. ADO.NET 2.0 prefixes the design-time schema with 258 lines of <x:annotation> information, which provide a full definition of the DataSet and its connection string, commands and their parameters, and column mapping data. The part of the schema that defines the elements for the table fields grows from 30 to 94 lines because element definitions now contain maxLength attribute values and use restrictionBase attributes to specify XSD data types.

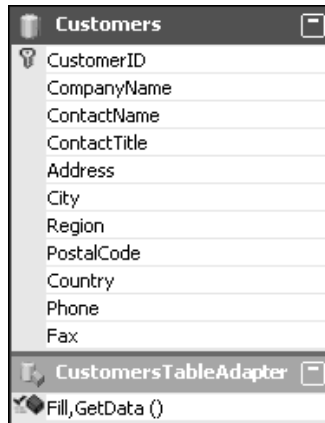


Figure 1-9

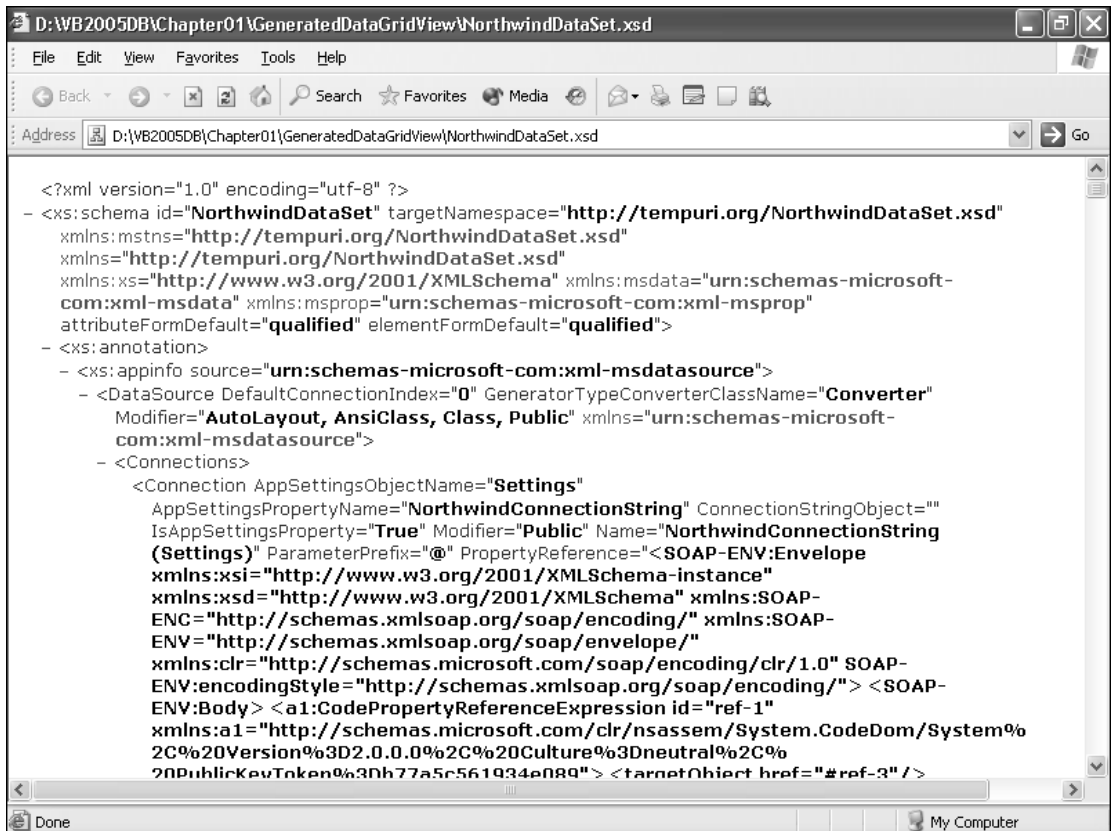


Figure 1-10

Using the `DataSet.WriteXml` and `DataSet.WriteXmlSchema` methods to persist `DataSets` to local files shows that the Customers `DataSet` schema, which differs greatly from the design-time version, is 9.31KB and the XML data document is 37.3KB. The section “Create a Complete Data Entry Form in One Step,” later in this chapter, includes code to save the schema for the Northwind Customers `DataSet`. You can’t open the saved schema in the project’s Schema Designer.

