8 ActiveX Data Objekts (ADO) und ADO.NET

Die *ActiveX Data Objects* wurden vor einigen Jahren als eine Technologie eingeführt, die den Datenzugriff nicht nur über ein lokales Netzwerk, sondern auch über das Internet ermöglicht. ADO löste damit sowohl RDO *(Remote Data Objects)* als auch DAO *(Data Access Objects)* ab, das ursprünglich für die Jet-Datenbankengine entwickelt wurde. Nun taucht zusätzlich ADO.NET auf.

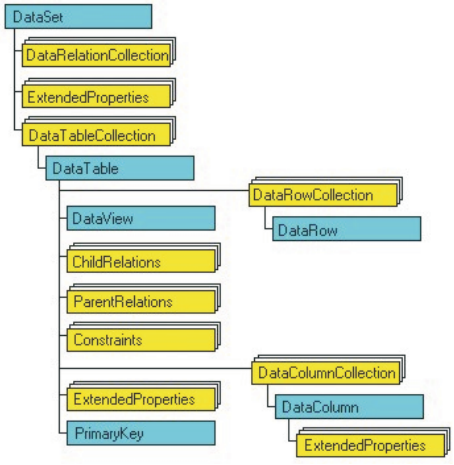
8.1 Die wesentlichen Unterschiede zwischen ADO und ADO.NET

Es gibt eine Reihe von Unterschieden zwischen ADO und ADO.NET, die hier zunächst nur stichpunktartig aufgeführt werden sollen:

- ADO arbeitet mit verbundenen Daten. Das heißt, dass wenn Sie Daten anzeigen oder aktualisieren, Sie eine Echtzeitverbindung zu ihnen haben.
- ADO.NET benutzt die Daten ohne Verbindung. Wenn Sie auf Daten zugreifen, legt ADO.NET eine Kopie der Daten mit Hilfe von XML an und hält nur während der Zeit die Verbindung zur Datenquelle aufrecht, in der die Daten abgefragt oder aktualisiert werden.
- ADO hat ein Hauptobjekt, das Recordset-Objekt, das dazu verwendet wird, auf Daten zuzugreifen. Es erlaubt eine einzige Ansicht Ihrer Daten, wobei diese natürlich auch relational sein kann. Mit ADO.NET stehen Ihnen viele Objekte zur Verfügung, die es Ihnen erlauben, auf Ihre Daten in unterschiedlichster Form zuzugreifen, darunter auch das DataSet-Objekt, das das relationale Modell Ihrer Datenbank repräsentiert.
- Mit ADO sind nur clientseitige Cursor möglich, ADO.NET hingegen lässt Ihnen die Wahl, entweder clientseitige oder serverseitige Cursor zu benutzen. In ADO.NET stehen für die Handhabung der Cursor spezielle Klassen zur Verfügung, so dass Sie sich um viele Details nicht kümmern müssen.
- Während ADO Ihnen nur erlaubt, Daten im XML-Format darzustellen, können Sie mit ADO.NET Ihre Daten auch mit Hilfe von XML manipulieren. Das ist nützlich, wenn Sie mit anderen Geschäftsanwendungen arbeiten oder Firewalls überwinden müssen, die Daten im HTML- und im XML-Format passieren lassen.

Diese Unterschiede sorgen dafür, dass ADO.NET im Bereich der Webanwendungen klare Vorteile beim Datenzugriff bietet. Aber auch bei den Desktopanwendungen kann der Einsatz von ADO.NET sinnvoll sein. Das werden Sie an den folgenden Beispielen sehen.

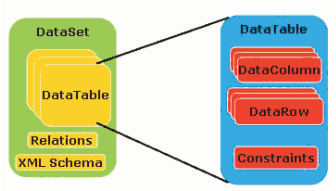
8.2 Objekte in ADO .NET



Wie bereits erwähnt, ist das meistgenutzte Objekt in ADO .NET das DataSet-Objekt. Sie sehen es mit seinen Eigenschaften, Methoden und Unterobjekten in Bild 1. Weitere Objekte, die Ihnen bei der Programmierung mit ADO.NET immer wieder begegnen werden, sind in Tabelle 1 beschrieben.

Tabelle 1: ADO .NET-Objekte für die Manipulation von Daten

`DataSet`	Das Objekt wird in Verbindung mit anderen Datensteuerelementen benutzt, um Ergebnisse von Commands und DataAdapters zu speichern. Im Gegensatz zum Recordset von ADO und DAO ist das Dataset in der Lage, Daten hierarchisch darzustellen. Mit Hilfe der Eigenschaften und Auflistungen des `DataSet`-Objekts können Sie alles von der Beziehung bis hin zur einzelnen Zeile oder Spalte erreichen.
`DataTable`	`DataTable` ist eines der Objekte des `DataSets`, das es Ihnen erlaubt, einzelne Datentabellen zu bearbeiten. Es ähnelt dem Recordset von ADO.
`DataView`	Mit diesem Objekt können Sie Ihre Daten filtern und sortieren, um verschiedene Ansichten der Daten zu haben. Jedes `DataTable`-Objekt hat einen `DefaultView`, der den Ausgangs-`DataView` darstellt. Dieser kann modifiziert und gespeichert werden.
`DataRow`	Dieses Objekt ermöglicht es Ihnen, einzelne Zeilen Ihrer `DataTable` zu modifizieren. Sie können sich das Objekt wie einen Datencache vorstellen, den Sie bearbeiten können, das heißt, Sie können Daten ändern, hinzufügen und löschen. Die Änderungen schreiben Sie dann zurück in das Recordset, indem Sie SQL-Befehle auf dem Server ausführen.
`DataColumn`	Das Objekt repräsentiert Spalten. Das Interessante daran ist, dass Sie sowohl Schemainformationen als auch Daten erhalten können. Möchten Sie zum Beispiel ein Listenfeld mit Feldnamen füllen, können Sie die DataColumn Collection einer `DataRow` durchlaufen und die Feldnamen auslesen.
`PrimaryKey`	Dieses Objekt erlaubt es Ihnen, einen Primärschlüssel für eine `DataTable` anzugeben, der zum Beispiel bei der Verwendung der `Find`-Methode wichtig ist.



