# Erstellen eines Adressbuchs

In diesem Windows Forms - Projekt wird Anhand der Erstellung eines Adressbuchs der Umgang mit Daten geübt. Im Vordergrund stehen dabei die Serialisierung/ Deserialisierung von Daten und das Drucken von Daten unter Windows. Die Daten werden in einer einfachen, flachen Datenbank ohne Schlüssel verwaltet.

## 1. Die Ausgangssituation

In einem Adressbuch sollen personenbezogene Daten gespeichert und abgerufen werden können. Ein Ausdruck des aktuell gewählten Datensatzes ist ebenfalls vorgesehen. Sie entschließen sich für ein Windows Forms – Projekt, da immer nur ein Datensatz gleichzeitig angezeigt und bearbeitet werden soll. Außerdem lassen sich die Daten durch die Trennung von Ansicht und Modell leicht verwalten. Der Form dient als Benutzerschnittstelle.

### 1.1 Die Analyse der Anforderungen

Zunächst müssen die Anforderungen an die Applikation geklärt werden:

☞ **Überlegen Sie, welche Daten in einem Datensatz zu speichern sind!**
☞ **Welche Aktionen sollten mit einem Datensatz möglich sein?**

### 1.2 Das grafische Design der Benutzeroberfläche

Die Ansichtsklasse der Applikation kann als Windows Form umgesetzt werden, auf dem die Steuerelemente abgelegt sind.

☞ **Skizzieren Sie den Aufbau der Ansicht.**

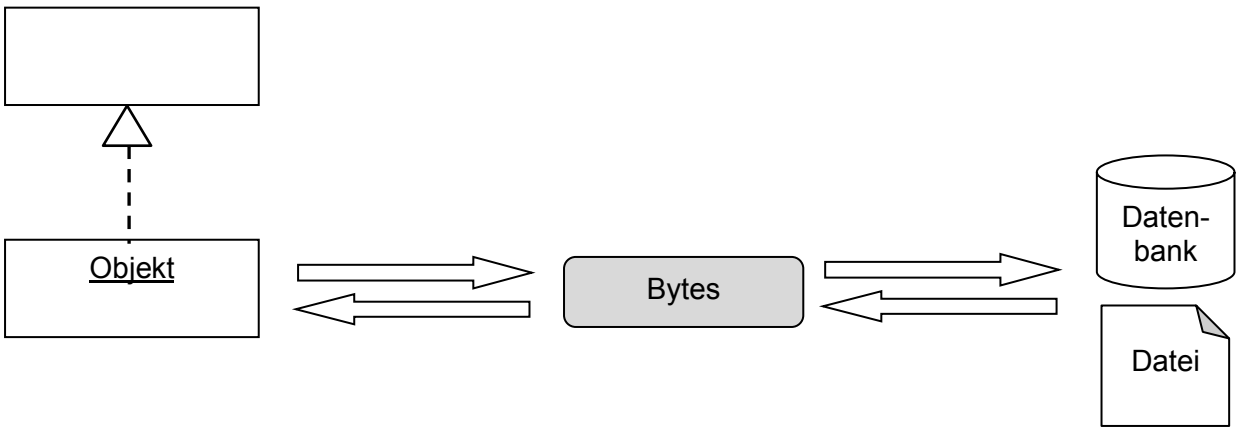
## 2. Serialisierung in Windows - Programmen

Die meisten Anwendungen bieten dem Benutzer die Möglichkeit, die eigene Arbeit zu speichern und wieder zu laden. Dabei kann es sich um ein Dokument einer Textverarbeitung, ein Tabellenblatt, eine Zeichnung oder eine Menge von Datensätzen handeln.
Da die Adressapplikation Daten speichern und wiederherstellen muss, sind die Datensätze in einer serialisierbaren Form zu erstellen.

### 2.1 Serialisierung unter C#

☞ **Erläutern Sie den Begriff Serialisierung!**

Die sogenannte Serialisierung zerfällt in zwei Teile. Wenn Objekte auf dem Systemlaufwerk in Form einer Datei gespeichert werden, spricht man von Serialisierung. Wird der Anwendungszustand des Objekts aus der Datei heraus wiederhergestellt, bezeichnet man diesen Vorgang als Deserialisierung. Die Kombination der beiden Teile ergibt die Serialisierung von Anwendungsobjekten unter Windows und .NET.



## 2.2 Serialisierbare Objekte erzeugen

☞ **Beschreiben Sie die Implementierung serialisierbarer Klassen in C#!**

In C# gibt es zwei Möglichkeiten um serialisierbare Objekte zu erstellen.

**Einfache Serialisierung:**

Benötigt man keine genaue Kontrolle über die Vorgänge der Serialisierung/ Deserialisierung, so genügt es, die Klasse mit dem Runtime Attribut [Serializable( )] zu kennzeichnen. Alle Elemente, die nicht mit serialisiert werden sollen, kennzeichnet man mit dem [NonSerialized( )] Attribute.
Die Serialisierung steuert das .NET Framework. Hierbei kann es allerdings zu Problemen bei unterschiedlichen Objektversionen kommen.

```
[Serializable( )]
public class BeispielEinfach
{
   // Element wird serialisiert
   private int i1;

   // Element wird nicht serialisiert
   [NonSerialized( )]
   private int i2;
}
```

**Benutzerdefinierte Serialisierung:**

Bei der benutzerdefinierten Serialisierung kann der Programmierer den Ablauf der Serialisierung und die serialisierten Elemente selbst bestimmen. Dazu muss die betreffende Klasse zusätzlich zum SerializableAttribute die ISerializable-Schnittstelle implementieren.

```
[Serializable( )]
public class BeispielBenutzerDefiniert : ISerializable
{
   private int i1;
   …
```

### 2.2.1 Die Methode GetObjectData(…)

In der überschriebenen Methode GetObjectData werden alle zu serialisierenden Daten mit der AddValue(…) Methode des SerializationInfo Objekts erfasst.

```
…
public void GetObjectData(SerializationInfo info,
StreamingContext context)
{
  info.AddValue("props", i1, typeof(int));
}
…
```

### 2.2.2 Der Überladene Konstruktor

Ein überladener Konstruktor dient dazu, die Werte bei der Deserialisierung zurück in das Objekt zu schreiben. Auch hier nutzt man wieder eine Methode des Serialization Objekts

```
…
public BeispielBenutzerDefiniert (SerializationInfo info,
StreamingContext context)
{
  i1 = (int) info.GetValue("props", typeof(int));
}
…
```

## 2.3 Die Umsetzung der Serialisierung

☞ **Beschreiben Sie die Umsetzung der Serialisierung in C#!**

In C# gibt es prinzipiell drei Arten der Serialisierung, die binäre Serialisierung, die Serialisierung im XML Format (letztere kann auch spezialisiert im SOAP Format zur Datenübertragung z.B. bei Webdiensten verwendet werden) sowie die Serialisierung als *.json Datei.

