

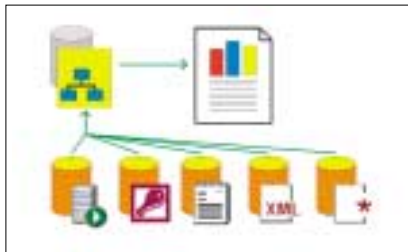
Berichte mit dem ADO.NET-DataSet

## Dröge Daten schick verpackt

Mit der Einführung des .NET Framework wurde das ADO.NET-DataSet als neuer Datenquelltyp für Berichte entdeckt. Im Vergleich zum klassischen Reporting, das bisher auf nativen und ODBC-Datenquellen basierte, kümmert sich der Anwendungsprogrammierer neuerdings selbst um die Bereitstellung der Daten. Dies nützt in vielen exklusiven Einsatzszenarien, wie dieser Artikel zeigt.

**E**in ADO.NET-DataSet-Bericht ist ein echter Anpassungskünstler in kniffligen Szenarien. Der Entwickler kann über das DataSet selbst gezielt die notwendigen Datenbank-Verbindungen aufbauen und dosiert die Datensätze anfordern, die benötigt werden. Dabei müssen die Datensätze noch nicht einmal aus derselben Datenbank stammen.

Da das DataSet als Datenbank im Arbeitsspeicher wiederum für vielfältige Modifikationen und Steuerungen des Entwicklers offen steht, dient es auch als Ausgangsbasis für trickreiche Reporting-Problemlösungen. Dieser relativ aufwändige Prozess wird auch als so genanntes *Push-Modell* bezeichnet, da sich der Bericht im Vergleich zum *Pull-Modell* relativ passiv verhält.



**Abbildung 1** Beim Push-Modell wird zur Laufzeit das DataSet mit den Daten befüllt, die im Bericht dargestellt werden sollen.

Modell bezeichnet, da sich der Bericht im Vergleich zum *Pull-Modell* relativ passiv verhält.

Bei der Architektur des Push-Modells kennt der Bericht lediglich die Datenstruktur und nicht deren Herkunft. Abbildung 1 stellt diese Beziehung dar: Zur Laufzeit wird das DataSet befüllt. In der Zeichnung soll der Zylinder das DataSet darstellen. Anschließend wird der Bericht auf der Basis dieser Daten erzeugt.

Nur ein extra zur Laufzeit mit Daten befülltes DataSet kann einem Berichts-Objekt zugrunde gelegt werden, um es anzuzeigen oder zu exportieren.

Aus welcher Datenbank die Datensätze stammen, spielt dabei keine Rolle. Diese Flexibilität zählt sich aus: Schließlich können auch Text- und XML-Dateien als Datenlieferanten dienen.

Zusammengefasst kann man sagen: Wer kreativ mit einem ADO.NET-DataSet umgehen kann, findet in dieser Berichtstechnik ein wahres Füllhorn an Lösungs-ideen für verschiedene Einsatzbeispiele, die sich bisher nur umständlich realisieren ließen.

In diesem Artikel werden sowohl die grundlegende Vorgehensweise zum Auf-

bau und Anzeigen ADO.NET-DataSet-basierter Berichte erläutert als auch einige Lösungsmuster zum Einsatz dieser raffinierten Technologie vorgestellt.

### Fünf Schritte zum DataSet-basierten Bericht

Um einen Bericht auf der Basis des Push-Modells berechnen zu lassen, sind die folgenden fünf Schritte erforderlich.

In der Entwurfsphase:

1. Erzeugen der *DataSet-Datei*: Mithilfe der IDE von Visual Studio .NET lässt sich komfortabel und grafisch orientiert ein DataSet-Objekt definieren und als XSD-Datei abspeichern. Die Tabellen des DataSet werden später im Feld-Explorer der Reporting-IDE auch als Tabellen mit den zugehörigen Feldern dargestellt.
2. *Berichtentwurf*: Die zuvor gewonnene DataSet-Datei dient dem Berichts-Assistenten als Datenstrukturschema zum Aufbau des Berichts. Zum Berichtsdesign ist anzumerken, dass eine Datenvorschau zur Entwurfszeit nicht möglich ist.