.NET umfasst so genannte Data-Provider-Klassen, die zusammen mit den ADO-.NET-Objekten den Datenzugriff ermöglichen. Bild 2 enthält einige dieser Klassen. Bei der Entwicklung von Visual-Studio-.NET-Anwendungen müssen Sie beachten, dass diese aus mehreren Assemblies bestehen, die wiederum mehrere Namensräume enthalten. Namensräume bestehen aus einer oder mehreren Klassen oder Objekten. Deshalb heißt der Namensraum für das `OleDb`-Objekt zum Beispiel `System.Data.OleDb`. Sie finden diese Objekte im Objektkatalog.

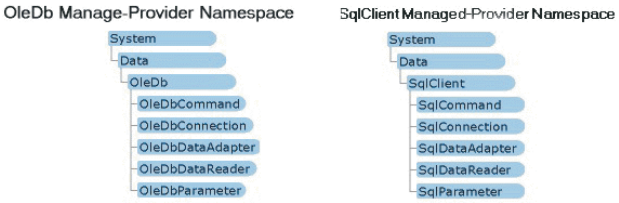


Tabelle 2 enthält eine Zusammenfassung einiger Objekte der .NET-Data-Provider-Klasse.

Tabelle 2: .NET-Data-Provider-Klassen dienen ebenfalls der Datenmanipulation.

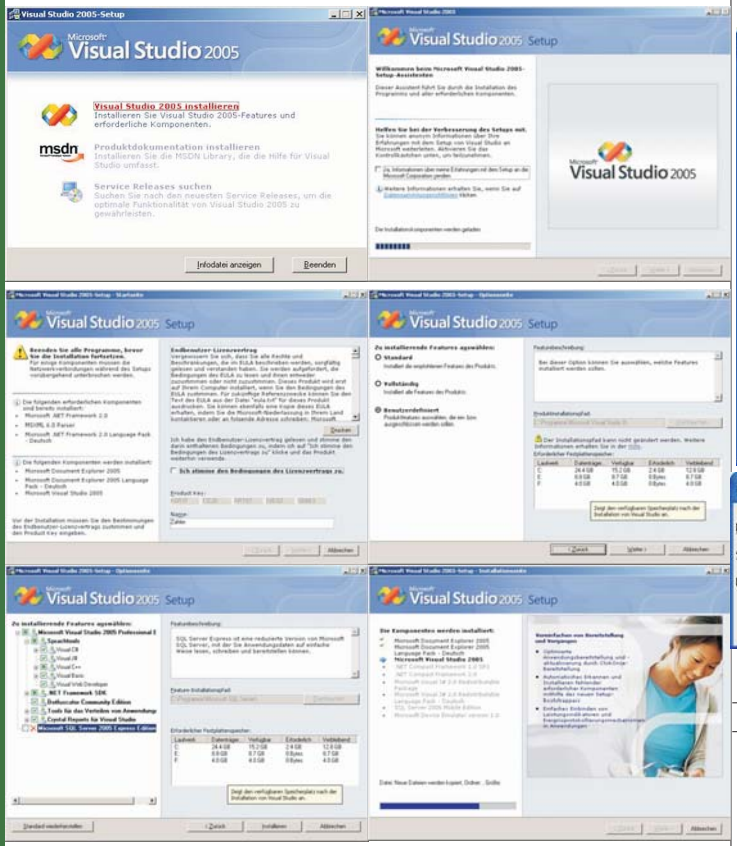
Objekt	Beschreibung
`Command`	Ähnlich dem ADO-Command-Objekt dient es dazu, Stored Procedures auszuführen. Im Gegensatz zu ADO können Sie ein `DataReader`-Objekt erstellen, indem Sie die Methode `ExecuteReader` ausführen.
`Connection`	Dieses Objekt öffnet eine Verbindung zum Server und zu der Datenbank, mit der Sie arbeiten wollen. Im Unterschied zum ADO-Connection-Objekt hängt es von dem Objekt ab, mit dem Sie arbeiten (`DataReader` oder `DataSet`), ob die Verbindung bestehen bleibt.
`DataAdapter`	Der `DataAdapter` ist ein echtes Arbeitstier. Das Objekt ermöglicht das Erzeugen von SQL-Befehlen und das Füllen von Datasets. Es erzeugt außerdem Aktionsabfragen wie Insert, `Update` und `Delete`.
`DataReader`	Erstellt einen read-only, forward-only Datastream, der sich besonders für Steuerelemente wie Listen- und Kombinationsfelder eignet.
`Parameter`	Dieses Objekt erlaubt es Ihnen, Parameter für DataAdapter-Objekte zu spezifizieren.

Die OleDb-Datensteuerelemente kommen in verschiedenen Backends zum Einsatz, während die SqlClient-Datensteuerelemente nur mit dem SQL-Server funktionieren. Das Gleiche gilt für die Objekte. Wenn Sie sicher sind, dass Sie nur den SQL-Server als Backend benutzen, ist die Performance besser, wenn Sie die SqlClient-Objekte benutzen.

