

Flexible Bindung

Die neue Datenbindung bei Visual Studio 2005 und Visual Basic 2005

von Peter Monadjemi

Das Thema Datenbindung bei Visual Basic war in der Vergangenheit ein Thema mit Höhen und Tiefen. Was bei Visual Basic 3.0 auf der Basis von DAO ganz brauchbar funktionierte, wurde mit ADO in einen fast unbrauchbaren Zustand modernisiert. Bei .NET spielte die Datenbindung anfangs eher eine unscheinbare Rolle. Mit dem kommenden Visual Studio 2005 sieht alles wieder ganz anders aus – fast scheint es so, als hätten es die Microsoft-Entwickler endlich richtig gemacht.

In diesem Artikel wird die neue Art der Datenbindung beim .NET Framework vorgestellt. Gab es denn auch eine alte? Im Prinzip ja, auch wenn viele Entwickler übersehen haben dürften, dass es überhaupt so etwas wie eine Datenbindung gab. Das mag zum einen an etwas seltsam klingenden Klassen wie *CurrencyManager* und *BindingContext* in der Version 1.1 gelegen haben, der wie immer etwas dürftigen Dokumentation und nicht zuletzt an der oft fehlenden Notwendigkeit für eine „vollautomatische“ Datenbindung. Der Hauptgrund für die mangelnde Akzeptanz dürfte allerdings der Umstand gewesen sein, dass die Datenbindung ein wenig „halbfertig“ wirkte und

es am Ende auch war. Vermutlich war schon zum Zeitpunkt, als das Featureset für die Version 1.1 festgelegt wurde, klar, dass mit 2.0 alles anders werden würde, sodass man sich bei Microsoft in diesem Punkt keine allzu große Mühe gegeben hat. Doch genug der Kritik und Spekulationen, denn kritische Untertöne sind bei der neuen Art der Datenbindung sicher nicht mehr angebracht. Auch wenn diesem Artikel nur die Beta 2 von Visual Studio 2005 zu Grunde lag, die allerdings „featurekomplett“ gewesen sein dürfte, lautet das vorläufige Prädikat: Einfach zu implementieren, flexibel, solide und beinahe schon ein wenig genial. Wenn im Folgenden nur Beispiele in Visual Basic präsentiert werden, hat dies nichts damit zu tun, dass die Datenbindung nur mit dieser einen Sprache funktioniert – alles funktioniert selbstverständlich auch mit C# und J#. Und niemand wird „gezwungen“ es so zu machen, die alte Form der Datenbindung steht selbstverständlich nach wie vor zur Verfügung und wer lieber alles „zu Fuß“ erledigen möchte, wird davon nicht abgehalten oder in seinen Möglichkeiten eingeschränkt. Dass in einem Projekt eine Datenquelle angelegt wurde, bedeutet nicht, dass nun auch Felder oder ganze Tabellen auf ein Formular gezogen werden müssen – wer A sagt, muss

in diesem Fall nicht B sagen. Allerdings ist die automatische Datenbindung, nicht zuletzt dank der partiellen Klassen, so attraktiv, dass es nur wenige Gründe geben dürfte, auf sie zu verzichten.

Alles ganz easy

Drag-&-Drop-Programmierung besitzt im Allgemeinen einen schlechten Ruf. Entweder werden Hunderte von Codezeilen generiert, die Seiteneffekte nach sich ziehen, von denen die Entwickler am Anfang noch nicht einmal etwas ahnen, oder ein Assistent „zaubert“ etwas und der Programmierer erhält keine Möglichkeit, an dem Ergebnis etwas nachträglich zu ändern oder, wie es beim Datenadapterkonfigurationsassistenten der Fall ist, um eine Änderung durchzuführen, muss der Assistent stets komplett erneut durchlaufen werden, wobei vorherige Einstellungen jedes Mal überschrieben werden, da der Code komplett neu generiert wird. Bei Visual Studio 2005 besitzt die Drag-&-Drop-Datenbindung diese Nachteile nicht. Der erzeugte Code ist „sauber“ und stört dank partieller Klassen nicht im „Hauptprogramm“, die Datenquelle wird durch ein typisiertes *DataSet* sauber gekaspelt und der Assistent zum Konfigurieren der Datenquelle macht einen äußerst flexiblen Eindruck. Auch erfahrene