### Listing 1

#### XML-Code der erstellten XSD-Datei.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="KundenDataSet" targetNamespace="http://tempuri.org/KundenDataSet.xsd" elementFormDefault="qualified"
attributeFormDefault="qualified" xmlns="http://tempuri.org/KundenDataSet.xsd"
xmlns:mstns="http://tempuri.org/KundenDataSet.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
<xs:element name="KundenDataSet" msdata:IsDataSet="true">
<xs:complexType>
<xs:choice maxOccurs="unbounded">
<xs:element name="Customers">
<xs:complexType>
<xs:sequence>
<xs:element name="CustomerID" type="xs:string" />
<xs:element name="CompanyName" type="xs:string" minOccurs="0" />
<xs:element name="ContactName" type="xs:string" minOccurs="0" />
<xs:element name="ContactTitle" type="xs:string" minOccurs="0" />
<xs:element name="Address" type="xs:string" minOccurs="0" />
<xs:element name="City" type="xs:string" minOccurs="0" />
<xs:element name="Region" type="xs:string" minOccurs="0" />
<xs:element name="PostalCode" type="xs:string" minOccurs="0" />
<xs:element name="Country" type="xs:string" minOccurs="0" />
<xs:element name="Phone" type="xs:string" minOccurs="0" />
<xs:element name="Fax" type="xs:string" minOccurs="0" />
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
<xs:unique name="KundenDataSetKey1" msdata:PrimaryKey="true">
<xs:selector xpath="."/>
<xs:field xpath="mstns:CustomerID" />
</xs:unique>
</xs:element>
</xs:schema>
```

Zur Laufzeit:

3. Ein *DataSet befüllen*: Ein Bericht-Berechnungslauf kann zur Laufzeit erfolgen. Dabei muss allerdings zun-erst ein DataSet-Objekt mittels

ADO.NET-Objektmodell mit Daten gefüllt werden.

4. *Datenquell-Zuweisung* an den Bericht: Anschließend wird das erzeugte DataSet-Objekt dem Bericht

mit Hilfe der Report Engine als Datenquelle zugewiesen.

5. *Bericht anzeigen*: Abschließend kann das erzeugte Report-Objekt zur Anzeige gebracht werden: Es wird dann an den entsprechenden Web oder Windows Form Viewer gebunden. Die Anzeige eines Berichts ist nicht zwingend. Eine denkbare Alternative wäre ein Berichtsexport – beispielsweise als *pdf-Datei*.

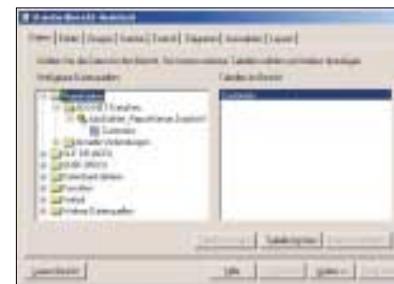
### Ein erstes Beispielprojekt

In dem folgenden Web-Beispielprojekt wurde ein DataSet-Objekt namens *KundenDataSet* angelegt. Es wird über die Dateivorlage *DataSet* erzeugt. Visual Studio .NET ermöglicht die grafische Darstellung und Bearbeitung einer solchen XSD-Schemadatei.

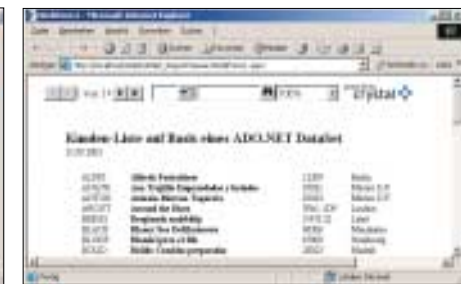
Die Eingabe einer Tabellenstruktur im DataSet kann manuell über Toolbox-Elemente erfolgen, was allerdings je nach Umfang etwas mühsam ist. Am bequemsten ist es, aus dem Server-Explorer heraus die entsprechende Tabelle im Entwurfsbereich des DataSet-Entwurfswindows zu positionieren. Es entstehen dann Objekte, deren Datentypen streng *Common Type System*-konform sind.

Bei der Erstellung des DataSet ist darauf zu achten, dass der Kontextbefehl *DataSet generieren* mit einem Häkchen versehen ist. Die entstehende XSD-Schemadatei wird im XML-Format abgespeichert und enthält die Eigenschaften der Datenquelle. Über den Kontextbefehl *XML-Code anzeigen* des DataSet-Entwurfsbereichs wird die Standard-XML-Struktur erkennbar (siehe Listing 1).

Der *KundenBericht* wird wie gewohnt mit dem Standardberichts-Assistenten



**Abbildung 2** Der Bericht-Assistent bietet als reguläre Datenquelle unter Projektdaten auch ADO.NET-DataSets an.



**Abbildung 3** Der CrystalReportViewer zeigt den ADO.NET-DataSet-Bericht in der Web-Form an.

### Listing 2

#### Einbindung des OleDb Namespace und Deklaration der Berichts-Objektvariable.