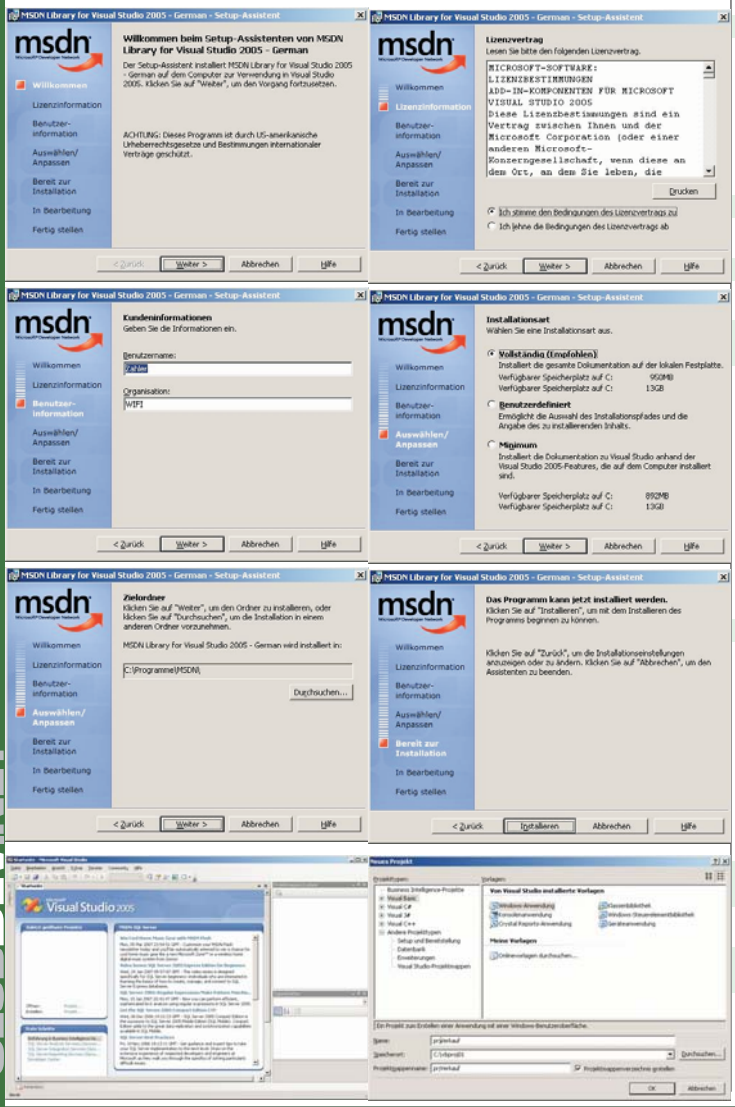
Im Folgenden lernen Sie den Einsatz der Objekte anhand von Beispielen kennen, die mit Windows Forms arbeiten. Das bedeutet jedoch nicht, dass die Mehrzahl der Objekte nicht auch in Web Forms benutzt werden kann.

8.3 Installation von Visual Studio 2005

Setupvorgang für Visual Studio 2005

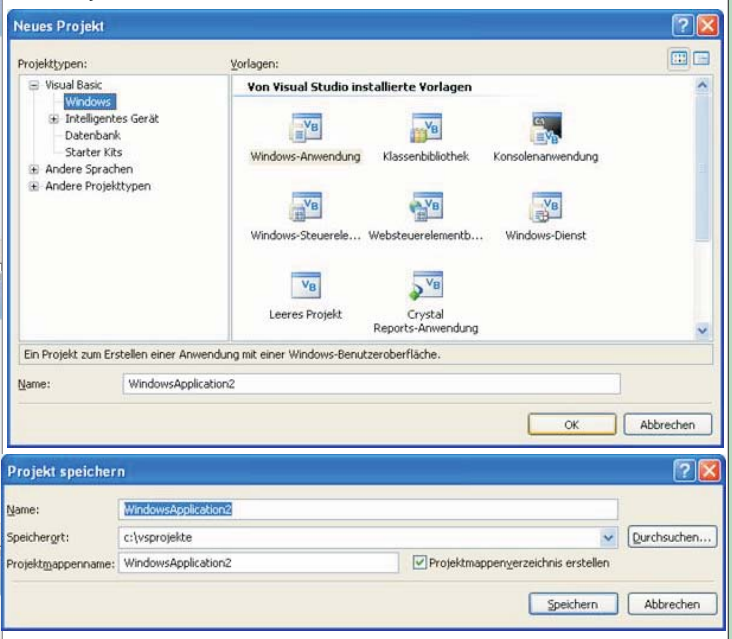


Die Visual Studio-Hilfe (Link *"Produktdokumentation"*) ist ein Bestandteil von MSDN und muss extra installiert werden:



8.4 Erstellen neuer Visual Basic.NET-Projekte

Der erste Schritt eines neuen VB.NET-Projekts besteht in der Anlage eines Projektordners, der mehrere Dateien enthält.



*.sln (Solution): ehemalige Projektdatei

8.5 ADO-Connection-Strings

Webquelle: `http://www.connectionstrings.com`

Um eine Datenquelle anzusprechen, verwendet ADO.NET (so wie auch ADO) Connection-Strings, die die Konfiguration der Schnittstelle zur Datenbank enthalten.

Dieser Connection-String ist für die Aktivierung eines `SqlConnection`-Objekts nötig.

```
Dim objConn As New SqlConnection(My.Settings.AuftragConnectionString)
```

In diesem Beispiel wird auf einen bereits im Projekt vorhandenen `ConnectionString` verwiesen.

Je nachdem, welche Möglichkeit des Datenbankzugriffs zur Verfügung steht, gibt es für `ConnectionString` folgende Möglichkeiten.

- Angabe einer vorbereiteten ODBC-Schnittstelle (DSN):

```
objConn.Open ConnectionString
```

Beispiel 1: Zugriff auf System-DSN

```
objConn.Open odbcAuftrag
```

Beispiel 2: Zugriff auf File-DSN

```
set cnn = server.createobject("ADODB.Connection")
cnn.open "FILEDSN=DSNName"
```

- Verwendung des Microsoft OLE DB-Providers für ODBC-Schnittstellen (= MSDASQL):

Mit dieser Variante kann auf ODBC-fähige Datenbanken zugegriffen werden, wobei die ODBC-Schnittstelle erst hier softwaremäßig konfiguriert wird.

Diese beiden Varianten sind älter und sollten – wenn möglich – nicht mehr verwendet werden.