Die **Binäre Serialisierung** erzeugt kleine und schnell lesbare Dateien. Alle Daten des Objekts (auch private Daten) werden serialisiert. Das Format ist aber unflexibel und nur in einer Anwendung lesbar. Binäre Serialisierung findet daher kaum noch Anwendung.

Die **XML Serialisierung** (E**x**tensible **M**arkup **L**anguage) bietet hingegen einen lesbareren Code sowie eine höhere Flexibilität bei der gemeinsamen Nutzung von Objekten und beim Datenaustausch. Es werden aber nur öffentliche Elemente serialisiert. Ebenso haben XML-Dateien durch den hierarchischen Aufbau und den Tags einen sehr großen Overhead.

Die **JSON Serialisierung** (**J**ava **S**cript **O**bject **N**otation) liefert kleine plattformunabhängige Textdateien. Hier ersetzen Sonderzeichen die XML Tags, was zu einer deutlichen Reduzierung des Overheads führt. Trotzdem bleiben die Daten lesbar.

### 2.3.1 Binäre Serialisierung

Bei der binären Serialisierung verwendet man ein Formatter Objekt zum Serialisieren und Deserialisieren eines Objekts.

Über einen Stream kann das Objekt geschrieben bzw. wieder eingelesen werden.

Beim Deserialisieren ist ein zusätzlicher Cast erforderlich, um das Objekt wieder herzustellen.

```
…
MyClass obj1 = new MyClass( );
IFormatter f = new BinaryFormatter( );
FileStream fs;
// Serialisieren
fs = new FileStream(@"C:\test.txt", FileMode.Create);
f.Serialize(fs, obj1);
fs.Close( );

// Deserialisierung
fs = new FileStream(@"C:\test.txt", FileMode.Open);
MyClass obj2 = (MyClass) f.Deserialize(fs);
fs.Close( );
…
```

### 2.3.2 XML Serialisierung

Bei der XML - Serialisierung verwendet man einen XMLSerializer (oder einen SoapFormatter) anstelle eines BinaryFormatter Objekts zum Serialisieren und Deserialisieren eines Objekts.

Über einen Stream kann das Objekt geschrieben bzw. wieder eingelesen werden.

Beim Deserialisieren ist wieder ein Cast erforderlich, um das Objekt herzustellen.

```
…
MyClass obj1 = new MyClass( );
XmlSerializer s = new XmlSerializer(typeof(MyClass));
FileStream fs;
// Serialisieren
fs = new FileStream(@"C:\test.xml", FileMode.Create);
s.Serialize(fs, obj1);
fs.Close( );

// Deserialisierung
fs = new FileStream(@"C:\test.xml", FileMode.Open);
MyClass obj2 = (MyClass) s.Deserialize(fs);
fs.Close( );
…
```

### 2.3.3 JSON Serialisierung

Für die JSON - Serialisierung werden zwei Schritte benötigt. Zuerst wandelt man das Objekt in einen JSON String um und anschließend schreibt man den String in eine Datei.
Ab Framework 4.0 verwendet man dazu ein JavaScriptSerializer Objekt, ab .NET 5.0 die statische Serialize(…) Methode der JsonSerializer Klasse.

Der StreamWriter legt den String in die angegebene Datei.

Beim Deserialisieren kann mit den Deserialize(…) Methoden auf das Casting verzichtet werden, da hier die Typumwandlung mit Hilfe von Generics (vgl. Templates) durchgeführt wird. Die Verwendung der Klassen JavaScriptSerializer bzw. JsonSerializer ist wieder abhängig vom eingesetzten Framework.

```
…
MyClass obj1 = new MyClass( );
// System.Web.Extensions Assembly (ab Framework 4.0)
JavaScriptSerializer serializer = new JavaScriptSerializer( );
string result = serializer.Serialize(obj1);
// System.Text.Json Assembly (ab .NET 5.0)
string result = JsonSerializer.Serialize< MyClass >(obj1);
string path = @"C:\test.json";
using (StreamWriter sw = new StreamWriter(path))
{
  sw.WriteLine(result);
}

// Deserialisierung
using (StreamReader sr = new StreamReader(path))
{
   string result = sr.ReadToEnd( );
}
// System.Web.Extensions Assembly (ab Framework 4.0)
JavaScriptSerializer serializer = new JavaScriptSerializer( );
MyClass obj2;
obj2 = serializer.Deserialize<MyClass>(result);
// System.Text.Json Assembly (ab .NET 5.0)
MyClass obj2;
obj2 = JsonSerializer.Deserialize<MyClass>(result);
```

# 3. Das Windows Forms Grundgerüst

Das Grundgerüst der Anwendung wird als Windows Forms – Applikation unter größtmöglicher Nutzung der Assistenten des Visual Studios erstellt.

- ☞ **Erstellen Sie unter C# eine neue Windows Forms - Anwendung „Adressbuch“ mit dem Visual Studio! (Framework 4.8x)**
- ☞ **Benennen Sie den Form1.cs im Projektmappenexplorer in AdressForm.cs um und bestätigen Sie die Änderungsabfrage.**
- ☞ **Setzen Sie die Textproperty des Forms auf „Adressbuch in C#“.**
- ☞ **Kompilieren, linken und testen Sie Ihre Applikation! Das Grundgerüst ist erstellt.**

## 3.1 Das Hauptmenü

Im nächsten Schritt müssen Sie das Standardgrundgerüst für die Adressbuch – Applikation entsprechend anpassen.