```
// Eingabe zu Beginn des Namespace-Definitionsbereiches;
using System.Data.OleDb;
// ...

namespace AdoDotNet_PowerReporting
{
    public class WebForm1 : System.Web.UI.Page
    {
        public KundenListe objRpt = null;
        //...
    }
}
```

generiert. Dieser kann im Projektmapen-Explorer über das Kontextmenü des Projektknotens, **Befehl Hinzufügen/Neues Element hinzufügen**, geöffnet werden. Nachdem als Elementvorlage *Crystal Report* ausgewählt wurde, kann der Bericht aufgebaut werden.

Dabei ist darauf zu achten, dass als Datenquelle unter dem Baumdiagrammzweig *Projektdaten ADO.NET-Datensätze* aufgekloppt ist und das zuvor erzeugte

### Listing 3

#### Das DataSet-Objekt wird mit Daten befüllt und dem Bericht als Datenquelle übergeben.

```
private void Page_Load(object sender, System.EventArgs e)
{
    InitializeComponent();
    base.OnInit(e);

    objRpt = new KundenListe();

    // Datenzugriff auf Datenbank per ADO.NET-Objektmodell:
    OleDbConnection oleConn = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" +
        "Data Source=C:\\DB_Access2000\\nwind.mdb");
    OleDbDataAdapter oleDataAdapter = new OleDbDataAdapter(
        "SELECT * FROM Customers", oleConn);
    DataSet1 KundenDataSet = new DataSet1();

    // Befüllung des DataSet-Objekts:
    // Wichtig: Tabellen-Aliasname entspricht dem DataSet-Tabellennamen
    oleDataAdapter.Fill(KundenDataSet, "Customers");

    // Übergabe des erzeugten DataSet-Objekts an den Bericht
    objRpt.SetDataSource(KundenDataSet);

    // Bindung des Bericht-Objekts mitsamt seinen Daten an
    // das Viewer-Control
    this.CrystalReportViewer1.ReportSource = objRpt;
}
```

*KundenDataSet* selektiert wird. Letzteres listet alle enthaltenen Objekte auf. In diesem Beispiel wird lediglich *Customers* angezeigt. Diese Tabelle kann dem Bericht zugrunde gelegt werden. Die weitere Vorgehensweise (Auswahl der Felder und so weiter) gleicht der bei konventionellen Berichten [1].

Nachdem der Bericht fertig gestellt ist, wird die XSD-Datei von diesem eigentlich nicht mehr benötigt, sofern zukünftig keine Änderungen an der Datenstruktur mehr anstehen. Der *DataSet*-Aufbau wird in der RPT-Berichtsdatei mit gespeichert. Sicherheitshalber sollte zur besseren Dokumentation beziehungsweise zur besseren Pflegebarkeit des Projekts aber doch noch die XSD-Datei als Projektdatei gespeichert bleiben.

Im nächsten Schritt kann in der Web Form ein *CrystalReportViewer*-Control positioniert werden, das später die Anzeige des Berichts übernehmen soll.

Der Bericht wurde lediglich auf einer Datenstruktur-Beschreibungsdatei entworfen und enthält daher selbst keine Daten. Bevor dieser Bericht angezeigt werden kann, muss als Erstes ein entsprechendes *DataSet*-Objekt mit Daten befüllt werden. Um die Anweisungseingabe zu vereinfachen,

wird der OleDb-Namespace eingebunden. Die Berichts-Objektvariable kann am Anfang der Klassendefinition deklariert werden (siehe Listing 2).

Nach der Anweisung *base.OnInit(e)* sind gemäß Listing 3 die Anweisungen zum ADO.NET-Datenzugriff, zur Befüllung des *DataSet*-Objekts und zur Übergabe des *DataSet*-Objekts an den Bericht einzugeben.

Ein besonderes Augenmerk verdient der *Tabellen-Aliasname* bei der Befüllung des *KundenDataSet* durch den *oleDataAdapter*. Dieser muss nämlich mit dem Tabellennamen, der in der *DataSet*-XSD-Datei definiert wurde, identisch sein. Weicht dieser in der Bezeichnung ab, wird der Bericht zwar geöffnet, zeigt allerdings keine Datensätze an.

Das Beispielprojekt kann nun gestartet werden und der Bericht wird angezeigt (siehe Abbildung 3).

#### Filtern des Datenstroms

Sollen nur bestimmte Datensätze im Bericht angezeigt werden, kann man dafür schon vor dem Bericht-Berechnungslauf die Datensatzauswahl mit bekannter SQL-Syntax filtern. Der folgende *WHERE*-Ausdruck beschränkt die Lösungsdatensätze entsprechend.