Beispiel 1: Zugriff auf Access-Datenbank

```
PROVIDER=MSDASQL;DRIVER={Microsoft Access Driver (*.mdb)};DBQ=C:\news.mdb
```

Provider	die MSDASQL Bibliothek
`Driver`	welcher ODBC-Treiber wird verwendet (Achtung: exakte Schreibweise erforderlich
`DBQ`	absoluter Pfad zur MDB-Datei

Beispiel 2: Zugriff auf Excel-Tabelle

```
Provider=MSDASQL; Driver={Microsoft Excel Driver (*.xls)};
DBQ=C:\path\filename.xls;
```

Beispiel 3: Zugriff auf Text-Datenbank

```
Provider=MSDASQL; Driver={Microsoft Text Driver (*.txt; *.csv)};
DBQ=C:\path\;
```

Hinweis: Es wird – solange kein konkreter Zugriff erfolgt – kein Dateiname angegeben!

Beispiel 4: Zugriff auf SQL Server-Datenbank

```
Provider=MSDASQL; Driver={SQL Server}; Server=server_name_or_address;
Database=database_name; UID=username; PWD=password;
```

- Verwendung des Microsoft Jet.OLEDB.4.0-Providers für Access:

Beispiel 1: Zugriff auf Access-Datenbank ohne Anmeldung

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\path\filename.mdb;
```

Beispiel 2: Zugriff auf Access-Datenbank mit Anmeldung

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\path\filename.mdb;User
ID=admin; Password=;
```

- Verwendung des Microsoft SQLOLEDB-Providers für SQL Server (empfohlen):

Syntaxbeispiel:

```
Provider=SQLOLEDB;Data Source=server_name_or_address;Initial
Catalog=database_name;User ID=username;Password=password;Network
Library=dbmssocn;
```

Für den Parameter `Network Library` können folgende Werte für die Netzwerkkommunikation zwischen Datenbank-Client und SQL Server gewählt werden:

Network Library	Library Name
TCP/IP	dbmssocn
Named Pipes	dbnmpntw
Multiprotocol (RPC)	dbmsrpcn
NWLink IPX/SPX	dbmsspxn
AppleTalk	dbmsadsn
Banyan VINES	dbmsvinn

Beispiel 1: Ansprechen von SQL Server über Windows-Authentifizierung ("vertraute Verbindung")

```
objConn.Open "Provider=sqloledb;" & _
          "Data Source=myServerName;" & _
          "Initial Catalog=myDatabaseName;" & _
          "Integrated Security=SSPI"
```

Beispiel 2: Dialog einblenden, um Benutzername und Kennwort abzufragen

```
objConn.Provider = "sqloledb"
objConn.Properties("Prompt") = adPromptAlways
objConn.Open "Data Source=myServerName;" & _
          "Initial Catalog=myDatabaseName"
```

Beispiel 3: Ansprechen einer SQL Server-Instanz auf demselben Computer

```
objConn.Open "Provider=sqloledb;" & _
          "Data Source=(local);" & _
          "Initial Catalog=myDatabaseName;" & _
          "User ID=myUsername;" & _
          "Password=myPassword"
```

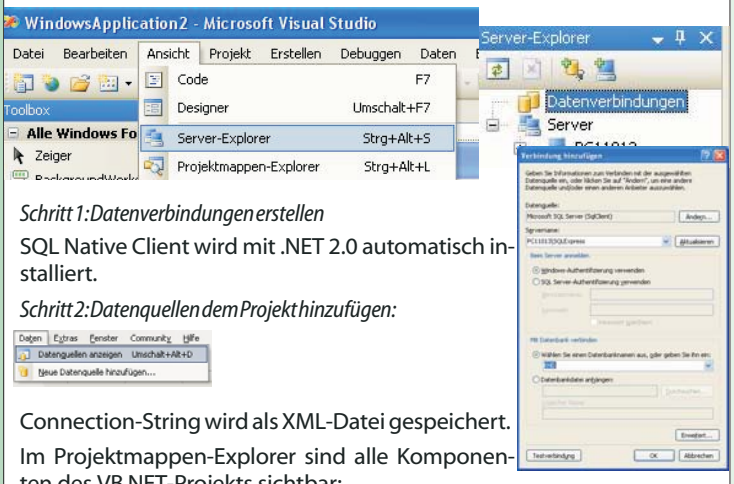
Beispiel 4: Ansprechen von SQL Server auf einem entfernten Computer über die IP-Adresse:

```
objConn.Open "Provider=sqloledb;" & _
          "Network Library=DBMSSOCN;" & _
          "Data Source=xxx.xxx.xxx.xxx,1433;" & _
          "Initial Catalog=myDatabaseName;" & _
          "User ID=myUsername;" & _
          "Password=myPassword"
```

xxx.xxx.xxx.xxx ist die IP-Adresse des SQL Servers, 1433 ist der standardmäßig von SQL Server verwendete TCP-Port.

8.6 Beispiel 1: Verwenden des Assistenten zum Hinzufügen von Datenquellen

Zeigen Sie zunächst den Server-Explorer an:



Schritt 1: Datenverbindungen erstellen

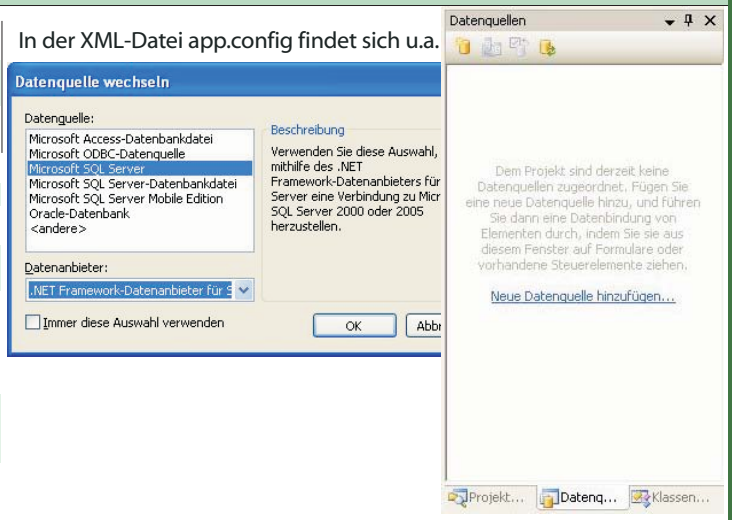
SQL Native Client wird mit .NET 2.0 automatisch installiert.