Add a `DataGridView` and `BindingNavigator` Controls

Opening `Form1` and the Data Sources window changes the appearance of the `DataSource` nodes. By default, the Customers `DataTable` icon now represents a `DataGridView` control. Dragging the Customers table node from the Data Sources window to your project’s default `Form1` autogenerates four components in the tray below the form designer and adds `DataGridView` and `DataNavigator` controls to a dramatically expanded form, as shown in Figure 1-11.

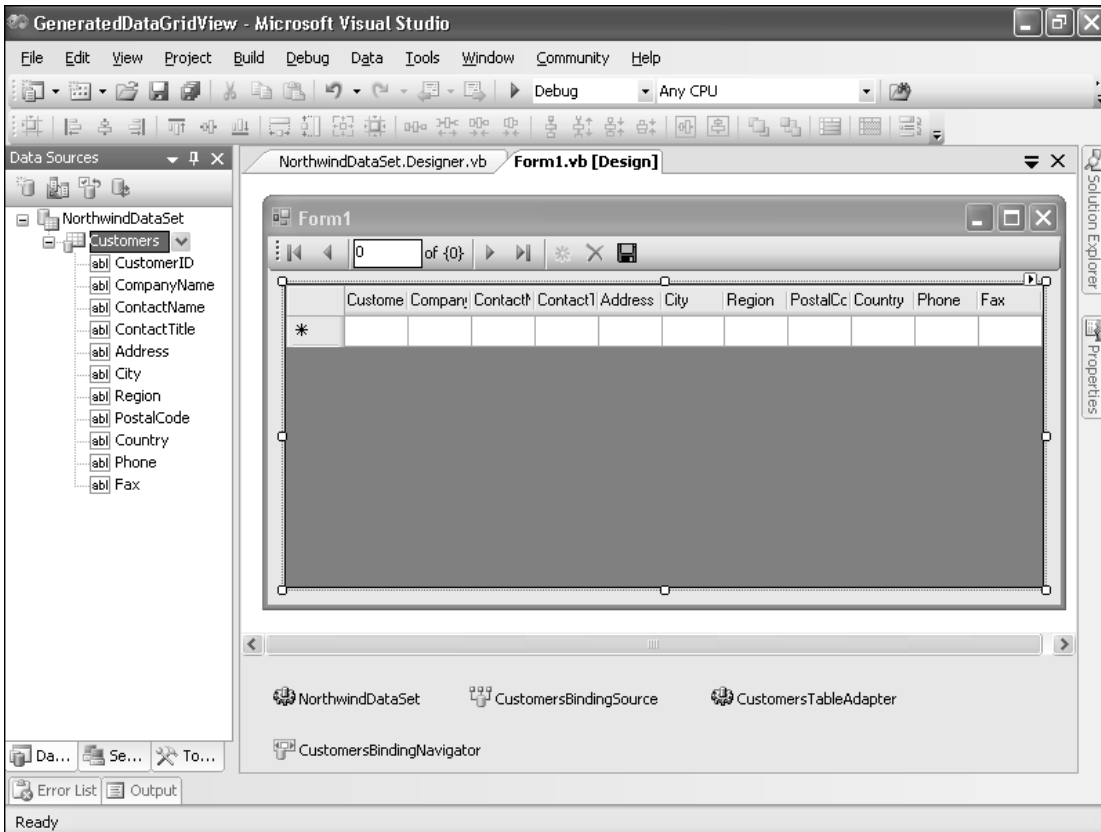


Figure 1-11

“Surfacing” is a common term for adding data and other components to the tray.