```
OleDbDataAdapter oleAdapter =
new OleDbDataAdapter("SELECT * FROM Customers
WHERE Country = 'Germany'", oleConn);
```

Die Befüllung des *DataSet*-Objekts beschränkt sich an dieser Stelle ausschließlich auf die Datensätze, die die geforderten Kriterien erfüllen. Damit besteht die Möglichkeit, den Netzwerk-Traffic auf das Nötigste zu beschränken.

#### DataSet mit heterogenen Datenquellen befüllen

Falls gewünscht, können die Tabellen eines *DataSet* durch unterschiedliche Datenquellarten befüllt und innerhalb des *DataSet*-Objekts verknüpft werden. Gemäß Abbildung 4 wäre es also möglich, die Datensätze einer Tabelle des *DataSet* beispielsweise aus einer Access-Datei und die der zweiten Tabelle aus einer SQL-Server-Datenbank zu lesen.

Folgende Schritte sind zur Implementierung einer solchen Lösungsvariante erforderlich:

- In der Visual-Studio-.NET-IDE wird die *DataSet*-Strukturdatei *Artikel-*



Abbildung 4 Die Daten für das Berichts-DataSet werden aus unterschiedlichen Datenquellen gelesen und innerhalb des DataSet-Objekts verknüpft.

- *DataSet.xsd* aufgebaut. Die grafisch orientierte Arbeitsweise erlaubt Drag-and-Drop aus dem Server-Explorer in die Entwurfs-Oberfläche des *DataSet*.
- Die Verknüpfung der beiden Tabellen

- wird über das Werkzeug *Relation* der Toolbox hinzugefügt.
- Der Bericht *ArtikelListe.rpt* wird auf Basis dieses *DataSet* erstellt.
- Der in Listing 4 abgebildete Code befüllt die Tabellen aus unterschiedlichen

### Listing 4

#### DataSet-Befüllung über unterschiedliche Datenquellen und Anzeige des Berichts.

```
private void frmDemo_Load(object sender, System.EventArgs e)
{
    string strOleCon;
    OleDbConnection oleCon;
    OleDbDataAdapter oleDataAdapter = new OleDbDataAdapter();
    string strOleSQL = "SELECT * FROM Categories";
    DataSet objDataSet = new DataSet();

    strOleCon = "Provider=Microsoft.Jet.OLEDB.4.0;" +
        "Data Source=C:\\DB_Access2000\\nwind.mdb";

    oleCon = new OleDbConnection(strOleCon);
    oleDataAdapter.SelectCommand = new OleDbCommand(strOleSQL, oleCon);
    oleDataAdapter.Fill(objDataSet, "Categories");

    string strSqlSvrConnection =
        "Data Source=(local);UID=DemoLogin;pwd=DemoPwd;" +
        "Database=Northwind";

    SqlConnection SqlSvrConnection;
    SqlDataAdapter SqlSvrDataAdapter = new SqlDataAdapter();

    string strSqlSvrSQL = "SELECT * FROM Products";

    SqlSvrConnection = new SqlConnection(strSqlSvrConnection);
    SqlSvrDataAdapter.SelectCommand = new SqlCommand(strSqlSvrSQL, SqlSvrConnection);
    SqlSvrDataAdapter.Fill(objDataSet, "Products");

    ArtikelListe oRep = new ArtikelListe();
    oRep.SetDataSource(objDataSet);
    repViewer.ReportSource = oRep;
}
```

Datenquellen: *Categories* aus einer Access-Datenbankdatei und *Products* aus einer SQL-Server-Datenbank.

Wer etwas ADO.NET-Erfahrung mitbringt, wird von diesem technischen Fallbeispiel viele Ideen ableiten können, diesen Berichtstyp für exklusive Einsatzbeispiele zu verwenden. Bei diesem Beispiel gilt es allerdings zu beachten, dass die Verknüpfung der Datenbestände der beiden Tabellen im *DataSet* zur Laufzeit den Arbeitsspeicher erheblich beansprucht. Daher wird man sich bemühen, eine Datenbeziehung nach Möglichkeit schon in der Datenbank selbst vorwegzunehmen – sofern es das Szenario erlaubt.

#### DataSet-Befüllung mit verknüpften Tabellen

Unter Verwendung des SQL-Ausdrucks *JOIN* lässt sich unter Umständen eine komplette Abfrage aufbauen, die als einzelne virtuelle Lösungstabelle dem *DataSet* übergeben werden kann. Wird auf diese Weise eine Verknüpfung der Datensätze innerhalb des *DataSet* vermieden und einfach vorweggenommen, verbessert sich automatisch die Performance des Berichts.

Das dargestellte Codebeispiel setzt ein *DataSet* mit einer Tabelle *CategoriesProducts* voraus und einen Bericht *KategorienProdukte*, der auf diesem *DataSet* aufgebaut wird.

#### DataSet-Befüllung per XML-Datei

Das XML-Datenformat ist nicht nur eine Moderscheinung, sondern ein hervorragendes *Datentaxi* für das Internet. Obwohl es sich bei einer XML-Datei praktisch nur

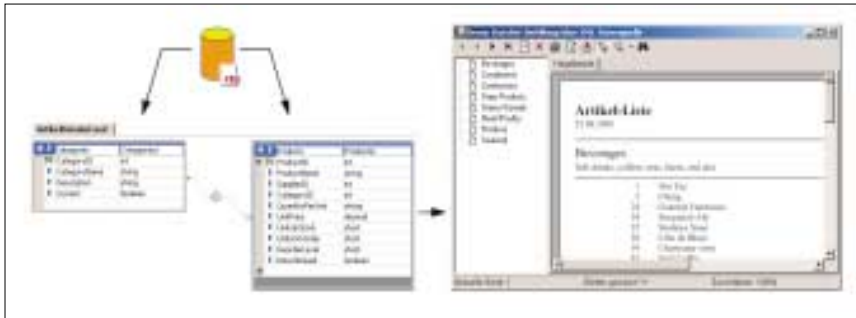


Abbildung 5 DataSet-Befüllung über eine XML-Datei.

um eine Textdatei handelt, beschreibt das XML-Format Daten perfekt strukturiert und kann diese natürlich auch speichern. Die Überführung von Daten eines DataSet in eine XML-Datei – auch als *Serialisierung* bezeichnet – ist mit Hilfe des .NET Framework besonders einfach.

Um auf Basis eines DataSet-Objekts eine XML-Datendatei zu erzeugen, kann die *WriteXml*-Methode verwendet werden.

