



Handwerkskammer  
Niederbayern-Oberpfalz

FEBRUAR 2022

---

# C# Anwendungsentwickler

---

Ergänzende Hinweise

Andreas Piehler  
Benedikt Geisler

# Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>3</b>
1.1	Pseudocode . . . . .	3
1.2	Clean Code . . . . .	3
1.3	Codeausschnitte . . . . .	5
1.4	Das Ausgabefenster . . . . .	5
<b>2</b>	<b>Abfangen von ungültigen Benutzereingaben</b>	<b>6</b>
2.1	Konvertierung eines Strings mit <code>TryParse</code> . . . . .	6
2.2	Abfrage eines Strings auf <code>null</code> oder <code>empty</code> . . . . .	7
<b>3</b>	<b>Arbeiten mit mehreren Formularen</b>	<b>9</b>
3.1	Formular aufrufen . . . . .	9
3.2	Rückgabe des Formularaufrufs auswerten . . . . .	9
3.3	Datenaustausch zwischen Formularen . . . . .	10
3.4	Anwendung beenden . . . . .	11
<b>4</b>	<b>Klasse <code>String</code> für Zeichenketten</b>	<b>13</b>
4.1	Eigenschaften . . . . .	13
4.1.1	<code>Length</code> . . . . .	13
4.1.2	<code>Chars</code> . . . . .	13
4.2	Methoden . . . . .	13
4.2.1	<code>Trim</code> : Leerzeichen am Anfang und Ende entfernen . . . . .	14
4.3	Formatierung von Strings . . . . .	14
4.3.1	Formatierungsoptionen . . . . .	14
<b>5</b>	<b>Datum und Uhrzeit</b>	<b>15</b>
5.1	Datumsauswahl mit dem <code>DateTimePicker</code> . . . . .	16
<b>6</b>	<b>Mathematik</b>	<b>17</b>
<b>7</b>	<b>Zugriff auf Dateien</b>	<b>19</b>
7.1	Dateien lesen . . . . .	19
7.2	Dateien schreiben . . . . .	20
7.3	csv-Datei einlesen . . . . .	21
7.4	Standardpfade . . . . .	22
<b>8</b>	<b>Datenbankzugriff mit dem Entity Framework Core</b>	<b>24</b>
8.1	SQL-Server Instanz anzeigen . . . . .	25
8.2	Anlegen einer bestehenden Datenbank in VisualStudio . . . . .	25
8.3	Installieren der EF CLI Tools . . . . .	25

8.4	NuGet-Paket einbinden . . . . .	26
8.5	Scaffolding der bestehenden Tabellen in das C# Projekt . . . . .	26
8.6	Datenbindung an Windows Forms Elemente . . . . .	27
8.6.1	Context-Instanz erzeugen und Datenquelle binden . . . . .	30
8.6.2	PositionChanged-Ereignis der Datenquelle . . . . .	31
8.6.3	Label oder Textbox . . . . .	31
8.6.4	Combobox . . . . .	32
8.7	Code um Änderungen in der Datenbank zu speichern . . . . .	32
8.7.1	Neuen Datensatz anlegen . . . . .	32
8.7.2	Datensatz löschen . . . . .	33
8.7.3	Datensatz ändern . . . . .	33
8.7.4	Datensatz suchen . . . . .	33
<b>9</b>	<b>LINQ-Abfragen</b>	<b>34</b>
9.1	Filtern . . . . .	34
9.2	Sortieren . . . . .	35
9.2.1	Aufsteigend . . . . .	35
9.2.2	Absteigend . . . . .	35
9.3	Erster Datensatz (Genau ein Datensatz) . . . . .	35
9.4	Anzahl Datensätze . . . . .	36
9.5	Summe bilden . . . . .	36
9.6	Maximum . . . . .	36
9.7	Minimum . . . . .	36

# 1 Einführung

## 1.1 Pseudocode

Wird man beim Programmieren vor eine Aufgabe gestellt so ist die erste Frage, die man sich stellt naturgemäß: *Was will ich überhaupt erreichen?*

Es ist überaus wichtig, sich diese Frage zu stellen. Nicht selten verliert man sich schnell in Lösungsansätzen, erschlagen von den Möglichkeiten, die uns technisch offen stehen. Das *Was* wird übersprungen, man startet sofort beim *Wie*. Stelle Dir diese Frage deswegen vor jeder Aufgabe!

Nachdem das geklärt wurde kann man über den besten Weg nachdenken, wie man das erreichen könnte. Und auch hier gilt wieder: Am Anfang lieber einen Gang zurückschalten und kurz nachdenken, wie man das Problem am effektivsten lösen könnte. Nicht sofort mit der Programmierung beginnen, sonst endet verfangt man sich schnell im *Spaghetti-Code*.

Hierbei hat sich *Pseudocode* als ein nützliches Werkzeug erwiesen. **Dabei beschreibt man die einzelnen Lösungsschritte zuerst in Klartext und schreibt sie als Kommentare in den Code.** Dadurch schafft man schnell eine Grundstruktur des Programms. Außerdem können Denkfehler schneller erkannt und auch behoben werden.

Sobald diese Grundstruktur steht, **ersetzt man die Kommentare durch tatsächlichen Code.** Kommentare sollten nur dann bestehen bleiben, wenn sie zum Code zusätzliche Informationen liefern. Kommentare, die nur aussagen, was der Code an dieser Stelle tut sollten gelöscht werden. Unverständlicher Code sollte durch gut lesbaren Code ersetzt werden. Mehr dazu im nächsten Kapitel.