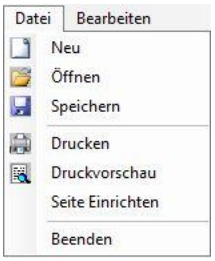
- ☞ **Fügen Sie dem Form einen Toolstripcontainer hinzu und benennen Sie diesen auf tscAdressen um!**

Der Toolstripcontainer ermöglicht es, dass später Toolbars bzw. Menübars frei angedockt werden können. Dazu besitzt er ein ContentPanel für den Form und vier Bereiche für Toolstripes/ Menüstripes (oben, unten, links, rechts).

- ☞ **Setzen Sie die Bereichssichtbarkeit des Containers auf oben und unten!**

Zur Benutzerführung erhält der Form ein Menü.

- ☞ **Fügen Sie dem Form einen Menüstrip hinzu und benennen Sie diesen auf „mainMenuStrip“ um!**
- ☞ **Bearbeiten Sie das Menü entsprechend der folgenden Tabelle!**
- ☞ **Die grafischen Ressourcen werden vom Lehrer gestellt, kopieren Sie diese in das Projektverzeichnis!**





| (Name) | Text | (Name) | Text |
|---|---|---|---|
| neuToolStripMenuItem | &Neu | neuerDatensatzToolStripMenuItem | &Neuer Datensatz |
| oeffnenToolStripMenuItem | &Öffnen | datensatzLoeschenToolStripMenuItem | Datensatz &Löschen |
| speichernToolStripMenuItem | &Speichern | | |
| druckenToolStripMenuItem | &Drucken | | |
| druckvorschauToolStripMenuItem | Druck&vorschau | | |
| seiteEinrichtenToolStripMenuItem | Seite &Einrichten | | |
| beendenToolStripMenuItem | &Beenden | | |

- ☞ **Setzen Sie die Image Property der Menüpunkte auf die passenden *.jpg Dateien. Importieren Sie die Bilder für jeden Menüpunkt dazu als „Lokale Ressource“!**

## 3.2 Die Toolbar

Analog zum Menü ist eine Toolbar mit den wichtigsten Aktionen zu erstellen.

- ☞ **Fügen Sie der Applikation einen neuen ToolStrip (adressToolStrip) hinzu!**
- ☞ **Bearbeiten Sie die Toolbar analog zum Menü!**

| (Name) |
|---|
| neuToolStripButton |
| oeffnenToolStripButton |
| speichernToolStripButton |
| neuerDatensatzToolStripButton |
| datensatzLoeschenToolStripButton |
| druckvorschauToolStripButton |
| druckenToolStripButton |

## 3.3 Beenden der Applikation

Die Applikation wird über debn Menüpunkt Beenden geschlossen.

- ☞ **Implementieren Sie die Handler Methode für das Klick Ereignis auf den Menüpunkt Beenden!**
- ☞ **Schließen (Close( )) und zerstören (Dispose( )) Sie den Form in der Methode!**

# 4. Der Entwurf des Basismodells

Beim Grundgerüst der Anwendung werden Daten und Benutzeroberfläche getrennt. Ebenso kommt eine vereinfachte Version des DAO Patterns (data access object) zum Tragen.
Bei der Verwaltung der Daten im Arbeitsspeicher entscheidet man sich für folgendes Modell:

- Als Benutzeroberfläche dient die Windows Forms Klasse Adressform.
- Die Daten eines jeden Datensatzes werden in der serialisierbaren Klasse Person gespeichert.
- Die DAOklasse (PersonDAO) bietet der Benutzeroberfläche über ein Interface IPersonDAO Zugriff auf die Datenserialisierung. Hier werden die Personenobjekte in einer Collection (List<Person>) abgelegt und verwaltet.

☞ **Erstellen Sie das grundlegende Klassenstrukturdiagramm für die Anwendung mit allen enthaltenen Klassen in Visio!**
☞ **Auf die Ausarbeitung der Methoden und Eigenschaften kann verzichtet werden.**

# 5. Die Klasse Person

In der Klasse Person sind für alle Einzeldaten private Attribute vom Typ string aufzunehmen. Für den Zugriff implementiert man öffentliche ZProperties mit **Get...( )** und **Set...(...)** Teil. Da die Personendaten auch gespeichert werden, ist die Klasse serialisierbar zu deklarieren.

☞ **Ergänzen Sie das Klassendiagramm von Person!**
☞ **Als Daten werden im Adressbuch Name, Vorname, Strasse, Hausnummer, PLZ, Ort, Telefon und Email gespeichert!**

Zur **Implementierung von Person** fügen Sie dem Projekt eine neue Klasse hinzu (Hinzufügen, Klasse...). Wählen Sie als Klassentyp eine C# Klasse.
Für die Attribute von Person deklariert man **private Attribute**. Der Zugriff Erfolgt über zugehörige **Properties.**
Diese Properties müssen nicht per Hand programmiert werden sondern lassen sich automatisch generieren:

- Rechtsklick auf das entsprechende Attribut
- Schnelle Aktionen…
- Feld kapseln

**Tipp**: C# ermöglicht für Attribute und Properties auch eine Kurzschreibweise: `public string Name{get; set;}`

```
private string name;
...
public string Name
{
        get
        {
           return name;
        }
        set
        {
           name = value;
        }
}
...
```

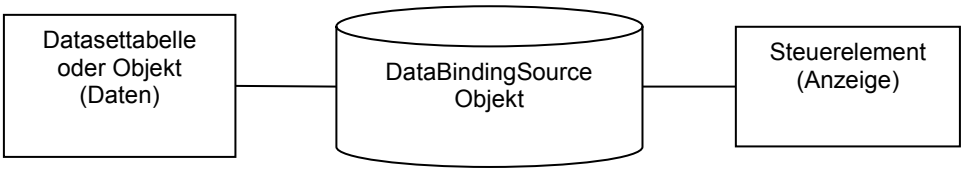
Zum Erzeugen der Serialisierung ist die Klasse mit einem Runtime Attribut zu kennzeichnen und das Interface ISerializable zu realisieren.