```
objDataSet.WriteXml("C:\\ ArtikelDaten.xml");
```

Die daraus resultierende Datendatei (siehe Listing 6) kann entweder in der Visual-Studio-.NET-IDE oder noch etwas

komfortabler im Internet Explorer betrachtet werden, da dieser sogar das Öffnen und Schließen der Objektknoten (+/-) erlaubt. Immer wenn Daten im XML-Format vorliegen, ist das Befüllen eines DataSet besonders einfach. Es steht dafür die *ReadXml*-Methode des DataSet zur Verfügung.

Auf den ersten Blick ist es verblüffend, dass nur eine einzige XML-Datei ein DataSet mit mehreren Tabellen befüllen kann (siehe Abbildung 5). Da die Daten aber sauber DataSet-gerecht in der XML-Datei vorliegen, stellt die Zuordnung der Datensätze in die richtige Zieltabelle kein Problem dar. Listing 7 zeigt ein komplettes

Beispiel dafür, wie ein Bericht auf der Basis von XML-Daten geöffnet wird. Allein die *ReadXml*-Methode des DataSet sorgt für die Befüllung des DataSet mit Datensätzen.

### Manuelle DataSet-Befüllung

Woher das DataSet seine Daten bezieht, spielt für den Bericht keine Rolle. Unter Umständen steht auch gar keine Datenbank als Datenlieferant zur Verfügung, sondern es sollen zur Laufzeit dem DataSet manuell Datensätze hinzugefügt werden. Eine manuelle Datenbefüllung ist in einigen Fällen gar nicht so abwegig. In dem nachfolgenden Beispiel soll ein Fragebogen-Dialog einer kleinen Webanwendung die Benutzereingaben als Bericht präsentationsgerecht zusammensetzen (siehe Abbildung 6). Dieser Bericht basiert nicht auf einer bestimmten Datenbank-Datenquelle, sondern auf den temporär zusammengestellten Tabellen-Datensätzen des DataSet.

Das Codebeispiel zeigt, wie zunächst die DataSet-Datenstruktur definiert und anschließend die einzelnen Informationen aus der Web-Form herausgelesen werden. Abschließend werden diese als Datensätze in der *tblWertung* abgespeichert.

Der Fragebogen-Bericht wird diesmal nicht über ein *CrystalReportViewer*-Control angezeigt, sondern als *pdf*-Datei exportiert, die dann im Webbrowser dargestellt wird. Der Anwender kann diese entweder lokal abspeichern oder komfortabel ausdrucken.

Gerade die Druckfunktion wird in Internet-Anwendungen gern über den Umweg eines PDF-Exports realisiert, da das Web-Form-basierte *CrystalReport-*



Abbildung 6 DataSet-Befüllung über die Eingabwerte einer Benutzeroberfläche.

*Viewer*-Control selbst keine Druck-Schaltfläche besitzt und der Ausdruck über den Internet-Browser durch die Kopf- und Fußzeilen überfrachtet wird. Beim Versuch, die Exportdatei zu erzeugen, kann es zu folgender Fehlermeldung kommen: *Zugriff auf Berichtsdatei verweigert*. In diesem Fall sollten die Sicherheitseinstellungen des Zielverzeichnisses erweitert werden. Der lokale Anwender ASP.NET benötigt hier nämlich Schreibrechte.

### Eine Textdatei als Datenquelle

Wenn eine Textdatei als Datenlieferant zur Berichts-Berechnung verwendet werden soll, ist lediglich ein passendes DataSet in das Projekt aufzunehmen. Dieses beschreibt, welche Felder in der Textdatei

enthalten sind, und es wird zur Laufzeit befüllt. Beim Auslesen der Textdatei wandert jede Textzeile als Datensatz in das DataSet-Objekt (siehe Abbildung 7). In dem abgebildeten Code-Beispiel wird ein DataSet-Objekt aufgebaut, das die Tabelle *tblTest* mit den Feldern *strFld1* und *strFld2* enthält. Beim zeilenweisen Auslesen der Textdatei wird dabei das Dataset gleich befüllt (siehe Listing 9).

Die übrigen Anweisungen sind wieder Routine: Zuerst wird ein Report-Objekt erstellt, dessen Datenquelle mit dem eben befüllten DataSet initialisiert wird. Abschließend wird das Bericht-Objekt der *ReportSource*-Eigenschaft des *CrystalReportViewer*-Controls zugewiesen. Im Beispiel-Code wird ersichtlich, wie die Feldinformationen aus jeder Zeile voneinander getrennt werden: Ein Semikolon ist

### Listing 7

#### DataSet-Befüllung über XML-Datendatei und Anzeige des Berichts.

