Kapitel 9

Die Microsoft Chart-Controls

In diesem Kapitel:

Mit Visual C# 2010 hält endlich auch ein *Chart*-Control Einzug in die Toolbox, der Programmierer ist also nicht mehr auf Fremdkomponenten angewiesen.

Die sowohl für Windows Forms als auch für ASP.NET verfügbaren *Microsoft Chart Controls* ermöglichen die grafische Präsentation von Daten auf vielfältige Weise.

So lassen sich mit den Komponenten 35 verschiedene Diagrammtypen in 2D/3D-Darstellung anzeigen, Sie als Programmierer haben Einfluss auf Farben, Schatten etc., es dürfte für jeden Anspruch bzw. Geschmack etwas dabei sein.

HINWEIS Gemäß der Zielstellung dieses Buchs richtet sich unser Fokus auf die *Chart*-Features zur Datenbindung. Das vorliegende Kapitel kann deshalb keine umfassende Beschreibung der *Chart*-Controls liefern.

Mehr zu den *Chart*-Controls siehe

WWW http://code.msdn.microsoft.com/mschart

Allgemeine Chart-Features

Da ein *Chart*-Control ein ziemlich komplexes und tief gestaffeltes Objektmodell hat und eine verwirrende Vielfalt von Datenbindungsmethoden anbietet, sollen zunächst einige Grundbegriffe geklärt und das Funktionsprinzip anhand der simplen *Points.AddXY*-Methode erläutert werden.

HINWEIS Im Weiteren beschränken wir uns auf die Darstellung des *Chart*s für Windows Forms-Anwendungen. Wie das How-to 9.3 »... mit ASP.NET ein Diagramm anzeigen?» beweist, ist es problemlos möglich, dieses Wissen auch im Rahmen einer ASP.NET-Anwendung zu nutzen, ohne große Änderungen vornehmen zu müssen.

Serien/Reihen und Datenpunkte direkt erzeugen

Von zentraler Bedeutung für die Diagrammdarstellung sind die Begriffe der Serien bzw. Reihen und der **Datenpunkte**.

HINWEIS Eine *Serie* bzw. *Reihe* besteht aus mehreren *Datenpunkten*, von denen jeder einzelne durch einen *X*- und einen *Y-Wert* festgelegt ist. Jeder einzelne Datenpunkt kann beispielsweise durch eine bestimmte Säule eines Balkendiagramms dargestellt werden.

Wenn Sie ein *Chart*-Control von der Toolbox abziehen und auf dem Formular absetzen, verfügt es bereits über eine Standard-Serie, die Sie aber, beispielsweise mittels der *Points.AddXY()*-Methode, noch mit Datenpunkten füllen müssen. Weitere Serien können Sie durch Aufruf der *Series.Add()*-Methode hinzufügen.

BEISPIEL

Ein einfaches Balkendiagramm mit zwei Serien erzeugen

Die Standard-Serie erhält eine neue Bezeichnung:

```
chart1.Series[0].Name = "Umsätze Kühlschränke";
```

Fünf Datenpunkte zuweisen:

```
chart1.Series[0].Points.AddXY(2006, 10);
chart1.Series[0].Points.AddXY(2007, 25);
chart1.Series[0].Points.AddXY(2008, 75);
chart1.Series[0].Points.AddXY(2009, 110);
chart1.Series[0].Points.AddXY(2010, 130);
```

Die zweite Serie wird hinzugefügt:

```
chart1.Series.Add("Umsätze Waschmaschinen");
```

Fünf Datenpunkte:

```
chart1.Series[1].Points.AddXY(2006, 150);
chart1.Series[1].Points.AddXY(2007, 75);
chart1.Series[1].Points.AddXY(2008, 25);
chart1.Series[1].Points.AddXY(2009, 10);
chart1.Series[1].Points.AddXY(2010, 15);
```

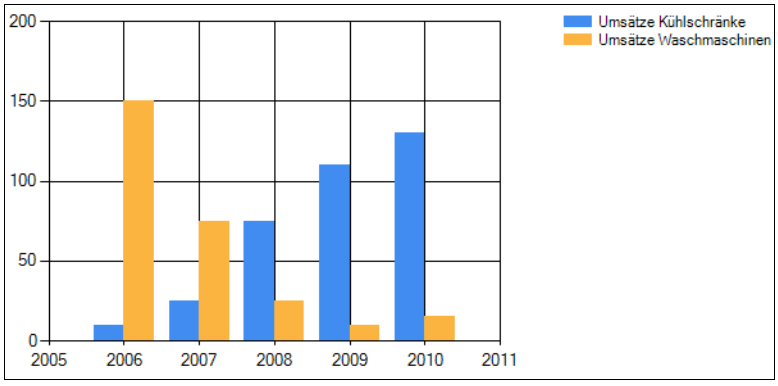


Abbildung 9.1 Das Anzeigeergebnis

Den Diagrammtyp ändern

Standardmäßig wird jede Serie als Balkendiagramm (*ChartType* = *Column*) angezeigt. Die Umstellung auf einen anderen Diagrammtyp ist für jede Serie einzeln vorzunehmen und kann entweder zur Entwurfszeit oder aber per Code erfolgen.

Es gibt insgesamt 35 Diagrammtypen für die unterschiedlichsten Ansprüche, man kann sie zur Entwurfszeit bequem im *Series-Auflistungs-Editor* zuweisen (erreichbar über die *Series*-Eigenschaft).

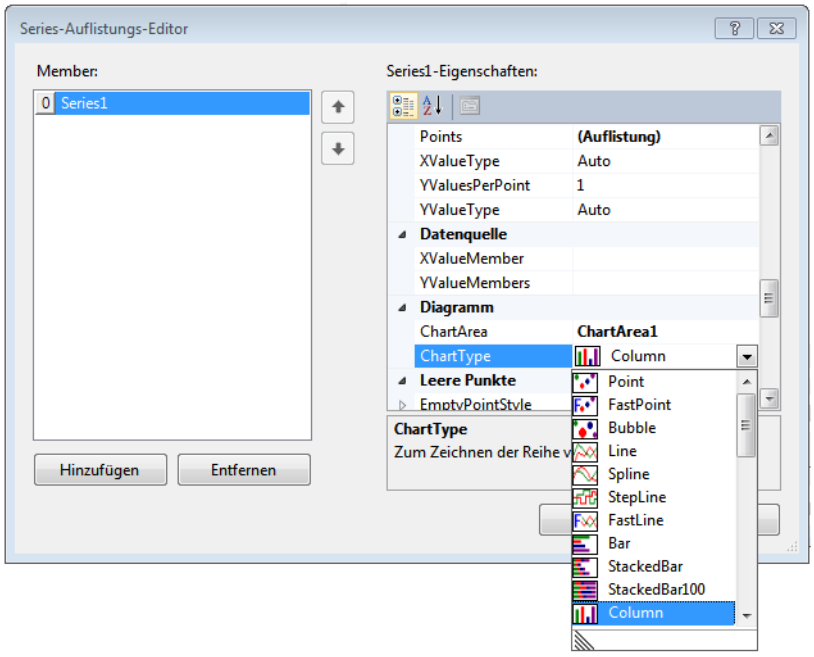


Abbildung 9.2 Auswahl Diagrammtyp

Überschaubarer und vor allem auch zur Laufzeit änderbar ist eine Zuweisung des Diagrammtyps per Code.

BEISPIEL

Das Vorgängerbeispiel soll als Liniendiagramm dargestellt werden. Es wird lediglich der ergänzende Code gezeigt:

```
using System.Windows.Forms.DataVisualization.Charting;
...
```

Zwecks besserer Optik wird die Linie verstärkt:

```
chart1.Series[0].BorderWidth = 5;
```

Das Zuweisen des neuen Diagrammtyps:

```
chart1.Series[0].ChartType = SeriesChartType.Line;
...
```

Dasselbe für die zweite Serie:

```
chart1.Series[1].BorderWidth = 5;
chart1.Series[1].ChartType = SeriesChartType.Line;
```

Das Ergebnis zeigt die Abbildung 9.3.

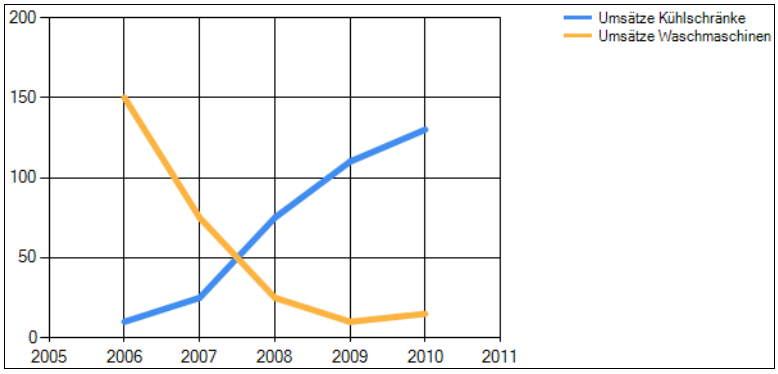


Abbildung 9.3 Ausgabeergebnis

HINWEIS Wie Sie sehen, werden bei einem Liniendiagramm die einzelnen Datenpunkte einer Serie/Reihe miteinander verbunden.

