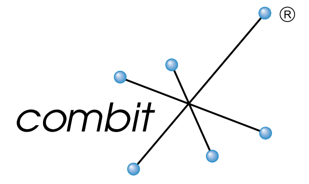


.Net Tutorial

List & Label® 15

Die in diesem Handbuch enthaltenen Angaben sind ohne Gewähr und können ohne weitere Mitteilung geändert werden. Die combit GmbH geht hiermit keinerlei Verpflichtungen ein. Die Verfügbarkeit mancher in dieser Anleitung beschriebener Funktionen (bzw. die Vorgehensweise, um darauf zuzugreifen), ist von Version, Releasestand, eingespielten Servicepacks u.ä. Ihres Systems (z.B. Betriebssystem, Textverarbeitung, eMailprogramm, etc.) sowie seiner Konfiguration abhängig.

Die in diesem Handbuch beschriebene Software wird auf Basis eines Lizenzvertrages geliefert. Der Lizenzvertrag befindet sich bei der Verpackung der CD, bzw. für die ESD-Version im Internet unter www.combit.net und wird auch durch das Installationsprogramm angezeigt.

Dieses Handbuch oder Ausschnitte aus diesem Handbuch dürfen ohne schriftliche Genehmigung der combit GmbH nicht kopiert oder in irgendeiner anderen (z.B. digitaler) Form vervielfältigt werden.

Copyright © combit GmbH 1992-2010; Rev. 15.000

http://www.combit.net

Alle Rechte vorbehalten.

Inhaltsverzeichnis

1. Einleitung 5
1.1 Integration in Visual Studio 5
1.2 Komponenten 5

2. Erste Schritte 7
2.1 List & Label integrieren 7
2.2 Komponente lizenzieren 8
2.3 Datenquelle anbinden 8
2.4 Design 8
2.5 Druck 10
2.6 Export 10
2.7 Wichtige Eigenschaften der Komponente 12
2.8 Webreporting 12

3. Weitere wichtige Konzepte 14
3.1 Datenprovider 14
AdoDataProvider 14
DataProviderCollection 15
DbCommandSetDataProvider 15
ObjectDataProvider 15
OleDbConnectionDataProvider 17
OracleConnectionDataProvider 18
SqlConnectionDataProvider 18
XmlDataProvider 18
3.2 Variablen, Felder und Datentypen 19
Variablen und Felder bei Datenbindung 19
Datentypen 20
3.3 Ereignisse 21
3.4 Projekttypen 21
Listen 22
Etiketten 22
Karteikarten 23
3.5 Verschiedene Drucker und Kopiendruck 23
Layoutbereiche 23
Ausfertigungen und Kopien 23
3.6 Designer anpassen und erweitern 24
Menüpunkte, Objekte und Funktionen sperren 24
Designer erweitern 25
3.7 Objekte im Designer 25
Text 25
Bild 26
Barcode26
RTF-Text 26
HTML 27

3.8 Berichtscontainer 27
3.9 Objektmodell 28
3.10 Fehlerhandling mit Exceptions 29
3.11 Debugging 30
Protokolldatei anfertigen 30

4. Beispiele 32
4.1 Einfaches Etikett 32
4.2 Einfache Liste 33
4.3 Sammelrechnung 33
4.4 Karteikarte mit einfachen Platzhaltern drucken 34
4.5 Unterberichte 35
4.6 Charts 36
4.7 Kreuztabellen 36
4.8 Datenbankunabhängige Inhalte 36
Zusätzliche Inhalte übergeben 36
Daten aus der Datenbindung unterdrücken 38
Vollständig eigene Datenstrukturen/-inhalte 39
4.9 Export 39
Export ohne Benutzerinteraktion 39
Einschränkung von Exportformaten 40
4.10 Designer um eigene Funktion erweitern 41
4.11 Vorschaudateien zusammenfügen und konvertieren 42
4.12 eMail-Versand 43
4.13 Druck im Netzwerk 44

5. Update von älteren Versionen 46

6. Index 47

1. Einleitung

Für die Verwendung von List & Label unter .NET stehen diverse Komponenten zur Verfügung, die die Erstellung von Berichten auf der .NET-Plattform so einfach wie möglich machen. Dieses Tutorial zeigt die wichtigsten Schritte, um schnell und produktiv mit List & Label zu arbeiten.

Die gesamte Programmierschnittstelle ist in der Komponentenhilfe für .NET ausführlich dokumentiert. Diese finden Sie im Startmenü unter Dokumentationen > Komponenten > List & Label .NET Hilfe

1.1 Integration in Visual Studio

Die List & Label-.NET-Komponente wird automatisch in Microsoft Visual Studio eingebunden. Für andere Programmierumgebungen oder bei einer Neuinstallation der Entwicklungsumgebung kann dies auch manuell erfolgen. Die Komponenten liegen als Assembly in den Verzeichnissen "Programmierbare Beispiele und Deklarationen\Microsoft .NET\" sowie "Redistributierbare Dateien\" der List & Label Installation. Die Einbindung geschieht folgendermaßen:

- Menüleiste **Extras** > **Toolboxelemente auswählen**
- Reiter **.NET Framework Komponenten** wählen
- Schaltfläche **Durchsuchen...** klicken
- **combit.ListLabel15.dll** auswählen

Nun können die List & Label-Komponenten wie üblich per Drag & Drop aus der Toolbox auf eine Form gezogen werden. Über das Eigenschaftsfenster können die einzelnen Eigenschaften bearbeitet und Ereignisbehandlungen eingefügt werden.

1.2 Komponenten

Im Reiter "combit LL15" in der Toolbox finden sich nach der Installation die folgenden Komponenten:

Komponente	Beschreibung
ListLabel	Die wichtigste Komponente. In dieser sind alle zentralen Funktionen wie Druck, Design und Export vereinigt.
ListLabelRTFControl	Eine RTF-Editor-Komponente zur Verwendung in eigenen Formularen.
ListLabelPreviewControl	Ein Vorschaucontrol, das ebenfalls in eigenen Formularen verwendet werden kann und z.B. den Direktexport nach PDF unterstützt.

Komponente	Beschreibung
ListLabelDocument	Eine Ableitung von PrintDocument. Mit dieser lassen sich auch die .NET-eigenen Preview-Klassen zur Anzeige von List & Label Vorschaudateien verwenden.
ListLabelWebViewer	Ein ASP.NET-Control zur Anzeige von Vorschaudateien im Internet Explorer.

2. Erste Schritte

Dieser Abschnitt führt durch die ersten Schritte, die benötigt werden um List & Label in eine bestehende Applikation zu integrieren.

2.1 List & Label integrieren

Zunächst muss dem Projekt ein Verweis auf die List & Label Assembly combit.ListLabel15.dll hinzugefügt werden. Diese befindet sich im Verzeichnis "Programmierbare Beispiele und Deklarationen\Microsoft .NET\" der Installation. Über Projekt > Verweis hinzufügen kann die Assembly dem Projekt hinzugefügt werden.

Im zweiten Schritt kann dann eine Instanz der Komponente erzeugt werden. Dies erfolgt entweder über die Entwicklungsumgebung direkt, indem die ListLabel-Komponente auf ein Formular gezogen wird. Alternativ kann die Komponente auch dynamisch erzeugt werden:

C#:

```
combit.ListLabel15.ListLabel LL = new combit.ListLabel15.ListLabel();
```

VB.NET:

```
Dim LL As New combit.ListLabel15.ListLabel()
```

In der Regel wird der Namespace combit.ListLabel15 für die ganze Datei vorreferenziert, je nach Programmiersprache über "using" (C#) oder "Imports" (VB.NET). Dies spart in der Folge viel Tipparbeit.

Bei der dynamischen Erzeugung sollte die Komponente nach Verwendung über die Dispose-Methode wieder freigegeben werden, damit die nicht-verwalteten Ressourcen möglichst schnell wieder freigegeben werden.

C#:

```
LL.Dispose();
```

VB.NET:

```
LL.Dispose()
```

Aus Performancegründen empfiehlt es sich aber auch bei dynamischer Erzeugung, immer eine Instanz des ListLabel-Objektes global im Speicher zu halten. Diese kann z.B. im Load-Ereignis des Applikationshauptfensters erzeugt und im FormClosed-Ereignis wieder freigegeben werden. Der entscheidende Vorteil dabei ist, dass die List & Label-Module nicht für jede neue Instanz ge- und entladen werden, was bei häufigen Aufrufen oder z.B. auch beim Seriendruck für unerwünschte Verzögerungen sorgen kann.

2.2 Komponente lizenzieren

Im Startmenü wird während der Installation der Vollversion (nicht bei der Trial-Version) eine Datei mit persönlichen Lizenz- und Supportinformationen angelegt. Damit List & Label bei Endkunden erfolgreich verwendet werden kann, muss unbedingt der darin enthaltene Lizenzschlüssel an alle Instanzen des ListLabel-Objektes übergeben werden. Das Objekt stellt hierfür die Eigenschaft LicensingInfo zur Verfügung. Ein Beispielaufruf könnte wie folgt aussehen:

C#:

```
LL.LicensingInfo = "A83jHd";
```

VB.NET:

```
LL.LicensingInfo = "A83jHd"
```

wobei der Teil in Anführungszeichen durch den Lizenzcode aus der Textdatei ersetzt werden muss.

2.3 Datenquelle anbinden

Für Design und Druck muss List & Label eine Datenquelle bekannt gemacht werden. Einen Überblick über die verfügbaren Datenquellen bietet der Abschnitt "Datenprovider". Natürlich können auch zusätzlich ungebundene, eigene Daten übergeben werden. Ein Beispiel hierfür zeigt der Abschnitt "Datenbankunabhängige Inhalte".

Um die Datenquelle an List & Label anzubinden stellt die Komponente die Eigenschaft DataSource zur Verfügung. Auch hier kann die Anbindung entweder interaktiv in der Entwicklungsumgebung über das Eigenschaftenfenster oder die SmartTags der Komponente vorgenommen werden oder alternativ auf Code-Ebene erfolgen:

C#:

```
LL.DataSource = CreateDataSet();
```

VB.NET:

```
LL.DataSource = CreateDataSet()
```

2.4 Design

Der Designer wird über die Methode Design() aufgerufen. Zuvor muss immer eine Datenquelle zugewiesen werden – diese ist die Basis für die im Designer verfügbaren Daten. Daher gibt es auch keine alleinstehende Design-Anwendung; die Daten werden immer direkt aus der Applikation zur Verfügung gestellt, List & Label selbst greift niemals direkt auf Daten zu.