Here are descriptions of the four tray components shown in Figure 1-11:

- ❑ `NorthwindDataSet` is the form's reference to the data source for the form, `NorthwindDataSource.xsd`.
- ❑ `CustomersTableAdapter` is the form's wrapper for an `SqlDataAdapter` object, which fills the `NorthwindDataSet`'s `Customers` `DataTable` by invoking the `CustomersTableAdapter.Fill` method. `Update`, `Insert`, and `Delete` methods send `DataSet` changes to the database server. The `CustomersTableAdapter.Adapter` property lets you access the underlying `SqlDataAdapter`.
- ❑ `CustomersBindingSource` is a form-based `BindingSource` object that unifies control data binding and row data navigation for the `Customers` `DataTable` by providing direct access to the `BindingManager` object. To make it easier for VB6 developers to adapt to ADO.NET 2.0, `BindingSources` have properties and methods that emulate `ADODB.Recordset` objects. Examples are `AllowEdit`, `AllowAddNew`, and `AllowRemove` (delete) properties, and corresponding `AddNew`, `CancelNew`, `EndNew`, `Edit`, `CancelEdit`, and `EndEdit` methods. Familiar `MoveFirst`, `MoveLast`, `MoveNext`, and `MovePrevious` methods handle row navigation. Enabling navigation requires binding a `DataGridView` or adding other controls to manipulate the `BindingSource`.
- ❑ `CustomersBindingNavigator` is a custom `ToolStrip` control that emulates the VCR and other buttons of an `ADODB.DataControl`. Binding the `CustomersBindingNavigator` to the `CustomersBindingSource` enables the buttons to invoke the `Move...`, `AddNew`, and `Cancel...` methods. By default, `BindingNavigators` dock to the top of the form. When you run the form, you can drag a `BindingNavigator` to a more natural position at the bottom of the form; alternatively, you can set a `DataNavigator`'s `Dock` property value to `Bottom` in the designer.

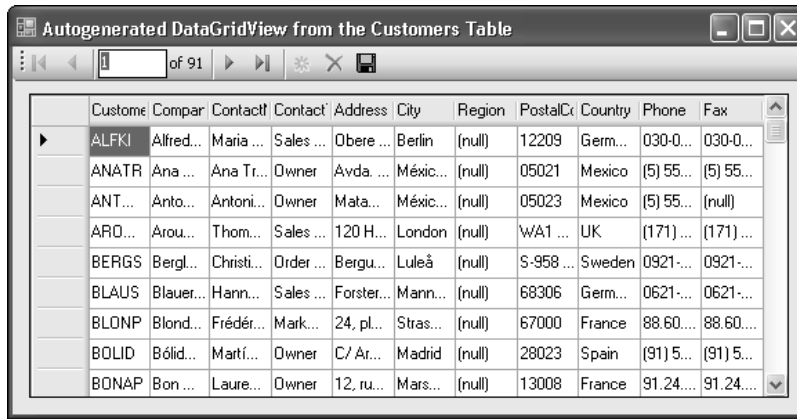
DataComponents, DataConnectors, and DataNavigators are new ADO.NET 2.0 components and controls that replace ADO.NET 1.x's form-based DataConnections and DataAdapters. VS 2005 data sources automatically create DataSet Relationships between tables, which previously required manual intervention. DataConnectors simplify code for navigating data tables. The DataSet.vb file contains the classes, interfaces, and event handlers for the data components.

The final step in the VS 2005 data form autogeneration process is adding the `CustomersComponent.Fill` method to the `Form1_Load` event handler, and code to save `DataSet` changes isn't added to the `bindingNavigatorSaveItem_Click` handler automatically, because of code complexity when the `DataSet` contains multiple `DataTables`. Saving multiple changes to parent and child tables requires sequencing inserts, updates, and deletions to maintain referential integrity.

```
Private Sub Form1_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load
    'TODO: This line of code loads data into the 'NorthwindDataSet.Customers' table.
    'You can move, or remove it, as needed.
    Me.CustomersTableAdapter.Fill(Me.NorthwindDataSet.Customers)
End Sub

Private Sub dataNavigatorSaveItem_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles dataNavigatorSaveItem.Click
    Me.CustomersBindingSource.EndEdit()
    Me.CustomersTableAdapter.Update(Me.NorthwindDataSet.Customers)
End Sub
```

Figure 1-12 shows the final form after reducing the form's size, expanding the DataGridView control to fill the available space, and pressing F5 to build, debug, and run the project.



	CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
▶	ALFKJ	Alfred...	Maria ...	Sales ...	Obere ...	Berlin	(null)	12209	Germ...	030-0...	030-0...
	ANATR	Ana...	Ana Tr...	Owner	Avda. ...	Méxic...	(null)	05021	Mexico	(5) 55...	(5) 55...
	ANT...	Anto...	Antoni...	Owner	Mata...	Méxic...	(null)	05023	Mexico	(5) 55...	(null)
	AROD...	Arou...	Thom...	Sales ...	120 H...	London	(null)	WA1 ...	UK	(171) ...	(171) ...
	BERGS	Bergl...	Christi...	Order ...	Bergu...	Luleå	(null)	S-958 ...	Sweden	0921...	0921...
	BLAUS	Blauer...	Hann...	Sales ...	Forster...	Mann...	(null)	68306	Germ...	0621...	0621...
	BLONP	Blond...	Frédér...	Mark...	24, pl...	Stras...	(null)	67000	France	88.60...	88.60...
	BOLID	Bólid...	Martí...	Owner	C/ Ar...	Madrid	(null)	28023	Spain	(91) 5...	(91) 5...
	BONAP	Bon ...	Laure...	Owner	12, ru...	Mars...	(null)	13008	France	91.24...	91.24...

Figure 1-12

The CustomersDataGridView is bound to the Northwind Customers table, and editing is enabled by default. Changes you make to the DataGridView don't propagate to the table until you click the Save Data button. To make editing easier, you can automate increasing the column widths to match the content by setting the DataGridView's AutoSizeColumnsMode property value to AllCells or DisplayedCells, which adds a horizontal scrollbar to the control.

Persist and Reopen the DataSet

The project's frmDataGridView_Load event handler includes the following code to save the NorthwindDataSet's XML data document—with and without an embedded schema—and the schema only. You can add similar code after the last DataComponent.Fill or DataAdapter.Fill invocation of any data project to persist its DataSet.

```
Private Sub frmDataGridView_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load
    Me.CustomersTableAdapter.Fill(Me.NorthwindDataSet.Customers)
    Dim strPath As String = Application.StartupPath
    With Me.NorthwindDataSet
        .WriteXml(strPath + "CustsNoSchema.xml", XmlWriteMode.IgnoreSchema)
        .WriteXml(strPath + "CustsWithSchema.xml", XmlWriteMode.WriteSchema)
        .WriteXmlSchema(strPath + "CustsSchema.xsd")
    End With
End Sub
```

Persisting the DataSet as an XML document without the embedded schema lets you support disconnected users by reloading the DataSet from the file. You can substitute the following statement for Me.CustomersTableAdapter.Fill(Me.NorthwindDataSet.Customers) when the user is disconnected:

```
Me.NorthwindDataSet.ReadXml(strPath + "CustsNoSchema.xml", XmlReadMode.Auto)
```

The real-world scenario for persisting and reloading DataSets is more complex than that shown here. Later chapters describe how to save and reload pending DataSet changes that haven't been committed to the base tables. The XmlReadMode.Auto argument is the default, so including it is optional.

The sample project at this point is GeneratedDataGridView.sln in your \VB2005DB\Chapter01\GeneratedDataGridView folder.

Change from a DataGridView to a Details Form

The default combination of DataGridView and DataNavigator controls speeds the creation of a usable form. However, a DataNavigator is much more useful for a details form that displays column values in text boxes or other bound controls, such as date pickers for DateTime and checkboxes for Boolean values. The Data Sources window makes it easy to change a DataGridView to a details form. Delete the DataGridView control, display the Data Sources window, open the dropdown list for the DataTable, and select Details, as shown in Figure 1-13.

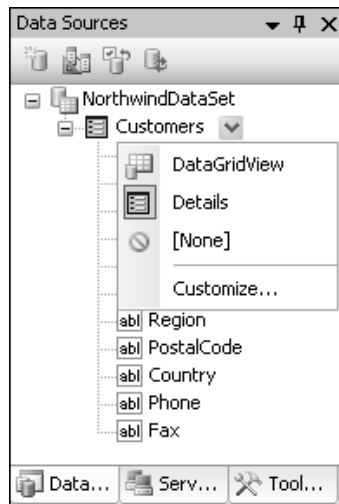


Figure 1-13

Drag the DataTable icon to the form to automatically add a column of labels with associated data-bound controls — text boxes for this example — to the form. Figure 1-14, which is a modified version of the GeneratedDataGridView project, shows the labels and text boxes rearranged to reduce form height.