kurz & bündig

Inhalt

Überblick über die Datenbindung bei Visual Studio 2005 und Visual Basic 2005

Zusammenfassung

Die Datenbindung bei Visual Studio 2005 macht, was den Komfort angeht, einen großen Sprung nach vorne, bleibt aber optional und vom Typ der Datenquelle unabhängig. Im Mittelpunkt stehen die neuen Datenquellen von Visual Studio 2005, die pro Projekt angelegt werden und Tabellen und Abfragen einer Datenbank oder eine Objektbibliothek repräsentieren



Abb. 1: Alle Datenquellen eines Projekts werden im Datenquellen-Explorer aufgelistet

Datenbankprofis sollten das neue Drag-&-Drop-Prinzip zum Herstellen einer Datenbindung einmal ausprobieren und sich positiv überraschen lassen. Wetten, dass Sie schon mit einer einzigen Befehlszeile zu einer vollständigen Datenbindung kommen, die auch Updates und das Einfügen neuer Datensätze umfasst?

Fügen Sie als erstes über DATA\ADD NEW DATA SOURCE (alle Angaben basieren noch auf der Beta 2) eine neue Datenquelle zum Projekt hinzu (dieser Schritt muss für jedes Projekt neu durchgeführt werden – anders als die Datenumgebung bei Visual Basic 6, als deren indirekter und inoffizieller Nachfolger die Datenquellen wohl angesehen werden dürfen, lässt sich eine Datenquelle nicht als Datei speichern). Als Datenquellen kommen neben Daten-

banken und Web Services (die ein *DataSet* zurückgeben) auch reguläre Objektbibliotheken in Frage, die dazu keine besonderen Schnittstellen implementieren, sondern lediglich eine öffentliche (Daten-) Klasse und entsprechende *Factory*-Methoden zum Füllen der Datensätze bereitstellen müssen. XML-Dateien als Datenquellen waren in Beta 1 noch dabei, fielen aber später dem Feature-Rotstift zum Opfer und dürften auch eher eine Nischenfunktion spielen. Nach Auswahl der Datenbank über das Anlegen einer Verbindung (wie wäre es mit der *Northwind*-Datenbank) kann eine Tabelle, eine gespeicherte Prozedur oder ein View ausgewählt werden (wie wäre es mit der *Employees*-Tabelle und den vier Feldern *EmployeeID*, *LastName*, *FirstName* und *City*?). Achten Sie auf den Namen des *DataSet*, denn dies ist auch jener Name, den die Datenquelle erhält. Damit steht die Datenquelle und wird im Datenquellen-Explorer angezeigt (der aber nicht wie zu erwarten gewesen wäre über das ANSICHT-Menü, sondern nur über das DATEN-Menü geöffnet werden kann).

Jetzt kommt der große Augenblick. Ziehen Sie einfach die Tabelle auf das Formular und lassen Sie sich hoffentlich positiv überraschen. Nach dem die Fest-

platte wieder zur Ruhe gekommen ist, erscheinen auf dem Formular ein Grid (das neue *DataViewGrid*) und eine ansprechende Navigationsleiste, die an das gute, alte Datensteuerelement von Visual Basic 3.0 erinnert (es handelt sich hier aber um kein Datensteuerelement, sondern lediglich um das neue *ToolStrip*-Control, auf dem bereits eine Reihe von Buttons und eine Textbox für die Anzeige der Datensatznummer platziert wurden).