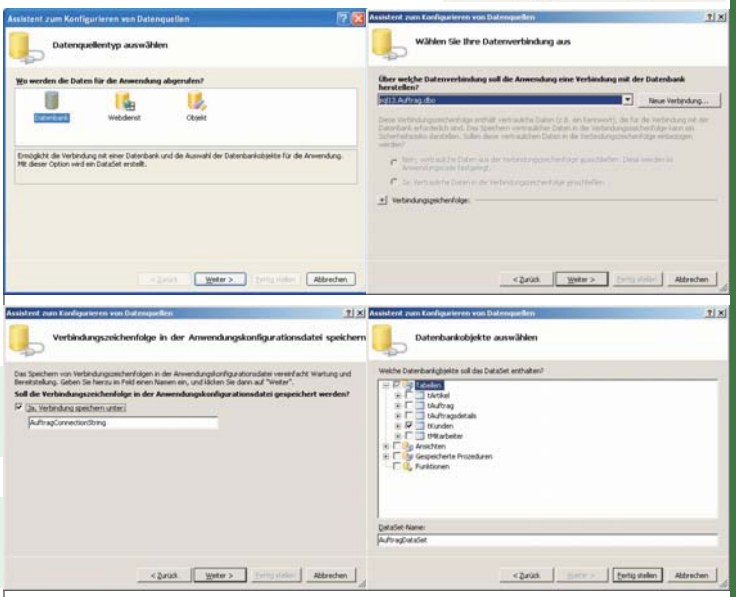
Schritt 2: Datenquellen dem Projekt hinzufügen:

Connection-String wird als XML-Datei gespeichert.

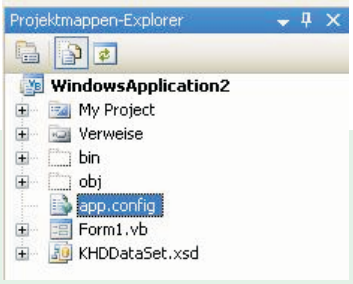
Im Projektmappen-Explorer sind alle Komponenten des VB.NET-Projekts sichtbar:

In der XML-Datei app.config findet sich u.a.





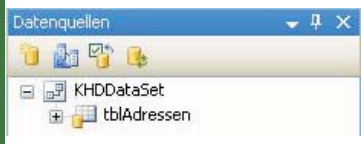
auch der ADO.NET ConnectionString:




```
<?xml version="1.0" encoding="utf-8"
?>
<configuration>
    <configSections>
    </configSections>
    <connectionStrings>
        <add
name="WindowsApplication2.My.MySettin
gs.KHDConnectionString"
            connectionString="Data Source=PC11013\SQLExpress;
              Initial Catalog=KHD;Integrated Security=True"
            providerName="System.Data.SqlClient" />
    </connectionStrings>
    <system.diagnostics>
        <sources>
            <!-- Dieser Abschnitt definiert die
                 Protokollierungskonfiguration für My.Application.Log -->
            <source name="DefaultSource" switchName="DefaultSwitch">
                <listeners>
                    <add name="FileLog"/>
                    <!-- Auskommentierung des nachfolgenden
                        Abschnitts aufheben, um in das
                        Anwendungsereignisprotokoll zu schreiben -->
                    <!--<add name="EventLog"/>-->
                </listeners>
            </source>
        </sources>
        <switches>
            <add name="DefaultSwitch" value="Information" />
        </switches>
        <sharedListeners>
            <add name="FileLog"
            type="Microsoft.VisualBasic.Logging.FileLogTraceListener,
             Microsoft.VisualBasic, Version=8.0.0.0, Culture=neutral,
             PublicKeyToken=b03f5f7f11d50a3a, processorArchitecture=MSIL"
```

```
                initializeData="FileLogWriter"/>
            <!-- Auskommentierung des nachfolgenden Abschnitts aufheben
und APPLICATION_NAME durch den Namen der Anwendung ersetzen, um in das
Anwendungsereignisprotokoll zu schreiben -->
            <!--<add name="EventLog"
                type="System.Diagnostics.EventLogTraceListener"
                initializeData="APPLICATION_NAME"/> -->
        </sharedListeners>
    </system.diagnostics>
</configuration>
```

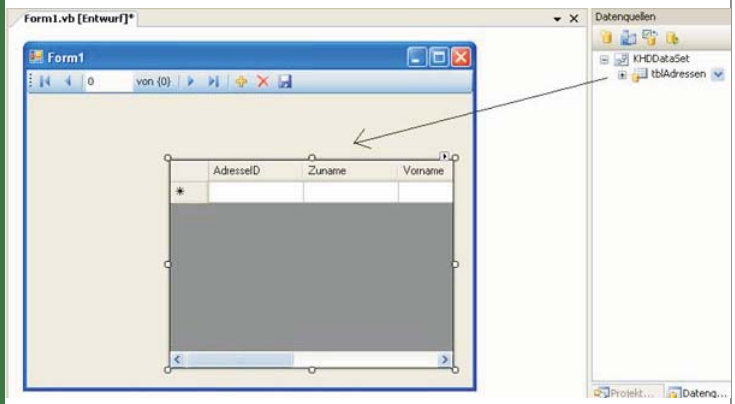
Ergebnis



Mit Drag & Drop in den Entwurfsdesigner-Bereich ziehen:

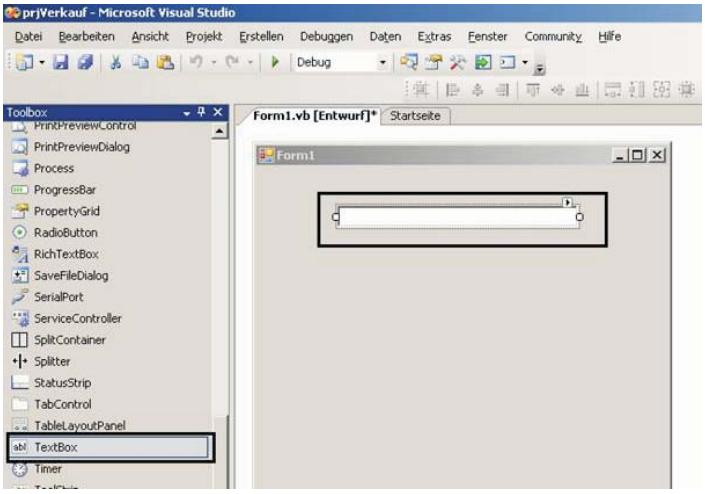
Es entsteht ein gebundenes `DataGrid`-Steuerelement, welches bereits funktioniert.