```
private void frmDemo_Load(object sender, System.EventArgs e)
{
    DataSet objDataSet = new DataSet();
    objDataSet.ReadXml("C:\\Demo\\ArtikelDaten.xml");

    ArtikelListe oRep = new ArtikelListe();
    oRep.SetDataSource(objDataSet);
    repViewer.ReportSource = oRep;
}
```

in diesem Beispiel das Feldtrennzeichen, das die Zeichenkette *fields* mit der *Split*-Methode in ein passendes Array zerlegt. Nach dem Start dieses Beispiels werden alle Datensätze aus der Textdatei durch das *CrystalReportViewer*-Control in der Seitenvorschau korrekt dargestellt.

#### DataSet-Dateien mit ADO.NET Klassenmethoden erstellen

Um ein DataSet zu erstellen, ist der Weg per Drag-and-Drop vom Server-Explorer auf die Entwurfsoberfläche sicherlich der eleganteste. Es kann jedoch Fälle geben, bei denen sich die Programmierung einiger Zeilen unter Verwendung der ADO.NET-Klassenmethoden durchaus lohnt:

- a) Je nach Umfang der anstehenden Reporting-Arbeiten soll die Datei-

### Listing 5

#### DataSet-Befüllung über ein SQL-Abfrage-Ergebnis und Anzeige des Berichts.

```
private void frmDemo_Load(object sender, System.EventArgs e)
{
    string strOleDbCon;
    OleDbConnection oleCon;
    OleDbDataAdapter oleDataAdapter = new OleDbDataAdapter();
    string strOleDbSQL = "SELECT * * +
        *FROM Categories INNER JOIN Products * +
        *ON Categories.CategoryID = Products.CategoryID";
    KategorieProdukte objDataSet = new KategorieProdukte();

    strOleDbCon = "Provider=Microsoft.Jet.OLEDB.4.0; * + *Data Source=C:\\VDR_Access2000\\nwind.mdb";

    oleCon = new OleDbConnection(strOleDbCon);
    oleDataAdapter.SelectCommand = new OleDbCommand(strOleDbSQL, oleCon);
    oleDataAdapter.Fill(objDataSet, "CategoriesProducts");

    KategorieProdukte oRep = new KategorieProdukte();
    oRep.SetDataSource(objDataSet);
    repViewer.ReportSource = oRep;
}
```

### Listing 6

#### Ein Auszug aus dem Aufbau der XML-Datendatei.