Der vollständige Aufruf – in diesem Beispiel mit einem DataSet als Datenquelle – wäre:

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();
LL.Design();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()
LL.Design()
LL.Dispose()
```

Standardmäßig wird hierbei ein Dateiauswahldialog für den Anwender angezeigt, in dem er entweder einen neuen Namen für die Berichtsdatei vergeben und so einen neuen Bericht erzeugen kann oder eine bestehende Datei zur Bearbeitung auswählen kann. Natürlich kann dies auch unterdrückt werden – der Abschnitt "Wichtige Eigenschaften der Komponente" beschreibt dies.

Die Verwendung des Designers selbst ist in der zugehörigen Onlinehilfe bzw. im Designerhandbuch detailliert erklärt. Das Ergebnis des Designprozesses sind in der Regel vier Dateien, die durch den Designer angelegt wurden. Die Dateiendungen können von der Applikation über die FileExtensions-Eigenschaft der ListLabel-Komponente frei bestimmt werden. Die folgende Tabelle beschreibt die Dateien für den Standardfall.

Datei	Inhalt
<Berichtsname>.lst	Die eigentliche Projektdatei. Diese enthält Informationen über die Formatierung der zu druckenden Daten, nicht aber die Daten selbst.
<Berichtsname>.lsv	Eine JPEG-Datei mit einer Miniaturdarstellung des Projektes für die Anzeige im Dateiauswahldialog.
<Berichtsname>.lsp	Datei mit benutzerspezifischen Drucker- und Exporteinstellungen. Diese Datei sollte nicht weitergegeben werden, wenn der Designrechner nicht mit dem Druckrechner identisch ist, da dann der darin angegebene Drucker meist nicht existiert.
<Berichtsname>.~lst	Wird ab dem zweiten Speichern im Designer angelegt und enthält eine Sicherung der Projektdatei.

Die wichtigste Datei ist dabei natürlich die Projektdatei. Die anderen Dateien werden von List & Label automatisch zur Laufzeit der Anwendung erstellt.

Zur Druckzeit wird dann aus der Kombination von Projektdatei und Datenquelle der eigentliche Bericht erstellt.

2.5 Druck

Der Druck wird über die Methode Print() aufgerufen. Zuvor muss im Designer eine Projektdatei für die Datenstruktur der gewählten Datenquelle erstellt werden. Am einfachsten bindet man die Komponente zur Druck- und Designzeit an die gleiche Datenquelle. Dann zeigt auch die Vorschau im Designer die richtigen Daten an und der Anwender kann sich ein gutes Bild vom Ergebnis zur Laufzeit machen. Ein vollständiger Aufruf des Drucks wäre:

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();
LL.Print();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()
LL.Print()
LL.Dispose()
```

Auch hier erscheinen in der Standard-Einstellung zunächst ein Dateiauswahl-, dann ein Druckoptionsdialog. Der Abschnitt "Wichtige Eigenschaften der Komponente" beschreibt, wie diese umgangen bzw. vorausgefüllt werden können, wenn dies erwünscht ist.

2.6 Export

Unter Export wird die Ausgabe auf eines der unterstützten Ausgabeformate wie PDF, HTML, RTF, XLS usw. verstanden. Der Start eines Exports ist codeseitig identisch mit dem eines Drucks, im Druckoptionsdialog kann der Anwender neben den "normalen" Ausgabeformaten Drucker, Datei und Vorschau auch ein beliebiges Exportformat wählen. Soll ein Format als Standardwert vorgewählt werden, kann dies vor Druckstart wie folgt erfolgen:

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();
LL.ExportOptions.Add(LlExportOption.ExportTarget, "PDF");
LL.Print();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()
LL.ExportOptions.Add(LlExportOption.ExportTarget, "PDF")
LL.Print()
LL.Dispose()
```

Die verfügbaren Exportziele listet die folgende Tabelle auf:

Exportziel	Wert für ExportTarget
PDF	PDF
HTML	HTML
RTF	RTF
Bitmap	PICTURE_BMP
EMF	PICTURE_EMF
TIFF	PICTURE_TIFF
Multi-TIFF	PICTURE_MULTITIFF
JPEG	PICTURE_JPEG
Excel	XLS
XPS	XPS
Multi-Mime HTML	MHTML
XML	XML
Text	TXT
Nadeldrucker	TTY
Vorschau	PRV
Drucker	PRN
Datei	FILE

Auch die weiteren Optionen (z.B. Schriftarteinbettung, Verschlüsselung etc.) lassen sich direkt aus dem Code mit Standardwerten vorbelegen. Dies erfolgt wie im Beispiel oben ebenfalls über die ExportOptions-Klasse, die LlExportOption-Enumeration beinhaltet für alle unterstützten Optionen eigene Werte.

Am häufigsten werden diese benötigt, um einen "stillen" Export durchzuführen. Ein Beispiel hierfür findet sich im Abschnitt "Export ohne Benutzerinteraktion".

2.7 Wichtige Eigenschaften der Komponente

Das Verhalten von Druck, Design und Export kann durch einige Eigenschaften der Komponente beeinflusst werden. Die Wichtigsten sind in der folgenden Tabelle zusammengestellt:

Eigenschaft	Funktion
AutoDesignerFile	Name der zu verwendenden Projektdatei. Dies ist der Vorgabename für den Anwender, wenn diesem ein Datei- Auswahldialog zur Verfügung gestellt wird. Ansonsten ist dies der Name des zu verwendenden Projekts (Voreinstellung: leer).
AutoDestination	Ausgabeformat. Wenn gewünscht, kann dem Anwender über diese Eigenschaft ein Format vorgegeben werden, z.B. Druck nur auf Drucker oder Vorschau erlaubt (Voreinstellung: LlPrintMode.Export). Wenn eine Auswahl von Exportformaten erlaubt werden soll, kann dies über das Setzen von LlOptionString.ExportsAllowed erfolgen. Dies wird im Abschnitt "Einschränkung von Exportformaten" gezeigt.
AutoFileAlsoNew	Bestimmt, ob der Benutzer für das Design auch einen noch nicht vorhandenen Dateinamen angeben darf, um ein neues Projekt zu erzeugen (Voreinstellung: true).
AutoProjectType	Legt den Projekttypen fest. Die verschiedenen Projekttypen sind im Abschnitt "Projekttypen" erklärt (Voreinstellung: LlProject.List).
AutoShowPrintOptions	Bestimmt, ob der Druckoptionsdialog angezeigt oder unterdrückt wird (Voreinstellung: true, anzeigen).
AutoShowSelectFile	Bestimmt, ob der Dateiauswahldialog angezeigt oder unterdrückt wird (Voreinstellung: true, anzeigen).
AutoMasterMode	Dient gemeinsam mit der DataMember-Eigenschaft dazu, bei 1:n-verknüpften Datenstrukturen die Haupttabelle als Variablen zu übergeben. Ein Beispiel findet sich im Abschnitt "Variablen, Felder".

2.8 Webreporting

Der Druck innerhalb einer Webapplikation ist im Grunde nichts anderes als ein Export z.B. auf das PDF-Format, bei dem alle Dialoge unterdrückt werden. Wie dies grundsätzlich funktioniert ist im Abschnitt "Export ohne Benutzerinteraktion" beschrieben. Nachdem der Bericht auf diese Weise erstellt wurde, kann der Browser

des Anwenders über die üblichen Mechanismen auf die erstellte Datei geleitet werden. Alternativ kann die Datei auch direkt per eMail an den Anwender gesendet werden, wenn die Erstellung z.B. zeitgesteuert erfolgen soll (s. Abschnitt "eMail-Versand").

Das Design ist nicht direkt im Browser möglich. In der Regel werden die Projektdateien innerhalb einer Client-Applikation erstellt und dann zusammen mit der Webapplikation veröffentlicht. Soll auch der Anwender der Webapplikation Designmöglichkeiten haben, kann dies über eine kleine Designanwendung verwirklicht werden, die auf dem Client installiert wird und z.B. per Webservice mit der Serveranwendung kommuniziert. Dies demonstriert das mitgelieferte Webreporting-Beispiel im Verzeichnis "Programmierbare Beispiele und Deklarationen\Microsoft .NET\Webreporting".

Auch auf dem Server muss zumindest ein Druckertreiber installiert sein, damit der Druck erfolgreich durchgeführt werden kann. List & Label arbeitet eng mit dem Windows-GDI zusammen und benötigt daher für alle Operationen einen Drucker-Gerätekontext. Auch der Microsoft XPS Druckertreiber (ab .NET Framework 3.5 auf allen Systemen automatisch installiert) eignet sich als Referenzdrucker.

3. Weitere wichtige Konzepte

3.1 Datenprovider

Die Datenversorgung in List & Label erfolgt über Datenprovider. Dies sind Klassen, die das Interface IDataProvider aus dem Namespace combit.ListLabel15.DataProviders implementieren. In diesem Namespace sind bereits eine Vielzahl von Klassen enthalten, die als Datenprovider fungieren können. Eine ausführliche Klassenreferenz ist in der .NET Komponentenhilfe enthalten.

Für augenscheinlich nicht direkt unterstützte Dateninhalte findet sich meist trotzdem ein passender Provider. Businessdaten aus Anwendungen können in der Regel über den Objektdatenprovider übergeben werden, liegen die Daten in kommaseparierter Form vor kann der Datenprovider aus dem "Dataprovider"-Beispiel verwendet werden. Viele andere Datenquellen unterstützen die Serialisierung nach XML, so dass dann der XmlDataProvider verwendet werden kann. Wenn nur einige wenige zusätzliche Informationen übergeben werden sollen, so ist auch dies direkt möglich – ein Beispiel zeigt der Abschnitt "Datenbankunabhängige Inhalte".

Die folgende Übersicht listet die wichtigsten Klassen und die von Ihnen unterstützten Datenquellen auf.

AdoDataProvider

Ermöglicht den Zugriff auf Daten der folgenden ADO.NET Elemente:

- DataView
- DataTable
- DataViewManager
- DataSet

Der Provider kann implizit zugewiesen werden, indem die DataSource-Eigenschaft auf eine Instanz einer der unterstützten Klassen gesetzt wird. Natürlich kann der Provider aber auch explizit zugewiesen werden. Beispiel:

C#:

```
ListLabel LL = new ListLabel();
AdoDataProvider provider = new AdoDataProvider(CreateDataSet());
LL.DataSource = provider;
LL.Print();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
Dim provider As New AdoDataProvider(CreateDataSet())
LL.DataSource = provider
LL.Print()
```

```
LL.Dispose()
```

DataProviderCollection

Ermöglicht die Kombination mehrerer anderer Datenprovider in einer Datenquelle. Der Provider kann zum Beispiel Daten aus mehreren DataSet-Klassen kombinieren oder unterstützt die Mischung aus XML und eigenen Objektdaten. Beispiel:

C#:

```
DataSet ds1 = CreateDataSet();
DataSet ds2 = CreateOtherDataSet();

// Daten von ds1 und ds2 in einer Datenquelle kombinieren
DataProviderCollection providerCollection = new DataProviderCollection();
providerCollection.Add(new AdoDataProvider(ds1));
providerCollection.Add(new AdoDataProvider(ds2));
ListLabel LL = new ListLabel();
LL.DataSource = providerCollection;
LL.Design();
LL.Dispose();
```

VB.NET:

```
Dim ds1 As DataSet = CreateDataSet()
Dim ds2 As DataSet = CreateOtherDataSet()

' Daten von ds1 und ds2 in einer Datenquelle kombinieren
Dim providerCollection As New DataProviderCollection()
providerCollection.Add(New AdoDataProvider(ds1))
providerCollection.Add(New AdoDataProvider(ds2))
Dim LL As New ListLabel()
LL.DataSource = providerCollection
LL.Design()
LL.Dispose()
```

DbCommandSetDataProvider

Erlaubt es, mehrere IDbCommand Implementierungen in einer Datenquelle zu kombinieren. Der Provider kann z.B. dazu verwendet werden, um auf mehrere SQL-Tabellen zuzugreifen und Relationen zwischen diesen zu definieren. Eine andere Möglichkeit ist es Daten aus bspw. Microsoft SQL- und Oracle-Datenbanken in einer Datenquelle zu kombinieren.

ObjectDataProvider

Erlaubt den Zugriff auf Objektstrukturen. Der Provider kann mit folgenden Typen/Schnittstellen zusammenarbeiten:

- IEnumerable (setzt jedoch mindestens einen Datensatz voraus)
- IEnumerable<T>

- IListSource

Die Eigenschaftsnamen und -typen können über die ITypedList Schnittstelle beeinflusst werden. Wenn nur der Name geändert werden soll ist es meist einfacher, das DisplayNameAttribute zu verwenden. Einzelne Member können über das Browsable(False) Attribut unterdrückt werden.

Der Provider kann leere Aufzählungen durchlaufen solange diese stark typisiert sind. Ansonsten wird mindestens ein Element in der Aufzählung vorausgesetzt. Dieses erste Element bestimmt den Typ der für das weitere Durchlaufen verwendet wird.

Der Provider unterstützt Sortierung automatisch sobald die Datenquelle die IBindingList Schnittstelle implementiert.

Dieser Datenprovider unterstützt auch die Bindung an LINQ Abfrageresultate, da diese IEnumerable<T> sind.

Bei Verwendung von EntityCollection<T>-Objekten als Datenquelle prüft der ObjectDataProvider zunächst mit Hilfe der IsLoaded-Eigenschaft den Zustand der Unterrelation und ruft gegebenenfalls dynamisch Load() auf. Damit werden die Daten bereitgestellt wenn sie benötigt werden. Beispiel:

C#:

```
class Car
{
    public string Brand { get; set; }
    public string Model { get; set; }
}

List<Car> cars = new List<Car>();
cars.Add(new Car { Brand = "VW", Model = "Passat"});
cars.Add(new Car { Brand = "Porsche", Model = "Cayenne"});
ListLabel LL = new ListLabel();
LL.DataSource = new ObjectDataProvider(cars);
LL.Design();
LL.Dispose();
```

VB.NET:

```
Public Class Car
    Dim _brand As String
    Dim _model As String

    Public Property Brand() As String
    Get
        Return _brand
    End Get
    Set(ByVal value As String)
        _brand = value
    End Set
    End Property
    Public Property Model() As String
    Get
```

```
        Return _model
    End Get
    Set(ByVal value As String)
        _model = value
    End Set
    End Property
End Class

Dim LL As New ListLabel
Dim Cars As New List(Of Car)()
Dim Car As New Car
Car.Model = "Passat"
Car.Brand = "VW"
Cars.Add(Car)
Car = New Car
Car.Model = "Cayenne"
Car.Brand = "Porsche"
Cars.Add(Car)
LL.DataSource = New ObjectDataProvider(Cars)
LL.AutoProjectType = LlProject.List
LL.Design()
LL.Dispose()
```

OleDbConnectionDataProvider

Ermöglicht das Binden an eine OleDbConnection (z.B. Access Datenbankdatei). Beispiel:

C#:

```
OleDbConnection conn = new
OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
DatabasePath);
OleDbConnectionDataProvider provider = new OleDbConnectionDataProvider(conn);
ListLabel LL = new ListLabel();
LL.DataSource = provider;
LL.Design();
LL.Dispose();
```

VB.NET:

```
Dim conn As New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=" + DatabasePath)
Dim provider As New OleDbConnectionDataProvider(conn)
Dim LL As New ListLabel()
LL.DataSource = provider
LL.Design()
LL.Dispose()
```

OracleConnectionDataProvider

Ermöglicht das Binden an eine OracleConnection.

SqlConnectionDataProvider

Ermöglicht das Binden an eine SqlConnection. Beispiel:

C#:

```
SqlConnection conn = new
SqlConnection(Properties.Settings.Default.ConnectionString);
SqlConnectionDataProvider provider = new SqlConnectionDataProvider(conn);
ListLabel LL = new ListLabel();
LL.DataSource = provider;
LL.Design();
LL.Dispose();
```

VB.NET:

```
Dim conn As New SqlConnection(Properties.Settings.Default.ConnectionString)
Dim provider As New SqlConnectionDataProvider(conn)
Dim LL As New ListLabel()
LL.DataSource = provider
LL.Design()
LL.Dispose()
```

XmlDataProvider

Ermöglicht den einfachen Zugriff auf XML-Daten. Es werden keine Schemainformationen aus XML-/XSD-Dateien verwendet und keine Constraints/Randbedingungen behandelt. Der Haupteinsatzzweck dieser Klasse ist der schnelle und einfache Zugriff auf verschachtelte XML-Daten. Beispiel:

C#:

```
XmlDataProvider provider = new XmlDataProvider(@"c:\users\public\data.xml");
ListLabel LL = new ListLabel();
LL.DataSource = provider;
LL.Design();
LL.Dispose();
```

VB.NET:

```
Dim provider As New XmlDataProvider("c:\users\public\data.xml")
Dim LL As New ListLabel()
LL.DataSource = provider
LL.Design()
LL.Dispose()
```

3.2 Variablen, Felder und Datentypen

Variablen und Felder sind die dynamischen Textblöcke für Berichte und enthalten den dynamischen Teil der Daten. Variablen ändern sich typischerweise einmal pro Seite oder Bericht – ein Beispiel sind die Kopfdaten einer Rechnung mit Rechnungsnummer und Adressat. Felder hingegen ändern sich in der Regel für jeden Datensatz, typische Vertreter sind also z.B. die Postendaten einer Rechnung.

Im Designer werden Variablen stets außerhalb, Felder nur innerhalb des Berichtscontainers (des "Tabellenbereichs") angeboten und können auch nur dort verwendet werden. Die Trennung dient vor allem dazu, dem Endanwender das Leben leichter zu machen – wenn er ein Feld in den "Außenbereich" platzieren würde, wäre das Ergebnis je nach Druckreihenfolge entweder der Inhalt des Ersten oder Letzten Datensatzes.

Für beide Baustein-Typen gilt, dass sie hierarchisch angeordnet werden können – im Designer macht sich dies durch eine Ordnerstruktur bemerkbar. Die Datenbank-Tabellennamen werden von der Datenbindung automatisch berücksichtigt, so dass alle Daten der "Bestelldaten"-Tabelle in einem Ordner "Bestelldaten" dargestellt werden.

Eigene Daten können ebenfalls hierarchisch angeordnet werden, indem ein Punkt als Hierarchietrenner verwendet wird (also z.B. "Zusatzdaten.Benutzername"). Wie eigene Daten hinzugefügt werden können und wie die Anmeldeinformationen der Datenbindung beeinflusst werden können zeigt der Abschnitt "Datenbankunabhängige Inhalte".

Variablen und Felder bei Datenbindung

Wenn in einer 1:n hierarchisch verknüpften Datenstruktur wie z.B. "Rechnungskopf" und "Rechnungsposten" die Bestelldaten-Tabelle als Variablen, die Bestellposten hingegen als Felder angemeldet werden sollen, kann dies über die Eigenschaften DataMember und AutoMasterMode der Komponente erreicht werden:

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// Bestelldaten als Variablen
LL.DataMember = "Rechnungskopf";
LL.AutoMasterMode = LlAutoMasterMode.AsVariables;

LL.Design();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()

' Bestelldaten als Variablen
```

```
LL.DataMember = "Rechnungskopf"
LL.AutoMasterMode = LlAutoMasterMode.AsVariables

LL.Design()
LL.Dispose()
```

Zur Druckzeit wird in diesem Falle automatisch ein Seriendruck generiert, wenn also z.B. ein Rechnungsformular designed wurde, wird für jeden Datensatz aus der Rechnungskopf-Tabelle eine eigene Rechnung mit eigener Seitennummerierung, Summierung etc. erzeugt.

Datentypen

Variablen und Felder werden typisiert übergeben, d.h. je nach Inhalt in der Datenbank als Text, Zahl etc. Dies besorgt die Datenbindung in der Regel automatisch, eine explizite Übergabe des Typs ist nur dann notwendig, wenn zusätzlich eigene Daten übergeben werden. Auch dann wird meist schon der passende Datentyp vorgewählt (z.B. bei Übergabe von DateTime-Objekten).

Die folgende Tabelle zeigt die wichtigsten Datentypen.