Auf dieselbe Art können Unterformulare erstellt werden.



8.7 Beispiel 2: Erstellen eigener Formulare

Wählen Sie die Klasse "`TextBox`" in der Toolbox aus und zeichnen Sie dann im Entwurfsfenster ein Rechteck. Sie erzeugen damit eine Instanz der Textboxklasse:



Nun legen Sie im Eigenschaftsfenster die Eigenschaften des neuen Textbox-Objekts fest:

(Name) `txtVorname`

Vergessen Sie nicht, auch für das Formular-Objekt einen Namen zu vergeben:

`frmDateneingabe`

Schritt 1: Importieren Sie den Namespace System.Data.SqlClient (wir wollen als Provider den SqlClient-Provider verwenden)

```
Imports System.Data.SqlClient
Public Class frmAdresseneingabe
End Class
```

Schritt 2: Erstellen Sie ein neues SqlConnection-Objekt

```
Dim objConn As New SqlConnection(My.Settings.AuftragConnectionString)
```

In unserem Beispiel können wir den bereits gespeicherten `ConnectionString` verwenden.

8.8 SqlCommand-Objekte

`SqlCommand`-Objekte sind vielseitig einsetzbar, weil sie in der Lage sind, sowohl mit SQL-Statements (insbesondere `INSERT`, `UPDATE` und `DELETE`-Statements) als auch – in Verbindung mit SQL Server – mit gespeicherten Prozeduren und Sichten umzugehen.

Eigenschaften

Eigenschaft	Bedeutung
`Connection`	Aktives `Connection`-Objekt, das die Verbindung zur Datenbank herstellt
`CommandType`	Was ist in der Eigenschaft `CommandText` enthalten? `CommandType.Text` ... SQL-Statement `CommandType.TableDirect` ... Tabellenname `CommandType.StoredProcedure` ... Name einer gespeicherten Prozedur
`CommandText`	SQL Statement oder Tabellenname oder gespeicherte Prozedur
`Parameters(n)`	`n` = Nummer des Parameters in der gespeicherten Prozedur
`Parameters(n).Value`	Wert des `n`-ten Parameters in der gespeicherten Prozedur

Methoden

Methode	Bedeutung
`ExecuteNonQuery`	Führt eine Änderungsabfrage (`INSERT`, `UPDATE`, `DELETE`) ohne `ResultSet` aus; gibt die Anzahl der betroffenen Zeilen zurück
`ExecuteReader`	Befüllt ein `SqlDataReader`-Objekt, welches nur zum Lesen von Daten verwendet werden kann
`ExecuteScalar`	Führt eine Abfrage durch und gibt die erste Zeile der ersten Spalte im `ResultSet` zurück
`ExecuteXmlReader`	Befüllt ein `XmlReader`-Objekt, welches nur zum Lesen von XML-Daten verwendet werden kann

8.9 Auslesen von Daten mit Hilfe eines SqlDataReader-Objekts

a) SQL-Anweisungen ohne Parameter

SqlDataReader-Objekte eignen sich ausschließlich zum Auslesen von Daten. Datenänderungen sind mit solchen Objekten nicht möglich.

Der SQL-Code selbst wird in einem eigenen Objekt der Klasse `SqlCommand` hinterlegt.

Schritt 1: Deklarieren Sie ein SqlDataReader-Objekt.

```
    Dim rAdresse As SqlClient.SqlDataReader
```

Schritt 2: Erstellen Sie ein neues SqlCommand-Objekt und befüllen Sie es mit einem SQL-Befehl. Führen Sie das SQL-Kommando aus und speichern Sie das ResultSet in einem SqlDataReader-Objekt.

```
    Private Sub prjEingabe_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load
        Dim sSQL As String
        Dim cmdKunden As SqlCommand
        sSQL = "select * from tKunden"
        cmdKunden = New SqlCommand(sSQL, objConn)
        objConn.Open()
        rAdresse = cmdKunden.ExecuteReader
        DatenAnzeigen()
    End Sub
```

Schritt 3: Erstellen Sie eine Prozedur DatenLesen(). Diese Prozedur stellt in der grafischen Oberfläche die Inhalte des DataReader-Objekts dar.

```
    Private Sub DatenAnzeigen()
        If rAdresse.HasRows = True Then
            rAdresse.Read()
            txtVorname.Text = rAdresse.Item("Vorname").ToString
            txtNachname.Text = rAdresse.Item("Nachname").ToString
        End If
    End Sub
```

b) SQL-Anweisungen mit Parameter

Wir adaptieren Beispiel a) wie folgt:

```
    Private Sub prjEingabe_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles MyBase.Load
        Dim sSQL As String
        Dim cmdKunden As SqlCommand
        sSQL = "SELECT * FROM tKunden " & "WHERE KdNr = @KdNr"
        cmdKunden = New SqlCommand(sSQL, objConn)
        cmdKunden.Parameters.Add("@KdNr", SqlDbType.Int).Value = 153
```

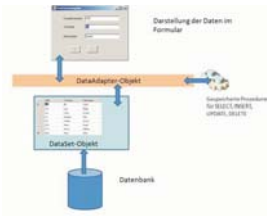
```
        objConn.Open()
        rAdresse = cmdKunden.ExecuteReader
        DatenAnzeigen()
    End Sub
```

8.10 Auslesen und Ändern von Daten mit Hilfe eines DataAdapter-Objekts

Konzept: Grundprinzip ist, dass während des Bearbeitungsvorgangs am Client keine aktive Verbindung zur SQL Server-Datenbank aufrechterhalten wird.