## 1.2 Clean Code

Als Programmierer verbringt man offensichtlich viel Zeit vor dem Bildschirm. Interessanterweise verwendet man deutlich mehr Zeit auf das *lesen* von bestehendem Code als auf das Schreiben von neuen Funktionen. Daher ist es essentiell, den Code so *verständlich* wie möglich zu gestalten. Idealerweise liest er sich nicht wie eine komplizierte Maschinensprache, sondern wie ein gut geschriebenes Buch.

Wie erreicht man das?

**Aussagekräftige Variablen- und Methodenbezeichner.** Verwenden Sie für Ihre Variablen und Methoden Namen, die präzise aussagen, was sich dahinter verbirgt. Methodennamen sollten in *Verb* beinhalten, da Methoden immer etwas ausführen,

sprich etwas *tun*. Verwenden Sie *CamelCasing*, um einzelne Wörter im Methoden- oder Variablennamen voneinander zu trennen. Diese Schreibweise hat sich als am einfachsten schreib- und lesbar herausgestellt.

```
1 // Positivbeispiel
2 var vorname = "Hans";
3 var nachname = "Müller";
4 var vorUndNachname = vorname + " " + nachname;
5
6 // Negativbeispiel
7 var a = "Hans"; // a = Vorname
8 var b = "Müller"; // b = Nachname
9 var c = a + " " + b;
```

Als Ausnahme zu dieser Regel sind Schleifenvariablen und LINQ-Abfragen zu betrachten: Dort hat es sich etabliert, als Indexvariable einen Buchstaben zu verwenden.

**White Space.** Achten Sie darauf, funktionell zusammengehörige Teile im Programm auch zueinander zu stellen. Fügen Sie Leerzeilen ein, um einzelne Segmente innerhalb von Methoden voneinander abzugrenzen.

**Kommentare.** Setzen Sie Kommentare spärlich ein. Nutzen Sie sie, um die Aussagekraft des Codes zu erhöhen oder Entscheidungen zu dokumentieren. Nutzen Sie sie, um eine Grundstruktur des Programms zu erstellen, bevor Sie zu programmieren beginnen.

Zu Beginn Ihrer Programmier-Karriere kann es sehr nützlich sein, in Kommentaren festzuhalten, was der Code tut. Tun Sie das immer, sobald Sie eine Codezeile nicht zu hundert Prozent verstehen. Sie werden sehen, dass Sie mit der Zeit auf immer mehr dieser Kommentare verzichten können, da sich Ihr Wissen und Verständnis gefestigt hat.

**Methoden.** Methoden (auch Funktionen genannt) sollten genau eine Sache ausführen. Funktionen, die mehr als das tun, werden allgemein als *Code Smell* betrachtet. Das führt zu kurzen, leicht lesbaren Funktionen. Als Daumenregel gilt, dass Funktionen nicht länger als eine Bildschirmseite sein sollten, im Idealfall deutlich kürzer.

**DRY.** **D**ont't **r**epeat **y**ourself. Dies ist eins der wichtigsten Programmierprinzipien. Sobald Sie den gleichen Code mehrfach schreiben, ist dies in den meisten Fällen ein Hinweis darauf, die Gemeinsamkeiten in eine eigene Funktion oder Klasse auszulagern und diese zu verwenden.

### 1.3 Codeausschnitte

Codeausschnitte<sup>1</sup> sind vorgefertigte Ausschnitte aus Code, die Sie schnell in Ihren Code einfügen können. Beispielsweise erstellt der `for`-Codeausschnitt eine leere `for`-Schleife.

<i>Kürzel</i>	<i>Erklärung</i>
<code>if</code>	Erstellt eine Gerüst für eine if-Verzweigung.
<code>for</code>	Erstellt eine Gerüst für eine for-Schleife.
<code>foreach</code>	Erstellt eine Gerüst für eine foreach-Schleife.
<code>while</code>	Erstellt eine Gerüst für eine while-Schleife.
<code>ctor</code>	Erstellt einen Konstruktor für die enthaltende Klasse.
<code>cw</code>	Erstellt einen Aufruf an <code>Console.WriteLine</code> .
<code>mbox</code>	Erstellt einen Aufruf an <code>System.Windows.Forms.MessageBox.Show</code> .
<code>prop</code>	Erstellt eine Deklaration mit automatisch implementierter Eigenschaft.
<code>propg</code>	Erstellt eine schreibgeschützte automatisch implementierte Eigenschaft mit einem privaten <code>set</code> -Accessor.
<code>propfull</code>	Erstellt eine Eigenschaftendeklaration mit <code>get</code> -Accessor und <code>set</code> -Accessor.
<code>try</code>	Erstellt einen <code>try-catch</code> -Block.

Tabelle 1: Nicht erschöpfende Auswahl an nützlichen Codeausschnitten

Es empfiehlt sich, Codeauschnitte immer zu verwenden. Zum einen ist es deutlich schneller, als alles von Hand zu schreiben und zum anderen vermeidet man Tippfehler.

### 1.4 Das Ausgabefenster

Unter **ANSICHT · AUSGABE** kann das Ausgabefenster im Visual Studio geöffnet werden. Dieses Ausgabefenster ähnelt der Eingabeaufforderung. Will man in diesem Fenster etwas ausgeben, so bedient man sich des Befehls `Console.WriteLine()`.

Warum ist dieses Ausgabefenster hilfreich? Man kann sich zur Programmlaufzeit Informationen aus dem Programm (Inhalt von Variablen...) anzeigen lassen, ohne dass es später der Endbenutzer mitbekommt. Vergisst man eine der `Console.WriteLine()`-Anweisungen im Programm, so ist es nicht weiter schlimm.