Datentyp	Verwendung
LlFieldType.Text	Text.
LlFieldType.RTF	RTF-formatierter Text. Dieser Feldtyp kann im Designer direkt in einem RTF-Feld bzw. RTF-Objekt verwendet werden.
LlFieldType.Numeric LlFieldType.Numeric_Integer	Zahl. Die Datenbindung unterscheidet automatisch zwischen Gleitkommazahlen und Integer-Werten.
LlFieldType.Boolean	Logische Werte.
LlFieldType.Date	Datums- und Zeitwerte (DateTime).
LlFieldType.Drawing	Grafik. In der Regel wird der Dateiname angegeben, für Bitmaps und EMF-Dateien ist auch eine direkte Übergabe des Speicherhandles möglich. Die Datenbindung untersucht automatisch Byte[]-Felder auf Ihren Inhalt und meldet diese als Grafik an, wenn sich ein passendes Format findet.
LlFieldType,Barcode	Barcode. Barcodes können am einfachsten als Instanzen der LlBarcode-Klasse direkt in den Add-Methoden der Variables- und Fields-Eigenschaft übergeben werden.
LlFieldType.HTML	HTML. Der Inhalt der Variablen ist ein gültiger HTML-Stream.

3.3 Ereignisse

Die folgende Tabelle zeigt einige wichtige Ereignisse der ListLabel-Komponente. Eine vollständige Referenz findet sich in der Komponentenhilfe für .NET.

Event	Verwendung
AutoDefineField/ AutoDefineVariable	Diese Ereignisse werden für jedes Feld bzw. jede Variable vor der Anmeldung bei List & Label aufgerufen. Über die Event-Argumente kann der Feldname und Inhalt angepasst werden oder die Anmeldung komplett unterdrückt werden. Beispiele hierfür im Abschnitt "Datenbankunabhängige Inhalte".
AutoDefineNewPage	Dieses Ereignis wird zu Beginn jeder Seite aufgerufen. Wenn die Anwendung seitenspezifische Zusatzdaten benötigt, die nicht aus der Datenquelle selbst stammen, können diese in diesem Event per LL.Variables.Add() hinzugefügt werden. Beispiele hierfür im Abschnitt "Datenbankunabhängige Inhalte".
AutoDefineNewLine	Dieses Ereignis wird für jede Zeile aufgerufen. Wenn die Anwendung zeilenspezifische Zusatzdaten benötigt, die nicht aus der Datenquelle selbst stammen, können diese in diesem Event per LL.Fields.Add() hinzugefügt werden. Beispiele hierfür im Abschnitt "Datenbankunabhängige Inhalte".
DrawObject DrawPage DrawTableLine DrawTableField	Diese Ereignisse werden jeweils einmal vor und einmal nach dem Druck des zugehörigen Elements aufgerufen, also z.B. für jede Tabellenzelle (DrawTableField). Die Ereignisargumente enthalten ein Graphics-Objekt und das Rechteck der Ausgabe, so dass die Anwendung eigene Informationen zusätzlich selber ausgeben kann. Es kann sich hierbei um eine besondere Schattierung, einen "Demo"-Schriftzug oder eine komplette eigene Ausgabe handeln.
VariableHelpText	Dient zur Anzeige eines Hilfetexts für Variablen und Felder für den Endanwender im Designer.

3.4 Projekttypen

Je nach Berichtstyp stehen drei verschiedene Modi des Designers zur Verfügung. Welcher Modus verwendet wird, hängt vom Wert der Eigenschaft AutoProjectType ab.

Listen

Dies ist der Standard und entspricht dem Wert LlProject.List für die AutoProjectType Eigenschaft.

Typische Anwendungsfälle sind Rechnungen, Adresslisten, Auswertungen mit Charts und Kreuztabellen, mehrspaltige Listen, kurz alle Berichtstypen für die ein tabellarisches Element benötigt wird. Nur in diesem Modus steht der Berichtscontainer zur Verfügung (s. Abschnitt "Berichtscontainer").

Etiketten

Dieser Projekttyp entspricht dem Wert LlProject.Label für die AutoProjectType Eigenschaft.

Er wird für die Ausgabe von Etiketten verwendet. Da es hier keine tabellarischen Bereiche und auch keinen Berichtscontainer gibt, stehen lediglich Variablen, nicht aber Felder zur Verfügung (vgl. Abschnitt "Variablen, Felder und Datentypen").

Wenn die verwendete Datenquelle mehrere Tabellen enthält, z.B. Produkte, Kunden etc., kann über die Eigenschaft DataMember der Komponente die Quelltabelle für den Etikettendruck ausgewählt werden:

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// Produkte als Quelldaten
LL.DataMember = "Produkte";

// Etikett als Projekttyp wählen
LL.AutoProjectType = LlProject.Label;

LL.Design();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()

' Produkte als Quelldaten
LL.DataMember = "Produkte"

' Etikett als Projekttyp wählen
LL.AutoProjectType = LlProject.Label

LL.Design()
LL.Dispose()
```

Karteikarten

Dieser Projekttyp entspricht dem Wert LlProject.Card für die AutoProjectType Eigenschaft.

Karteikartenprojekte sind ein Spezialfall von Etikettenprojekten mit genau einem seitenfüllenden Etikett. Typische Anwendungsfälle sind der Karteikartendruck (z.B. alle Kundeninformationen auf einen Blick) oder Serienbriefe. Die Ansteuerung erfolgt genau analog zum Abschnitt "Etiketten", es gelten die gleichen Hinweise und Einschränkungen.

3.5 Verschiedene Drucker und Kopiendruck

List & Label bietet eine komfortable Unterstützung für die Aufteilung eines Berichts auf verschiedene Drucker oder die Ausgabe von Kopien mit "Kopie"-Vermerk. Das Beste daran – es handelt sich um reine Designer-Features, die automatisch von List & Label unterstützt werden.

Layoutbereiche

Die Layout-Bereiche dienen zur Aufteilung des Projekts auf mehrere Seitenbereiche mit unterschiedlichen Eigenschaften. Typische Anwendungsfälle sind z.B. unterschiedliche Drucker für erste Seite, Folgeseiten und letzte Seite. Weitere Anwendungsmöglichkeiten sind Mischung von Hoch- und Querformat innerhalb eines Berichts.

Demonstriert wird die Verwendung z.B. in der List & Label Beispielanwendung (im Startmenü direkt auf der Basisebene) unter Design > Erweiterte Beispiele > Mischung von Hoch- und Querformat.

Ausfertigungen und Kopien

Sowohl Ausfertigungen als auch Kopien dienen zur Ausgabe mehrerer Exemplare eines Berichts. Die Kopien sind dabei "echte" Hardwarekopien, d.h. hier wird der Drucker angewiesen, mehrere Exemplare der Ausgabe zu erstellen. Naturgemäß sind diese Exemplare untereinander alle identisch und werden mit den gleichen Druckereinstellungen erzeugt.

Wenn die Ausgaben unterschiedliche Eigenschaften haben sollen (z.B. Original aus Schacht 1, Kopie aus Schacht 2) oder ein "Kopie"-Wasserzeichen ausgegeben werden soll, kann dies über die Ausfertigungssteuerung erreicht werden. Hierfür wird im Designer die Eigenschaft "Anzahl der Ausfertigungen" auf einen Wert größer eins gesetzt. Dann steht für die Layoutbereiche die Funktion "IssueIndex" zur Verfügung, so dass z.B. ein Bereich mit der Bedingung "IssueIndex()==1" (Original) und ein weiterer mit der Bedingung "IssueIndex()==2" (Kopie) erstellt werden kann.

Die Objekte im Designer erhalten eine neue Eigenschaft "Darstellungsbedingung für Ausfertigungsdruck", mit der auf ganz ähnliche Weise der Wasserzeichendruck realisiert werden kann.

Demonstriert wird die Verwendung z.B. in der List & Label Beispielanwendung (im Startmenü direkt auf der Basisebene) unter **Design** > **Rechnung** > **Rechnung mit Ausfertigungsdruck**.

3.6 Designer anpassen und erweitern

Der Designer ist für die Anwendung keine "Black Box", sondern kann in vielen Bereichen beeinflusst werden. Neben dem Sperren von Funktionen und Menüpunkten können eigene Elemente hinzugefügt werden, die Berechnungen, Aktionen oder Ausgaben in die Funktionslogik verlegen.

Menüpunkte, Objekte und Funktionen sperren

Dreh- und Angelpunkt für die Designereinschränkung ist die DesignerWorkspace-Eigenschaft des ListLabel-Objekts. Diese bietet die in der folgenden Tabelle aufgelisteten Eigenschaften für die Designereinschränkung.

Eigenschaft	Funktion
ProhibitedActions	Diese Eigenschaft dient dazu, einzelne Menüpunkte aus dem Designer zu entfernen.
ProhibitedFunctions	Diese Eigenschaft dient dazu, einzelne Funktionen aus dem Designer zu entfernen.
ReadOnlyObjects	Diese Eigenschaft dient dazu, Objekte im Designer gegen Bearbeitung zu sperren. Die Objekte sind weiterhin sichtbar, können aber innerhalb des Designers nicht bearbeitet oder gelöscht werden.

Das folgende Beispiel zeigt, wie der Designer so angepasst werden kann, dass kein neues Projekt mehr angelegt werden kann. Zudem wird die Funktion "ProjectPath$" entfernt und das Objekt "Demo" gegen Bearbeitung gesperrt.

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// Designer einschränken
LL.DesignerWorkspace.ProhibitedActions.Add(LlDesignerAction.FileNew);
LL.DesignerWorkspace.ProhibitedFunctions.Add("ProjectPath$");
LL.DesignerWorkspace.ReadOnlyObjects.Add("Demo");

LL.Design();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
```

```
LL.DataSource = CreateDataSet()

' Designer einschränken
LL.DesignerWorkspace.ProhibitedActions.Add(LlDesignerAction.FileNew)
LL.DesignerWorkspace.ProhibitedFunctions.Add("ProjectPath$")
LL.DesignerWorkspace.ReadOnlyObjects.Add("Demo")

LL.Design()
LL.Dispose()
```

Designer erweitern

Der Designer kann um eigene Funktionen, Objekte und Aktionen erweitert werden.

Eigene Funktionen können dafür verwendet werden, komplexere Berechnungen in die Applikation zu verlegen bzw. Funktionalitäten nachzurüsten, die im Standardumfang des Designers nicht vorhanden sind.

Ein Beispiel für das Hinzufügen einer eigenen Funktion findet sich im Abschnitt "Designer um eigene Funktion erweitern".

Beispiele für eigene Objekte oder Aktionen sowie eine weitere eigene Funktion zeigt das "Designer Erweiterungs-Beispiel", das unter "Sonstiges" im .NET-Beispielbereich im Startmenü zu finden ist.