In das Komponentenfach wurden platziert: Das typisierte *DataSet*, eine *BindingSource*-Komponente, welche für die Datenbindung zuständig ist, eine Komponente mit dem Namen *EmployeesTableAdapter*, welche die Datensätze aus dem typisierten *DataSet* holt und auch das Update übernimmt, und zum Schluss eine Komponente mit dem Namen *EmployeesBindingNavigator*, welche die über die *BindingSource* zur Verfügung gestellte Datenquelle an die Navigationsleiste bindet. Auch wenn das Komponentenfach ein wenig „übervölkert“ wirkt, bleibt alles recht übersichtlich. Testen Sie das Ganze, indem Sie das Projekt starten. Sie werden feststellen, dass sich a) die Datensätze navigieren lassen (das war zu erwarten), sich b) einzelne Felder aktualisieren lassen, allerdings ohne dass eine Bestätigung erscheint (das war nicht unbedingt zu erwarten, erscheint aber auch nicht besonders spektakulär), und c) sich einzelne Datensätze hinzufügen lassen (das war nicht unbedingt zu erwarten, zumal es in diesem Punkt bei ADO.NET in der aktuellen Version „Probleme“ gab, wenn die Datensätze nicht den für das Update erforderlichen *RowVersion*-Wert besaßen). Auch das Löschen von Datensätzen funktioniert, allerdings nur in der *DataTable*, nicht in der Datenbank. Aus gutem Grund, denn das Löschen hätte aufgrund der referentiellen

Listing 1

Die Funktion *Delete* löscht einen Datensatz über die *Delete*-Methode des spezialisierten *TableAdapter*

```
Public Overloads Overridable Function Delete(ByVal CType(1,Integer)
    Original_EmployeeID As Integer, ByVal Original_
    LastName As String, ByVal Original_FirstName
    As String, ByVal Original_City As String) As Integer
    Me.Adapter.DeleteCommand.Parameters(0).Value =
        CType(Original_EmployeeID,Integer)
    If (Original_LastName Is Nothing) Then
        Throw New System.ArgumentNullException
            ("Original_LastName")
    Else
        Me.Adapter.DeleteCommand.Parameters(1).Value =
            CType(Original_LastName,String)
    End If
    If (Original_FirstName Is Nothing) Then
        Throw New System.ArgumentNullException
            ("Original_FirstName")
    Else
        Me.Adapter.DeleteCommand.Parameters(2).Value =
            CType(Original_FirstName,String)
    End If
    If (Original_City Is Nothing) Then
        Me.Adapter.DeleteCommand.Parameters(3).Value =
            Me.Adapter.DeleteCommand.Parameters(4).Value =
                System.DBNull.Value
    Else
        Me.Adapter.DeleteCommand.Parameters(3).Value =
            CType(0,Integer)
        Me.Adapter.DeleteCommand.Parameters(4).Value =
            CType(Original_City,String)
    End If
    Dim previousConnectionState As System.Data.
        ConnectionState = Me.Adapter.DeleteCommand.
            ConnectionState
    Me.Adapter.DeleteCommand.Connection.Open
    Try
        Return Me.Adapter.DeleteCommand.ExecuteNonQuery
    Finally
        If (previousConnectionState = System.Data.
            ConnectionState.Closed) Then
            Me.Adapter.DeleteCommand.Connection.Close
        End If
    End Try
End Function
```

Listing 2

Hinter dem *DeleteCommand* steckt ein *SqlCommand*-Objekt

```
Me.m_adapter.DeleteCommand.CommandText = "DELETE
FROM [dbo].[Employees] WHERE ((([EmployeeID]
=@Original_EmployeeID) AND ([&_
'LastName'] = @Original_LastName) AND ([FirstName] =
@Original_FirstName) AND ((@I'&_
'sNull_City = 1 AND [City] IS NULL) OR ([City] =
@Original_City)))"
```

Sie suchen die wirklich wichtigen Informationen rund um .NET?

Wir haben Sie!