☞ **Implementieren Sie die Klasse Person als serialisierbare Klasse (siehe Punkt 2.2 auf der Seite 2 im Skript)!**
☞ **Erstellen Sie neben dem Standardkonstruktor auch die GetObjectData(…) Methode sowie den überladenen Konstruktor!**

# 6. Das Binden der Daten an die Steuerelemente

## 6.1 Data Binding in .NET

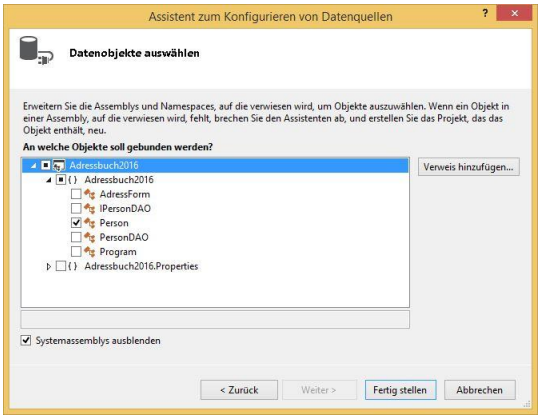
Im .NET-Framework können alle Properties eines Steuerelements an eine Datenquelle gebunden werden. Dabei unterscheidet man zwischen **simple** und **complex binding**. Beim simple binding bindet man nur eine Eigenschaft an einen Datensatz (z. B. die Text-Property eines Textfeldes an den Namen des Personenobjekts). Verwendet man das complex binding, so werden mehrere Datensätze an ein Steuerelement gebunden (z. B. alle Daten mehrerer Datensätze an ein DataGrid oder alle Namen an eine ComboBox).
Beide Arten der Bindung können grafisch oder im Quelltext implementiert werden.
Im Visual Studio übernimmt die Datenbindung ein **DataBindingSource** Objekt.




Den Quelltext zur Datenbindung lässt man sich üblicherweise vom Visual Studio generieren, sofern sich die Datenbindungen nicht zur Laufzeit ändern.

## 6.2 Das Erstellen der Data Bindings

Die Datenbindungen sollen zunächst über das Visual Studio automatisiert generiert werden.




- **Öffnen Sie im Projekt die Datenquellenansicht (Ansicht/ Weitere fenster/ Datenquellen)!**
- **Fügen Sie eine neue Datenquelle hinzu und wählen Sie ein Objekt!**
- **Erweitern Sie den Knoten Adressbuch und wählen Sie Ihre Klasse Person!**

Nach dem Klick auf „Fertig stellen“ sind die grundlegenden Voraussetzungen zum Data Binding geschaffen.


- **Erweitern Sie den Knoten Person in der Datenquellenansicht und schalten Sie in der ComboBox auf Details um.**
- **Ziehen Sie Person per Drag and Drop auf Ihren AdressForm!**

Die Benutzeroberfläche ist bis auf kleinere Änderungen fertig erstellt. Ebenso werden automatisch die beiden Objekte personBindingSource (DataBindingSource – Objekt) und personBindingNavigator (Navigation) generiert

- **Ordnen Sie die Steuerelemente passend auf der Oberfläche an und ändern Sie ggf. die Labels!**
- **Der Binding Navigator kann in die untere Leiste des Toolstripcontainers gelegt werden!**

# 7. Die Klasse PersonDAO

Die Klasse PersonDAO dient als Datenverwalter der Applikation. Sie speichert die Personen in einer privaten List Collection (List<Person> pList) und ist für die Serialisierung und Deserialisierung zuständig.
Für den internen Gebrauch zur Deserialisierung/ Serialisierung werden zwei private Methoden (void Load( ) und void Save( )) implementiert. Der Zugriff von außen erfolgt über drei public void Methoden (LoadData( ), SaveData( ), NewData( )).

- **Fügen Sie dem Projekt die neue Klasse PersonDAO hinzu!**
- **Implementieren Sie die Collection und die fünf Methoden mit leerem Methodenrumpf!**

Der Zugriff auf die Personenliste kann von außen über eine öffentliche Property PList zur Verfügung gestellt werden. Da die Liste selbst nicht geändert werden darf, wird nur ein lesender Zugriff (readonly) erlaubt. Der Set – Teil der Property ist zu löschen!

```
public List<Person> PList
{
  get
  {
    return pList;
  }
}
```

☞ **Generieren Sie die Property PList und löschen Sie den Set – Teil der Property!**

Im letzten Schritt kann über die Klasse ein Interface erstellt werden. Dies bewerkstelligt das Visual Studio automatisiert.

☞ **Erzeugen Sie mit dem Visual Studio ein Interface für die Klasse PersonDAO (Rechtsklick auf die Klasse im Code/ schnelle Aktionen/ Interface extrahieren)!**

☞ **Ergänzen Sie das grundlegende Klassendiagramm!**

# 8. Benutzeroberfläche und Dateninterface

Beim Data Binding bindet man Daten einer Datenquelle an die Steuerelemente einer Benutzeroberfläche. Bisher sind die Steuerelemente der GUI bereits zur Designzeit an ein Person Objekt gebunden. Die Datenbindung muss sich im Programm aber auf die Collection mit Personenobjekten in PersonDAO beziehen.

## 8.1 RefreshBinding( )

Für die eigentliche Verknüpfung der Daten und der GUI erstellt man eine neue, private Methode in AdressForm. Ebenso benötigt die GUI Zugriff auf das DAO – Interface.

☞ **Definieren Sie ein private Attribut vom Typ der PersonDAO Klasse im AdressForm!**

☞ **Implementieren Sie die private Methode RefreshBinding( ) zum Setzen der Datenbindung!**

Die DataSource Property der BindingSource wird jetzt mit der Person Collection verbunden.

```
…
private IPersonDAO pers = new PersonDAO( );
…
private void RefreshBinding( )
{
   personBindingSource.DataSource = pers.PList;
}
…
```

## 8.2 Die Handler Methoden für die GUI

Zur Verbindung der GUI mit den Daten sind zunächst die Handler-Methoden für die Menüereignisse "**Neu**", "**Öffnen**" und "**Speichern**" zu erstellen.