Soll der Diagrammtyp für alle Serien gleichzeitig geändert werden (das ist meistens der Fall), so kann der Code mit einem Schleifenkonstrukt vereinfacht werden (siehe folgendes Beispiel):

BEISPIEL

Die Serien des Vorgängerbeispiels werden als Flächendiagramme dargestellt

```
foreach (Series ser in chart1.Series)
{
    ser.ChartType = SeriesChartType.Area;
}
```

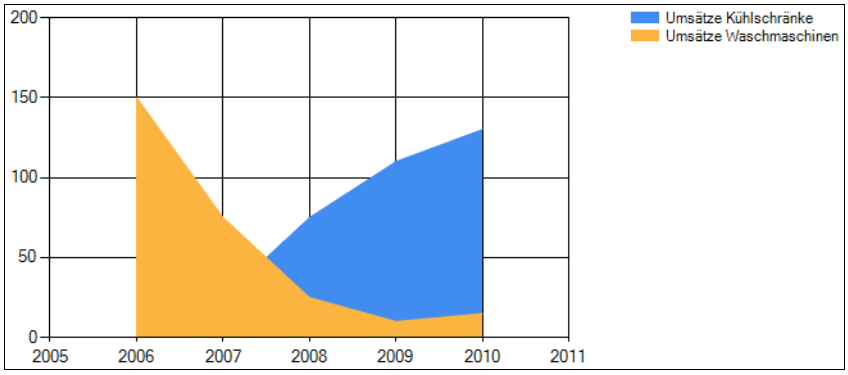


Abbildung 9.4 Laufzeitdarstellung des Diagramms

3D-Darstellung

Alle Diagrammtypen lassen sich in der Regel auch räumlich darstellen. Sie können dazu den *Chart-Area-Auflistungs-Editor* verwenden, mit welchem allgemeine Eigenschaften eines Diagramms (Achsenbeschriftungen etc.) zugewiesen werden können[1].

BEISPIEL

Das Vorgängerbeispiel in 3D-Darstellung

```
chart1.ChartAreas[0].Area3DStyle.Enable3D = true;
```

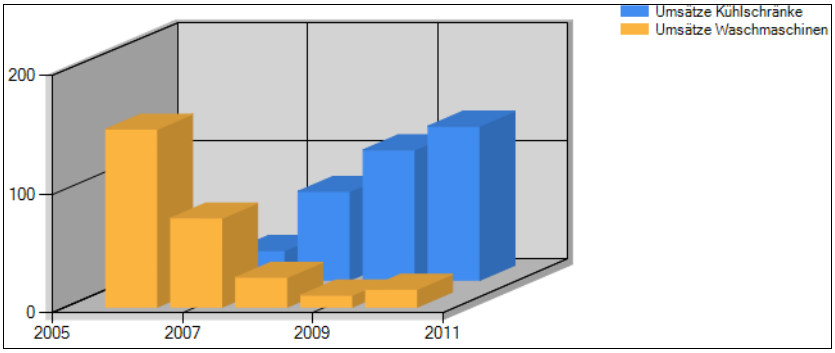


Abbildung 9.5 Ausgabeergebnis

Farben für Serien und Datenpunkte

Das *Chart*-Control bietet verschiedene Wege, um Farben für Serien und Datenpunkte zuzuweisen.

Farbpalette auswählen

Es gibt 12 eingebaute Paletten, von denen jede einzelne etwa 10 unterschiedliche Farben aufweist.

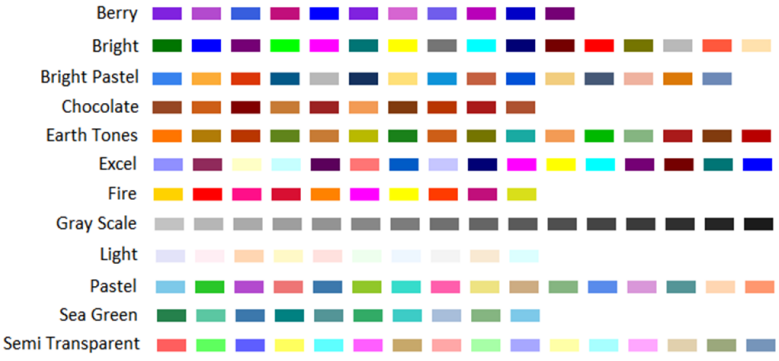


Abbildung 9.6 Übersicht Farbpaletten

[1] In einem *Chart* können auch mehrere Diagramme (*ChartAreas*) gleichzeitig dargestellt werden, z.B. untereinander.

Chart-Typen, wie beispielsweise *Column* und *Line*, weisen jeder Serie automatisch eine bestimmte Farbe aus der gewählten Palette zu. Wenn das Angebot erschöpft ist, wird wieder von vorn begonnen.

BEISPIEL

Zuweisen einer Farbpalette

```
using  System.Windows.Forms.DataVisualization.Charting;
...
chart1.Palette = ChartColorPalette.EarthTones;
```

Ein *Chart*-Typ wie beispielsweise *Pie* (Tortendiagramm) verwendet für jeden einzelnen Datenpunkt eine bestimmte Farbe. Hier können Sie nach wie vor die *Palette*-Eigenschaft verwenden oder aber Sie definieren unterschiedliche Paletten für unterschiedliche Serien mittels der *Series.Palette*-Eigenschaft.

Benutzerdefinierte Paletten

Falls Ihnen keine der Standard-Paletten gefällt, können Sie auch eigene Paletten mit beliebig vielen Farben erzeugen.

BEISPIEL

Zuweisen einer benutzerdefinierten Farbpalette

```
chart1.Palette = ChartColorPalette.None;
chart1.PaletteCustomColors = new Color[] { Color.Red, Color.Blue, Color.Green};
```

Color-Eigenschaft

Sie haben, unabhängig von den bisher beschriebenen Techniken, immer die Möglichkeit, die *Color*-Eigenschaft für eine Serie oder einen einzelnen Datenpunkt direkt zu setzen.

BEISPIEL

Farbe für die gesamte Serie zuweisen

```
chart1.Series[0].Color = Color.Blue;
```

BEISPIEL

Farbe für einen einzelnen Datenpunkt setzen

```
chart1.Series[0].Points[5].Color = Color.Yellow;
```

HINWEIS Um wieder auf die Paletten-Farben umzuschalten müssen Sie die *Color*-Eigenschaften der Serien und Datenpunkte auf *Color.Empty* setzen.

Leere Datenpunkte

Wenn Sie *DBNull*-Werte an die *Chart* binden wollen, werden die Datenpunkte automatisch als »empty points« markiert. Sie können das auch selbst tun, indem Sie Datenpunkte mittels der *DataPoint.IsEmpty*-

Eigenschaft setzen. Jeder leere Datenpunkt verwendet die in der *Series.EmptyPointStyle*-Eigenschaft definierten visuellen Attribute.

BEISPIEL

Alle leeren Datenpunkte einer Serie werden versteckt, indem sie transparent gemacht werden

```
chart1.Series[0].EmptyPointStyle.Color = Color.Transparent;
```

Diagramm drucken

Ein Diagramm auf dem Bildschirm sieht ja recht schön aus, doch in vielen Fällen soll das Ergebnis auch zu Papier gebracht werden. Das ist kein Problem, über die *Printing*-Eigenschaft des *Chart*-Controls werden alle Aktivitäten rund um die Druckausgabe gebündelt.

Folgende Methoden stehen zur Verfügung:

Methode	Beschreibung
PageSetup	... zeigt den bekannten Pagesetup-Dialog an.
Print	... druckt das vorliegende Diagramm. Übergeben Sie als Parameter *true*, wird. der bekannte Druckdialog zur Druckerauswahl angezeigt.
PrintPaint	Ausgabe des Diagramms auf einem *Graphics*-Objekt.
PrintPreview	Statt der direkten Druckauswahl wird eine Druckvorschau angezeigt.

Tabelle 9.1 Methoden zur Druckausgabe

BEISPIEL

Eine einfache Druckvorschau ist mit

```
chart1.Printing.PrintPreview();
```

realisierbar.

Diagramm exportieren/abspeichern

Mit der *SaveImage*-Methode können Sie Ihr Diagramm in insgesamt 8 verschiedenen Bildformaten exportieren bzw. abspeichern: *Bmp*, *Emf*, *EmfDual*, *EmfPlus*, *Gif*, *Jpeg*, *Png*, *Tiff*.

BEISPIEL

Die *Chart* wird als Bitmap-Datei im Anwendungsverzeichnis abgelegt.

```
chart1.SaveImage("Chart1.bmp", ChartImageFormat.Bmp);
```

BEISPIEL

Die *Chart* wird in einem *MemoryStream*-Objekt gespeichert und dann als *Byte*-Array zurück gegeben.

```
private Byte[] GetChart()
```

```
{
    ...
    using (var chartImage = new System.IO.MemoryStream())
    {
        chart1.SaveImage(chartImage, ChartImageFormat.Png);
        return chartImage.GetBuffer();
    }
}
```

Einführung in die Chart-Datenbindung

Wie bereits in der Einführung gezeigt, verfügen Sie mit der *Points.AddXY*-Methode über ein einfaches und universelles Werkzeug, um beliebige Diagramme darzustellen. Damit lassen sich, quasi »in Handarbeit«, auch beliebige Datenbankinhalte an das *Chart* anbinden.

Nachdem wir diese Technik am Beispiel demonstriert haben, wollen wir uns einen Überblick über die verfügbaren speziellen Datenbindungsmethoden verschaffen, mit welchen die Arbeit des Programmierers teilweise drastisch vereinfacht werden kann.

Manuelle Datenbindung mittels Points.AddXY-Methode

Sie haben immer die Möglichkeit, die *Chart* manuell anzubinden, indem Sie über die Datenquelle iterieren und die einzelnen Datenpunkte zu den Serien nach Bedarf selbst hinzufügen.

Das gilt insbesondere natürlich auch für die Auswertung von Datenbankinhalten.

BEISPIEL

Eine Access-Datenbanktabelle *PKW_Verkauf_3* hat die Spalten *Verkäufer* und für die Jahre 2000 ... 2010 jeweils eine weitere Spalte, in welcher die pro Jahr erzielten Verkaufssummen enthalten sind:

Feldname	Felddatentyp	Beschreibung
Verkäufer	Text	Name des Verkäufers
2000	Währung	Verkaufssumme für das Jahr 2000
2001	Währung	
2002	Währung	
2003	Währung	
2004	Währung	
2005	Währung	
2006	Währung	

Abbildung 9.7 Tabellenlayout

Gewünscht wird ein Diagramm, welches je nach Verkäufer die Abhängigkeit der Verkaufssumme von der Jahreszahl zeigt.