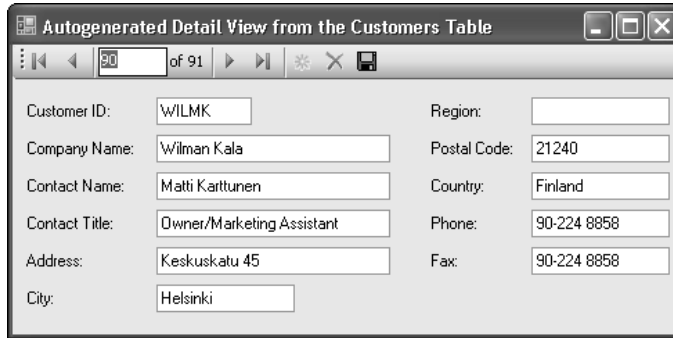


Figure 1-14

The completed GeneratedDetailView.sln project is in the \VB2005DB\Chapter01\GeneratedDetailView folder.

Add a Related DataBound Control

You can add a related table to the Data Sources window and then add a control, such as a DataGridView, that you bind to the related BindingAdapter. To add a related OrdersDataGridView control to a copy of the GeneratedDetailView.sln project, do the following:

- 1.** Copy and paste the GeneratedDetailView folder, and rename the new folder **OrdersDetailView**. Don't rename the solution or project.
- 2.** Press F5 to build and compile the project. Correct any object name errors that the debugger reports.
- 3.** Open the Data Source window, and click the Configure DataSet with Wizard button to open the Choose Your Database Objects page.
- 4.** Expand the Tables node, mark the Orders table checkbox, and click Finish, which adds in the Data Sources window a related Orders node to the Customers table and a standalone Orders node (see Figure 1-15).