3.7 Objekte im Designer

Einige Objekte im Designer dienen nur der grafischen Gestaltung (z.B. Linie, Rechteck, Ellipse). Die meisten anderen Objekte interagieren aber mit den zur Verfügung gestellten Daten. Hierfür stehen z.T. eigene Datentypen zur Verfügung oder es gibt Konvertierungsfunktionen, die die Umwandlung von Inhalten erlauben, so dass diese im jeweiligen Objekt verwendet werden können. Die folgende Aufstellung gibt einen Überblick über die am häufigsten verwendeten Objekte, die zugehörigen Datentypen und Designerfunktionen zur Umwandlung von Inhalten.

Die Hinweise für die Einzelobjekte gelten in gleicher oder ähnlicher Weise auch für (Bild-, Barcode-, usw.) Spalten in Tabellenelementen.

Text

Ein Textobjekt besteht aus mehreren Absätzen. Jeder dieser Absätze hat einen eigenen Inhalt. Dies kann entweder direkt eine Variable sein oder alternativ eine Formel, die mehrere Dateninhalte kombiniert. Für die Darstellung einzelner Variablen ist in der Regel keine spezielle Konvertierung notwendig. Sollen mehrere Variablen unterschiedlichen Typs (s. Abschnitt "Datentypen") innerhalb einer Formel kombiniert werden, müssen die einzelnen Bestandteile auf den gleichen Datentypen (z.B. Zeichenkette) konvertiert werden. Ein Beispiel für die Kombination von Zahlen und Zeichenketten wäre:

```
"Gesamtsumme: "+Str$(Sum(Artikel.Preis),0,2)
```

Die folgende Tabelle listet einige der Konvertierungsfunktionen auf, die in diesem Zusammenhang häufiger gebraucht werden.

Von/In	Datum	Zahl	Zeichnung	Barcode	Text
Datum	-	DateToJulian	-	-	Date$
Zahl	JulianToDate	-	-	Barcode(Str$)	FStr$ Str$
Zeichnung	-	-	-	-	Drawing$
Barcode	-	Val(Barcode$)		-	Barcode$
Text	Date	Val	Drawing	Barcode	-

Bild

Der Inhalt eines Bildobjekts wird über die Eigenschaftsliste bestimmt. Die Eigenschaft **Datenquelle** bietet die drei Werte **Dateiname**, **Formel** und **Variable** an.

- Die Einstellung **Dateiname** dient zur Verwendung einer festen Datei wie z.B. eines Firmenlogos. Wenn die Datei nicht mitgeliefert werden soll, kann sie auch direkt in den Bericht eingebettet werden, der Dateiauswahldialog bietet eine entsprechende Option.
- Über **Formel** kann der Inhalt über eine Zeichenkette festgelegt werden, die einen Pfad enthält. Die dafür benötigte Funktion ist "Drawing".
- Über **Variable** können schon als Bild übergebene Inhalte dargestellt werden (s. Abschnitt "Datentypen").

Barcode

Der Inhalt eines Barcodeobjekts wird über einen Dialog bestimmt. Dieser bietet als Datenquelle die drei Optionen **Text**, **Formel** und **Variable** an.

- Die Einstellung **Text** dient zur Verwendung eines festen Texts/Inhalts im Barcode. Zusätzlich zum Inhalt können der Typ und – z.B. bei 2D-Barcodes – weitere Eigenschaften zur Fehlerkorrektur oder Codierung eingestellt werden.
- Über **Formel** kann der Inhalt über eine Zeichenkette festgelegt werden, die den Barcodeinhalt enthält. Die dafür benötigte Funktion ist "Barcode".
- Über **Variable** können schon als Barcode übergebene Inhalte dargestellt werden (s. Abschnitt "Datentypen").

RTF-Text

Der Inhalt eines RTF-Textobjekts wird über einen Dialog bestimmt. Dieser bietet unter **Quelle** die Optionen **(freier Text)** oder eine Auswahl evtl. übergebener RTF-Variablen (s.u.)

- Die Einstellung **(freier Text)** dient zur Verwendung eines festen Texts/Inhalts im RTF-Objekt. Innerhalb des Objekts kann an jeder Stelle ein Dateninhalt verwendet werden (z.B. für personalisierte Serienbriefe), indem in der Toolleiste auf das Formelsymbol geklickt wird.
- Über die Auswahl einer Variablen können schon als RTF übergebene Inhalte dargestellt werden (s. Abschnitt "Datentypen").

HTML

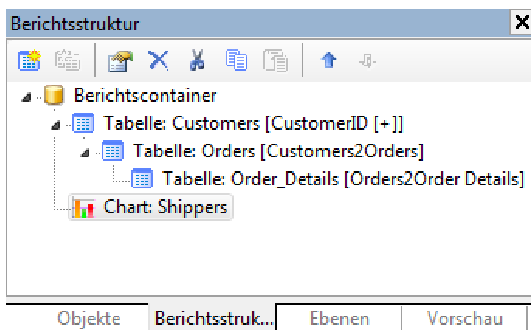
Der Inhalt eines HTML-Objekts wird über einen Dialog bestimmt. Dieser bietet als Datenquelle die drei Optionen **Dateiname**, **URL** und **Variable** an.

- Die Einstellung **Dateiname** dient zur Verwendung einer festen HTML-Datei.
- Über **URL** kann eine URL angegeben werden, von der der HTML-Inhalt heruntergeladen werden soll.
- Über **Variable** können schon als HTML-Stream übergebene Inhalte dargestellt werden (s. Abschnitt "Datentypen").

3.8 Berichtscontainer

Der Berichtscontainer ist das zentrale Element für Listen-Projekte. Er erlaubt die Darstellung von tabellarischen Daten (auch mehrspaltig oder verschachtelt), Statistiken und Charts sowie Kreuztabellen. Die Daten können auch mehrfach in unterschiedlicher Form ausgegeben werden – z.B. zunächst eine grafische Auswertung der Verkäufe über die Jahre und anschließend eine detaillierte tabellarische Aufstellung.

Für die Inhalte des Containers steht ein eigenes Toolfenster, die Berichtsstruktur, innerhalb des Designers zur Verfügung. Über dieses Fenster können neue Inhalte hinzugefügt oder bestehende bearbeitet werden. Das Fenster dient als eine Art "Drehbuch" für den Bericht, da darin exakt der Ablauf der einzelnen Berichtselemente zu sehen ist.



Das Berichtsstruktur-Toolfenster im Designer

Damit der Berichtscontainer zur Verfügung steht, muss ein Datenprovider (vgl. Abschnitt "Datenprovider") als Datenquelle verwendet werden. Prinzipiell ist es auch möglich, den gesamten Druck über die low-level API-Funktionen des LlCore-Objekts selbst durchzuführen, dies ist aber nicht die empfohlene Vorgehensweise, da dann viele Features (Designervorschau, Drilldown, Berichtscontainer,...) eigens unterstützt werden müssen. Viel sinnvoller ist es, im Zweifel einen eigenen Datenprovider zu schreiben. Hinweise dazu im Abschnitt "Datenbankunabhängige Inhalte".

Alle mitgelieferten Listen-Beispiele für das .NET Framework verwenden den Berichtscontainer und liefern so Anschauungsmaterial für die verschiedenen Einsatzzwecke.

Eine detaillierte Beschreibung für die Verwendung dieses Elements findet sich im Designerhandbuch im Abschnitt "Berichtscontainer einfügen".

3.9 Objektmodell

Während der Designer eine sehr komfortable und mächtige Oberfläche zur Bearbeitung der Projektdateien bietet, kann es oft auch gewünscht sein, Objekt- oder Berichtseigenschaften direkt per Code zu bestimmen. So kann die Anwendung z.B. dem Anwender einen vorgeschalteten Dialog zur Datenvorauswahl anbieten und den Designer bereits mit einem so vorbereiteten Projekt öffnen. Ein Beispiel hierfür zeigt das Beispiel "Einfaches DOM-Beispiel", das unter "Sonstiges" im .NET-Beispielbereich im Startmenü zu finden ist.

Der Zugriff auf das Objektmodell ist erst ab der Professional Edition möglich.

Die folgende Tabelle listet die wichtigsten Klassen und Eigenschaften aus dem Namespace combit.ListLabel15.Dom auf.

Klasse	Funktion
ProjectList ProjectLabel ProjectCard	Die eigentlichen Projektklassen. Diese stellen das Wurzelelement des Projektes dar. Schlüsselmethoden sind Open, Save und Close.
<Projekt>.Objects	Eine Auflistung der Objekte innerhalb des Projekts. Die Objekte sind von ObjectBase abgeleitet und verfügen jeweils über eigene Eigenschaften und ggf. Auflistungen (z.B. Textabsätze).
<Projekt>.Regions	Eine Auflistung der Layoutbereiche des Projekts. Hierüber kann z.B. eine seitenabhängige Druckersteuerung realisiert werden. Weitere Informationen finden sich im Abschnitt "Layoutbereiche".
ObjectText	Repräsentiert ein Textobjekt. Schlüsseleigenschaft ist Paragraphs, der eigentliche Inhalt des Texts.
ObjectReportContainer	Repräsentiert einen Berichtscontainer. Schlüsseleigen-

Klasse	Funktion
	schaft ist SubItems, der eigentliche Inhalt des Berichtscontainers.
SubItemTable	Repräsentiert eine Tabelle innerhalb des Berichtscontainers. Diese besteht aus verschiedenen Zeilenbereichen (Lines-Eigenschaft), die wiederum verschiedene Spalten (Columns-Eigenschaft einer Zeile) haben.

Innerhalb der einzelnen Klassen findet sich über die IntelliSense-Unterstützung recht einfach die gesuchte Eigenschaft. Eine vollständige Referenz über alle Klassen liefert die Komponentenhilfe für .NET.