Zunächst lesen wir auf gewohnte Weise die gewünschten Daten in eine *DataTable* ein:

```
string sql = "SELECT * FROM PKW_Verkauf_3";

OleDbConnection conn = new OleDbConnection(connStr);
OleDbCommand cmd = new OleDbCommand(sql, conn);
OleDbDataAdapter da = new OleDbDataAdapter(cmd);
```

```
DataSet ds = new DataSet();
conn.Open();
da.Fill(ds,"query1");
conn.Close();

DataTable dt = ds.Tables["query1"];
```

Alle Verkäufer durchlaufen:

```
foreach(DataRow row in dt.Rows)
{
```

Pro Verkäufer eine neue Serie hinzufügen:

```
    string serName = row["Verkäufer"].ToString();
    chart1.Series.Add(serName);
```

Jeder Verkäufer = eine Serie von Punkten = eine Linie:

```
    chart1.Series[serName].ChartType = SeriesChartType.Line;
    chart1.Series[serName].BorderWidth = 5;
```

Alle Jahres-Spalten durchlaufen:

```
        for(int colNr = 1; colNr < dt.Columns.Count; colNr++)
        {
```

Pro Jahres-Spalte den Y-Wert als Punkt hinzufügen:

```
            decimal YVal = (decimal) row[colNr];
```

Für X-Achse (Beschriftung!):

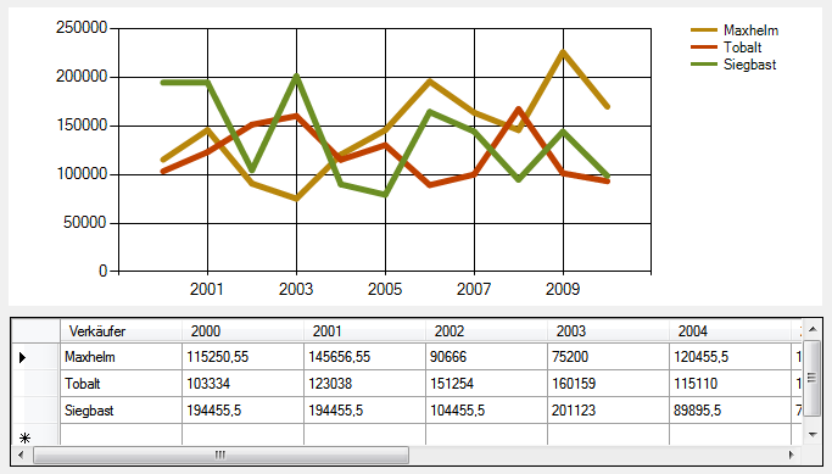
```
            string colName = dt.Columns[colNr].ColumnName;

            chart1.Series[serName].Points.AddXY(colName, YVal);
        }
    }
```

Kontrollanzeige der *DataTable* im Datengitter:

```
   dataGridView1.DataSource = ds;
   dataGridView1.DataMember = "query1";
   ...
```

Das Ergebnis zeigt die Abbildung 9.8.

HINWEIS Die Verwendung der Methode *Points.AddXY* ist auch dann zu empfehlen, wenn die Daten der Serien mittels Formeln berechnet wurden.



Verkäufer	2000	2001	2002	2003	2004
Maxhelm	115250,55	145656,55	90666	75200	120455,5
Tobalt	103334	123038	151254	160159	115110
Siegbast	194455,5	194455,5	104455,5	201123	89895,5

Abbildung 9.8 Ausgabe des Beispiels

Übersicht über die speziellen Datenbindungsmethoden

Wie Sie obigem Beispiel entnehmen können, ist der Programmieraufwand bei manueller Datenbindung (Methode *Points.AddXY*) doch ziemlich erheblich, denn Sie müssen Ihre Serien Punkt für Punkt selbst zusammenbauen. Damit Sie sich diese mühevolle Handarbeit sparen können, verfügt das *Chart*-Control über eine Reihe spezieller Datenbindungsmethoden, die das Erzeugen von Serien und Datenpunkten für unterschiedliche Datenquellen mehr oder weniger automatisieren.

Die folgende Tabelle gibt einen Überblick über die speziellen Datenbindungsmethoden des *Chart*-Controls:

Bindungsoption	Vorteile	Nachteile
DataBindTable-Methode	Einfaches Binden für X und Y Werte. Automatisches Erzeugen von Serien, basierend auf der Spaltenanzahl der Datenquelle.	Keine mehrfachen Y-Werte pro Serie. Alle Serien haben denselben X Wert, oder der ist nicht gesetzt. Keine Bindung für erweiterte *Chart*-Eigenschaften wie z.B. Tooltips.
DataBind-Methode und *DataSource*-Eigenschaft	Können bereits zur Entwurfszeit verwendet werden. Unterstützen mehrfache Y-Werte.	Keine Bindung für erweiterte *Chart*-Eigenschaften, wie z.B. Tooltips.
Points.DataBind(X)Y	Unterstützt multiple Datenquellen, inklusive separate Datenquellen für X und Y Werte. Unterstützt multiple Y Werte. Ist flexibler als obige Methoden.	Keine Bindung für erweiterte *Chart*-Eigenschaften, wie z.B. Tooltips.
Points.DataBind	Wie oben plus Bindungen für erweiterte *Chart*-Eigenschaften wie z.B. Tooltips.	Unterstützt keine unterschiedlichen Datenquellen für X- und Y-Werte von Serien.
DataBindCrossTab	Für jeden eindeutigen Wert in spezifizierten Spalten werden Serien automatisch erzeugt um die Daten zu gruppieren.	Nur Gruppieren auf einem Level (Single Level Grouping) wird unterstützt.

Tabelle 9.2 Datenbindungsmethoden

Unterstützte Datenquellen

Von den *Chart*-Bindungsmethoden wird eine Vielfalt von Datenquellen unterstützt:

- *OleDbDataReader/SqlDataReader*
- *OleDbCommand/SqlCommand*
- *OleDbDataAdapter/SqlDataAdapter*
- *DataView*
- *DataSet/DataTable*
- *List/Array*
- alle Objekte die *IEnumerable* implementieren

HINWEIS Nicht alle Bindungsmethoden unterstützen alle Datenquellen-Typen. So eignen sich *DataSet/DataTable*, *SqlCommand/OleDbCommand* und *SqlDataAdapter/OleDbDataAdapter* nur für Datenbindung per *DataSource*-Eigenschaft!

Spezielle Chart-Datenbindungsmethoden

In diesem Abschnitt wollen wir jede der *Chart*-Bindungsmethoden näher erläutern und am konkreten Beispiel demonstrieren.

Die DataBindTable-Methode

Diese Methode ermöglicht eine verblüffend einfache Datenbindungstechnik. Es kommt allerdings nur eine einzige Datenquelle infrage. Die Serien werden automatisch erzeugt, basierend auf der Anzahl von Spalten in der Datenquelle (eine Serie pro Datenspalte). Jeder Spalteneintrag erzeugt einen Datenpunkt in der entsprechenden Serie und wird für den ersten Y Wert des Punkts verwendet. Um eine Spalte für die X-Werte aller Serien zu spezifizieren verwendet man die überladene Methodendefinition, die einen *xField*-Parameter enthält.

HINWEIS Diese Methode durchfährt die Datenquelle nur einmal um alle Daten zu binden.

Syntax:

```
public void DataBindTable (IEnumerable dataSource, string xField)
```

Zu den Parametern:

dataSource: Datenquelle - ein beliebiges Objekt, welches *IEnumerable* implementiert

xField: Name des Felds für die X-Werte der Serien

HINWEIS Jede Tabellenspalte wird zu einem Y-Wert einer Serie. Außerdem kann das Feld für den X-Wert bereitgestellt werden.

BEISPIEL

Die Verkäufer Maxhelm, Tobalt und Siegbast arbeiten in einem Autohaus. Die folgende Tabelle *PKW_Verkauf* mit den Spalten *Verkäufer*, *Mercedes*, *BMW*, *Opel*, *Ford*, *Mazda* und *Toyota* gibt an, wie viele Autos eines bestimmten Typs von einem bestimmten Verkäufer verkauft worden sind:

Verkäufer	Mercedes	BMW	Opel	Ford	Mazda	Toyota
Maxhelm	12	9	6	10	4	5
Tobalt	8	2	7	5	3	9
Siegbast	5	5	8	11	5	7

Abbildung 9.9 Tabellenlayout

Nachdem Sie eine *Chart*-Komponente auf dem Formular abgesetzt haben, sind zunächst die ADO.NET-typischen Vorbereitungen (Erzeugen von *Connection*- und *Command*-Objekten, Öffnen der *Connection*, Erzeugen des *DataReader*) vorzunehmen:

```
using System.Data.OleDb;
...
string connStr = "PROVIDER=Microsoft.Jet.OLEDB.4.0;Data Source=ChartTest.mdb";
string sql = "SELECT * FROM PKW_Verkauf";
OleDbConnection conn = new OleDbConnection(connStr);
OleDbCommand cmd = new OleDbCommand(sql, conn);
cmd.Connection.Open();
OleDbDataReader dr = cmd.ExecuteReader(CommandBehavior.CloseConnection);
```

Erst jetzt kommen wir zum interessanten Teil, dem Anbinden des *Chart*-Controls: Da ein *DataReader* die Schnittstelle *IEnumerable* implementiert, kann er, zusammen mit dem Namen der Spalte *Verkäufer* die als X-Wert dient, direkt an die *DataBindTable*-Methode übergeben werden:

```
chart1.DataBindTable(dr, "Verkäufer");
dr.Close();
conn.Close();
```

Das Ergebnis als standardmäßiges Balkendiagramm:

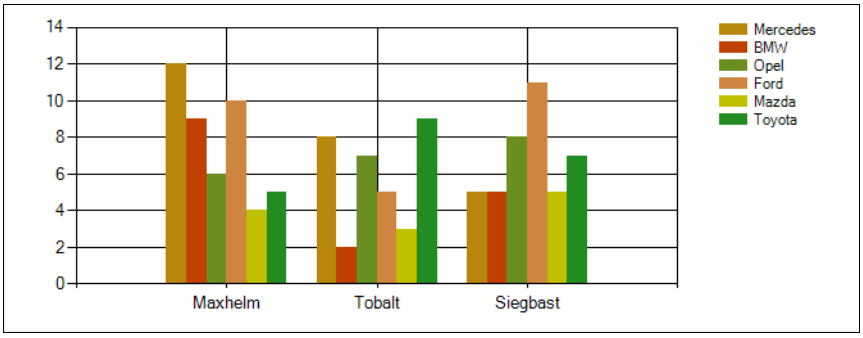


Abbildung 9.10 Das Diagramm zur Laufzeit

Wie Sie sehen, zeigt die Spalte *Verkäufer* die Werte auf der (horizontalen) X-Achse. Die übrigen sechs Spalten (PKW-Typen) bilden die Serien, die auf der (vertikalen) Y-Achse dargestellt werden. Jeder der sechs PKW-Typen (bzw. Serien) hat eine bestimmte Farbe, die Zuordnung ist auf den Legenden (rechts oben) ersichtlich.