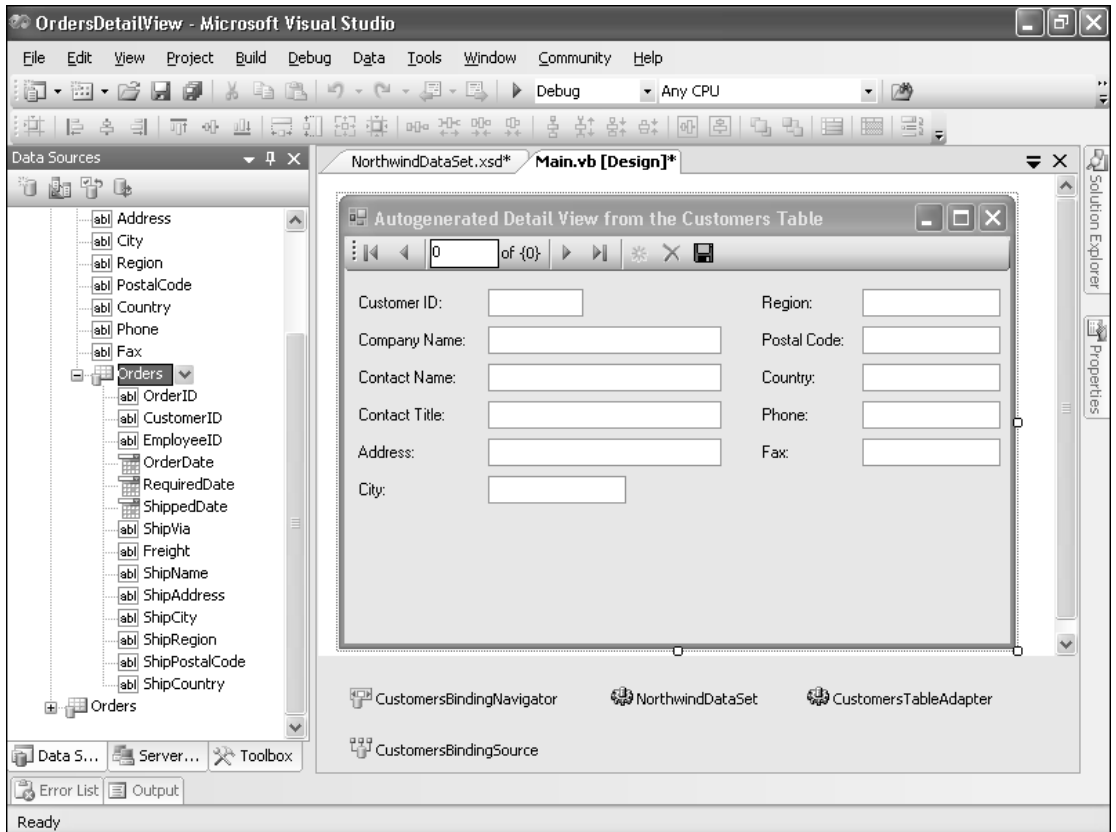


Figure 1-15

5. With DataGridView selected in the dropdown list, drag the related Orders node below the bound text boxes of the form to autogenerate an OrdersDataGridView control.
6. Adjust the size and location of the controls, and set the OrdersDataGridView .AutoSizeColumnsMode property value to DisplayedCells. Optionally, alter the form's Text property to reflect the design change.
7. Press F5 to build and run the project. The form appears as shown in Figure 1-16.

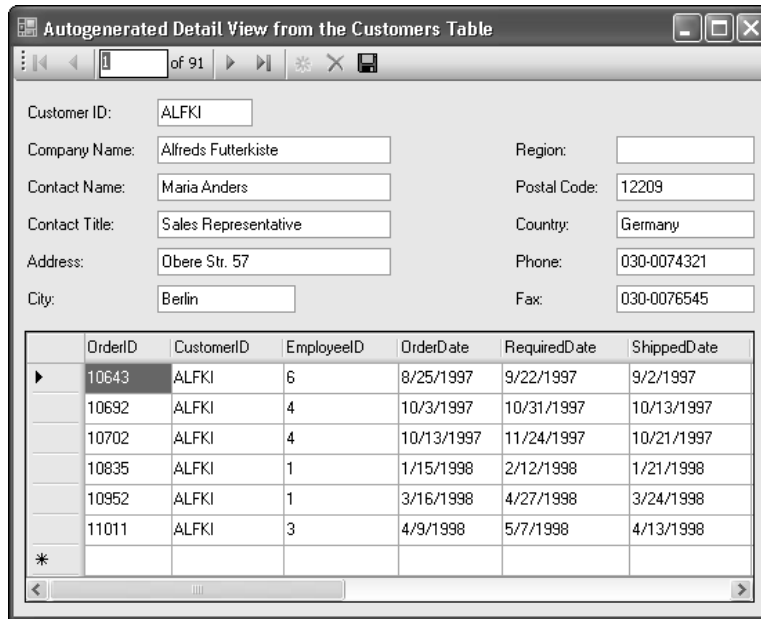


Figure 1-16

Dragging the related Orders table node to the form adds an OrdersTableAdapter and OrdersBindingSource to the tray and the OrdersDataGridview control to the form. The OrdersDataGridview control's DataSource property value is the OrdersBindingSource. The OrdersBindingSource's DataSource property value is CustomersBindingSource and the DataMember property value is FK_Orders_Customers, which is the foreign-key relationship on the CustomerID field between the Customers and Orders tables. To verify the properties of FK_Orders_Customers, open NorthwindDataSet.xsd in the DataSet Designer, right-click the relation line between the Orders and Customers tables, and choose Edit Relation to open the Relation dialog (see Figure 1-17).