☞ **Generieren Sie die Handler Methoden für die drei Klick Ereignisse mit dem Visual Studio!**

In den drei Handler Methoden sind nur die zugehörigen Interface Methoden aufzurufen. Bei neuen Daten oder beim Laden von Daten muss zusätzlich RefreshBinding( ) aufgerufen werden.

```
…
public void neuToolStripMenuItem_Click(…)
{
   pers.NewData( );
   RefreshBinding( );
}
…
```

☞ **Verknüpfen Sie über die Eigenschaftenseite auch die Klick Ereignisse auf die Toolbar Buttons mit den zugehörigen (bereits erstellten) Handler-Methoden!**

☞ **Testen Sie die Applikation! Die Daten sollten bereits temporär im Arbeitsspeicher behalten werden. Betätigen Sie dazu den „+" Button im Bindingnavigator!**

## 8.3 Speichern der Daten

### 8.3.1 Binäres Speichern

Das Speichern der Daten übernimmt die DAO Klasse in der SaveData( ) Methode. Hierzu verwendet man am besten den bereits vorgefertigten SaveFileDialog des .NET Frameworks. Dieser Standarddialog ist den Benutzern bereits aus anderen Windows Applikationen bekannt und kann somit ohne Probleme genutzt werden.

Neben dem SaveFileDialog sind auch Attribute zum Dateipfad, zum Startpfad und der Dateiart zu implementieren.

```
private SaveFileDialog sfDialog = new SaveFileDialog( );
private string path = string.Empty;
private string initialPath = "K:";
private string initialFilter = "Adressdateien (*.adr) | *.adr";
```

☞ **Erstellen Sie die Attribute in der PersonDAO Klasse!**

In der SaveData( ) Methode setzt man die Attribute des SaveFileDialogs.
Ist bereits früher eine Speicherung der Daten erfolgt, so ist die path Eigenschaft nicht leer und kann als Dateipfadangabe für den Dialog genutzt werden.
Über den Dialog und dessen Rückgabe (DialogResult) leitet man den Speichervorgang ein. Die Save( ) Methode muss dazu noch implementiert werden.

☞ **Programmieren Sie die SaveData( ) Methode in der PersonDAO Klasse!**

```
public void SaveData( )
{
 sfDialog.InitialDirectory = initialPath;
 sfDialog.Filter = initialFilter;
 if (path != string.Empty)
 {
   sfDialog.FileName = path;
 }
 if (sfDialog.ShowDialog( ) == DialogResult.OK)
 {
   path = sfDialog.FileName;
   Save( );
 }
}
```

Das eigentliche Speichern (Serialisierung) der Personen Liste erfolgt dann in der privaten Methode Save( ) der PersonDAO Klasse.
In der Save( ) Methode nutzt man ein BinaryFormatter und ein FileStream Objekt zur Serialisierung der Personen.
Der using Block sorgt für die sofortige Freigabe des FileStream Objekts im Freispeicher, nachdem der Block verlassen wird.
Zur Sicherheit ist die Aktion in einen try – catch – Block mit Fehlerhandling gesetzt.

☞ **Erstellen Sie die Save( ) Methode zum Speichern der Daten!**

```
private void Save( )
{
 try
 {
   IFormatter f = new BinaryFormatter( );
   using (FileStream fs = new FileStream(path,
                FileMode.OpenOrCreate, FileAccess.Write))
   {
     f.Serialize(fs, pList);
     fs.Close( );
   }
 }
 catch(Exception ex)
 {
   MessageBox.Show(ex.Message, "Fehler!",
         MessageBoxButtons.OK,MessageBoxIcon.Error);
 }
}
```

Da die Speicherung der Daten auch dann durchgeführt werden soll, wenn der Benutzer die Adressbuch Applikation schließt, muss dies in einem Event Handler abgefangen werden. Dazu eignet sich der FormClosing Event des AdressForms. FormClosing wird abgefeuert, wenn der Form gerade geschlossen wird, jedoch bevor dieser bereits geschlossen ist.


☞ **Generieren Sie die Handler Methode zum FormClosing Event des AdressForms mit dem Visual Studio!**

In der Methode prüft man mit einer MessageBox und dem zurück gegebenen DialogResult, ob der Benutzer die Daten tatsächlich speichern will.
Entsprechend der Benutzerauswahl ist die Speicherung der Daten aus der DAO Klasse aufzurufen.

```
private void AdressForm_FormClosing(…)
{
  if(MessageBox.Show("Wollen Sie…",…
  {
    // Daten speichern
    …
  }
}
```

### 8.3.2 Laden der Daten

Analog zum binären Speichern der Daten verläuft das Laden neuer Daten aus einer Datei (Deserialisierung). Anstelle des SaveFileDialog Objekts verwendet man ein OpenFileDialog Objekt aus der .NET Bibliothek.

☞ **Implementieren Sie die beiden Methoden Load( ) und LoadData( ) analog zum Speichern der Daten! (siehe auch Punkt 2.3, ab Seite 2)**

## 8.4 Zusatzprogramm: XML und JSON Serialisierung

☞ **Implementieren Sie die Methoden LoadXML( ), LoadJSON( ) sowie SaveXML( ) und SaveJSON( ) für die Arbeit mit den entsprechenden Dateiformaten!**
☞ **Erweitern Sie den initialFilter und rufen Sie die passende Serialisierungsmethode auf!**

```
initialFilter = "Adressdateien (*.adr)|*.adr|XMLAdressen (*.xml)|*.xml|JSONAdressen (*.json)|*.json";
```

## 8.5 Erstellen eines neuen Adressbuchdatensatzes

Möchte der Benutzer nicht nur neue Daten in einem Adressbuch sondern eine komplett neue Adressbuchdatei erstellen, so klickt er auf den Menüpunkt „Neu“. Dabei wird die NewData( ) Methode von PersonDAO aufgerufen.

Bevor eine neue Liste angelegt wird, sollte jedoch die bestehende Liste gespeichert werden.
Sind bereits Einträge in der Liste, so prüft man mit einer MessageBox, ob der User die Daten speichern will.
Anschließend erzeugt man eine neue Liste mit Personen und setzt den Speicherpfad der Adressdatei zurück.