---

<sup>1</sup> Alle Codeausschnitte finden sich unter <https://docs.microsoft.com/de-de/visualstudio/ide/visual-csharp-code-snippets?view=vs-2019>

## 2 Abfangen von ungültigen Benutzereingaben

Zur Validierung von Benutzereingaben hat sich die *fail fast*-Methode<sup>2</sup> bewährt: Dabei wird die Benutzereingabe auf Gültigkeit geprüft und im Fehlerfall eine Ausnahme ausgelöst, die detailliert beschreibt, welche Benutzereingabe fehlerhaft war. Wichtig hierbei ist es, die *fail fast*-Anweisungen immer innerhalb eines `try-catch`-Blocks zu verwenden.

### 2.1 Konvertierung eines Strings mit TryParse

Sie haben bereits die Klasse `Convert` mit ihren verschiedenen Methoden kennengelernt, die Daten eines Datentyps in einen anderen umwandelt. So wurde sie beispielsweise oft dazu verwendet, um den in eine `TextBox` eingegebenen Text in eine Zahl umzuwandeln:

```
1 int textAlsZahl = Convert.ToInt32(txtEingabe.Text);
```

Für einfache Testzwecke war dies ausreichend. Allerdings birgt diese Methode den entscheidenden Nachteil, dass das Programm abstürzt, sobald der Benutzer etwas in die Textbox eingibt, das nicht als Zahl interpretiert werden kann. Wie man diesen Fall abfängt und eine entsprechende Fehlermeldung ausgibt, sehen wir uns im Folgenden an.

Mithilfe der `TryParse`-Methode kann ein `string` in ein Zahlen- oder Datumsformat umgewandelt werden. Die Methode liefert dabei einen `Boolean`-Wert zurück, der angibt, ob die Umwandlung erfolgreich war. Schlägt die Umwandlung fehl (Rückgabe = `false`), so kann eine Ausnahme vom Typ `ArgumentException` ausgelöst werden. Als Argument dieser Ausnahme kann der Fehlertext übergeben werden.

```
1 private void BtnSpeichern_Click(object sender, EventArgs e)
2 {
3     try
4     {
5         // Fail fast : Abfangen von ungültigen Eingaben
6         if (!double.TryParse(reparatur.Kosten.ToString(), out double
7             kosten)) throw new ArgumentException("Bitte Kosten eingeben!");
8         if (kosten <= 0) throw new ArgumentException("Bitte gültige
9             Kosten eingeben!");
10    }
11    catch (Exception ex)
12    {
13        // Meldung im Fehlerfall ausgeben
14        MessageBox.Show(ex.Message);
15    }
16 }
```

---

<sup>2</sup>Weitere Informationen:

<https://enterprisecraftsmanship.com/posts/fail-fast-principle/>

```

13 }
14 }

```

Listing 1: Umwandlung von `string` in `double` mithilfe der `TryParse()`-Methode.

Das Ausrufezeichen vor `double.TryParse()` invertiert das Ergebnis - schlägt die Umwandlung fehl, so wird die Anweisung hinter der `if`-Abfrage ausgeführt.

Es ist wichtig, die Umwandlung innerhalb eines `try-catch`-Blocks auszuführen und die Fehlermeldung abzufangen. Die Fehlermeldung kann mit `MessageBox.Show(ex.Message)` ausgegeben werden.

Die `TryParse`-Methode kann auf eine Vielzahl von Datentypen angewandt werden, z.B. `double.TryParse()`, `Int32.TryParse()` oder `DateTime.TryParse()`.

Schauen wir uns die Funktion einmal genauer an:

```

bool erg = double.TryParse(reparatur.Kosten.ToString(), out double kosten)
      1         2           3                               4

```

1. `bool erg`: In der Variable „erg“ vom Typ `bool` soll das Ergebnis der Umwandlung gespeichert werden. Das Ergebnis kann zwei Zustände einnehmen:
  - `true`: Die Umwandlung war erfolgreich
  - `false`: Die Umwandlung schlug fehl
2. `double`: In diesen Datentyp soll der übergebene String konvertiert werden
3. Der hier übergebene `string` soll konvertiert werden
4. Die Funktion liefert außerdem das Ergebnis der Umwandlung zurück:
  - `out`: Dieses Schlüsselwort bedeutet, dass die nachfolgende Parameter von der Funktion zurückgegeben wird
  - `double`: Der Rückgabewert ist vom Typ `double`
  - `kosten`: Der Rückgabewert soll in der Variable mit dem Namen „kosten“ gespeichert werden.

## 2.2 Abfrage eines Strings auf null oder empty

Ein `string` sollte immer auf `null` (String-Objekt existiert nicht) oder `empty` (Zeichenfolge ist leer) überprüft werden. Dies geschieht mit der `string.IsNullOrEmpty()`-Methode. Falls der `string` `null` oder leer ist, so liefert die Methode `true` zurück.



```
1 private void BtnSpeichern_Click(object sender, EventArgs e)
2 {
3     try
4     {
5         if (string.IsNullOrEmpty(MeinFahrzeug.Halter)) throw new
            ArgumentException("Bitte Halter eingeben!");
6     }
7     catch (Exception ex)
8     {
9         MessageBox.Show(ex.Message);
10    }
11 }
```

Listing 2: Abfrage eines strings auf null oder empty.

## 3 Arbeiten mit mehreren Formularen

Um dem Projekt ein neues Formular hinzuzufügen, wählen Sie **PROJEKT · FORMULAR HINZUFÜGEN** aus.

### 3.1 Formular aufrufen

Erstellen Sie auf Ihrem Hauptformular einen Button, mit dem Sie das neue Formular aufrufen können. In die **Click**-Methode dieses Buttons übernehmen Sie den Code zum Aufruf des Formulars. Im folgenden Beispiel ist **frmNeueForm** der Name des neu erstellen Formulars:

```
1 var frm2 = new frmNeueForm();  
2 frm2.Show();
```

Listing 3: Neues Formular aufrufen.