Abb. 2: Die neue Art der Datenbindung in Aktion – das Ergebnis kann sich sehen lassen

Integritätsregeln auch das Löschen abhängiger Datensätze zur Folge. Wer das ebenfalls möchte, kommt um die Programmierung doch nicht ganz herum.

Wie gut ist die Erweiterbarkeit?

Diese Funktion nachträglich einzubauen, ist die erste Nagelprobe für die Qualität der neuen Bindung. Bekommt der Programmierer wieder einmal eine Custom-Lösung ohne echte Erweiterbarkeit oder handelt es sich um ein durchdachtes Konzept, das beliebig erweiterbar ist? Ein beherzter Doppelklick auf den *Delete*-Button bringt schnell ans Licht, dass gar kein Code eingefügt wurde – also konnte auch nichts gelöscht werden. Ein Blick in den Code, der mit dem *Update*-Button verknüpft ist, macht das Prinzip deutlich. Es ist der *TableAdapter*, der die komplette Funktionalität für das Abrufen, das Aktualisieren, das Einfügen und entsprechend auch für das Löschen von Datensätzen bereitstellt. Und siehe da, dieser besitzt tatsächlich eine *Delete*-Methode,

die allerdings für jeden Feldnamen den Originalwert erwartet. Bevor die Frage geklärt wird, woher diese Werte kommen, soll die nicht weniger wichtige Frage beantwortet werden, was genau die *Delete*-Methode macht. Führt sie irgendein vorbereitetes SQL-Kommando aus oder greift sie vielleicht im Blackbox-Modus á la ADO direkt auf die Datenbank zu? Die Antwort liegt sehr nahe, genauer gesagt, in jener Quelltextdatei, die das durch das Anlegen der Datenquelle entstandene typisierte *DataSet NwDs* begleitet. Ihr Name lautet entsprechend *NwDs.Designer.vb* und sie muss im Projekt-Explorer erst sichtbar gemacht werden. Auf diese Weise stellt sich heraus, dass es sich um eine kleine Funktion handelt (Listing 1), die automatisch beim Anlegen der Datenquelle generiert wurde. Auch wenn die Frage, welchen „Informationswert“ ein von einem Tool generiertes Stück Code haben soll, berechtigt ist, Listing 1 besitzt einen hohen Lehrwert, den es zeigt, wie Datensätze offiziell gelöscht werden. Das ei-

Klasse	Namespace	Bedeutung
<i>BindingSource</i>	<i>System.Windows.Forms</i>	Kapselt die Funktionalität von <i>BindingManager</i> und <i>BindingContext</i> und managt damit die Dateibindung. Der aktuelle Datensatz wird als <i>DataRowView</i> über die <i>Current</i> -Eigenschaft zur Verfügung gestellt. Leitet sich direkt von der <i>Component</i> -Klasse ab
<i>BindingNavigator</i>	<i>System.Windows.Forms</i>	Stellt die Verbindung zwischen der Datenquelle über das <i>BindingSource</i> -Objekt und der Navigationsleiste her.
<i>DataTableAdapter</i>	Gehört nicht zur .NET-Klassenbibliothek	Wird in der mit dem typisierten <i>DataSet</i> angelegten Quelltextdatei definiert.
<i>DataConnector</i>	<i>System.Windows.Forms</i>	Der alte Name der <i>BindingSource</i> -Klasse in Beta 1.
<i>DataNavigator</i>	<i>System.Windows.Forms</i>	Der alte Name der <i>BindingNavigator</i> -Klasse in Beta 1.