☞ **Erstellen Sie die NewData( ) Methode!**

```
public void NewData( )
{
   if(pList.Count > 0)
   {
     if (MessageBox…
     {
       // Daten speichern
       …
     }
   }
   pList = new List<Person>( );
   path = string.Empty;
}
```

## 8.6 Letzte Anpassungen der Benutzeroberfläche

Bei genauer Betrachtung der Benutzeroberfläche sollten noch drei kleinere Ungereimtheiten ins Auge stechen:


- Beim Start der Applikation und beim Erstellen eines neuen Adressbuchs wird kein leerer Datensatz angelegt (die Anzahl der Datensätze in der Datenquelle ist 0),
- die Toolbar des BindingNavigators enthält ein „Speichern“ – Icon, das noch nicht mit Funktionalität vorbelegt ist,
- im Menü und der Standard Toolbar finden sich zwei Einträge („Neuer Datensatz“, Datensatz Löschen“), die noch nicht implementiert sind.


Da die eigentlichen Funktionalitäten für alle Icons bzw. Menüpunkte aber bereits existieren, sind nur die entsprechenden Handler Methoden zu ergänzen bzw. zu erstellen.

☞ **Fügen Sie dem Konstruktor des AdressForms den Aufruf von neuToolstripMenuItem hinzu!**

```
public AdressForm( )
{
    InitializeComponent( );
    neuToolStripMenuItem.PerformClick( );
}
```

☞ **Ergänzen Sie die Handler Methode zum Klick Event um das Hinzufügen eines ersten Datensatzes in die leere Datenquelle!**

```
private void neuToolStripMenuItem_Click(…)
{
    …
    personBindingSource.AddNew( );
}
```

☞ **Generieren Sie die Handler Methode zum Klick Ereignis des personNavigatorSaveItems!**
☞ **Rufen Sie in der Handler Methode die PerformClick( ) Methode des speichernToolStripMenuItems auf!**

Auch die Funktionalität zum Erstellen und Löschen von einzelnen Datensätzen im Adressbuch ist bereits implementiert. Testen Sie dazu den Klick auf das „Plus“ bzw. das „X“ des BindingNavigators. Mit dem Erzeugen der BindingSource liefert das Visual Studio die passenden Methoden im BindingSource Objekt und verbindet diese mit den Icons des BindingNavigators.

☞ **Erstellen Sie die Handler Methoden zum Klick Ereignis auf die beiden Menüpunkte „Neuer Datensatz“ und „Datensatz Löschen“!**
☞ **Rufen Sie die Methoden „AddNew( )“ bzw. „RemoveCurrent( )“ des personBindingSource Objekts darin auf!**
☞ **Verbinden Sie auch die Klick Events auf die zugehörigen Toolbarbuttons mit den beiden Handler Methoden!**

# 9. Drucken unter Visual C#

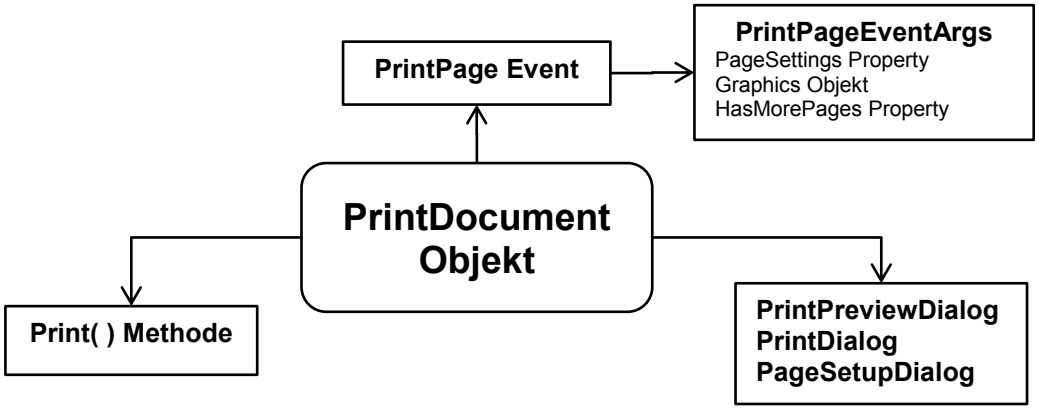
## 9.1 Allgemeines zum Druckvorgang

Für die geräteunabhängige Grafikausgabe arbeitet Windows mit sogenannten Gerätekontexten. Diese kann der Programmierer zum Zeichnen nutzen. Durch die installierten Treiber (z.B. der Grafikkarte) erscheint die Ausgabe auf unterschiedlichen Hardwareplattformen im richtigen Format. Genau wie beim Zeichnen in ein Fenster mit dem GDI+ (Grafic Device Interface), verwendet Windows auch einen Gerätekontext, um über das Betriebssystem und den Druckertreiber unterschiedliche Drucker - Hardware ansprechen zu können. Alles, was in diesen Gerätekontext gezeichnet wird, erscheint in der Druckervorschau und wird entsprechend ausgedruckt.

☞ **Erklären Sie, weshalb ein Programm keine Druckvorschau erzeugen kann, wenn kein Drucker installiert ist!**

## 9.2 Drucken unter Windows Forms

☞ **Erläutern Sie den prinzipiellen Druckvorgang unter Windows Forms!**




Beim Drucken unter Windows Forms und C# steht das **PrintDocument** Objekt im Mittelpunkt. Dieses kann im Quelltext oder per Toolbox hinzugefügt werden.
In der Handler Methode zum **PrintPage Event** implementiert man die Programmlogik zum Druckinhalt und zum mehrseitigen Druck. Der übergebene **PrintPageEventArgs** Parameter enthält dazu nützliche Elemente wie die PageSettings Property mit den Seiteneinstellungen, das Graphics Objekt (kapselt den Device Context zur Grafikausgabe) und die HasMorePages Property für mehrseitigen Druck.
Zusätzlich bietet das .NET Framework gewohnte Benutzerdialoge zum Konfigurieren und Durchführen des Druckvorgangs an. Dazu zählen die Druckvorschau (**PrintPreviewDialog**), die Seiteneinstellungen (**PageSetupDialog**) sowie der eigentliche Druckdialog (**PrintDialog**).
Die **Print( ) Methode** des PrintDocument Objekts sendet die Daten schließlich an den Drucker.