### 3.2 Rückgabe des Formularaufrufs auswerten

Sie können die Rückgabe des Formulars auswerten, indem Sie in Ihrem Hauptformular das neue Formular mit **frm2.ShowDialog()** aufrufen und das Ergebnis vom Typ **DialogResult** auswerten.

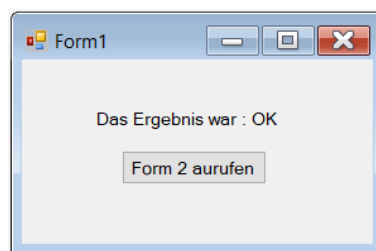


Abbildung 1: Form1: Aufruf und Auswerten der Rückgabe von Form2

```
1 var frm2 = new frmForm2();  
2 DialogResult dr = frm2.ShowDialog();  
3 if (dr == DialogResult.OK) LblAusgabe.Text = "Das Ergebnis war:OK";  
4 if (dr == DialogResult.Cancel) LblAusgabe.Text = "Das Ergebnis war:  
   Abbrechen";
```

Listing 4: Aufruf des neuen Formulars und Auswertung der Rückgabeparameter.

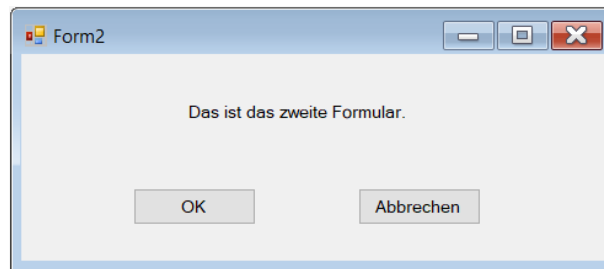


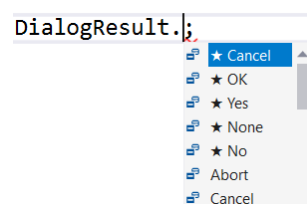
Abbildung 2: Form2: Buttons OK und ABBRECHEN

Im Code des zweiten Formulars muss die Rückgabe vom Typ `DialogResult` erstellt werden:

```
1 private void BtnOK_Click(object sender, EventArgs e)
2 {
3     DialogResult = DialogResult.OK;
4     Close();
5 }
6
7 private void BtnAbbrechen_Click(object sender, EventArgs e)
8 {
9     DialogResult = DialogResult.Cancel;
10    Close();
11 }
```

Listing 5: Rückgabe eines `DialogResult` in der Form die aufgerufen wurde

Neben `OK` und `Cancel` gibt es noch weitere Rückgabe-Möglichkeiten, die sich einfach mit *IntelliSense* herausfinden lassen:

Abbildung 3: Von *IntelliSense* vorgeschlagene Rückgabewerte

### 3.3 Datenaustausch zwischen Formularen

Sollen Daten, z.B. in Form eines Objekts, zwischen zwei Formularen ausgetauscht werden, so empfiehlt es sich, in dem Formular, das aufgerufen werden soll, eine Eigenschaft vom Typ dieses Objekts als `public` zu deklarieren. Im Folgenden Beispiel soll ein Objekt vom Typ `FAHRZEUG` übergeben werden:

```
1 public partial class frmForm2 : Form
2 {
3     public Fahrzeug MeinFahrzeug; // Eigenschaft deklarieren
4
5     public Form2()
6     {
7         InitializeComponent();
8     }
9 }
```

Listing 6: Form2: Öffentliche Eigenschaft in Form2 deklarieren

In der Hauptform muss folgendes passieren:

1. Ein neues Objekt vom Typ `Fahrzeug` muss erzeugt werden.
2. Ein neues Objekt vom Typ `frmForm2` muss erzeugt werden.
3. Das `Fahrzeug`-Objekt wird der neuen Form übergeben.
4. Die neue Form wird aufgerufen.

```
1 private void BtnForm2Aufrufen_Click(object sender, EventArgs e)
2 {
3     // Fahrzeug-Objekt und Form2-Objekt erzeugen
4     var neuesFahrzeug = new Fahrzeug();
5     var neueForm = new frmForm2();
6
7     // Daten übergeben
8     neueForm.MeinFahrzeug = neuesFahrzeug;
9
10    // Form 2 aufrufen
11    neueForm.ShowDialog();
12 }
```

Listing 7: Form1: Aufruf und Datenübergabe an die Form die aufgerufen wird

Da das übergebene Objekt `neuesFahrzeug` ein Referenztyp ist, kann es in FORM2 bearbeitet und danach in FORM1 weiterverwendet werden.

Alternativ können Sie auch in der aufgerufenen Form eine Eigenschaft mit der Sichtbarkeit `public` deklarieren, die Sie in der aufrufenden Form auswerten.

### 3.4 Anwendung beenden

Mit `Application.Exit()` beenden Sie die Anwendung.

Ein Formular schließen Sie mit `Close()`. Ist nur ein Formular aktiv, so beendet dies auch die Anwendung.

Um eine Funktion zu erstellen, die ausgeführt wird, wenn die Anwendung beendet wird, klicken Sie im Eigenschaftsfenster der Form auf **METHODEN** (gelber Blitz) und dann doppelt auf **FormClosing**. Um doch zu verhindern, dass das Formular geschlossen wird, verwenden Sie in dieser Methode die Anweisung `e.Cancel = true;`

## 4 Klasse String für Zeichenketten

Die Klasse `string` für Zeichenketten stellt eine wichtige Klasse in C# dar. In einer Variablen des Datentyps `string` lassen sich Zeichenfolgen unverändert speichern.

Da `string` eine Klasse ist, stehen Methoden und Eigenschaften zur Verfügung.

### 4.1 Eigenschaften

`string` besitzt zwei Eigenschaften: `Length` und `Chars`. Dabei liefert `Length` die Anzahl der Zeichen zurück und mithilfe von `Chars` können Sie auf ein einzelnes Zeichen zugreifen.

