

Combit List & Label 15

Lieferservice

Die mit List & Label 15 eingeführten Daten-Provider für .NET erlauben den direkten Zugriff auf viele Datenquellen. Auf diese Weise können auch eigene Datenquellen angebunden werden und profitieren automatisch von Designervorschau und Drilldown.

Die Datenbankunabhängigkeit war und ist das heilige Mantra der List-&-Label-Entwickler. Sie erlaubt eine einfache und schlanke Weitergabe der Applikation, da keine eigenen Datenbanktreiber mitgeliefert werden müssen. Unter .NET ist schon seit einigen Versionen eine Datenbindung über das .NET Framework möglich. In Version 15 von List & Label wurden die beiden Welten nun über einen Satz von Schnittstellen miteinander verbunden.

Aus der Box

Im neuen Namespace `combit.ListLabel15.DataProviders` finden sich die mitgelieferten Datenlieferanten. Mit Hilfe der `DataSource`-Eigenschaft kann ein solcher Provider an die List-&-Label-Komponente gebunden werden. Um zum Beispiel eine komplette SQL-Server-Datenbank mit allen Relationen zu übergeben, sind nur die folgenden Zeilen Code erforderlich:

```
SqlConnection connection = new SqlConnection(
    Properties.Settings.Default.
    AdventureWorksConnectionString);
SqlConnectionDataProvider provider = new
    SqlConnectionDataProvider(connection);
LL.DataSource = provider;
LL.Design();
```

Kaum Handarbeit:

Der Designer mit angebundener AdventureWorks-Testdatenbank



Enthält eine Datenbank neben Tabellen auch gespeicherte Views, können diese wahlweise angebunden werden – entweder zusätzlich zu den Tabellen oder an deren Stelle.

Neben Microsoft SQL-Datenbanken werden weitere Datenbankformate direkt durch eigene Daten-Provider unterstützt:

- Oracle (über `System.Data.OracleClient`)
- PostgreSQL (über `Npgsql`)
- MySQL (über `MySql.Data`)
- SQLite (über `System.Data.SQLite`)
- OleDb (über `System.Data.OleDb`)
- DB2 (über `IBM.Data.DB2`)

Entwickler sind nicht auf die schon fertigen Datenbank-Provider beschränkt. Im Namespace findet sich auch eine abstrakte Basisklasse, über die sich weitere Datenbankformate aus der SQL-Familie anbinden lassen. Voraussetzung dafür ist, dass .NET-Konnectoren vorhanden sind, welche die `IDbCommand`- und `IDbConnection`-Schnittstellen unterstützen. Alles, was von einem eigenen Provider noch geliefert werden muss, sind die Tabellen und Relationen. Um das Generieren von Abfragen, das Verwalten der Verbindung und die gesamte Drucklogik kümmert sich die `DbConnectionDataProvider`-Basisklasse. Dabei werden SQL-Abfragen auf Wunsch automatisch so optimiert, dass zur Druckzeit nur die tatsächlich benötigten Felder aus der Datenbank geholt werden. Zur Auflösung von Relationen werden parametrisierte Abfragen erzeugt und alle Daten über schnelle `DataReader`-Objekte vom Server abgerufen. Die Datenbankverbindung wird so kurz wie möglich offen gehalten, damit der Server über `Connection-Pooling` für die bestmögliche Lastenverteilung sorgen kann.

Weniger ist mehr

Soll nicht die gesamte Datenbank als Datenquelle zur Verfügung stehen, können die angebotenen Datenbankschemata eingegrenzt und beispielsweise nur das Sales-Schema der Datenbank angemeldet werden. Dies ist über eine einfache Überladung des Provider-Konstruktors möglich. Ist ein noch feinerer Zugriff auf die Daten nötig, kommt der `DbCommandSetProvider` zum Einsatz. Über ihn lassen sich die Ergebnisse einzelner SQL-Anweisungen direkt als Datenquelle verwenden. Dabei können beliebig viele Anweisungen kombiniert und über Schlüsselfelder mit Relationen verknüpft werden:

```
OleDbCommand command = new OleDbCommand(„Select *
    from [Customers]“, connection);
OleDbCommand command2 = new OleDbCommand(„Select *
    from [Orders]“, connection);
DbCommandSetDataProvider provider = new
    DbCommandSetDataProvider();
provider.AddCommand(command, „Customers“);
provider.AddCommand(command2, „Orders“);
provider.AddRelation(„Customers2Orders“,
    „Customers“, „Orders“, „CustomerID“,
    „CustomerID“);
LL.DataSource = provider;
LL.Design();
```

Die verwendeten Abfragen müssen nicht unbedingt die Felder einer einzigen Tabelle selektieren. Es können auch Felder aus anderen 1:1-verknüpften Tabellen per Join mit in die Ergebnismenge genommen werden. Es können sogar Tabellen aus verschiedenen Datenbanksystemen gemischt werden.