BEISPIEL

Anstatt eines *DataReader* wollen wir diesmal eine generische Liste als Objektdatenquelle verwenden, welche wir äquivalent zur Datenbanktabelle *PKW_Verkauf* des Vorgängerbeispiels erstellen.

Die Elemente der Liste sind Objekte des Typs *CVerkäufer*:

```
public class CVerkäufer
{
     public string Name { get; set; }
     public int Mercedes { get; set; }
     ...
}
```

Die generische Liste (per Typinferenz) erzeugen:

```
var verkäufe = new List<CVerkäufer>();
```

Liste füllen:

```
verkäufe.Add(new CVerkäufer{Name = "Maxhelm", Mercedes = 12, BMW = 9, Opel = 6, Ford = 10,
                                                                    Mazda = 4, Toyota = 5});
verkäufe.Add(new CVerkäufer { Name = "Tobalt", Mercedes = 8, BMW = 8, Opel = 7, Ford = 5,
                                                                   Mazda = 3, Toyota = 9 });
...
```

Anbinden des *Chart*-Controls:

```
chart1.DataBindTable(verkäufe,"Name");
```

Das Ergebnis ist äquivalent zum Vorgängerbeispiel. Damit es uns aber nicht zu langweilig wird, wollen wir diesmal anstatt des standardmäßigen Balkendiagramms einen der zahlreichen anderen Diagrammtypen zeigen, wie wäre es mit *Bubble*? Hierfür ist allerdings etwas zusätzlicher Programmieraufwand erforderlich, da der neue Diagrammtyp jeder einzelnen Serie extra zugewiesen werden muss:

```
foreach (Series ser in chart1.Series)
{
    ser.ChartType = SeriesChartType.Bubble;
}
```

Überflüssige Legende für Standardserie ausblenden:

```
chart1.Series[0].IsVisibleInLegend = false;
```

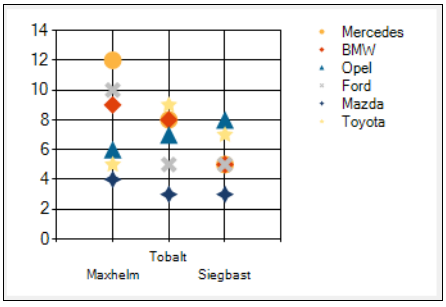


Abbildung 9.11 Ausgabeergebnis

DataBind-Methode/DataSource-Eigenschaft

Wollen Sie die *DataBind*-Methode verwenden, so müssen Sie noch diverse andere Eigenschaften des *Chart*-Controls setzen. Am wichtigsten ist die *DataSource*-Eigenschaft, sie bietet die einzige Möglichkeit, das *Chart* bereits zur Entwurfszeit anzubinden, ihr können alle oben aufgeführten Datenquellen direkt zugewiesen werden. Im Unterschied zur *DataBindTable*-Methode müssen Sie die Serien selbst erzeugen und deren *YValueMembers*- und optional *XValueMember*-Eigenschaften festlegen.

In den folgenden Beispielen verzichten wir im Interesse der Transparenz auf mögliche Vorteile der Entwurfszeit-Datenbindung mittels der verschiedenen über das Eigenschaftenfenster erreichbaren Auflistungs-Editoren und nehmen eine komplette Laufzeit-Anbindung vor. Die Ergebnisse jedes der Beispiele sind identisch und entsprechen den Vorgängern, weshalb wir auf Abbildungen verzichten können.

BEISPIEL

Eine alternative Realisierung des Vorgängerbeispiels mit einem *DataReader* als Datenquelle (der Vorbereitungscode, bis einschließlich dem Öffnen der *Connection*, ist identisch).

Der Datenquelle wird direkt der *DataReader* zugewiesen:

```
chart1.DataSource = cmd.ExecuteReader(CommandBehavior.CloseConnection);
```

Im Unterschied zum Vorgängerbeispiel müssen wir uns nun um die sechs Serien selbst kümmern.

Die erste Serie ist zwar bereits vorhanden, allerdings müssen wir noch die Namen der entsprechenden X- und Y-Werte setzen:

```
chart1.Series["Series1"].XValueMember = "Verkäufer";
chart1.Series["Series1"].YValueMembers = "Mercedes";
```

Auch um die Anzeige/Beschriftung der Legende müssen wir uns selbst kümmern:

```
chart1.Series["Series1"].IsVisibleInLegend = true;
chart1.Series["Series1"].LegendText = "Mercedes";
```

Dasselbe ist für die restlichen Serien zu tun, diese müssen zunächst mittels *Add()*-Methode zur *Series*-Collection hinzugefügt werden:

```
chart1.Series.Add("Series2");
chart1.Series["Series2"].XValueMember = "Verkäufer";
chart1.Series["Series2"].YValueMembers = "BMW";
chart1.Series["Series2"].LegendText = "BMW";
...
```

Den analogen Code für die übrigen PKW-Typen (Opel, Ford, Mazda, Toyota) sparen wir uns (siehe Begleitdaten).

Nun erst kann das *Chart* an die Datenquelle gebunden werden:

```
chart1.DataBind();
conn.Close();
}
```

Das Resultat ist identisch zum Vorgängerbeispiel, der Codeaufwand ist allerdings größer.

BEISPIEL

Das im Vorgängerbeispiel verwendete *Command*-Objekt wird direkt als Datenquelle eingesetzt.

```
...
OleDbConnection conn = new OleDbConnection(connStr);
OleDbCommand cmd = new OleDbCommand(sql, conn);
chart1.DataSource = cmd;
...
chart1.DataBind()
...
```

BEISPIEL

Ein *DataAdapter*-Objekt als Datenquelle

```
...
OleDbDataAdapter da = new OleDbDataAdapter(cmd);
chart1.DataSource = da;
...
chart1.DataBind()
...
```

BEISPIEL

Auch der etwas umständlichere Weg über eine *DataTable* ist möglich.

```
...
OleDbDataAdapter da = new OleDbDataAdapter(cmd);
DataTable dt = new DataTable();
da.Fill(dt);
chart1.DataSource = dt;
...
chart1.DataBind()
...
```

HINWEIS Das *Chart* bindet sich noch vor dem Rendering automatisch an die spezifizierte Datenquelle. Sie können aber durch Aufruf der *DataBind()* Methode erzwingen, dass sich das *Chart* in einem beliebigen Moment anbindet.

Die DataBindCrossTable-Methode

Diese Methode unterscheidet sich von allen anderen Bindungsmethoden darin, dass sie das Gruppieren eindeutiger Werte in einer Spalte erlaubt. Jeder eindeutige Wert in der spezifizierten gruppierten Spalte führt zum Erzeugen von Datenserien. Weiterhin können erweiterte Datenpunkt-Eigenschaften (also nicht nur X- und Y Werte) gebunden werden.

Die Syntax:

```
public void DataBindCrossTable(
 IEnumerable dataSource,
 string seriesGroupByField,
 string xField,
 string yFields,
```

```
 string otherFields,
 PointSortOrder sortingOrder
)
```

Parameter	Bedeutung
dataSource	Die Datenquelle.
seriesGroupByField	Der Name des Felds zum Gruppieren von Daten in die Serien.
xField	Der Name des Felds für die x-Werte.
yFields	Eine durch Trennzeichen separierte Liste von Namen der Felder für y-Werte.
otherFields	Weitere Datenpunkteigenschaften *(AxisLabel, Tooltip, Label, LegendText, LegendTooltip* und *CustomPropertyName*).
sortingOrder	Serien werden nach Gruppenfeldwerten in der angegebenen Reihenfolge sortiert.