```
<?xml version="1.0" standalone="yes"?>
<NewDataSet>
  <Categories>
    <CategoryID></CategoryID>
    <CategoryName>Beverages</CategoryName>
    <Description>Soft drinks, coffees, teas, beers, and ales</Description>
    <Current>true</Current>
  </Categories>
  <Categories>
    <CategoryID>2</CategoryID>
    <CategoryName>Condiments</CategoryName>
    <Description>Sweet and savory sauces, relishes, spreads, and seasonings</Description>
    <Current>false</Current>
  </Categories>
  ...
```

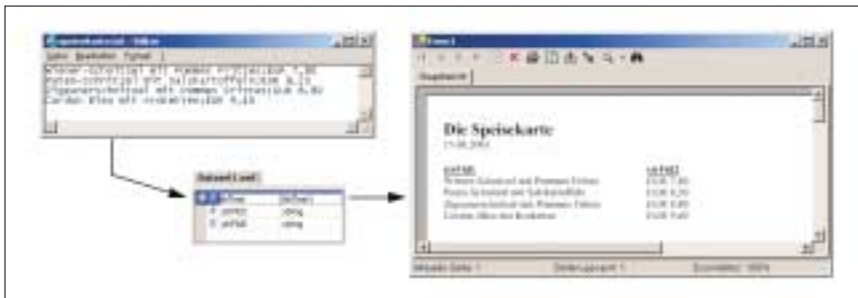


Abbildung 7 DataSet-Befüllung mit einer Textdatei.

erstellung der DataSets automatisiert werden, oder  
b) es steht für das DataSet gar keine greifbare Datenquelle in Form einer XSD-Datei zur Verfügung. Die Anwendung baut eventuell erst zur Laufzeit die Tabellenstrukturen in einem DataSet auf. Diese temporär vorliegende Datenstruktur kann bei Bedarf quasi als Schnappschuss abgespeichert werden.

Listing 10 demonstriert, wie eine XSD-Datei geschrieben wird. Zunächst entsteht die Tabelle `tblArtikel` mit den Feldern `ID` und `Bez` für das DataSet. Die DataSet-Struktur wird mit der `WriteXmlSchema`-Methode des DataSet als XSD-Datei geschrieben.

### Datenstruktur ändern

Sollten Änderungen in einer Datenbank erforderlich werden und damit DataSets und

Berichte aktualisiert werden müssen, gilt es folgende wichtige Punkte zu beachten:  
DataSet-Dateien können entweder manuell korrigiert werden, beispielsweise durch Änderung eines Feldnamens mit der Texteingabemaske, oder sie werden einfach gelöscht und per Drag-and-Drop aus dem Server-Explorer neu aufgebaut.  
Der Befehl zum Aktualisieren von Datenstrukturinformationen eines Berichts befindet sich unter dem Kontextmenü:

### Listing 8

#### Manuelle DataSet-Befüllung mit Benutzereingabewerten und Bericht-Export in das Adobe-Acrobat-Format.

```
private void btnFragebogen_Click(object sender, System.EventArgs e)
{
    DataSet WertungsDataSet = new DataSet();
    DataTable tblWertung = new DataTable("Bewertung");
    DataColumn Datenfeld;
    DataRow Datensatz;

    //erstes Feld (WertungsArt) erstellen:
    Datenfeld = new DataColumn();
    Datenfeld.DataType = System.Type.GetType("System.String");
    Datenfeld.ColumnName = "WertungsArt";
    tblWertung.Columns.Add(Datenfeld);

    //zweites Feld (WertungsNote) erstellen:
    Datenfeld = new DataColumn();
    Datenfeld.DataType = System.Type.GetType("System.String");
    Datenfeld.ColumnName = "WertungsNote";
    tblWertung.Columns.Add(Datenfeld);

    WertungsDataSet.Tables.Add(tblWertung);

    //Datensätze erzeugen:
    Datensatz = tblWertung.NewRow();
    Datensatz["WertungsArt"] = lblWertung1.Text;
    Datensatz["WertungsNote"] = cboNote1.SelectedItem.Value;
    tblWertung.Rows.Add(Datensatz);

    Datensatz = tblWertung.NewRow();
    Datensatz["WertungsArt"] = lblWertung2.Text;
    Datensatz["WertungsNote"] = cboNote2.SelectedItem.Value;
    tblWertung.Rows.Add(Datensatz);

    Datensatz = tblWertung.NewRow();
    Datensatz["WertungsArt"] = lblWertung3.Text;
    Datensatz["WertungsNote"] = cboNote3.SelectedItem.Value;
    tblWertung.Rows.Add(Datensatz);

    Fragebogen objFragebogenBericht = new Fragebogen();
    objFragebogenBericht.SetDataSource(WertungsDataSet);

    CrystalDecisions.Shared.DiskFileDestinationOptions dskOpt =
    new CrystalDecisions.Shared.DiskFileDestinationOptions();
    objFragebogenBericht.ExportOptions.ExportDestinationType =
    CrystalDecisions.Shared.ExportDestinationType.DiskFile;

    objFragebogenBericht.ExportOptions.ExportFormatType =
    CrystalDecisions.Shared.ExportFormatType.PortableDocFormat;

    dskOpt.DiskFileName = Server.MapPath("") + "\\BerichtsExport.pdf";
    objFragebogenBericht.ExportOptions.DestinationOptions = dskOpt;
    //Zur Erstellung der Exportdatei benötigt der
    //lokale Anwender ASPNET-Schreibrechte!
    objFragebogenBericht.Export();
    Response.Redirect("BerichtsExport.pdf");
}
```

### Listing 9

#### DataSet-Befüllung über eine Textdatei und Anzeige mit dem CrystalReportViewer-Control.