#### 4.1.1 Length

Die Eingabe von

```
1 string text = "Hallo";  
2 Console.WriteLine("Länge des Strings: " + text.Length);
```

liefert als Ausgabe:

```
1 Länge des Strings: 5
```

#### 4.1.2 Chars

Die Eingabe von

```
1 string text = "Hallo";  
2 Console.WriteLine("Zeichen an der Stelle 2: " + text.Chars[2]);
```

liefert als Ausgabe:

```
1 Zeichen an der Stelle 2: l
```

### 4.2 Methoden

`string` besitzt eine Vielzahl von Methoden. Hier gehen wir exemplarisch nur auf die Methode `Trim()` ein, die Leerzeichen am Anfang und Ende einer Zeichenkette entfernt. *IntelliSense* liefert jedoch jederzeit einen guten Überblick, welche weiteren Methoden Ihnen zur Verfügung stehen.

### 4.2.1 Trim: Leerzeichen am Anfang und Ende entfernen

Über die Methode `Trim` entfernen Sie die Leerzeichen am Anfang und Ende einer Zeichenkette.

```
1 string text = "  Hallo  ";
2 Console.WriteLine(">" + text.Trim() + "<");
```

liefert als Ausgabe:

```
1 >Hallo<
```

Die Pfeile dienen der Verdeutlichung, dass die Leerzeichen entfernt wurden.

## 4.3 Formatierung von Strings

Zur Formatierung der Ausgabe einer Zeichenketten stehen Ihnen zwei Möglichkeiten zur Verfügung: Formatierung mit `String.Format()` oder mit `ToString()`.

### 4.3.1 Formatierungsoptionen

Zur Ausgabeformatierung von Zahlen stehen verschiedene Optionen zur Verfügung<sup>3</sup>:

<i>Zeichen</i>	<i>Beschreibung</i>	<i>Beispiele</i>
0	Ersetzt die Ziffer 0 ggf. durch eine entsprechende vorhandene Ziffer; andernfalls wird die Ziffer 0 in der Ergebniszeichenfolge angezeigt.	1234.5678 ("00000") → 01235
#	Ersetzt das #-Symbol ggf. durch eine entsprechende vorhandene Ziffer; andernfalls wird keine Ziffer in der Ergebniszeichenfolge angezeigt.	1234.5678 ("#####") → 12345
.	Bestimmt die Position des Dezimaltrennzeichens in der Ergebniszeichenfolge.	0.45678 ("0.00") → 0,46
,	Dieses Zeichen wird als Gruppentrennzeichen verwendet und repräsentiert typischerweise den 1000er-Trennpunkt.	2147483647 ("###,##") → 2.147.483.647

Tabelle 2: Benutzerdefinierte Zahlenformatzeichenfolgen

Diese Formatierungsoptionen können bei `String.Format()` oder `ToString()` angegeben werden.

<sup>3</sup>Quelle:

<https://docs.microsoft.com/de-de/dotnet/standard/base-types/custom-numeric-format-strings>

## 5 Datum und Uhrzeit

C# stellt zur Speicherung von Datum und Uhrzeit die Struktur `DateTime` zur Verfügung. Sie liefert zwei wichtige statische Eigenschaften: `Now` (aktuelles Datum und Uhrzeit) und `Today` (aktuelles Datum, Uhrzeit = 00:00:00).

Um das Datum oder die Uhrzeit auszugeben, können drei verschiedene `ToString()`-Methoden verwendet werden:

<i>Methode</i>	<i>Erläuterung</i>
<code>ToString()</code>	liefert Datum und Uhrzeit Ausgabe: 25.02.2020 10:47:31
<code>ToShortTimeString()</code>	liefert nur die Uhrzeit Ausgabe: 10:47
<code>ToShortDateString()</code>	liefert nur das Datum Ausgabe: 25.02.2020

Tabelle 3: Verschiedene `ToString()`-Methoden von `DateTime`.

Dieser Code

```

1 Console.WriteLine("Datum und Uhrzeit: ");
2 Console.WriteLine(DateTime.Now.ToString());
3
4 Console.WriteLine("Datum: ");
5 Console.WriteLine(DateTime.Today.ToString());
6
7 Console.WriteLine("Nur Uhrzeit:" );
8 Console.WriteLine(DateTime.Now.ToShortTimeString());
9
10 Console.WriteLine("Nur Datum: ");
11 Console.WriteLine(DateTime.Now.ToShortDateString());

```

Listing 8: Ausgabe von Datum und Uhrzeit

liefert folgende Ausgabe:

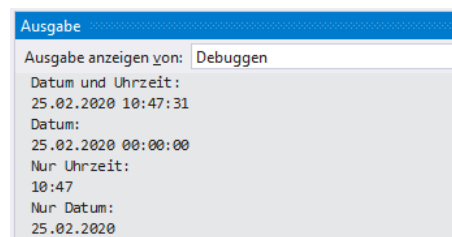


Abbildung 4: Verschiedene Möglichkeiten, `DateTime` in `string` umzuwandeln.



## 5.1 Datumsauswahl mit dem DateTimePicker

Um die Datumsauswahl benutzerfreundlich zu gestalten, kann das Steuerelement `DateTimePicker` verwendet werden.

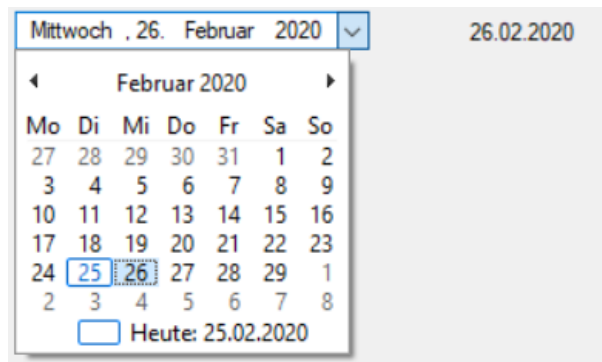


Abbildung 5: Datumsauswahl mit dem `DateTimePicker`.