## 9.3 Vorbereitungen zum Druck

Um in der Adressbuchapplikation einen Ausdruck erstellen zu können muss zuerst ein PrintDocument Objekt erstellt werden.

☞ **Ziehen Sie in der Entwurfsansicht ein PrintDocument aus der Toolbar auf die GUI der Applikation!**

Das Visual Studio deklariert und initialisiert das PrintDocument automatisch. Über das neue PrintDocument kann im Anschluss daran die Handler Methode zum PrintPage Event erstellt werden.

☞ **Generieren Sie die Handler Methode zum PrintPage Event mit dem Visual Studio!**

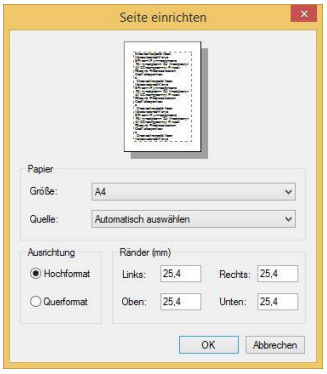
Für die Seiteneinstellungen verwendet man ein PageSettings Objekt, auf das alle am Druck beteiligten Objekte Zugriff erhalten.

```
private PageSettings pageSet = new PageSettings( );
```

☞ **Definieren Sie ein PageSettings Objekt als Attribut im AdressForm!**

## 9.4 Das Einrichten der Seite

Wie beim Laden und Speichern von Daten stellt das .NET Framework auch für den Druckvorgang entsprechende Standarddialoge zu Verfügung. Mit dem PageSetupDialog lassen sich beispielsweise Seiteneinstellungen zum Drucken durch den Benutzer beeinflussen.




- **Fügen Sie dem AdressForm einen PageSetupDialog über die Toolbox hinzu!**
- **Benennen Sie den Dialog in psDialog um!**
- **Erstellen Sie die Handler Methode für das Klick Event auf den Menüpunkt Seite Einrichten!**

In der Handler Methode sind die vier Seitenränder auf einen Standardwert zu setzen.
Die Margins beziehen sich auf 0.01 inch und müssen auf Millimeter umgerechnet werden.

PrintDocument und PageSetupDialog teilen sich ein PageSettings Objekt mit den gemeinsamen Einstellungen. Das PageSettings Objekt ist deshalb für beide zu setzen.

```
private void seiteEinrichtenToolStripMenuItem_Click(…)
{
    // Seitenränder setzen
    pageSet.Margins.Left = 100;
    pageSet.Margins….
    …
    // Umrechnung in mm
    psDialog.EnableMetric = true;
    printDocument.DefaultPageSettings = pageSet;
    psDialog.PageSettings = pageSet;
    psDialog.ShowDialog( );
}
```

## 9.5 Die Druckvorschau

Analog zum Einrichten der Seite verläuft die Druckvorschau. Auch hier kann auf vorgefertigte Elemente des .NET Frameworks zugegriffen werden.

- **Fügen Sie dem AdressForm einen PrintPreviewDialog über die Toolbox hinzu!**
- **Benennen Sie den Dialog in ppDialog um!**
- **Erstellen Sie die Handler Methode für das Klick Event auf den Menüpunkt Druckvorschau!**
- **Verknüpfen Sie auch den ToolbarButton mit der Handler Methode!**

In der Methode selbst setzt man die Document Property auf das verwendete PrintDocument Objekt.
Damit später alle Datensätze in der Vorschau angezeigt werden setzt man die Datenquelle auf den ersten Datensatz bevor der Dialog angezeigt wird.

```
private void druckvorschauToolStripMenuItem_Click(…)
{
    ppDialog.Document = …
    personBindingSource.MoveFirst( );
    // Dialog anzeigen
    …
}
```

## 9.6 Die Drucklogik in der Handler Methode

Die eigentliche Programmlogik zum Ausdruck findet sich in der Handler Methode zum PrintPage Event der Applikation.

- **Implementieren Sie die Handler Methode zum PrintPage Event anhand der folgenden Hilfestellungen!**

### 9.6.1 Die Datenlogik

Beim Ausdruck soll jeder Datensatz auf einer extra Seite dargestellt werden. Ebenso enthält die Überschrift die Nummer des Datensatzes sowie die Gesmtanzahl der Datensätze in der Adress-datei. Die entsprechenden Daten liefert die verknüpfte Datenquelle über das BindingSource Objekt. Da die Position Property den Index wiedergibt ist diese um 1 zu erhöhen.
Mit den Daten kann die Überschrift zusammengesetzt werden.

```
private void printDocument_PrintPage(…)
{
    // Daten
    Person p = (Person)personBindingSource.Current;
    int number = personBindingSource.Position + 1;
    int count = personBindingSource.Count;
    string header = "Adressdatensatz " + number + " von " + count;
    …
```

Am Ende jeder Seite muss vor dem Schließen der Methode auf den nächsten Datensatz gesprungen werden.
Sind weitere Datensätze vorhanden, so ist die HasMorePages Property für den Ausdruck weiterer Seiten auf true zu setzen. Die Handler Methode wird erneut aufgerufen.

```
…
// Nach dem Ausdruck eines Datensatzes…
personBindingSource.MoveNext( );
if (number < count)
    e.HasMorePages = true;
else
    e.HasMorePages = false;
}
```

### 9.6.2 Grafikelemente und Seitenaufteilung

Der Ausdruck oder das Zeichnen in den Device Context erfolgt beim GDI+ mit Hilfe verschiedener Grafikobjekte wie Pinsel, Stifte, Schriften etc. Das Graphics Objekt besitzt dazu Methoden, welche diese Elemente als Parameter verwenden.