Tabelle 9.3 Parameter der *DataBindCrossTable*-Methode

BEISPIEL

Gegeben sei die folgende Datenbanktabelle, in welcher die jährlichen Verkaufssummen und Provisionen unserer drei glorreichen Autoverkäufer enthalten sind:

Name	Jahr	Verkaufssumme	Provision
Maxhelm	2007	145.656,55 €	20.699,33 €
Tobalt	2007	103.334,00 €	22.299,00 €
Siegbast	2007	194.455,50 €	33.636,00 €
Maxhelm	2008	189.783,00 €	24.355,00 €
Tobalt	2008	81.999,00 €	12.487,00 €
Siegbast	2008	156.449,00 €	19.794,00 €
Maxhelm	2009	162.994,00 €	23.593,00 €
Tobalt	2009	124.993,00 €	22.599,00 €
Siegbast	2009	178.993,00 €	25.852,00 €
Maxhelm	2010	120.567,00 €	8.234,00 €
Tobalt	2010	56.789,00 €	5.239,00 €
Siegbast	2010	151.421,00 €	17.345,00 €

Abbildung 9.12 Ausgangstabelle

```
...
 OleDbConnection conn = new OleDbConnection(connStr);
 OleDbCommand cmd = new OleDbCommand("SELECT * FROM PKW_Verkauf_2", conn);
 cmd.Connection.Open();
 OleDbDataReader dr = cmd.ExecuteReader(CommandBehavior.CloseConnection);
```

Wir wollen nach der »Name« Spalte gruppieren, d.h., pro Verkäufer wird eine Datenserie erzeugt. Die X-Werte werden an die »Jahr«-Spalte gebunden, die Y-Werte an die Spalte »Verkaufssumme«. und die *Label*-Eigenschaft der resultierenden Datenpunkte (einer pro Datensatz) an die Spalte »Provision«.

```
 chart1.DataBindCrossTable(
                dr,
                "Name",
                "Jahr",
                "Verkaufssumme",
                "Label=Provision{C}");
dr.Close();
conn.Close();
```

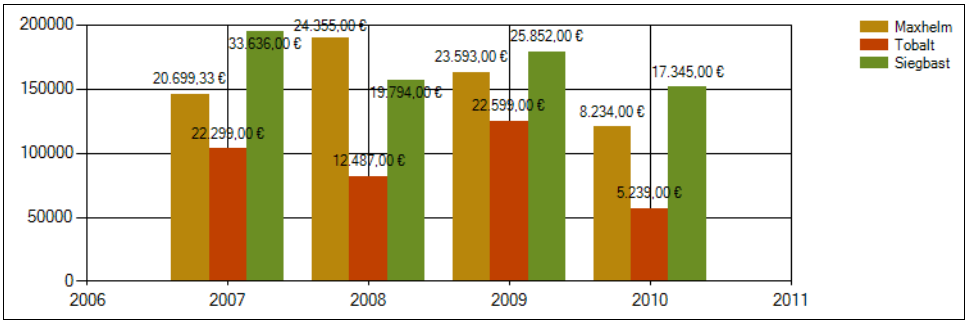


Abbildung 9.13 Die Standarddarstellung als Balkendiagramm:

Eine von vielen weiteren Möglichkeiten – die alternative Darstellung als Liniendiagramm:

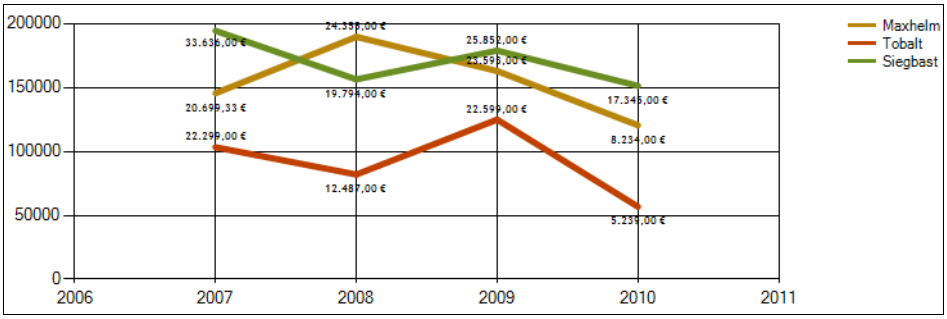


Abbildung 9.14 Alternative Darstellung

Wie Sie sehen, werden die Serien automatisch zum *Chart* hinzugefügt, abhängig von der Anzahl eindeutiger Werte in der durch den Parameter *seriesGroupByField* spezifizierten Spalte der Datenquelle.

Als Datenquelle kommen *DataReader*, *DataView*, *DataSet* oder *DataRow* infrage, sowie alle Auflistungen, die *IEnumerable* implementieren

HINWEIS Das *DataTable*-Objekt können Sie mit der *DataCrossTable*-Methode nicht verwenden, da es kein *IEnumerable*-Interface besitzt. Benutzen Sie stattdessen einen *DataView*.

Die Points.DataBind-Methode

Die *Points.DataBind*-Methode fügt Punkte zu einer spezifischen Serie hinzu, sie ermöglicht es, neben den X- und Y-Werten auch andere Properties an die Datenspalten zu binden. Zu diesen Properties gehören *Label*, *AxisLabel*, *Tooltip*, *LegendText*, *LegendTooltip* und *CustomPropertyName*, wie wir sie bereits als *otherFields*-Parameter aus der *DataBindCrossTable*-Methode kennen (diese Methode splittet *automatisch* Daten in Serien auf, basierend auf einem bestimmten Gruppierungsfeld).

BEISPIEL

Aus der Datenbanktabelle (*PKW_Verkauf_2*) des Vorgängerbeispiels werden nur die Verkäufe für das Jahr 2010 abgefragt. Die X- und Y-Werte der Serie werden den Spalten »Name« bzw. »Verkaufssumme« zugeordnet. Die Properties *Tooltip* und *Label* der resultierenden Datenpunkte werden mit den Spalten »Jahr« und »Provision« verbunden.

```
...
OleDbCommand cmd = new OleDbCommand("SELECT * FROM PKW_Verkauf_2 WHERE Jahr=2010", conn);
cmd.Connection.Open();
OleDbDataReader dr = cmd.ExecuteReader(CommandBehavior.CloseConnection);
chart1.Series["Series1"].Points.DataBind(
                              dr,
                              "Name",
                              "Verkaufssumme",
                              "Tooltip=Jahr,
                               Label=Provision{C}");
...
```

Was Sie jetzt sehen ist gewissermaßen ein Ausschnitt aus dem Diagramm des Vorgängerbeispiels (es wird nur eine einzige Serie angezeigt). Das Zuweisen der *Tooltip*-Eigenschaft bewirkt, dass beim Verweilen des Mauszeigers auf einem Balken das Jahr *2010* als Quickinfo angezeigt wird.

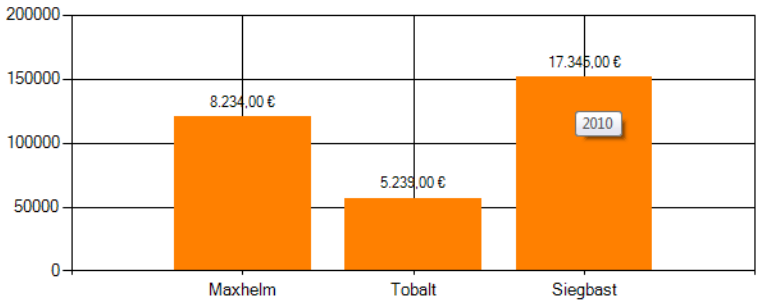


Abbildung 9.15 Ausgabeergebnis

Die Points.DataBind(X)Y-Methoden

Die *Points.DataBind*-Methode erlaubt es nicht, verschiedene Datenquellen für die X- und Y-Werte einer Serie festzulegen, eine Lösung bieten die *DataBind(X)Y*-Methoden.

Points.DataBindXY

Die *Points.DataBindXY*-Methode ermöglicht das Binden von X- und von Y-Werten einer Serie. Dabei können auch unterschiedliche Datenquellen verwendet werden.

BEISPIEL

zwei *DataReader*

```
...
OleDbCommand cmd = new OleDbCommand(
                        "SELECT Name, Provision FROM PKW_Verkauf_2 WHERE Jahr = 2010", conn);
cmd.Connection.Open();
```

```
OleDbDataReader dr = cmd.ExecuteReader(CommandBehavior.CloseConnection);

chart1.Series[0].Points.DataBindXY(
                        dr,
                        "Name",
                        dr,
                        "Provision");
...
```

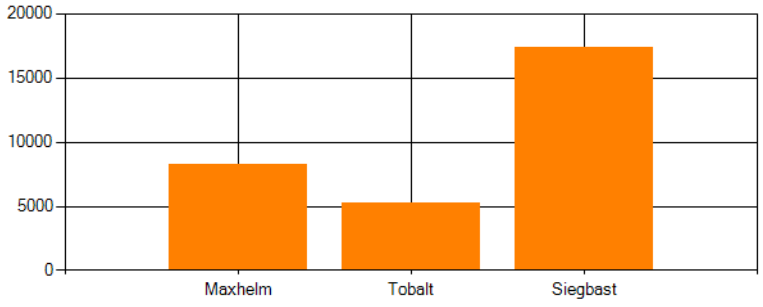


Abbildung 9.16 Ausgabeergebnis

BEISPIEL

Ein einfaches Diagramm

Array für X-Werte initialisieren:

```
string[] xval = { "Maxhelm", "Siegbast", "Tobalt", "Friedhelm", "Susanne" };
```

Array für Y-Werte initialisieren:

```
double[] yval = { 1, 6, 4, 5.35, 8 };
```

Anbinden des Y-Arrays an die Punkte der Y-Achse der Standard-Datenserie:

```
chart1.Series[0].Points.DataBindXY(
                    xval,
                    yval);
```

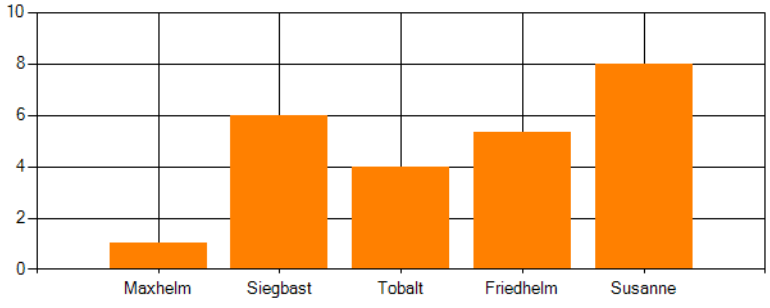


Abbildung 9.17 Ausgabeergebnis

Points.DataBindY

Im Unterschied zur *DataBindXY*-Methode bindet *DataBindY* nur Y-Werte.