Tabelle 1: Klassen, die bei der Datenbindung eine Rolle spielen

Ihre Abo-Vorteile:

- Sie erhalten den 128 MB dot.net-**USB-Stick***
- Sie sparen rund 10% gegenüber dem Einzelverkauf
- Sie erhalten das **Riesen-Poster** „.NET Framework 2.0“
- Mit der **Jahres-CD** erhalten Sie alle Ausgaben des vergangenen Jahres gratis
- Sie verpassen keine Ausgabe
- Jede Ausgabe inklusive **Heft-CD** mit vielen Tools

Weitere Informationen und Bestellung unter www.dotnet-magazin.de

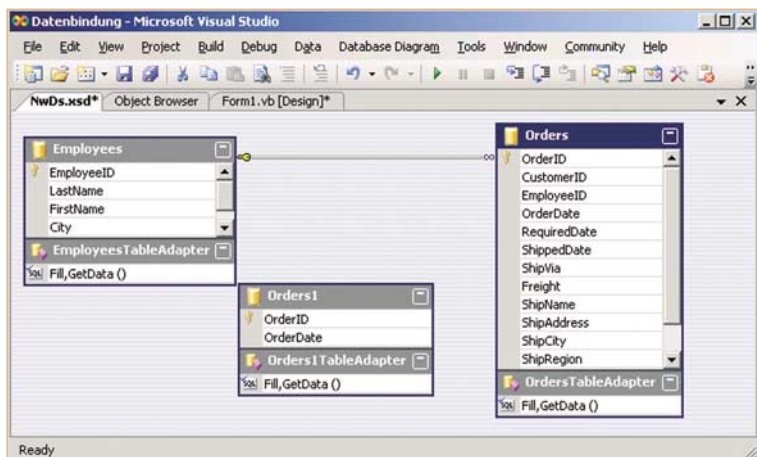


Abb. 3: Das DataSet wird im Designer bearbeitet, wobei für jede neue Abfrage automatisch die benötigten DataTable-Adapter generiert werden

gentliche *Delete*-Kommando ist ein *SqlCommand*-Objekt, dessen Initialisierung in Listing 2 enthalten ist. Um einen Datensatz zu löschen, muss daher im *Click*-Handler des Button lediglich die *Delete*-Methode des *TableAdapter* aufgerufen werden, wobei der Rückgabewert die Anzahl der gelöschten Datensätze angibt.

Bitmaps anzeigen

Wie flexibel die Datenbindung ist, soll zum Abschluss noch ein weiteres Mal getestet werden. Was war die „mit Abstand größte Herausforderung“, mit der sich ganze Generationen von Visual-Basic-Programmierern in der Vergangenheit konfrontiert sahen? Ganz klar, die Anzeige von Bitmaps. Und da die *Employees*-Tabelle mit *Photo* über ein Bildfeld verfügt, lautet die Frage natürlich, wie sich die Datenquelle dahingehend erweitern lässt, dass auch diese Bitmap angezeigt wird? Auch diese kleine Herausforderung lässt sich elegant lösen. Klicken Sie im Datenquellenfenster das *DataSet* mit der rechten Maustaste an und wählen Sie *CONFIGURE*

WITH WIZARD. Es erscheint das Datenquellendetailfenster, in dem Sie lediglich das *Photo*-Feld ankreuzen und auf *Finish* klicken müssen. Dass das *Photo*-Feld im Datenquellenexplorer mit einem typischen „Geht-nicht“-Symbol versehen wird, hat einen einfachen Grund: Es wurde noch kein Steuerelement ausgewählt, mit dem das Feld dargestellt werden soll. Wählen Sie in der Auswahlliste *PICTUREBOX* (über *Customize* können Sie jedes beliebige Steuerelement auswählen) und ziehen Sie das Feld in einer „vernünftigen“ Größe auf das Formular, wählen Sie aus der Aufgabenliste (die bei jedem Steuerelement bei Visual Studio 2005 über einen kleinen Pfeil in der rechten oberen Ecke angeboten wird) für *SizeMode StretchImage* aus und fertig ist die *PictureBox*. Starten Sie das Projekt erneut und Sie werden (hoffentlich erfreut) feststellen, dass zu dem aktuellen selektierten Angestellten auch das Konterfei angezeigt wird. Und das alles ohne einen weiteren Befehl. Übrigens kann das neue *DataGridView*-Control auch Bitmaps anzeigen – allerdings sieht das Ergebnis bei großen Bitmaps naturgemäß nicht sehr gut aus.