// Beitrag // Beitrag // Beitrag // Beitrag // Beitrag // Beitrag // Beitrag // Beitrag // Beitrag // Beitrag //

Jenseits von SQL

Neben den Providern, die den direkten Datenbankzugriff erlauben, finden sich weitere Klassen im DataProviders-Name-space. Der ObjectDataProvider erlaubt die Übergabe kompletter Business-Objekte an List & Label. Dabei werden folgende Schnittstellen unterstützt und automatisch als Relationen umgesetzt:

- IEnumerable
- IEnumerable<T>
- IListSource

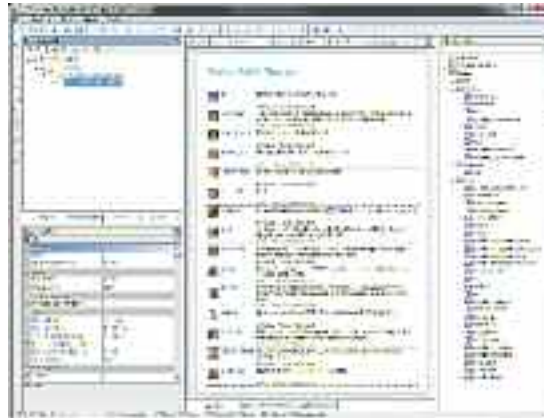
Objektstrukturen können dabei beliebig tief aufgelöst werden. Auch zirkuläre Referenzen sind zulässig und werden bis zu einer einstellbaren Rekursionstiefe aufgelöst. Die üblichen Mechanismen zur Anzeigesteuerung (Browsable- und DisplayName-Attribute, ITypedList-Interface) werden dabei berücksichtigt. Im einfachsten Fall ist die Anbindung mit vier Codezeilen erledigt:

```
List<Customer> customerList =
    ReceiveAllCustomers();
ObjectDataProvider provider = new
    ObjectDataProvider(customerList);
LL.DataSource = provider;
LL.Design();
```

Als Quellobjekte für den Objekt-Daten-Provider bieten sich auch LINQ-Abfrageergebnisse oder EntityCollection-Objekte an.

Ein weiterer Provider schließlich erlaubt das Binden an XML-Streams. Der Schwerpunkt liegt hierbei nicht auf schematisierten XML-Dateien sondern auf mehrfach verschachtelten, untypisierten XML-Dateien, wie sie zum Beispiel häufig von Webdiensten zurückgeliefert oder als Exportergebnisse von XML-fähigen Applikationen erzeugt werden. So liefert etwa die URL http://twitter.com/statuses/public_timeline.xml die öffentliche Timeline von Twitter im XML-Format. Folgender Code bindet List & Label an diese Datenquelle:

```
string url = „http://twitter.com/statuses/
    public_timeline.xml“;
string xml;
using (WebClient client = new WebClient())
{
    using (Stream stream = client.OpenRead(url))
    {
        using (StreamReader reader = new
            StreamReader(stream))
        {
            xml = reader.ReadToEnd();
        }
    }
}
XmlDocument xmlDocument = new XmlDocument();
xmlDocument.LoadXml(xml);
XmlDataProvider provider = new
    XmlDataProvider(xmlDocument);
LL.DataSource = provider;
LL.Design();
```



ii. Datengezwitscher:
List & Label
als Twitter-Client

Und jetzt alle zusammen

Die DataProviderCollection erlaubt es, verschiedene Datenquellen frei zu mischen und gemeinsam für das Reporting zu verwenden. So können Business-Objekte, Ergebnisse einer Abfrage an den SQL-Server und Einstellungen aus einer XML-Datei einfach kombiniert werden:

```
ObjectDataProvider objectData = new
    ObjectDataProvider(GetCustomerList());
DbCommandSetDataProvider sqlData = new
    DbCommandSetDataProvider();
sqlData.AddCommand(new SqlCommand(„Select * From
    Sales“), „Sales“);
XmlDataProvider xmlData = new
    XmlDataProvider(@.\\settings.xml“);

DataProviderCollection provider = new
    DataProviderCollection();
provider.Add(objectData);
provider.Add(sqlData);
provider.Add(xmlData);
LL.DataSource = provider;
LL.Design();
```

Die einzige Einschränkung besteht darin, dass keine übergreifenden Relationen zwischen den verschiedenen Quellen definiert werden können. Doch das ist in der Praxis kein Problem.

Selbst gemacht

Wer unter den mitgelieferten Daten-Providern nicht fündig wird, kann einen eigenen generieren. Die Schnittstellen, die ein Daten-Provider unterstützen muss, sind dokumentiert.



ii. Überschaubar:
Benötigte Schnittstellen
für einen Daten-Provider

Jochen Bartlau ist seit 1998 beim Konstanzer Softwareunternehmen combit beschäftigt. Als Projektleiter ist er zusammen mit seinem Team für die technische Weiterentwicklung der Berichtskomponente List & Label verantwortlich.