Die Standardmethode (die mit einem Doppelklick auf den `DateTimePicker` erzeugt wird) ist `ValueChanged` und wird aufgerufen, sobald der Benutzer ein Datum ausgewählt hat. Das ausgewählte Datum ist in der Eigenschaft `Value` gespeichert.

Mit folgendem Code kann einem `string` (zum Beispiel der Eigenschaft `Text` eines `label`s) das Datum zugewiesen werden:

```
1 private void datPicker_ValueChanged(object sender, EventArgs e)
2 {
3     lblDatum.Text = datPicker.Value.ToShortDateString();
4 }
```

Listing 9: Datumszuweisung

## 6 Mathematik

Zum Rechnen stellt C# die Klasse `Math` zur Verfügung. Gibt man im Quellcode `Math.` ein, so hilft *IntelliSense* bei der Auswahl einer geeigneten Funktion.

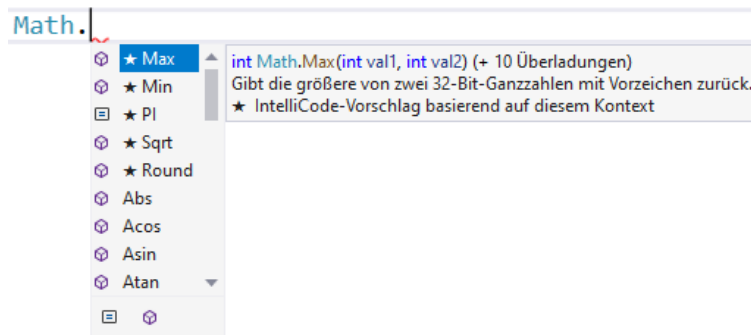


Abbildung 6: Von *IntelliSense* vorgeschlagene Methoden der `Math`-Klasse

Methode	Erläuterung
<code>Ceiling</code>	nächsthöhere ganze Zahl (aus 2,7 wird 3, aus -2,7 wird -2).
<code>Floor</code>	nächstniedrigere ganze Zahl (aus 2,7 wird 2, aus -2,7 wird -3).
<code>Round</code>	nächste ganze Zahl (gerundet, aus 2,7 wird 3, aus -2,7 wird -3, zusätzlich ist die Stellenzahl einstellbar)
<code>Truncate</code>	Abschneiden der Nachkommastellen (aus 2,7 wird 2, aus -2,7 wird -2).
<code>DivRem</code>	Berechnet den Quotienten zweier Zahlen und gibt außerdem den Rest als Ausgabeparameter zurück.

Tabelle 4: Wichtige Methoden der Klasse `Math`

**Division mit Rest** Die Methode `DivRem` erschließt sich nicht auf Anhieb, wird aber doch oft gebraucht. Dazu ein Beispiel: Von der Rechnung  $10 \div 3$  soll sowohl die *Vorkommastelle* als auch der *Rest* berechnet werden.

```
1 var zahl = 10;    // Diese Zahl soll durch 3 geteilt werden
2 int vorkommastelle; // Vorkommastelle des Ergebnisses
3 int rest;         // Rest des Ergebnisses
4
5 vorkommastelle = Math.DivRem(zahl, 3, out rest);
6
7 Console.WriteLine("Operation: 10 / 3 = ");
8 Console.WriteLine("Vorkomma: " + vorkommastelle);
9 Console.WriteLine("Rest: " + rest);
```

Listing 10: Division mit Rest

Dies führt zur Ausgabe:

```
Operation: 10 / 3 =
Vorkomma: 3
Rest: 1
```

## 7 Zugriff auf Dateien

Um auf Dateien zugreifen zu können, binden Sie den Namensraum `System.IO` ein:

```
1 using System.IO;
```

**Hinweis** Bei Dateizugriffen kann erfahrungsgemäß einiges schiefgehen: Datei existiert nicht, keine Berechtigung... Kapseln Sie daher den Dateizugriff immer in einem try-catch-Block!

**Pfadname in string-Variable speichern** In C# ist der *Backslash* ein Sonderzeichen und kann nicht ohne weiteres in strings stehen. Fügen Sie deswegen ein @ vor den Anführungszeichen des Strings hinzu:

```
1 string pfad = @"C:\temp\datei.txt";
```

### 7.1 Dateien lesen

```
1 string text = File.ReadAllText(@"c:\temp\datei.txt");
```

Listing 11: Lesen einer ganzen Datei

```
1 try
2 {
3     String text = "";
4
5     // 1. Öffnen der Datei
6     StreamReader sr = new StreamReader(@"C:\temp\datei.txt");
7
8     // 2. Einlesen der Datei
9     while (!sr.EndOfStream)
10    {
11        string line = sr.ReadLine();
12        text += line + Environment.NewLine;
13    }
14
15    // 3. Schließen der Datei
16    sr.Close();
17 }
18 catch (Exception ex)
19 {
20     MessageBox.Show(ex.Message);
21 }
```

Listing 12: Datei zeilenweise lesen

## 7.2 Dateien schreiben

```
1 // Neue Datei anlegen
2 File.WriteAllText(@"c:\temp\datei.txt", "Inhalt der Datei");
3
4 // An Datei anhängen
5 File.AppendAllText(@"c:\temp\datei.txt", "Anhang");
```