BEISPIEL

Die Spalte *Verkaufssumme* der Datenbanktabelle *PKW_Verkauf_2* hat 12 Datensätze, sie wird mit den Y-Werten der Standard-Punkteserie verbunden. Die X-Achse ist mit den laufenden Datensatznummern beschriftet.

```
OleDbCommand cmd = new OleDbCommand("SELECT * FROM PKW_Verkauf_2", conn);
cmd.Connection.Open();
OleDbDataReader dr = cmd.ExecuteReader(CommandBehavior.CloseConnection);

chart1.Series["Series1"].Points.DataBindY(
                                dr,
                                "Verkaufssumme");
```

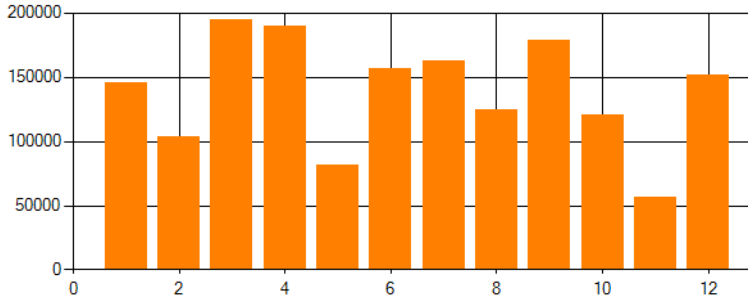


Abbildung 9.18 Ausgabeergebnis

BEISPIEL

Einfacher geht es wohl kaum noch: eine Punkteserie an ein *Double*-Array binden

Array initialisieren:

```
double[] dArr = { 2, 4, 6, 8, 3, 5, 7, 9, 19 };
```

Array an die Punkte der Y-Achse der Datenserie binden:

```
chart1.Series["Series1"].Points.DataBindY(dArr);
```

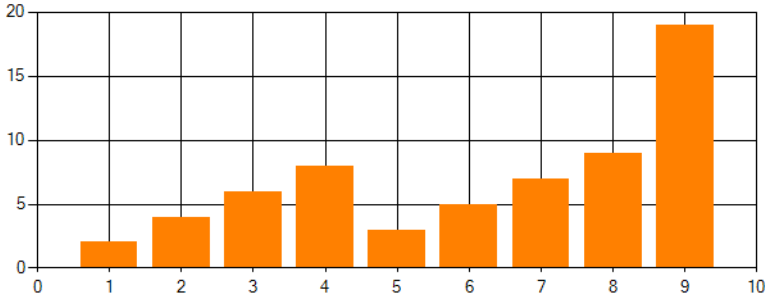


Abbildung 9.19 Ausgabeergebnis

How-to-Beispiele

9.1 ... das Chart-Control zur Laufzeit mit Daten füllen?

Chart-Control: *DataSource*-Eigenschaft, *DataBind*-Methode; *DataSetBindingSource*-Objekt: *ListChanged*-Ereignis:

An einem einfachen Beispiel wollen wir Ihnen den interaktiven Einsatz des *Chart*-Controls demonstrieren, d.h., Sie können zur Laufzeit Einträge hinzufügen und ändern. Um mit möglichst wenig Quellcode auszukommen, nutzen wir ein *DataSet* als Datenspeicher, an das wir ein *DataGridView* zur Eingabe und ein *Chart*-Control zur Ausgabe anbinden.

Dataset entwerfen

Nachdem Sie eine neue Windows Forms-Anwendung erstellt haben, fügen Sie dem Projekt zunächst ein neues *DataSet* hinzu (*Projekt/Neues Element hinzufügen*). Erstellen Sie eine Tabelle »Umsatzentwicklung« mit folgendem Aufbau:

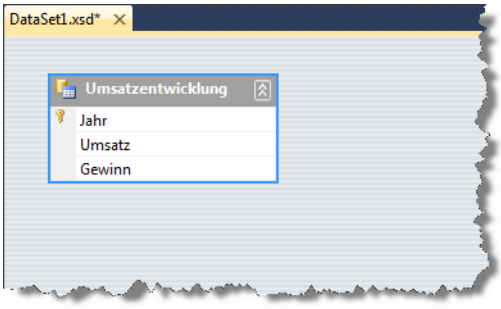


Abbildung 9.20 Das DataSet-Layout

Speichern Sie das *DataSet* ab.

Oberfläche

Fügen Sie *Form1* zunächst einen *SplitContainer* hinzu, dessen *Orientation*-Eigenschaft Sie auf *Horizontal* setzen. Um das Control im Formular auszurichten, legen Sie *Dock* auf *Fill* fest. In den oberen Teil des *SplitContainers* fügen Sie ein *DataGridView*, in den unteren Teil ein *Chart*-Control ein.

Über das Aufgaben-Menü des *DataGridView* weisen Sie unser *DataSet* als Datenquelle zu, es werden automatisch ein *DataSet* und eine *BindungSource* in das Fenster eingefügt. Legen Sie für die *BindingSource* die *DataMember*-Eigenschaft auf »Umsatzentwicklung« fest. Damit ist das *DataGridView* mit dem *DataSet* verbunden.

Nachfolgend können wir uns dem *Chart*-Control zuwenden. Legen Sie zunächst *DataSource* auf die bereits vorhandene *BindingSource*-Komponente fest. Nun lassen sich auch die einzelnen Datenreihen an die Tabellenspalten binden. Öffnen Sie dazu den *Series*-Editor und legen Sie sowohl den *XValueMember* als auch den *YValueMember* entsprechend Ihren Wünschen fest:

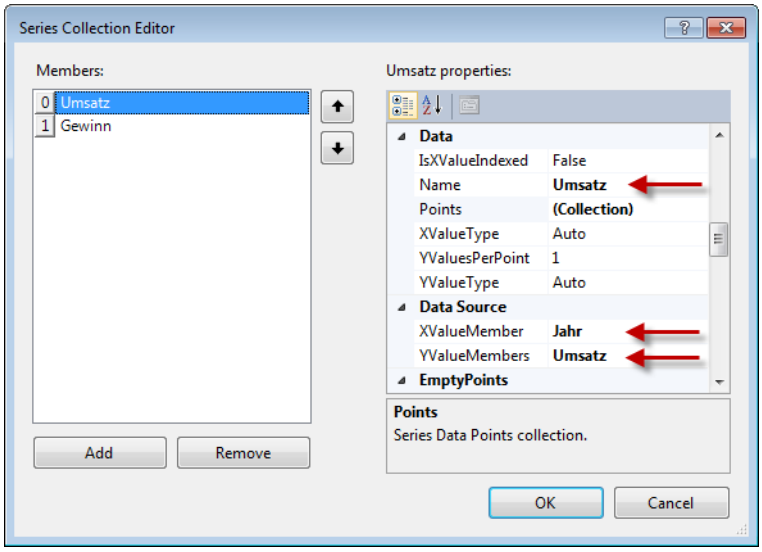


Abbildung 9.21 *XValueMember* und *YValueMember* festlegen

Die Beschriftung der Datenreihe wird über die *Name*-Eigenschaft bestimmt.

Da wir zwei Datenreihen anzeigen wollen, müssen wir im obigen Editor noch eine weitere Datenreihe einfügen und entsprechend konfigurieren.

Quelltext

Leider aktualisiert das *Chart*-Control nicht automatisch die Anzeige und so bleibt uns nichts anderes übrig, als eine »umfangreiche« Ereignisbehandlung für die *BindingSource* zu realisieren:

```
        private void dataSet1BindingSource_ListChanged(object sender, ListChangedEventArgs e)
        {
            chart1.DataBind();
        }
```

Das obige Ereignis wird bei jeder Datenänderung im *DataSet* ausgelöst.

Test

Zur Laufzeit können Sie Werte in das Datengitter eingeben und die Reaktionen des Diagramms unmittelbar beobachten (Abbildung 9.22).

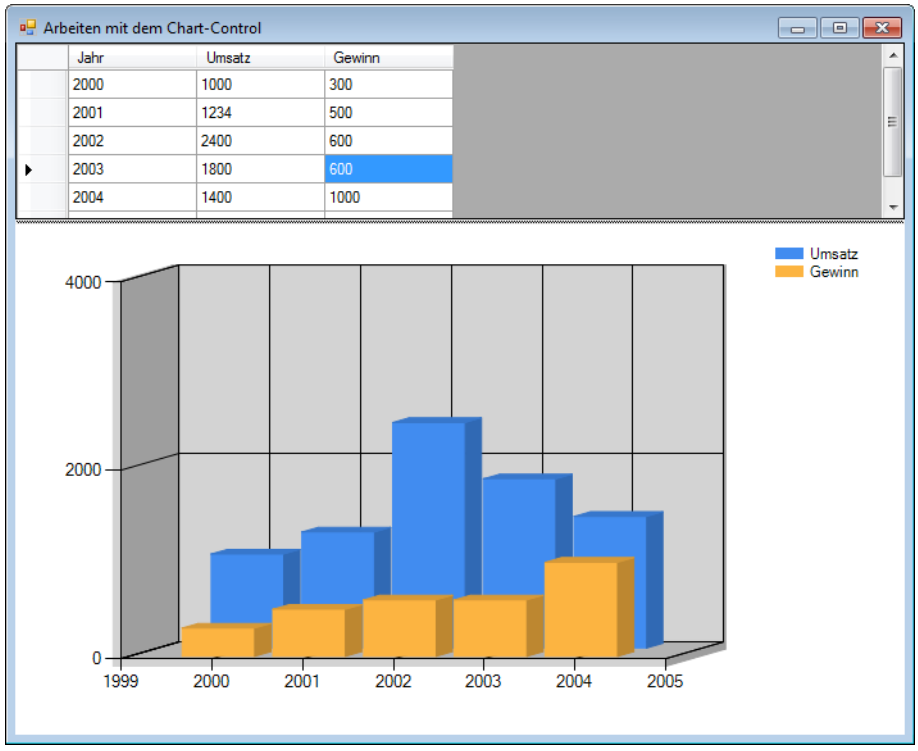


Abbildung 9.22 Laufzeitansicht

9.2 ... das Chart mit einer LINQ to SQL-Abfrage verbinden?