Das DataSet-Objekt bildet dabei den gemeinsamen Rahmen für Daten, die aus der Server-Datenbank ausgelesen werden sollen. Das Befüllen des DataAdapter-Objekts geschieht folgendermaßen:

```
        Dim sSQL As String = "select * from tKunden"
        da = New SqlDataAdapter(sSQL, objConn)
        ds = New DataSet
        da.Fill(ds)  'DataAdapter wird mit Daten aus dem DataSet befüllt
```



Erstellen Sie am SQL Server eine gespeicherte Prozedur, die einen durch die Kundennummer vorgegebenen Datensatz der Tabelle tKunden aktualisiert, wie folgt:

```
create proc dbo.pKundeAendern
@KdNr int,
  @Vorname nvarchar(50),
  @Nachname nvarchar(50)
as
  update dbo.tKunden
    set Vorname=@Vorname, Nachname=@Nachname
    where KdNr=@KdNr
```

Idee der folgenden kleinen Applikation ist es, die Navigation durch die Datensätze der Kundentabelle zu ermöglichen, wobei eventuelle Änderungen am Client sofort auch serverseitig gespeichert werden sollen.


```
Imports System.Data.SqlClient
Public Class frmDateneingabe
    Dim objConn As New SqlConnection(My.Settings.AuftragConnectionString)
    Dim rAdresse As SqlClient.SqlDataReader
    Dim da As SqlDataAdapter
    Dim ds As DataSet
    Dim i As Integer
    Dim cmdKunden As SqlCommand
    Private Sub prjEingabe_Load(ByVal sender As System.Object, ByVal e As
        System.EventArgs) Handles MyBase.Load
        Dim sSQL As String = "select * from tKunden"
        da = New SqlDataAdapter(sSQL, objConn)
        ds = New DataSet
        da.Fill(ds)  'DataAdapter wird mit Daten aus dem DataSet befüllt
        cmdKunden = New SqlCommand("pKundeAendern", objConn)
        cmdKunden.CommandType = CommandType.StoredProcedure
        cmdKunden.Parameters.Add("@KdNr", SqlDbType.Int, 4, "KdNr")
        cmdKunden.Parameters.Add("@Vorname", SqlDbType.NVarChar, 50,
"Vorname")
        cmdKunden.Parameters.Add("@Nachname", SqlDbType.NVarChar, 50,
"Nachname")
        DatenAnzeigen(0)
        da.UpdateCommand = cmdKunden
    End Sub
    Private Sub DatenAnzeigen(ByVal i As Integer)
        txtKdNr.Text = ds.Tables(0).Rows(i)("KdNr").ToString
        txtVorname.Text = ds.Tables(0).Rows(i)("Vorname").ToString
        txtNachname.Text = ds.Tables(0).Rows(i)("Nachname").ToString
    End Sub
    Private Sub DatenSpeichern()
        ds.Tables(0).Rows(i).Item("KdNr") = CInt(txtKdNr.Text)
        ds.Tables(0).Rows(i).Item("Vorname") = txtVorname.Text
        ds.Tables(0).Rows(i).Item("Nachname") = txtNachname.Text
        da.Update(ds)
    End Sub
    Private Sub cmdForward_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles cmdForward.Click
        DatenSpeichern()
        i = i + 1
        If i >= ds.Tables(0).Rows.Count Then
            MsgBox("Ende der Datensatzgruppe erreicht")
            i = ds.Tables(0).Rows.Count - 1
        End If
        DatenAnzeigen(i)
    End Sub
    Private Sub cmdBack_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles cmdBack.Click
        DatenSpeichern()
        i = i - 1
        If i < 0 Then
            MsgBox("Anfang der Datensatzgruppe erreicht")
            i = 0
        End If
        DatenAnzeigen(i)
    End Sub
End Class
```

8.11 Arbeiten mit Fehlermeldungen

```
ALTER proc dbo.pKundeAendern
  @KdNr int,
  @Vorname nvarchar(50),
  @Nachname nvarchar(50)
as
  SET NOCOUNT ON
  if (isnull(@KdNr,'')='') or
    (isnull(@Nachname,'')='')
  begin
    raiserror(50011,1,16)
    return
  end;
    update dbo.tKunden
    set Vorname=@Vorname, Nachname=@Nachname
    where KdNr=@KdNr;
  SET NOCOUNT OFF
/* Hinzufügen neuer benutzerdefinierter Fehlermeldung */
exec sp_addmessage 50011, 16,
    'Datensatz konnte nicht geändert werden, da Nachname NULL
ist','us_english'
```

1.Schritt:

```
    Dim WithEvents objConn As _
      New SqlConnection(My.Settings.AuftragConnectionString)
    'bewirkt, dass bei objCOnn Ereignisse zur Auswahl stehen
```

2.Schritt: Ereignis InfoMessage behandeln, indem die Nummer und der Text der in der Stored Procedure definierten Fehlermeldung ausgegeben wird.

```
    Private Sub objConn_InfoMessage(ByVal sender As Object, _
      ByVal e As System.Data.SqlClient.SqlInfoMessageEventArgs)
      Handles objConn.InfoMessage
        For i As Integer = 0 To e.Errors.Count - 1
            MsgBox(Str(e.Errors(i).Number) + e.Errors(i).Message)
        Next
    End Sub
```

Löschen

Schritt 1: Gespeicherte Prozedur erstellen

```
create proc dbo.pKundeLoeschen
  @KdNr int
as
  SET NOCOUNT ON
  if (isnull(@KdNr,'')='')
  begin
    raiserror(50011,1,16)
    return
  end;
    delete dbo.tKunden
    where KdNr=@KdNr;
  SET NOCOUNT OFF
```