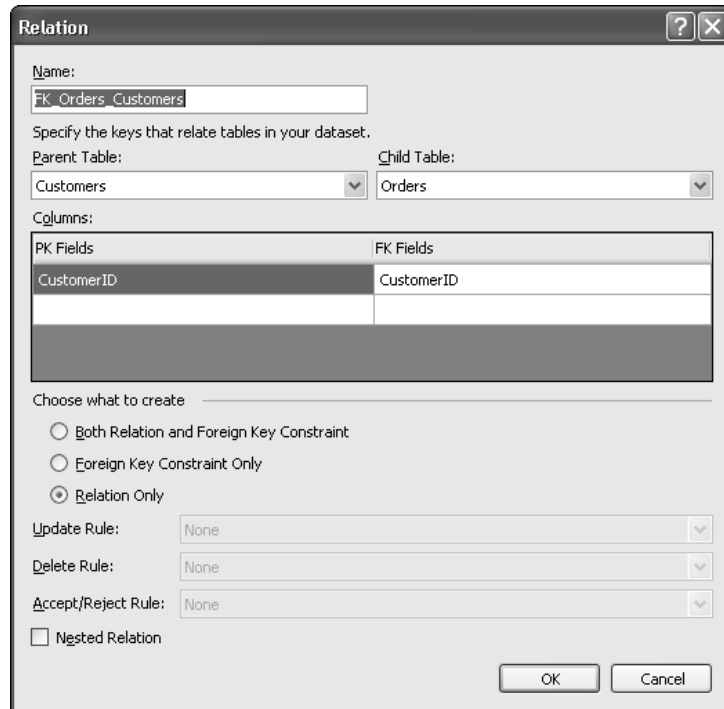


Figure 1-17

Relations you define by adding related tables to the Data Sources window don't enforce referential integrity by default. You must change the default Relation Only property value to one of the other options to maintain referential integrity. You also can specify cascade or other options for Update, Delete, and Accept/Reject Rules.

Summary

Microsoft designed the basic ADO.NET feature set to resemble that of ADO. The code to create a database connection with ADO.NET's `SqlConnection`, `OleDb`, or `Odbc` managed providers is quite similar to that for `ADODB.Connection` objects. The same is true for ADO.NET's connection-specific commands and parameters. The primary differences between ADO and ADO.NET involve processing resultsets. `DataReaders` correspond to ADO's default forward-only, read-only `Recordsets`. The `SqlConnection` data provider provides a substantial performance boost by eliminating the COM-based OLE DB layer and communicating with SQL Server 7.0 and later using SQL Server's native TDS protocol.

ADO.NET data binding to typed `DataSet` objects and data-related event handling differ radically from ADO. Many experienced VB6 database developers discovered that migrating from `ADODB.Recordsets` to ADO.NET 1.x `DataAdapters`, typed `DataSets`, and databound controls wasn't a walk in the park. Creating an ordinary data entry form with ADO.NET 1.x's `DataGrid` or other controls bound to a `DataSet's DataTable` and adding record navigation buttons involved writing much more code than that

required for a corresponding VB6 project. To ease the pain of the transition from VS 6 to VS 2005, ADO.NET 2.0 provides drag-and-drop methods for autogenerating the components and controls to create a basic, single-table form with the new DataGridView and DataNavigator controls, plus DataComponent and DataContainer components. Changing the DataGridView to a details view with individual databound controls takes only a minute or two.

The new drag-and-drop methods and component configuration wizards are useful for product demos by Microsoft's .NET evangelists, which elicit "oohs" and "aahs" from conference and user-group attendees. Autogenerated data entry forms can help programmers gain a basic understanding of ADO.NET data binding and flatten the ADO.NET learning curve. But you'll probably find that autogenerated forms aren't useful in real-world production applications. A major shortcoming is the default to parameterized SQL statements for data retrieval and update operations; most DBAs require stored procedures for *all* operations on base tables. Fortunately, you can intervene in the autogeneration process to specify and create the required stored procedures. Another issue is the BindingNavigator's lack of shortcut keys, which are a necessity for heads-down data entry. You'll discover other limitations of autogenerated forms and their workarounds as you progress through the book.

The preceding comments on databound control autogeneration doesn't apply to generating typed DataSets. Writing VB 2005 code for typed DataSets isn't a practical alternative. You can, however, create lightweight, untyped DataSets with only a few lines of code. Later chapters provide code examples to create untyped DataSets at runtime.

The following chapters of Parts I and II show you how to create production-quality Windows data entry forms by combining some of the techniques you learned in this chapter with DataSets, TableAdapters, and VB 2005 code to manage data retrieval, DataTable navigation, and multiple base table updates.