Chart-Control: *Titles*-Collection, *SaveImage*-Methode; *ChartAreas*-Collection: *AxisX*-, *AxisY*-, *BackColor*-, *LabelStyle.Font*-, *LabelStyle.Angle*-Eigenschaften; *Series*-Collection: *IsVisibleInLegend*-Eigenschaft;

Mit dem *Chart* wollen wir das Ergebnis einer verknüpften Abfrage der Tabellen *Employees* und *Orders* der *Northwind*-Datenbank des SQL-Servers grafisch auswerten. Die Verbindung zwischen Datenbank und *Chart* wird unter Verwendung von LINQ to SQL (siehe Kapitel 17) realisiert.

Oberfläche

Öffnen Sie eine neue Windows Forms-Anwendung.

Basis für den Einsatz von LINQ to SQL ist immer ein so genannter Datenkontext (*DataContext*). Da dessen Erzeugung bereit im Einführungsbeispiel 1.3 »... eine einfache LINQ to SQL-Anwendung schreiben?« ausführlich beschrieben wurde, können wir uns hier kurz fassen:

- Ziehen Sie die Datenbankdatei *Northwind.mdf* auf den Projektmappen-Explorer (Assistent abbrechen!)
- Fügen Sie eine neue *LINQ to SQL*-Klasse hinzu (Menü *Projekt/Neues Element hinzufügen ...*)
- Öffnen Sie den Server Explorer (Menü *Ansicht/Server-Explorer*) und ziehen Sie die Tabellen *Employees* und *Orders* auf die Oberfläche des LINQ to SQL-Designers
- Im Eigenschaftendialog des Designers ändern Sie den Namen des Datenkontexts in *NWDataContext*

Wechseln Sie nun in die Entwurfsansicht von *Form1* und setzen Sie ein *Chart*-Control und einen *Button* auf das Formular.

Quelltext

```
...
using System.Windows.Forms.DataVisualization.Charting;


namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        ...
        private void button1_Click(object sender, EventArgs e)
        {
```

Unser Datenkontext:

```
            var db = new NWDataContext();
```

Die folgende LINQ Abfrage liefert einen anonymen Typ welcher die Angestellten (*Employee*-Objekte) enthält, zusammen mit der Anzahl der von ihnen generierten Bestellungen (*Orders*).

```
            var query = from o in db.Orders
                        group o by o.Employees
                            into g
                            select new
                            {
                                Employee = g.Key,
                                NoOfOrders = g.Count()
                            };
```

Einige Properties für das Rendering der *Chart* werden gesetzt, wir beginnen mit der Überschrift:

```
            chart1.Titles.Add("Sales By Employees");
            chart1.Titles[0].Font = new Font("Arial", 16f);
```

Die Bezeichnungen für X- und Y-Achse:

```
            chart1.ChartAreas[0].AxisX.Title = "Employee";
            chart1.ChartAreas[0].AxisY.Title = "Sales";

            chart1.ChartAreas[0].AxisX.TitleFont = new Font("Arial", 12f);
            chart1.ChartAreas[0].AxisY.TitleFont = new Font("Arial", 12f);

            chart1.ChartAreas[0].AxisX.LabelStyle.Font = new Font("Arial", 10f);
```

Der Schriftwinkel für die Beschriftung der X-Achse soll um 45 Grad geneigt sein:

```
            chart1.ChartAreas[0].AxisX.LabelStyle.Angle = -45;
```

Die Hintergrundfarbe:

```
            chart1.ChartAreas[0].BackColor = Color.LightYellow;
```

Die Legende ist hier überflüssig, da nur eine Serie vorhanden ist:

```
            chart1.Series[0].IsVisibleInLegend = false;
```

Die resultierenden Daten der LINQ Abfrage werden mittels *AddXY()*-Methode an das *Chart* gebunden:

```
            foreach (var q in query)
            {
                var Name = q.Employee.FirstName + ' ' + q.Employee.LastName;
                chart1.Series[0].Points.AddXY(Name, Convert.ToDouble(q.NoOfOrders));
            }
```

Ein kleines Schmankerln zum Schluss, die *Chart* wird als Bilddatei abgelegt:

```
            chart1.SaveImage("ChartTest.bmp", ChartImageFormat.Bmp);
        }
    }
}
```

Test

Das Diagramm zeigt die Anzahl der von jedem Verkäufer insgesamt betreuten Bestellungen.

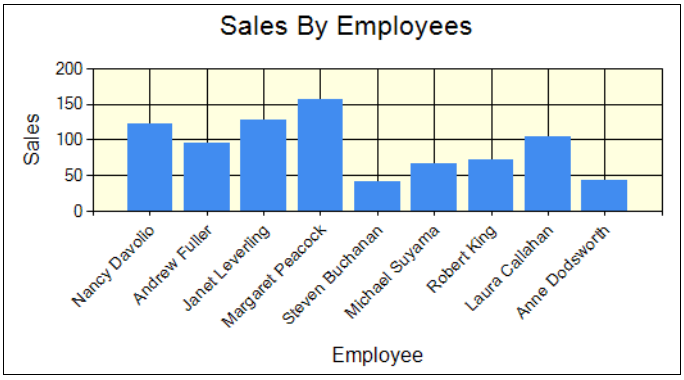


Abbildung 9.23 Laufzeitansicht

9.3 ... mit ASP.NET ein Diagramm anzeigen?

Chart; Entity Framework; Stored Procedures als Funktionsimport

Das *Chart*-Control findet sich nicht nur in der Toolbox für Windows Forms-, sondern auch in der für ASP.-NET-Anwendungen. An einem relativ einfachen Beispiel wollen wir Ihnen demonstrieren, wie Sie per Entity Framework auf eine vorhandene Stored Procedure zugreifen und deren Daten für die Diagrammanzeige verwenden können.

Oberfläche

Erzeugen Sie zunächst eine neue leere ASP.NET Website (*Datei/Neu/Web Site ...*) und legen Sie als Ziel »Dateisystem« fest. Fügen Sie dem Projekt ein neues *Web Form* hinzu (Kontextmenü *Neues Element hinzufügen*).

Als Datenquelle soll uns die schon bekannte Datenbank *Northwind.mdf* dienen. Bevor Sie die Datei per Drag & Drop in den Projektmappen-Explorer ziehen, erzeugen Sie zunächst noch einen Ordner *App_Data* (Kontextmenü *ASP.NET-Ordner hinzufügen*) in welchem wir die Datei ablegen werden.

Nachfolgend sollte das Projekt folgende Struktur haben:

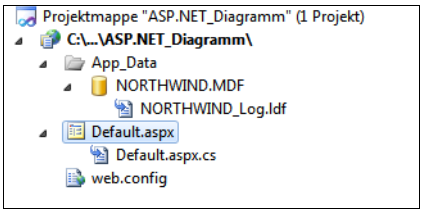


Abbildung 9.24 Projektstruktur

Datenzugriff realisieren

Für unser Beispiel wollen wir das Entity Framework als Datenzugriffstechnologie verwenden. Erzeugen Sie deshalb ein neues »ADO.NET Entity Data Model« (Kontextmenü *Hinzufügen neues Element*) mit dem Namen *NWModel.edmx*.

HINWEIS Das Modell muss bei einem Website-Projekt im Unterordner *App-Code* abgelegt werden, da sonst kein Zugriff auf die erzeugten Klassen möglich ist. Der Assistent wird einen entsprechenden Hinweis anzeigen.

Im Assistenten für das Entity Data Model wählen Sie die bereits eingefügte Datenbank *Northwind.mdf* aus, unsere Daten wollen wir aus der Stored Procedure »Ten Most Expensive Products« beziehen, wählen Sie diese in der Liste der zu importierenden Datenbankobjekte aus (siehe folgende Abbildung 9.25).

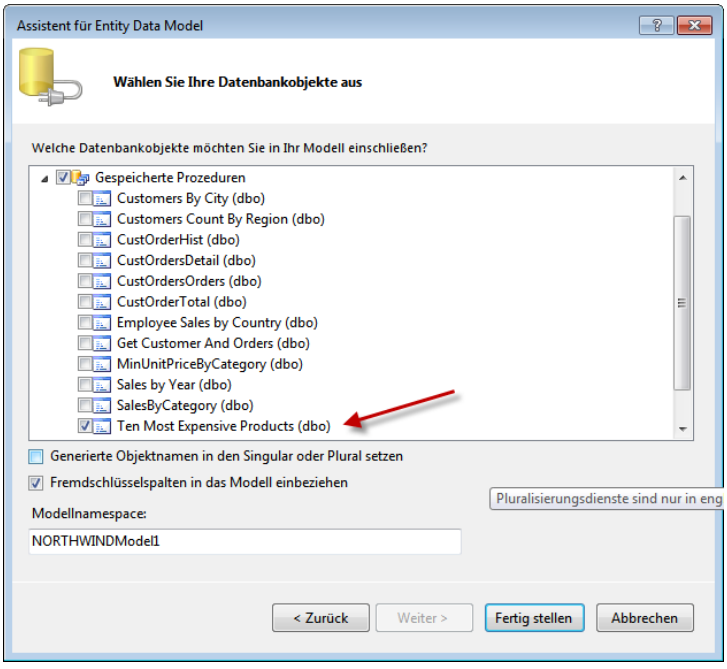


Abbildung 9.25 Auswahl der zu importierenden Datenbankobjekte

Nachfolgend finden Sie ein scheinbar leeres Entity Data Model vor, lassen Sie sich davon nicht täuschen, ein Blick in den Modellbrowser zeigt in der Rubrik »Gespeicherte Prozeduren« unsere importierte Stored Procedure »Ten Most Expensive Products«. Wollen wir diese auch in unserem Programm nutzen, ist es erforderlich, sie als Funktion in das Datenmodell zu importieren. Verwenden Sie dazu das Kontextmenü des Modellbrowsers, wie in der folgenden Abbildung gezeigt:

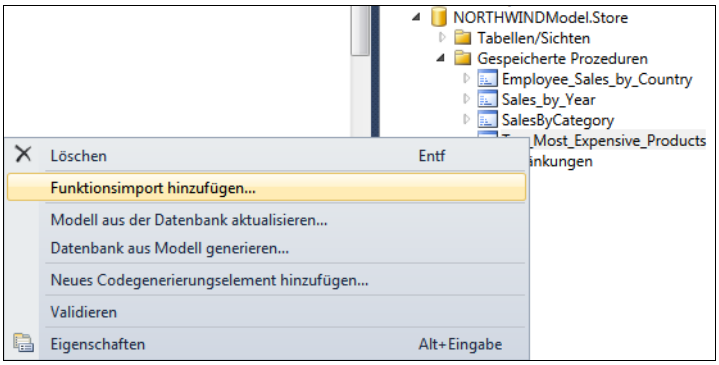


Abbildung 9.26 Die Stored Procedure als Funktion importieren.