Listing 13: Schreiben einer ganzen Datei

```
1 try
2 {
3     // 1. Öffnen der Datei
4     StreamWriter sw = new StreamWriter(@"c:\temp\datei3.txt", false);
5     // true: Anhängen, false: Neue Datei
6
7     // 2. Schreiben in die Datei
8     sw.WriteLine("Erste Zeile");
9     sw.WriteLine("Zweite Zeile");
10
11    // 3. Schließen der Datei
12    sw.Close();
13 }
14 catch (Exception ex)
15 {
16     MessageBox.Show(ex.Message);
17 }
```

Listing 14: Datei zeilenweise lesen

### 7.3 csv-Datei einlesen

Excel-Dokumente können im `csv`-Format gespeichert werden. `csv` bedeutet *comma-separated values*, die einzelnen Einträge in dieser Datei sind also durch `;` voneinander abgegrenzt. Um auf die Einträge zuzugreifen, liest man Zeile für Zeile mit `reader.ReadLine()` ein und greift mit `line.Split(';')` auf die Einträge zu.

Folgende Datei soll eingelesen werden:

```
1;SAD-AP-348;VW;Golf;Schreiner, Franz;Schwandorf
2;R-EM-331;BMW;X1;Maurer, Johann;Regensburg
```

Dabei sollen die Einträge in einer Liste von Fahrzeug-Objekten gespeichert werden.

```
1 using (var reader = new StreamReader(@"C:\test.csv"))
2 {
3     List<Fahrzeug> fahrzeugListe = new List<Fahrzeug>();
4
5     while (!reader.EndOfStream)
6     {
7         // Neues Fahrzeug-Objekt erzeugen
8         var neuesFahrzeug = new Fahrzeug();
9
10        // Zeile einlesen
11        var line = reader.ReadLine();
12        // Einträge in dem Array values[] speichern
13        var values = line.Split(';');
14
15        // Eigenschaften einlesen
16        neuesFahrzeug.Nr = Convert.ToInt16(values[0]);
17        neuesFahrzeug.Kennzeichen = values[1];
18        neuesFahrzeug.Marke = values[2];
19        neuesFahrzeug.Modell = values[3];
20        neuesFahrzeug.Halter = values[4];
21        neuesFahrzeug.Ort = values[5];
22
23        // Fahrzeug zur Liste hinzufügen
24        fahrzeugListe.Add(neuesFahrzeug);
25    }
26 }
```

Listing 15: csv-Datei einlesen

## 7.4 Standardpfade

Die Enumeration<sup>4</sup> `Environment.SpecialFolder` Enumeration gibt Enumerationskonstanten an, mit denen Verzeichnispfade für besondere Systemordner abgerufen werden. Folgend eine kurze Auswahl der wichtigsten Konstanten:

<i>Element</i>	<i>Erläuterung</i>
<code>Desktop</code>	Desktop.
<code>MyComputer</code>	Der Ordner Arbeitsplatz.
<code>MyDocuments</code>	Der Ordner Eigene Dateien.
<code>ProgramFiles</code>	Das Verzeichnis für Programmdateien.

Tabelle 5: Enumerationskonstanten für Verzeichnispfade besonderer Systemordner

```
1 // Pfad des Ordners "Eigene Dateien" abrufen
2 string pfad = Environment.GetFolderPath(Environment.SpecialFolder.
    MyDocuments);
```

Listing 16: Standardpfad abrufen

Wenn möglich, sollten diese Standardpfade verwendet werden, um dem Benutzer einen sinnvollen Startpunkt zu geben.

---

<sup>4</sup>Weitere Informationen:

<https://docs.microsoft.com/de-de/dotnet/api/system.environment.specialfolder?view=netframework-4.8>