Den DataTableAdapter um Abfragen erweitern

Die *DataTableAdapter* sind die neuen „Kunstgebilde“, die durch das eine (auf einer Datenbank basierenden) Datenquelle stets begleitende typisierte *DataSet* zur Verfügung gestellt werden. Die Idee ist, dass der *DataTableAdapter* alle für das Abrufen und Aktualisieren der Daten erforderlichen Methoden zur Verfügung stellt. Ein Pluspunkt ist die leichte Erweiterbarkeit. Anders als ein generischer

Datenadapter, der immer nur für eine Abfrage steht, lassen sich die *DataTableAdapter* um beliebige Abfragen erweitern. Selektieren Sie dazu den *DataTableAdapter* im Komponentenfach und wählen Sie *Add Query*. Es öffnet sich eine Dialogbox, in der das SQL-Kommando entweder eingetippt oder „designt“ wird. Allerdings muss die neue Abfrage auf dem Datenbestand des *DataTableAdapter* basieren. Möchten Sie z.B. alle Aufträge der *Employees* sehen, geht dies zunächst nicht, da die *Orders*-Tabelle nicht Teil des Datenbestands ist. Sie müssen daher zunächst die Datenquelle neu konfigurieren und dabei einen weiteren *DataTableAdapter* hinzufügen, der z.B. für die Abfrage *SELECT OrderID, OrderDate From Orders, Employees Where Orders.EmployeeID=Employees.EmployeeID* steht. Sie werden feststellen, dass die neue Abfrage sehr elegant im Designer des *DataSet* zusammengestellt werden kann (Abbildung 3). Auch wenn hier genau wie bei der Vorgängerversion eine *XSD*-Datei im Hintergrund steht, kann ihr Inhalt in der IDE nicht sichtbar gemacht werden, da dies nicht notwendig ist. Damit wurde die Rolle von XML als Datenformat im Rahmen von Visual Studio endgültig zementiert. Es ist, im Zusammenspiel mit *XML Schema (XSD)*, unverzichtbar, doch der Entwickler bekommt es nicht mehr direkt zu sehen – was auch gut so ist. ●

Peter Monadjemi ist Chefredakteur des dot.net magazin. Seine Lieblingslektüre war für eine Zeit das Handbuch „Advanced Data Access“, das einst Visual Basic 2.0 begleitete, und das ihm lange Zeit ein großes Rätsel blieb. Inzwischen ist er zur Einsicht gelangt, dass die beste Datenbindung jene ist, die einfach nur funktioniert und ansonsten nicht weiter auffällt. Und das scheint auf die neue Art der Datenbindung bei Visual Studio 2005 zuzutreffen. Sie können ihn unter peterm@dotnet-magazin.de erreichen.

● Links & Literatur

- [1] Transkript eines Chats zum Thema Verbesserung bei der Datenbindung bei Visual Basic 2005 – msdn.microsoft.com/chats/transcripts/vstudio/vstudio_101404.aspx
- [2] Unterschiede bei der Datenbindung zwischen Beta 1 und 2 – blogs.msdn.com/vbteam/archive/2005/01/12/351495.aspx

Datenbindungsereignisse

Die Datenbindungsereignisse bei ADO waren praktisch unbrauchbar. Praktisch nicht dokumentiert, basierend auf einem etwas eigenartigen Konzept und bestimmt auch nicht fehlerfrei implementiert. Das liegt zum Glück lange zurück. Die Datenbindung beim .NET Framework 2.0 wird durch einen Satz überschaubarer Events geleitet. Die Wichtigsten finden Sie mit *AddingNew*, *CurrentChanged*, *CurrentItemChanged* und *PositionChanged* bei der *BindingSource*-Klasse. Ein kleiner Tipp: Bei Visual Studio 2005 werden die Events endlich auch im Eigenschaftsfenster aufgelistet.