3.10 Fehlerhandling mit Exceptions

List & Label definiert eine Reihe eigener Exceptions, die alle von der gemeinsamen Basisklasse ListLabelException abgeleitet sind und so zentral abgefangen werden können. Wenn die Applikation ein eigenes Exception-Handling vornehmen soll, können Aufrufe an List & Label in einen Exceptionhandler gekapselt werden:

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

try
{
    LL.Design();
}
catch (ListLabelException ex)
{
    MessageBox.Show(ex.Message);
}
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()

Try
	LL.Design()
Catch ex As ListLabelException
	MessageBox.Show(ex.Message)
End Try
LL.Dispose()
```

Die Message-Eigenschaft der Exception-Klasse enthält einen Fehlertext, der – wenn ein entsprechendes Sprachkit vorhanden ist – in der Regel auch lokalisiert ist und so direkt dem Anwender angezeigt werden kann.

Eine vollständige Referenz über alle Exception-Klassen liefert die Komponentenhilfe für .NET.

3.11 Debugging

Probleme die auf dem Entwicklerrechner auftreten können meist leicht gefunden werden – hier kann direkt mit den üblichen Features der Entwicklungsumgebung gearbeitet werden und ein Problem so recht schnell eingegrenzt werden. Der erste Schritt besteht darin, eventuell auftretende Exceptions abzufangen und deren Ursache zu überprüfen (vgl. "Abschnitt Fehlerhandling mit Exceptions").

Als Entwicklungskomponente wird List & Label aber natürlich unter einer Vielzahl verschiedener Konstellationen bei den Endanwendern ausgeführt. Um Probleme dort möglichst einfach zu finden, steht ein eigenes Debug-Tool zur Verfügung, das bei selten oder nur auf bestimmten Systemen auftretenden Problemen eine Protokollierungsfunktion bietet, mit deren Hilfe Probleme auch auf Systemen ohne Debugger untersucht werden können.

Natürlich kann die Logging-Funktion auch auf dem Entwicklerrechner genutzt werden und bietet auch dort die Möglichkeit, sämtliche Aufrufe und Rückgabewerte schnell auf einen Blick zu prüfen.

Protokolldatei anfertigen

Tritt ein Problem nur auf einem Kundensystem auf, sollte auf diesem zunächst eine Protokolldatei erstellt werden. Hierzu dient das Tool Debwin3, welches im "Tools"-Verzeichnis der List & Label-Installation installiert wird.

Debwin3 muss vor der Applikation gestartet werden. Über Logging > Force Debug Mode wird die Protokollierung erzwungen. Wenn anschließend die Applikation gestartet wird, werden sämtliche Aufrufe an die Komponente mit ihren Rückgabewerten sowie einige Zusatzinformationen zu Modulversionen, Betriebssystem etc. protokolliert.

Jede unter .NET geworfene Exception entspricht im Protokoll einem negativen Rückgabewert einer Funktion. Im Protokoll finden sich meist weitere hilfreiche Informationen, eine typische Ausgabe könnte wie folgt aussehen.

```
CMLL15  : 12:30:02.082 00000df0/02 3  LlSelectFileDlgTitleEx(2,0X001310BE,
'(NULL)',0x00008002,0X0344F7C8,260,00000000)
CMLL15  : 12:30:03.672 00000df0/02 4  =-99 (Der Benutzer hat den Vorgang
abgebrochen.) -> '*.lst'
```

Man sieht zunächst die DLL, die die Ausgabe erzeugt hat, eine Timestamp für die Ausgabe, die Thread-ID des ausgebenden Threads, eine fortlaufende Nummer sowie den eigentlichen Aufruf mit allen Parametern. In der Folgezeile erfolgt dann die

Rückgabe eines Fehlercodes (-99) und eine Erklärung dafür – in diesem Falle hat der Benutzer den Dateiauswahldialog mit "Abbrechen" beendet.

Soll die Anwendung ohne Hilfe von Debwin3 Debugprotokolle erstellen, kann dies z.B. über die Konfigurationsdatei der Anwendung erreicht werden. Eine Protokollierung kann darin wie folgt erzwungen werden:

```
<configuration>
 <appSettings>
  <add key="ListLabel DebugLogFilePath"
    value="c:\users\public\debug.log "/>
  <add key="ListLabel EnableDebug" value="1"/>
 </appSettings>
</configuration>
```

4. Beispiele

Die Beispiele in diesem Abschnitt zeigen, wie einige typische Aufgabenstellungen gelöst werden. Der Code kann als Kopiervorlage für eigene Erweiterungen dienen.

Aus Übersichtlichkeitsgründen wird auf die übliche Fehlerbehandlung verzichtet. Alle Exceptions werden somit direkt in der Entwicklungsumgebung aufgefangen. Für "echte" Applikationen empfehlen wir ein Exception-Handling wie im Abschnitt "Fehlerhandling mit Exceptions" beschrieben.

4.1 Einfaches Etikett

Um ein Etikett auszugeben, wird der Projekttyp LlProject.Label benötigt. Wenn eine Datenquelle mit mehreren Tabellen wie z.B. ein DataSet angebunden wird, kann über die Eigenschaft DataMember ausgewählt werden, welche Daten im Etikett bereitgestellt werden sollen.

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// Produkte als Quelldaten
LL.DataMember = "Produkte";

// Etikett als Projekttyp wählen
LL.AutoProjectType = LlProject.Label;

// Designer aufrufen
LL.Design();

// Drucken
LL.Print();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()

' Produkte als Quelldaten
LL.DataMember = "Produkte"

' Etikett als Projekttyp wählen
LL.AutoProjectType = LlProject.Label

' Designer aufrufen
LL.Design()

'Drucken
```

```
LL.Print()
LL.Dispose()
```

4.2 Einfache Liste

Der Druck und das Design von einfachen Listen ist der "Standardfall" und kann schon mit wenigen Codezeilen gestartet werden:

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// Designer aufrufen
LL.Design();

// Drucken
LL.Print();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()

' Designer aufrufen
LL.Design()

'Drucken
LL.Print()
LL.Dispose()
```

4.3 Sammelrechnung

Eine Sammelrechnung ist ein impliziter Seriendruck. Die Kopf- oder Elterndaten enthalten für jeden Beleg einen Datensatz, der 1:n mit den Detail- oder Kinddaten verknüpft ist. Um einen solchen Beleg zu designen und zu drucken, muss List & Label die Elterntabelle über die DataMember-Eigenschaft bekannt gegeben werden. Zudem muss die AutoMasterMode-Eigenschaft auf AsVariables gesetzt werden, wie im folgenden Beispiel gezeigt:

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// Bestelldaten als Variablen
LL.DataMember = "Rechungskopf";
LL.AutoMasterMode = LlAutoMasterMode.AsVariables;
```

```
// Designer aufrufen
LL.Design();

// Drucken
LL.Print();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()

' Bestelldaten als Variablen
LL.DataMember = "Rechnungskopf"
LL.AutoMasterMode = LlAutoMasterMode.AsVariables

' Designer aufrufen
LL.Design()

'Drucken
LL.Print()
LL.Dispose();
```

4.4 Karteikarte mit einfachen Platzhaltern drucken

Der Druck eines ganzseitigen Projektes, das einfach nur an verschiedenen Stellen durch die Applikation bestimmte Platzhalter enthält ist am einfachsten über das Binden an ein passendes Objekt zu bewerkstelligen:

C#:

```
public class DataSource
{
    public string Text1 { get; set; }
    public double Number1 { get; set; }
    ...
}

// Datenquelle vorbereiten
object dataSource = new DataSource { Text1 = "Test", Number1 = 1.234 };
ListLabel LL = new ListLabel();
LL.DataSource = new ObjectDataProvider(dataSource);
LL.AutoProjectType = LlProject.Card;

// Designer aufrufen
LL.Design();

// Drucken
LL.Print();
LL.Dispose();
```

VB.NET:

```
Public Class DataSource
	Dim _text1 As String
	Dim _number As Double

	Public Property Text1() As String
		Get
			Return _text1
		End Get
		Set(ByVal value As String)
			_text1 = value
		End Set
	End Property
	Public Property Number1() As Double
		Get
			Return _number
		End Get
		Set(ByVal value As Double)
			_number = value
		End Set
	End Property
End Class

' Datenquelle vorbereiten
Dim dataSource As Object = New DataSource()
dataSource.Text1 = "Test"
dataSource.Number1 = 1.234
Dim LL As New ListLabel()
LL.DataSource = New ObjectDataProvider(dataSource)
LL.AutoProjectType = LlProject.Card

' Designer aufrufen
LL.Design()

' Drucken
LL.Print()
LL.Dispose()
```

4.5 Unterberichte

Die Strukturierung von Berichten mit Hilfe des Berichtscontainers ist ein reines Designerfeature. Insofern unterscheidet sich die Ansteuerung nicht von der für "normale" Listen, wie im Abschnitt "Einfache Liste" gezeigt.

Für die Ausgabe von Untertabellen wird vorausgesetzt, dass Eltern- und Kinddaten durch eine Relation miteinander verknüpft sind. Dann kann im Berichtscontainer zunächst ein Tabellenelement für die Elterntabelle gestaltet werden. Im nächsten Schritt kann über die Funktionsleiste im Berichtsstrukturfenster ein Unterelement mit den Kinddaten eingefügt werden.

Zur Druckzeit wird dann für jeden Datensatz der Elterntabelle automatisch der passende Kind-Unterbericht eingefügt. Demonstriert wird die Verwendung z.B. in der List & Label Beispielanwendung (im Startmenü direkt auf der Basisebene) unter Design > Erweiterte Beispiele > Unterberichte und Relationen.

4.6 Charts

Auch die Diagrammfunktion wird durch den Berichtscontainer automatisch unterstützt. Die Ansteuerung erfolgt also auch hier wie im Abschnitt "Einfache Liste" gezeigt.

Die List & Label Beispielanwendung (im Startmenü direkt auf der Basisebene) enthält unter Design > Erweiterte Beispiele eine Vielzahl von verschiedenen Chart-Beispielen.

4.7 Kreuztabellen

Wenig überraschend werden Kreuztabellen analog den Hinweisen im Abschnitt "Einfache Liste" angesteuert.

Die List & Label Beispielanwendung (im Startmenü direkt auf der Basisebene) enthält unter Design > Erweiterte Beispiele eine Vielzahl von verschiedenen Kreuztabellen-Beispielen.

4.8 Datenbankunabhängige Inhalte

Nicht immer liegen alle Daten in einer Datenbank oder einem DataSet vor. So kann es erwünscht sein, zusätzlich zu den Daten aus der Datenquelle weitere Daten wie z.B. den Benutzernamen innerhalb der Applikation, den Projektnamen oder ähnliche Informationen auszugeben. In anderen Fällen scheint auf den ersten Blick kein passender Datenprovider in Sicht zu sein. Diese Fälle werden in den folgenden Abschnitten betrachtet.

Zusätzliche Inhalte übergeben

Um nur einige wenige Variablen oder Felder zusätzlich zu den Daten der Datenbindung hinzuzufügen, genügt die Behandlung zweier Ereignisse. Zusätzliche Variablen können innerhalb des AutoDefineNewPage-Ereignisses per LL.Variables.Add, Felder innerhalb des AutoDefineNewLine-Ereignisses per LL.Fields.Add übergeben werden.