Am Anfang steht die IDataProvider-Schnittstelle, sozusagen das relationale Datenbanksystem an sich. Es besteht aus Tabellen und gegebenenfalls Relationen zwischen ihnen. Eine Relation (ITableRelation) wird dabei durch ihren Namen und die Schlüsselfelder in der Eltern- und Kindtabelle definiert. Zur Druckzeit fordert List & Label das jeweils aktuelle Tabellenobjekt (ITable) vom Provider an. Tabellen können dabei selbst

entscheiden, ob beispielsweise das Zählen der enthaltenen Zeilen, das Filtern oder das Sortieren derselben möglich ist. List & Label stellt sich flexibel darauf ein und zeigt gegebenenfalls keinen Fortschrittsbalken beim Druck an, wenn die Datensatzanzahl unbekannt ist.

Zentrale Eigenschaft einer Tabelle ist die Auflistung ihrer Zeilen (ITableRow), also die Rows-Property. Eine Tabellenzeile wiederum besteht aus Spalten, die den eigentlichen Dateninhalt liefern. Ausgestattet mit diesem Instrumentarium kann List & Label alle Features wie den Berichtscontainer, die Echtdatenvorschau im Designer oder die in Version 15 neu hinzugekommenen Drilldown-Reports mit selbst geschriebenen Providern unterstützen.

Im Lieferumfang von Version 15 findet sich dafür auch ein Beispiel, welches eine Bindung an kommaseparierte Werte (CSV) erlaubt. Mit diesem Beispiel als Startpunkt können schnell eigene Provider realisiert werden, die auch exotischere Formate oder eine individuelle Mischung von Datenquellen ansteuern können.

Fazit

Mit den neuen Daten-Providern steht eine mächtige neue Abstraktionsebene für den Datenzugriff zur Verfügung. Die mitgelieferten Provider ermöglichen bisher kaum erschwierlichen Komfort – alleine der Code für die DbConnection-

DataProvider-Basisklasse füllt 25 eng beschriebene DIN-A4-Seiten. Dieser Code arbeitet nun bequem und automatisch für alle im Hintergrund, so dass Sie sich als Entwickler auf die wirklich wichtigen Dinge konzentrieren können – sei es die Freizeitplanung oder das Schreiben eines eigenen Daten-Providers.

Anzeige

.NET- und PowerShell-Seminare von und mit Dr. Holger Schwichtenberg

.NET Framework 4.0 und Visual Studio 2010 – die Neuerungen im Überblick (4 Tage)

15.03.2010 – 18.03.2010 in Essen
14.06.2010 – 17.06.2010 in Hamburg

.NET-Einsteiger-Crashkurs (2 Tage)

.NET 4.0 im Komplettüberblick für Umsteiger von C++, Java, Delphi, VB etc. - ideal für Entscheider und Projektleiter!
15.02.2010 – 16.02.2010 in Essen
12.04.2010 – 13.04.2010 in Essen
07.06.2010 – 08.06.2010 in Essen

.NET-Basisseminar: Softwareentwicklung mit dem .NET Framework 4.0, C# 4.0 und Visual Studio 2010 (4 Tage)

Besteht aus dem .NET-Crashkurs und zwei Tagen Hands-On-Lab mit einer mehrschichtigen Desktop- und Web-Anwendung!
12.04.2010 – 15.04.2010 in Essen

Windows-Desktop-Anwendungen mit Windows Presentation Foundation (WPF) (3 Tage)

Dieser Aufbaukurs kann vergünstigt in Verbindung mit dem .NET-Crashkurs gebucht werden!
17.02.2010 – 19.02.2010 in Essen
09.06.2010 – 11.06.2010 in Essen

Web-Anwendungen mit ASP.NET und C#: IIS, ASP.NET, ASP.NET AJAX, XML-Webservices (3 Tage)

Dieser Aufbaukurs kann vergünstigt in Verbindung mit dem .NET-Crashkurs gebucht werden!
17.02.2010 – 19.02.2010 in Essen
09.06.2010 – 11.06.2010 in Essen

Windows PowerShell (3 Tage)

Für Administratoren und Entwickler
09.03.2010 – 11.03.2010 in Essen
27.04.2010 – 29.04.2010 in Essen
22.06.2010 – 24.06.2010 in Essen

10% Rabatt für DevDorado-Leser

Veranstalter:

www.IT-Visions.de
Dr. Holger Schwichtenberg

In Zusammenarbeit mit:



Seminarinformationen und Anmeldungen: <http://www.it-visions.de/go.aspx?DD39>

DevDorado-Leser erhalten mit dem Gutscheincode DEVDO-10 einen Rabatt von 10% auf den Seminarpreis!