Im folgenden Assistenten klicken Sie auf den Eintrag »Spalteninformationen abrufen«, um zunächst die Rückgabewerte der Stored Procedure zu ermitteln.

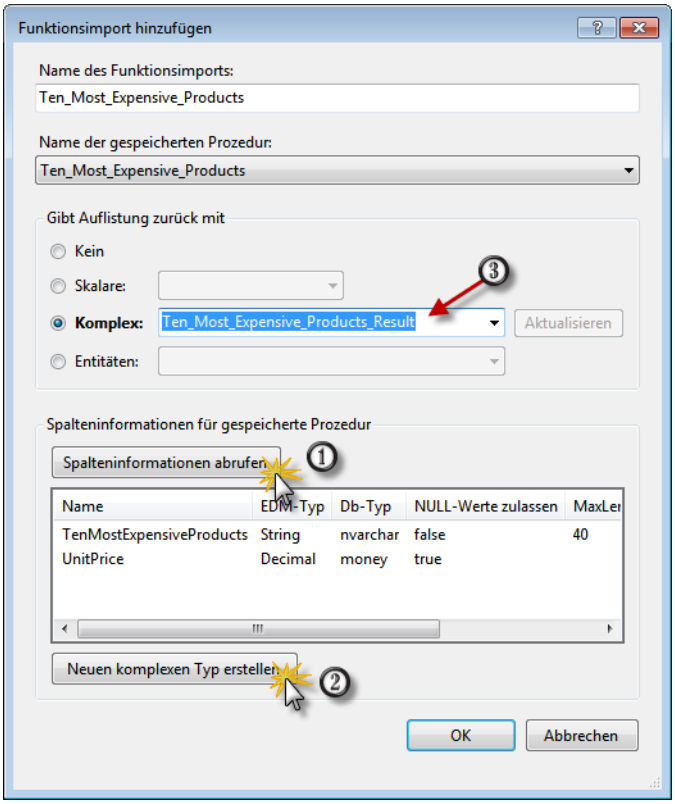


Abbildung 9.27 Assistent für den Funktionsimport

Über die Schaltfläche »Neuen komplexen Typ erstellen« erzeugen wir einen passenden Typ in unserem Modell, der nachfolgend bereits zugewiesen wird (siehe obige Abbildung 9.27).

Kompilieren Sie nachfolgend das Projekt um sicherzustellen, dass das Datenmodell auch erzeugt wird und wir im Weiteren von der Intellisense Gebrauch machen können:

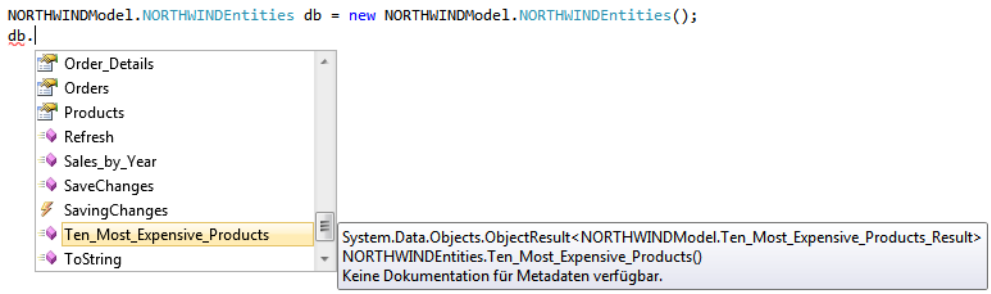


Abbildung 9.28 So einfach ist jetzt der Zugriff auf die Stored Procedure!

Bevor wir uns um den Quelltext kümmern, müssen wir noch ein *Chart*-Control in unser neu erzeugtes Web Form (*default.aspx*) einfügen. Sparen Sie nicht mit der Größe, die inneren Ränder des Diagramms sind später recht groß.

Quelltext

Damit können wir uns den inneren Werten zuwenden. Mit dem Laden des Formulars rufen wir zunächst die Daten per Entity Data Model ab und »basteln« uns dann das Diagramm nach unseren Wünschen zusammen.

Der Namenspace für die Elemente unseres Diagramms:

```
using System.Web.UI.DataVisualization.Charting;

public partial class _Default : System.Web.UI.Page
{
```

Laden der Seite:

```
    protected void Page_Load(object sender, EventArgs e)
    {
```

Daten abrufen:

```
        NORTHWINDModel.NORTHWINDEntities db = new NORTHWINDModel.NORTHWINDEntities();
        var tenprod = db.Ten_Most_Expensive_Products();
```

Wir löschen die bereits vorhandenen *ChartAreas* und *Series*:

```
        Chart1.ChartAreas.Clear();
        Chart1.Series.Clear();
```

Zuweisen der Datenquelle:

```
        Chart1.DataSource = tenprod;
```

Wir erzeugen ein neues *ChartArea*, konfigurieren dieses und fügen es der Auflistung hinzu:

```
        ChartArea chartArea1 = new ChartArea();
        chartArea1.Name = "ChartArea1";
        chartArea1.Area3DStyle.Enable3D = true;
        chartArea1.Area3DStyle.Inclination = 5;
        chartArea1.Area3DStyle.Rotation = 10;
        chartArea1.AxisX.Interval = 1;
        Chart1.ChartAreas.Add(chartArea1);
```

Eine Legende erzeugen:

```
        Legend legend1 = new Legend();
        legend1.Name = "Legend1";
        Chart1.Legends.Add(legend1);
```

Die eigentlichen Daten werden per *Series*-Objekt zugewiesen:

```
        Series series1 = new Series();
```

Horizontale Balken:

```
        series1.ChartType = SeriesChartType.Bar;
        series1.ChartArea = "ChartArea1";
        series1.Legend = "Legend1";
        series1.Name = "Die teuersten Produkte";
```

Hier wird die Brücke zu den beiden Spalten unserer Auflistung geschlagen:

```
        series1.XValueMember = "TenMostExpensiveProducts";
        series1.YValueMembers = "UnitPrice";
```

HINWEIS Da es sich um den Typ »Bar« handelt, sind X- und Y-Achse vertauscht!

Hinzufügen der Serie:

```
        Chart1.Series.Add(series1);
```

Datenbindung:

```
        Chart1.DataBind();
    }
}
```

Das war doch recht einfach, wer es komfortabler mag, kann auch eine *ObjectDataSource* verwenden und dann das *Chart* per Assistent konfigurieren.

Test

Ein Start der Anwendung sollte zum folgenden Ergebnis führen:

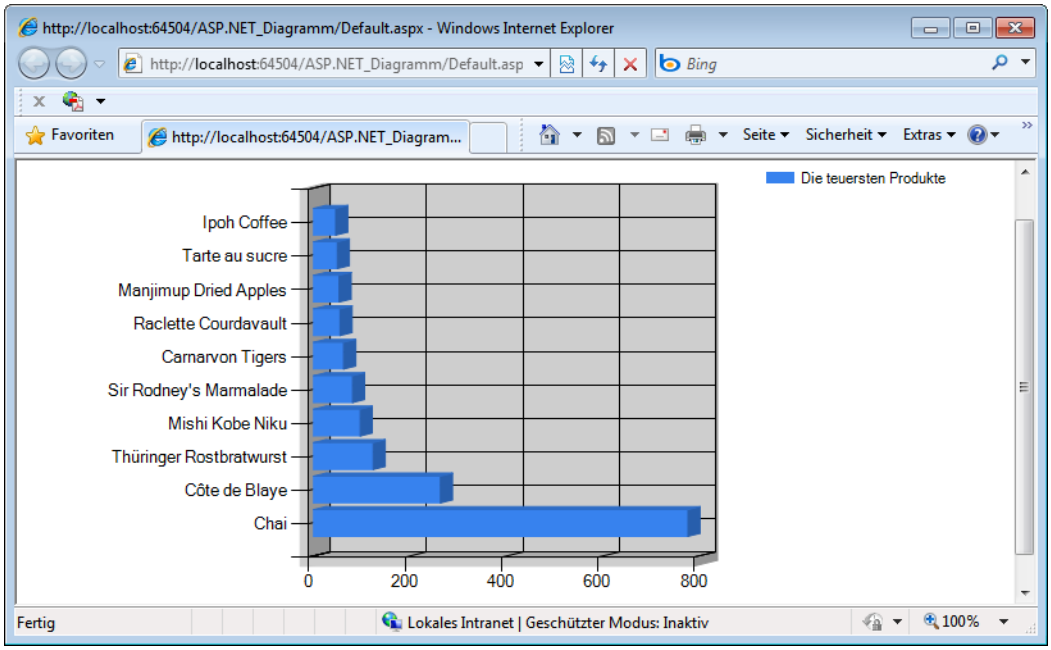


Abbildung 9.29 Die Laufzeitansicht im Internet Explorer