Das folgende Beispiel zeigt beide Ansätze:

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

…

// Ereignisbehandlung für eigene Informationen hinzufügen
```

```
LL.AutoDefineNewLine += new AutoDefineNewLineHandler(LL_AutoDefineNewLine);
LL.AutoDefineNewPage += new AutoDefineNewPageHandler(LL_AutoDefineNewPage);


// Designer aufrufen
LL.Design();

// Drucken
LL.Print();
LL.Dispose();

…

void LL_AutoDefineNewLine(object sender, AutoDefineNewLineEventArgs e)
{
    // ggf. zum nächsten Datensatz wechseln, wenn dies notwendig ist
    // GetCurrentFieldValue ist eine Funktion Ihrer Applikation, die
    // den Inhalt des Datenfeldes liefert.
    LL.Fields.Add("Zusatzdaten.Zusatzfeld", GetCurrentFieldValue());
}

void LL_AutoDefineNewPage(object sender, AutoDefineNewPageEventArgs e)
{
    // Zusätzliche Variablen anmelden
    LL.Variables.Add("Zusatzdaten.Benutzername", GetCurrentUserName());
    LL.Variables.Add("Zusatzdaten.Projektname", GetCurrentProjectName());
}
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()
…

' Designer aufrufen
LL.Design()

' Drucken
LL.Print()
LL.Dispose()

…

Sub LL_AutoDefineNewLine(sender As Object, e As AutoDefineNewLineEventArgs)_
Handles LL.AutoDefineNewLine
    ' ggf. zum nächsten Datensatz wechseln, wenn dies notwendig ist
    ' GetCurrentFieldValue ist eine Funktion Ihrer Applikation, die
    ' den Inhalt des Datenfeldes liefert.
    LL.Fields.Add("Zusatzdaten.Zusatzfeld", GetCurrentFieldValue())
End Sub
```

```
Sub LL_AutoDefineNewPage(sender As Object, e As AutoDefineNewPageEventArgs)_
Handles LL.AutoDefineNewPage
    ' Zusätzliche Variablen anmelden
    LL.Variables.Add("Zusatzdaten.Benutzername", GetCurrentUserName())
    LL.Variables.Add("Zusatzdaten.Projektname", GetCurrentProjectName())
End Sub
```

Daten aus der Datenbindung unterdrücken

Einzelne, nicht benötigte Felder oder Variablen (z.B. ID-Felder die im Druck nicht benötigt werden) können mit Hilfe der AutoDefineField- und AutoDefineVariable-Ereignisse unterdrückt werden.

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// Ereignisbehandlung für Feldunterdrückung hinzufügen
LL.AutoDefineField += new AutoDefineFieldHandler(LL_AutoDefineField);

// Designer aufrufen
LL.Design();

// Drucken
LL.Print();
LL.Dispose();

…

void LL_AutoDefineField(object sender, AutoDefineElementEventArgs e)
{
    if (e.Name.EndsWith("ID"))
        e.Suppress = true;
}
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()

' Designer aufrufen
LL.Design()

' Drucken
LL.Print()
LL.Dispose()

…

Sub LL_AutoDefineField(sender As Object, e As AutoDefineElementEventArgs)_
Handles LL.AutoDefineNewField
```

```
        If e.Name.EndsWith("ID") Then
                e.Suppress = True
        End If
End Sub
```

Vollständig eigene Datenstrukturen/-inhalte

Für augenscheinlich nicht direkt unterstützte Dateninhalte findet sich meist trotzdem ein passender Provider. Businessdaten aus Anwendungen können in der Regel über den Objektdatenprovider übergeben werden, liegen die Daten in kommaseparierter Form vor kann der Datenprovider aus dem "Dataprovider"-Beispiel verwendet werden. Viele andere Datenquellen unterstützen die Serialisierung nach XML, so dass dann der XmlDataProvider verwendet werden kann.

Natürlich kann aber auch eine eigene Klasse verwendet werden, die die DataProvider-Schnittstelle unterstützt. Ein guter Startpunkt hierfür ist das DataProvider-Beispiel, das einen einfachen CSV-Datenprovider demonstriert. Oft kann auch eine der vorhandenen Klassen als Basisklasse verwendet werden. Wenn z.B. eine Datenquelle angebunden werden soll, die die IDbConnection-Schnittstelle implementiert, kann von DbConnectionDataProvider geerbt werden. Hier muss dann lediglich die Init-Methode überschrieben werden, in der die verfügbaren Tabellen und Relationen bereitgestellt werden müssen. In der Komponentenhilfe für .NET findet sich ein Beispiel, wie dies z.B. für SQL-Serverdaten im SqlConnectionDataProvider gemacht wird. Die meisten Datenbanksysteme stellen ähnliche Mechanismen zur Verfügung.

4.9 Export

Die Exportformate lassen sich per Code vollständig "fernsteuern", so dass keine Benutzeraktion mehr notwendig ist. Zudem kann die Auswahl der Formate so eingeschränkt werden, wie es für den jeweiligen Bericht notwendig oder gewünscht ist.

Export ohne Benutzerinteraktion

Damit der Export im Hintergrund erfolgen kann, muss zumindest das Format, der Pfad und der Dateiname für die Erzeugung bei List & Label angemeldet werden. Die Ansteuerung erfolgt über die ExportOptions-Eigenschaft der ListLabel-Komponente. Zudem wird die Projektdatei für den Druck vorgegeben und etwaige Dialoge werden unterdrückt.

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// Projektdatei angeben, Dialoge unterdrücken
LL.AutoDesignerFile = "<Projektdateiname mit Pfad>";
LL.AutoShowSelectFile = false;
```

```
LL.AutoShowPrintOptions = false;

// Ziel und Pfad (hier: PDF)
LL.ExportOptions.Add(LlExportOption.ExportTarget, "PDF");
LL.ExportOptions.Add(LlExportOption.ExportFile, "<Zieldateiname>");
LL.ExportOptions.Add(LlExportOption.ExportPath, "<Zielpfad>");

// Dateiauswahldialog unterdrücken
LL.ExportOptions.Add(LlExportOption.ExportQuiet, "1");

// Ergebnis anzeigen
LL.ExportOptions.Add(LlExportOption.ExportShowResult, "1");

// Export starten
LL.Print();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()

' Projektdatei angeben, Dialoge unterdrücken
LL.AutoDesignerFile = "<Projektdateiname mit Pfad>"
LL.AutoShowSelectFile = False
LL.AutoShowPrintOptions = False

' Ziel und Pfad (hier: PDF)
LL.ExportOptions.Add(LlExportOption.ExportTarget, "PDF")
LL.ExportOptions.Add(LlExportOption.ExportFile, "<Zieldateiname>")
LL.ExportOptions.Add(LlExportOption.ExportPath, "<Zielpfad>")

' Dateiauswahldialog unterdrücken
LL.ExportOptions.Add(LlExportOption.ExportQuiet, "1")

' Ergebnis anzeigen
LL.ExportOptions.Add(LlExportOption.ExportShowResult, "1")

' Export starten
LL.Print()
LL.Dispose()
```

Einschränkung von Exportformaten

Wenn dem Endanwender nur einzelne Exportformate erlaubt sein sollen, kann die Liste der Formate auf genau diese eingeschränkt werden. Dies ist über das Setzen der Option LlOptionString.Exports_Allowed möglich. Eine Liste der verfügbaren Formate findet sich im Abschnitt "Export".

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// nur PDF und Vorschau erlauben
LL.Core.LlSetOptionString(LlOptionString.Exports_Allowed, "PDF;PRV");

// Drucken
LL.Print();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()

' nur PDF und Vorschau erlauben
LL.Core.LlSetOptionString(LlOptionString.Exports_Allowed, "PDF;PRV")

' Drucken
LL.Print()
LL.Dispose()
```

4.10 Designer um eigene Funktion erweitern

Das folgende Beispiel zeigt, wie eine Funktion hinzugefügt werden kann, die es ermöglicht, den Wert eines Registrierungsschlüssels innerhalb eines Berichts abzufragen. Das Ergebnis der Funktion könnte dann z.B. in Darstellungsbedingungen für Objekte verwendet werden. Natürlich können die Eigenschaften der DesignerFunction-Klasse alternativ auch direkt im Eigenschaften-Fenster der Entwicklungsumgebung angelegt werden.

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// Funktion initialisieren
DesignerFunction RegQuery = new DesignerFunction();
RegQuery.FunctionName = "RegQuery";
RegQuery.GroupName = "Registrierung";
RegQuery.MinimalParameters = 1;
RegQuery.MaximumParameters = 1;
RegQuery.ResultType = LlParamType.String
RegQuery.EvaluateFunction += new
EvaluateFunctionHandler(RegQuery_EvaluateFunction);

// Funktion hinzufügen
LL.DesignerFunctions.Add(RegQuery);
```

```
LL.Design();
LL.Dispose();

…

void RegQuery_EvaluateFunction(object sender, EvaluateFunctionEventArgs e)
{
    // Registrierungsschlüssel auslesen
    RegistryKey key = Registry.CurrentUser.OpenSubKey(@"Software\combit\");
    e.ResultValue = key.GetValue(e.Parameter1.ToString()).ToString();
}
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()

' Funktion initialisieren
Dim RegQuery As New DesignerFunction()
RegQuery.FunctionName = "RegQuery"
RegQuery.GroupName = "Registrierung"
RegQuery.MinimalParameters = 1
RegQuery.MaximumParameters = 1
RegQuery.ResultType = LlParamType.String

' Funktion hinzufügen
LL.DesignerFunctions.Add(RegQuery)

LL.Design()
LL.Dispose()

…

Sub RegQuery_EvaluateFunction(sender As Object,
      e As EvaluateFunctionEventArgs) Handles RegQuery.EvaluateFunction
    ' Registrierungsschlüssel auslesen
    Dim key As RegistryKey
    RegistryKey = Registry.CurrentUser.OpenSubKey("Software\combit\")
    e.ResultValue = key.GetValue(e.Parameter1.ToString()).ToString()
End Sub
```

4.11 Vorschaudateien zusammenfügen und konvertieren

Das Vorschauformat kann als Ausgangsformat verwendet werden, wenn z.B. mehrere Berichte zu einem zusammengefügt werden sollen oder neben einem direkten Ausdruck auch eine Archivierung als PDF gewünscht wird. Das folgende Beispiel zeigt einige Möglichkeiten der PreviewFile-Klasse.

C#:

```
// Vorschaudateien öffnen, Deckblatt mit Schreibzugriff
```

```
PreviewFile cover = new PreviewFile(@"<Pfad>\deckblatt.ll", false);
PreviewFile report = new PreviewFile(@"<Pfad>\report.ll", true);

// Bericht an Deckblatt anhängen
cover.Append(report);

// Gesamtbericht drucken
cover.Print();

// Bericht als PDF konvertieren
cover.ConvertTo(@"<Pfad>\report.pdf");

// Vorschaudateien freigeben
report.Dispose();
cover.Dispose();
```

VB.NET:

```
' Vorschaudateien öffnen, Deckblatt mit Schreibzugriff
Dim cover As New PreviewFile("<Pfad>\deckblatt.ll", False)
Dim report As New PreviewFile("<Pfad>\report.ll", True)

' Bericht an Deckblatt anhängen
cover.Append(report)

' Gesamtbericht drucken
cover.Print()

' Bericht als PDF konvertieren
cover.ConvertTo("<Pfad>\report.pdf")

' Vorschaudateien freigeben
report.Dispose()
cover.Dispose()
```

4.12 eMail-Versand

Der eMail-Versand kann ebenfalls über die Liste der Exportoptionen angesteuert werden (vgl. Abschnitt "Export ohne Benutzerinteraktion"), wenn Export und Versand in einem Arbeitsgang erfolgen sollen. Ein Beispiel hierfür zeigt das Export-Beispiel, das unter "Sonstiges" im .NET-Beispielbereich im Startmenü zu finden ist.

Unabhängig von einem vorherigen Export ist es aber über die MailJob-Klasse auch möglich, beliebige Dateien per eMail zu versenden. Dies ist insbesondere dann interessant, wenn aus einer Vorschaudatei als Quelle z.B. eine PDF-Datei generiert wird (vgl. Abschnitt "Vorschaudateien zusammenfügen und konvertieren") und diese versendet werden soll.

C#:

```
// Mailjob instanzieren
```

```
MailJob mailJob = new MailJob();

// Optionen setzen
mailJob.AttachmentList.Add(@"<Pfad>\report.pdf");
mailJob.To = "info@combit.net";
mailJob.Subject = "Hier kommt der Report";
mailJob.Body = "Bitte sehen Sie sich das Attachment an.";
mailJob.Provider = "XMAPI";
mailJob.ShowDialog = true;

// eMail versenden
mailJob.Send();
mailJob.Dispose();
```

VB.NET:

```
' Mailjob instanzieren
Dim mailJob As New MailJob()

' Optionen setzen
mailJob.AttachmentList.Add("<Pfad>\report.pdf")
mailJob.To = "info@combit.net"
mailJob.Subject = "Hier kommt der Report"
mailJob.Body = "Bitte sehen Sie sich das Attachment an."
mailJob.Provider = "XMAPI"
mailJob.ShowDialog = True

' eMail versenden
mailJob.Send()
mailJob.Dispose()
```

4.13 Druck im Netzwerk

Beim Druck im Netzwerk gilt es, zwei Dinge zu beachten:

- Vorschaudateien werden in der Regel mit dem Namen der Projektdatei und der Endung "LL" im gleichen Verzeichnis wie die Projektdatei angelegt. Wenn also zwei Nutzer die gleiche Datei auf die Vorschau drucken wollen, wird der zweite Anwender eine Fehlermeldung erhalten. Dies kann durch die Verwendung von LlPreviewSetTempPath verhindert werden (s. Beispiel unten).
- Ähnliches gilt für die Druckereinstellungsdateien. Auch diese werden – mit der aktuell gewählten Endung – im Verzeichnis der Projektdatei gesucht bzw. angelegt. Hier sollte LlSetPrinterDefaultsDir verwendet werden.

C#:

```
ListLabel LL = new ListLabel();
LL.DataSource = CreateDataSet();

// lokalen Temporärpfad setzen
```

```
LL.Core.LlPreviewSetTempPath(Path.GetTempPath());

// Druckoptionen sollten in benutzerspezifischem Unterverzeichnis
// abgelegt werden, damit Änderungen dauerhaft übernommen werden
LL.Core.LlSetPrinterDefaultsDir(<Pfad>);

LL.Print();
LL.Dispose();
```

VB.NET:

```
Dim LL As New ListLabel()
LL.DataSource = CreateDataSet()

' lokalen Temporärpfad setzen
LL.Core.LlPreviewSetTempPath(Path.GetTempPath())

' Druckoptionen sollten in benutzerspezifischem Unterverzeichnis
' abgelegt werden, damit Änderungen dauerhaft übernommen werden
LL.Core.LlSetPrinterDefaultsDir(<Pfad>)

LL.Print()
LL.Dispose()
```

5. Update von älteren Versionen

In der Regel genügt es, den Verweis auf die combit.ListLabel15.dll durch einen Verweis auf die combit.ListLabel15.dll auszutauschen und die Namespace-Verweise zu aktualisieren. Sie sollten zusätzlich die alten Komponenten aus der Toolbox entfernen und durch die neuen Komponenten ersetzen.

Die Datenbindung ist in Version 15 erheblich erweitert worden. Sollten Sie in Ihren alten Projekten DataTable bzw. DataView-Elemente als Datenquelle verwendet haben, müssen Sie die Eigenschaft "AdvancedDatabinding" der Komponente auf False setzen. Für neue Projekte sollten Sie die Standardeinstellung "True" beibehalten. Sie profitieren so von den Features des Berichtscontainer-Objekts und haben eine Designervorschau für diese Datenquellen zur Verfügung.

Datenquellen, die die Schnittstellen IEnumerable oder IEnumerable<T> unterstützen werden nun ebenfalls über den Berichtscontainer abgebildet. Zudem entfällt das Caching als DataSet im Speicher, was bei umfangreicheren Objektmodellen wie Sie z.B. LINQ-Abfragen generieren, einen erheblichen Vorteil bietet. Allerdings sind bestehende Projekte nicht mehr kompatibel, um diese zu drucken, können Sie die Option "LegacyEnumBinding" auf "True" setzen. Der Default dieser Option ist "False".

Die Namensgebung im DOM-Namespace hat sich geändert. Aus z.B.

combit.ListLabel14.Dom.**ListLabelDomProjectList**

wird nun

combit.ListLabel15.Dom.**ProjectList**

Ebenso wurde "DesignerObject" in "ExtensionObject" umbenannt.

Ein entsprechender Suchen & Ersetzen Vorgang in Ihrem Editor sollte durchgeführt werden.

6. Index

A

Access 17
AdoDataProvider 14
Ausfertigungen 23
Ausgabeformat vorgeben 12
AutoDefineField 21
AutoDefineNewLine 21, 36
AutoDefineNewPage 21, 36
AutoDefineVariable 21
AutoDestination 12
AutoFileAlsoNew 12
AutoMasterMode 12, 19
AutoProjectFile 12
AutoProjectType 12, 21
AutoShowPrintOptions 12
AutoShowSelectFile 12

B

Barcode 20
Beispiele 32
Berichtscontainer 27
Bild 20

C

Charts 36

D

DataBinding 8
DataMember 19
DataProviderCollection 15
DataTable 14
DataView 14
DataViewManager 14
Dateiauswahldialog 12
Dateiendungen 9
Dateitypen 9
Daten unterdrücken 38
datenbankunabhängig 36
Datenprovider 14
Datenquelle 8
Datentyp
  Barcode 20
  Datum 20
  Grafik 20
  HTML 20
  Logisch 20
  RTF 20
  Text 20
  Zahl 20
Datentypen 19
Datenübergabe 8
DateTime 20
DbCommandSetDataProvider 15
Debugging 30
Debwin3 30
Design 8
Designer
  anpassen 24
  erweitern 24, 41
DesignerFunction 41
DOM 28
DrawObject 21
DrawPage 21
DrawTableField 21
DrawTableLine 21
Druck 10
  Netzwerk 44
Drucker 23
Druckereinstellungen 9
Druckertreiber 13
Druckoptionsdialog 12

E

eMail-Versand 43
Entity Framework 15
EntityCollection 16
Ereignisse 21
Etikett 32
Etiketten 22
Export 10, 39

Formate 11
Formate einschränken 40
ohne Benutzerinteraktion 39

F

Felder 19, 22
FileExtensions 9
Funktionen sperren 24

H

Hochformat 23
HTML 20

I

IDbCommand 15
IEnumerable<T> 15
IListSource 16
Instanzierung 7
Integration 7
ITypedList 16

K

Karteikarten 23
Komponente
Eigenschaften 9, 10, 12
Komponenten 5
Konzepte 14
Kopien 23
Kreuztabellen 36

L

Layoutbereiche 23
LINQ 16
List<T> 15
Liste 33
Listen 22
ListLabel 5
ListLabelDocument 6
ListLabelPreviewControl 5
ListLabelRTFControl 5
ListLabelWebViewer 6
Lizenzierung 8
LlExportOption 11

M

Menüpunkte sperren 24

N

Neues Projekt anlegen 12

O

ObjectDataProvider 15
ObjectReportContainer 28
Objects 28
ObjectText 28
Objekte 25
Barcode 26
Bild 26
HTML 27
RTF-Text 26
Text 25
Objekte sperren 24
Objektmodell 28
OleDbConnectionDataProvider 17
OracleConnection 18
OracleConnectionDataProvider 18

P

P-Datei 9
Platzhalter 34
Preview 42
PreviewFile 43
ProhibitedActions 24
ProhibitedFunctions 24
ProjectCard 28
ProjectLabel 28
ProjectList 28
Projektdatei vorgeben 12
Projekttyp 12
Projekttypen 21
Etiketten 22
Karteikarten 23
Listen 22

Protokolldatei anfertigen 30

Q

Querformat 23

R

ReadOnlyObjects 24
Rechnung 20
Regions 28

S

Sammelrechnung 33
Seriendruck 20
SqlConnection 18
SqlConnectionDataProvider 18
SubItemTable 29
Subreports 35

T

Textbaustein 34
Textblöcke 19

U

ungebundene Daten 36
Unterberichte 35
Update 46

V

VariableHelpText 21
Variablen 12, 19, 22
Visual Studio 5
Vorschaudateien
  konvertieren 42
  zusammenfügen 42

W

Webreporting 12

X

XML 18
XmlDataProvider 18

Z

Zusatzdaten übergeben 21