Schritt 2: Verknüpfen Sie zunächst ein neues SqlCommand-Objekt mit der Gespeicherten Prozedur "Löschen" und weisen Sie dieses neue SqlCommand-Objekt dann der DeleteCommand-Eigenschaft des DataAdapters zu.

```
    Dim cmdDelKunden As SqlCommand
    Private Sub frmKundenanzeige_Load(…)
…
        cmdDelKunden = New SqlCommand("pKundeLoeschen", objConn)
        cmdDelKunden.CommandType = CommandType.StoredProcedure
        cmdDelKunden.Parameters.Add("@KdNr", SqlDbType.Int, 4, "KdNr")
…
        da.DeleteCommand = cmdDelKunden
    End Sub
```

Schritt 3: Programmieren Sie einen Butten "Löschen" mit folgendem Ereigniscode:

```
    Private Sub butLoeschen_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butLoeschen.Click
        ds.Tables(0).Rows(i).Delete()
```

```
        da.Update(ds)
        If i = ds.Tables(0).Rows.Count Then
            i = i - 1
        End If
        DatenAnzeigen(i)
    End Sub
```

Einfügen neuer Datensätze

Schritt 1: Programmieren Sie eine Gespeicherte Prozedur

```
create proc dbo.pKundeInsert
  @KdNr int,
  @Vorname nvarchar(50),
  @Nachname nvarchar(50)
as
  SET NOCOUNT ON
  if (isnull(@KdNr,'')='') or
    (isnull(@Nachname,'')='')
  begin
    raiserror(50011,1,16)
    return
  end;
     insert dbo.tKunden (KdNr, Vorname, Nachname)
     values
        (@KdNr, @Vorname, @Nachname);
     SET NOCOUNT OFF
```

Schritt 2: Erstellen Sie einen Button mit der Beschriftung "Neu". Beim Klicken auf diesen Button verschwinden die Standard-Schaltflächen, denn zunächst soll der User sinnvolle Daten bereitstellen, bevor wieder navigiert werden kann. Stattdessen wird ein Button "Speichern" eingeblendet.

```
    Dim zeile As DataRow
    Private Sub butNeu_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles butNeu.Click
        zeile = ds.Tables(0).NewRow()
        txtKdNr.Text = ""
        txtVorname.Text = ""
        txtNachname.Text = ""
        butNeu.Visible = False
        butBack.Visible = False
        butForward.Visible = False
        butSpeichern.Visible = True
    End Sub
   Private Sub butSpeichern_Click(ByVal sender As _
          System.Object, ByVal e As System.EventArgs) _
          Handles butSpeichern.Click
        zeile.Item("KdNr") = CInt(txtKdNr.Text)
        zeile.Item("Vorname") = txtVorname.Text
        zeile.Item("Nachname") = txtNachname.Text
        ds.Tables(0).Rows.Add(zeile)
        da.Update(ds)
        butNeu.Visible = True
        butBack.Visible = True
        butForward.Visible = True
        butSpeichern.Visible = False
    End Sub
```

8.12 Nutzen von Anwendungsrollen *(Application Roles)*

Vorgangsweise

1. Erstellen Sie eine Applikationsrolle

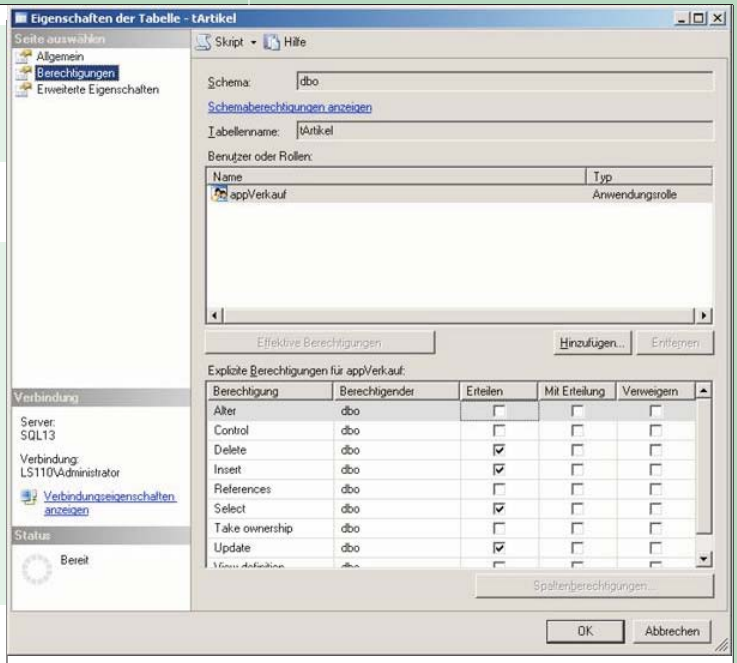
```
create application role appManager
  with password = 'wifi@wifi1'
```

2. Vergeben Sie Berechtigungen an die Applikationsrolle

```
grant select,insert,update,delete
  on dbo.tArtikel
  to AppVerkauf;
```

3. In der Client-Applikation erstellen Sie ein `SqlCommand`-Objekt, mit dem Sie die gespeicherte Prozedur `sp_setapprole` ausführen, um in den Kontext der Applikationsrolle zu wechseln:

```
Public Class frmKundenanzeige
…
    Dim cmdAppRole As SqlCommand
   …
    Private Sub frmKundenanzeige_Load(…) Handles MyBase.Load
   …
        objConn.Open()
        cmdAppRole = New SqlCommand("EXEC sp_setapprole AppVerkauf,
          'wifi@wifi1'", objConn)
```




```
        cmdAppRole.ExecuteNonQuery()
    End Sub
End Class
```

8.13 Befüllen von ComboBox- und ListBox-Steuerelementen

```
        For t As Integer = 0 To ds.Tables(0).Rows.Count - 1
            cmbKunden.Items.Add(ds.Tables(0).Rows(t)("Nachname"))
        Next
```