Innerhalb der Handler Methode kann über die PrintPageEventArgs auf das Graphics Objekt zugegriffen werden.
Für die Überschrift nutzt man später eine blaue Schrift.

```
…
// Grafikelemente
Graphics g = e.Graphics;

Font headerFont = new Font ("Arial",18,FontStyle.Bold);
Brush headerBrush = Brushes.Blue;
…
```

☞ **Erzeugen Sie eine Schriftart für den regulären Text (textFont, Arial 14) mit schwarzer Farbe (textBrush)!**

Zeilenabstand und Spalten der Ausgabe richten sich nach der Überschrift und dem bedruckbaren Bereich.
Die MeasureString(…) Methode des Graphics Objekts sowie die MarginBounds Property liefern die benötigten Angaben um den Text später richtig zu platzieren.

```
// Zeilenabstand und Ränder
float headerWidth = g.MeasureString(header, headerFont).Width;
float headerHeight = g.MeasureString(…)…

float row = e.MarginBounds.Height / headerHeight;

float column1 = e.MarginBounds.Left;
float column2 =  e.MarginBounds.Left + e.MarginBounds.Width / 3.0f;

// Nach dem Ausdruck eines Datensatzes…
…
```

### 9.6.3 Die Drucklogik

In der folgenden Ausgabe setzt man die DrawString(…) Methode des Graphic Objekts mit den oben erstellten Variablen ein. DrawString(…) übernimmt dazu den Text, die Schriftart sowie eine Brush. Zusätzlich benötigt die Methode noch x und y Koordinaten für die Platzierung des Textes.
Da die Überschrift mittig liegen soll, ist die x Koordinate über die Schriftbreite und den Seitenrand zu berechnen.

```
…
// Überschrift drucken
g.DrawString(header, headerFont, headerBrush,
                e.MarginBounds.Left + (e.MarginBounds.Width - headerWidth) / 2,  e.MarginBounds.Top);
…
```

☞ **Erklären Sie die Berechnung der Position der Überschrift!**

Die Datensätze selbst druckt man in Spalte eins und zwei beginnend ab der 3. Zeile. Die Zeilenhöhe richtet sich dabei nach der Höhe der Überschrift.

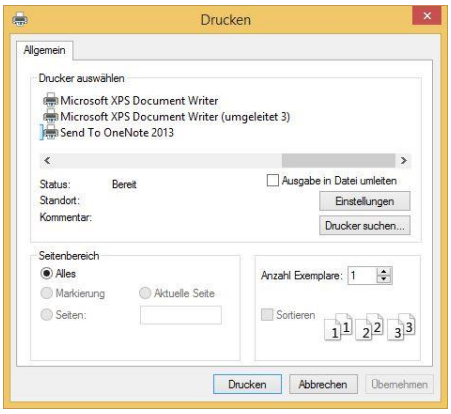
```
…
// Daten drucken
int line = 3;
g.DrawString("Name:", textFont, textBrush, column1, line * row + e.MarginBounds.Top);
g.DrawString(p.Name, textFont, textBrush, column2, line++ * row + e.MarginBounds.Top);
g.DrawString("Vorname:", …
…

// Nach dem Ausdruck eines Datensatzes…
…
```

☞ **Ergänzen Sie den Druck um die fehlenden Daten einer Person!**

Nach Abschluss der Drucklogik kann die Druckvorschau mit Inhalt gestartet werden.
Für alle Adressdaten sollte die Druckvorschau des Adressbuchs in etwa folgendermaßen aussehen:
Nach einer mittigen Überschrift mit der Nummer und der Anzahl der Datensätze folgen tabellerisch die Daten der Personenobjekte des Adressbuchs.


### 9.7 Drucken mit dem Druckdialogfeld

Für den Ausdruck steht ein weiteres vorgefertigtes Dialogfeld des .NET Frameworks zu Verfügung.
Der PrintDialog enthält bereits alle notwendigen Funktionalitäten zur Druckerauswahl, Seitenauswahl und zum Drucken allgemein.




- ☞ **Fügen Sie dem Adressform aus der Toolbox einen neuen PrintDialog hinzu!**
- ☞ **Benennen Sie das Dialogfeld in pDialog um!**

Die Anzeige und Auswertung des Dialogs erfolgt über die entsprechende Handler Methode.

- ☞ **Erstellen Sie die Handler Methode für das Klick Event auf den Menüpunkt Drucken!**
- ☞ **Verknüpfen Sie auch den ToolbarButton mit der Handler Methode!**

Innerhalb der Methode zeigt man den Druckdialog an und wertet das DialogResult in einer if- Auswahl aus.
Hat der Benutzer den Dialog mit OK verlassen, so springt man zum Drucken auf den ersten Datensatz und gleicht die PrinterSettings Property des PrintDocuments mit denen des Dialogs ab. Anschließend startet man den Ausdruck mit dem PrintDocument Objekt.

```
private void druckenToolStripMenuItem_Click(…)
{
    // Dialog anzeigen, wenn der Benutzer OK klickt
    if(…
    {
        // Auf ersten Datensatz springen
        personBindingSource.MoveFirst( );
        // PrinterSettings setzen
        printDocument.PrinterSettings = …
        // Druck starten
        …
    }
}
```

## 10. Zusatzprogramm

### 10.1 Die Suche nach Adressdaten

In der Applikation sollte auch eine Suche/ Filterung nach Adressdatensätzen möglich sein. Die List Collection verfügt dazu über eine ganze Reihe von Möglichkeiten, um die darin enthaltenen Elemente zu Sortieren oder zu Suchen.

- ☞ **Implementieren Sie eine Suche nach bestimmten Nachnamen!**
- ☞ **Nutzen Sie dazu beispielsweise die FindAll(…) Methode der List Collection!**

### 10.2 Die Erweiterung des Ausdrucks

Natürlich können Sie auch Rechtecke, Kreise, Linien etc. mit dem Graphics Objekt „zeichnen“. Dies ermöglicht das Erstellen eines individuellen Adressberichts mit Zeilen, Spalten, farbigen Feldern und Überschriften. Spicken Sie dazu in der MSDN.

- ☞ **Gestalten Sie den Ausdruck der Datensätze mit Hilfe der MSDN ansprechender!**
- ☞ **Nutzen Sie dazu das Graphics Objekt sowie die PrintPageEventArgs Elemente!**