```
private void Form1_Load(object sender,
System.EventArgs e)
{
    string fields;
    string [] fieldsArray;

    System.IO.StreamReader file = new
System.IO.StreamReader(
@"C:\TextRep\speisekarte.txt");

    DataSet ds = new DataSet();
    DataTable dt = new DataTable("tblTest");
    dt.Columns.Add("strFld1");
    dt.Columns.Add("strFld2");
    ds.Tables.Add(dt);
    DataRow r;

    while((fields = file.ReadLine()) != null)
    {
        fieldsArray = fields.Split(';');
        r = dt.NewRow();
        r["strFld1"] = fieldsArray[0];
        r["strFld2"] = fieldsArray[1];
        dt.Rows.Add(r);
    }
    file.Close();

    CrystalReport1 oRep = new CrystalReport1();
    oRep.SetDataSource(ds);
    this.crystalReportViewer1.ReportSource = oRep;
}
```

**Datenbank/Datenbank aktualisieren.** Es erscheint ein Hinweis für jede geänderte Tabelle. Unter Umständen wird auch ein Dialog geöffnet, der die Zuordnung von alten zu neuen Feldnamen erlaubt (siehe Abbildung 8). **Wichtiger Bug-Hinweis:** Änderungen am DataSet werden zurzeit



Abbildung 8 Dialog zur Neuordnung von Feldern nach einer Datenbankstruktur-Änderung.

nicht sofort von der Visual-Studio-.NET-IDE erkannt und dem Berichtdesigner übergeben. Daher sind zunächst Änderungen am DataSet abzuspeichern und anschließend Visual Studio .NET neu zu starten.

Es ist wichtig zu wissen, dass Fehler in Datenbank-Aktualisierungen nicht unbedingt das Nicht-Öffnen eines Berichts zur Folge haben. Weitaus unangenehmer können nämlich logische Fehler sein, die dann auftreten können, wenn das DataSet mit Tabellenfeldern befüllt wird, die nicht genau der DataSet-Spezifikation entsprechen. Ein kleiner Tippfehler kann ein typisches Beispiel dafür sein. Das führt eventuell zu einem Feldwert-Mix, der sich nur durch Zufall oder große Aufmerksamkeit aufspüren lässt.

### Fazit

Die ADO.NET-DataSet-Berichte eignen sich für viele Probleme, die man bisher nur sehr umständlich oder gar nicht hat lösen können.  
Die Architektur des technischen Konzepts überzeugt: Der Bericht selbst dient lediglich als eine Art Daten-Formatierungsmaske und kümmert sich nicht, wie im klassischen Pull-Modell bisher üblich, um das Einlesen der Daten. Vielmehr wird er extern, nämlich durch Codeanweisungen, mit einem Datenstrom versorgt, den der Entwickler zuvor bedarfsgerecht an den jeweiligen Fall angepasst hat. Es fällt aber auch auf, dass die Basistechnologie für ADO.NET-Berichte noch relativ jung ist.

Grundsätzlich sollte es möglich sein, DataSet-basierte Berichte auch auf Basis eines `DataView` zu öffnen. Dies ist aufgrund eines Software-Fehlers zurzeit leider nicht möglich. Einziger Workaround dafür wäre die Erzeugung eines `DataSet`- oder `DataTable`-Objekts, welches mit diesen ge-

### Listing 10

#### DataSet-Datenstruktur als XSD-Datei speichern.

```
using System;
using System.IO;
using System.Text;
using System.Xml;
using System.Data;

namespace DataSetAutomat
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            // Aufbau eines DataSet zur Laufzeit
            DataSet ds = new DataSet();
            DataTable tblArtikel = new
            DataTable("Artikel");
            tblArtikel.Columns.Add("ID");
            tblArtikel.Columns.Add("Bez");
            ds.Tables.Add(tblArtikel);

            // Schreiben der XSD-Datei
            FileStream XmlDaten = new
            FileStream(@"C:\DataSet\Artikel.xsd",
            FileMode.Create);
            XmlTextWriter XmlText = new
            XmlTextWriter(XmlDaten,
            Encoding.Unicode);
            ds.WriteXmlSchema(XmlText);

            Console.ReadLine();
        }
    }
}
```

filterten Daten befüllt wird und dann dem Bericht als Datenquelle zugrunde gelegt wird. Die Lösung des Problems wird durch zukünftige monatlich erscheinende HotFixes beziehungsweise Service Packs [2] in Aussicht gestellt.

Die Leistungsfähigkeit eines umfangreichen Bericht-Distributionsystems wird zukünftig sehr stark von der Auswahl des geeigneten Bericht-Quelldatentyps und der passenden Befüllungstechnik abhängen. ADO.NET-DataSets werden dabei mit Sicherheit eine zentrale Rolle spielen. |||||

- [1] Alexander Bräumer, Start-up mit Crystal Reports für Visual Studio .NET, dotnetpro 1/2/2003, Seite 34 ff.
- [2] Aktuelle Service Packs zu Crystal Reports .NET: [http://support.crystaldecisions.com/fix/hot/si75?ref=default.asp\\_selectlist](http://support.crystaldecisions.com/fix/hot/si75?ref=default.asp_selectlist)