## 8 Datenbankzugriff mit dem Entity Framework Core

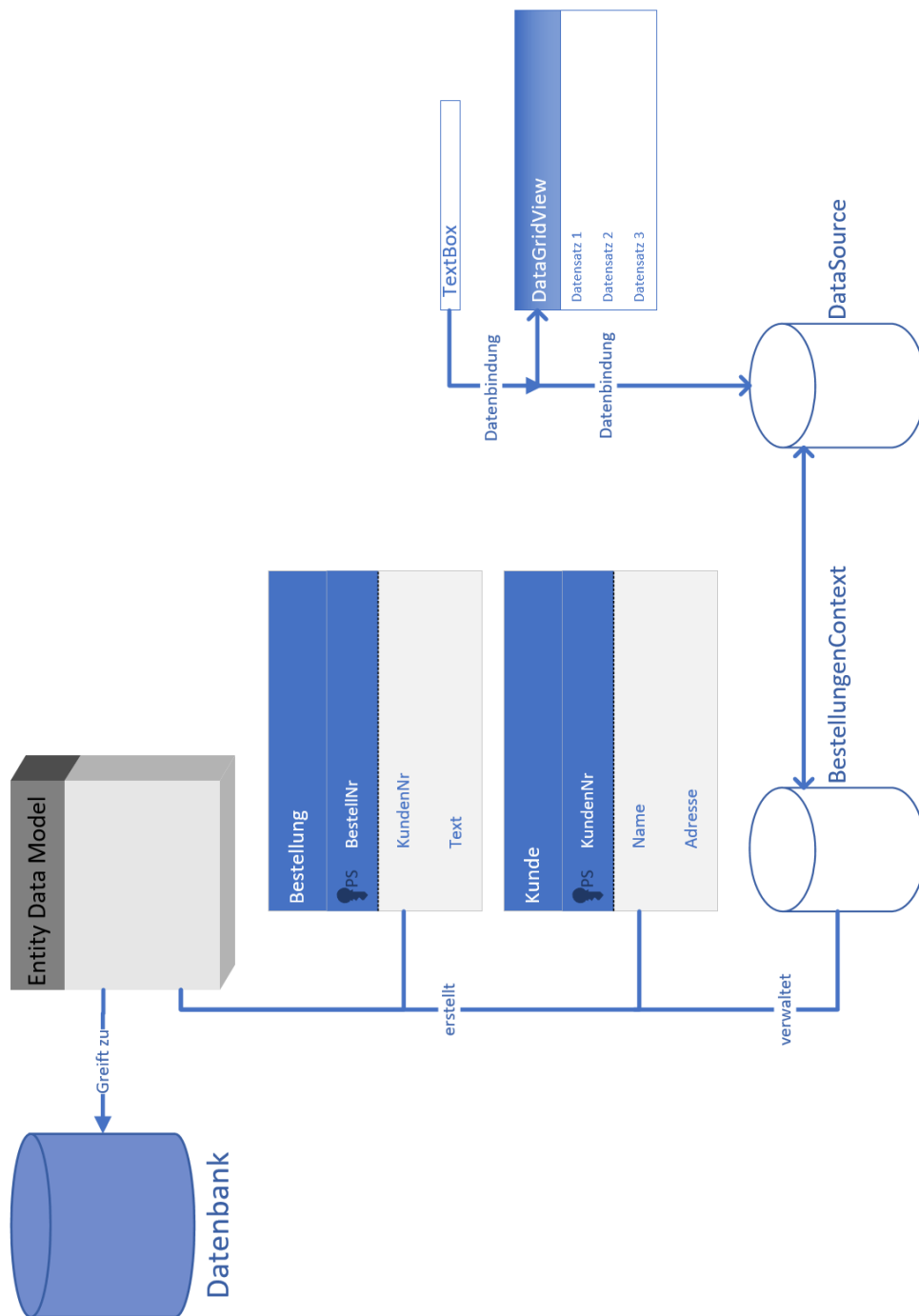


Abbildung 7: Komponenten beim Datenbankzugriff mit dem Entity Framework.

## 8.1 SQL-Server Instanz anzeigen

Alle Instanzen des lokalen SQL-Servers lassen sich in der Kommandozeile mithilfe des Befehls `SqlLocalDB.exe info` anzeigen.

## 8.2 Anlegen einer bestehenden Datenbank in VisualStudio

Zum Anlegen der Datenbank öffnen Sie in Visual Studio den **SQL Server Object Explorer: ANSICHT · SQL SERVER-OBJEKT-EXPLORER**.

**Neue Datenbank anlegen** *Dieser Schritt ist nur notwendig, wenn in der `sql`-Datei keine Datenbank erzeugt wird.* In Ihrer Datenbank-Instanz (z.B. `(localdb)\MSSQLLocalDB`) klicken Sie mit der rechten Maustaste auf **DATENBANKEN** und **NEUE DATENBANK HINZUFÜGEN**.

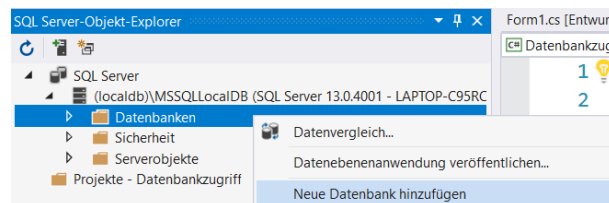


Abbildung 8: Datenbank hinzufügen

Vergeben Sie einen aussagekräftigen Namen (z.B. den Ihres Projekts) und hängen Sie das Suffix `Db` hinten dran.

**Datenbank befüllen** Klicken Sie auf die neu angelegte Datenbank mit der rechten Maustaste und wählen Sie **NEUE ABFRAGE**.

Kopieren Sie den Inhalt ihrer bestehenden Datenbank (Endung: `.SQL`) in das Fenster und führen sie die Abfrage aus:

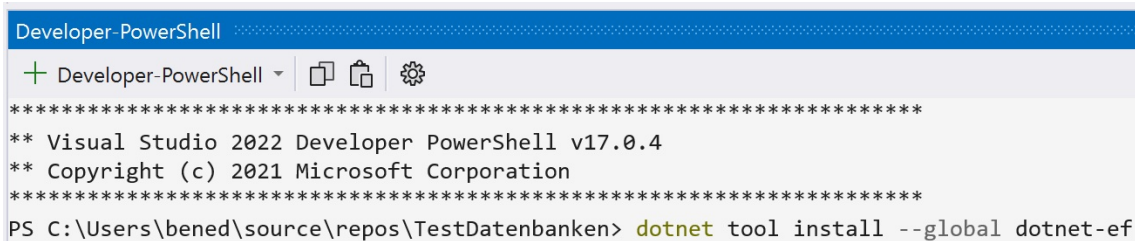


Abbildung 9: Datenbankanfrage ausführen

## 8.3 Installieren der EF CLI Tools

Um mit *database-first* (Klassen in C# automatisch aus bestehender Datenbank erzeugen) zu starten, müssen die *Command Line Interface* (CLI) Tools installiert

werden, da die Befehle zum „Gerüstbau“ über die Kommandozeile ausgeführt werden.



```
Developer PowerShell
+ Developer-PowerShell | [Icons]
*****
** Visual Studio 2022 Developer PowerShell v17.0.4
** Copyright (c) 2021 Microsoft Corporation
*****
PS C:\Users\bened\source\repos\TestDatenbanken> dotnet tool install --global dotnet-ef
```

Abbildung 10: Installieren der EF CLI Tools

Mit folgendem Befehl können Sie prüfen, ob die Tools erfolgreich installiert wurden:

```
1 dotnet ef
```

Listing 17: Überprüfen der EF Tools

Sollten Sie die EF Tools updaten müssen, verwenden Sie folgenden Befehl:

```
1 dotnet tool update --global dotnet-ef
```

Listing 18: Update der EF Tools

## 8.4 NuGet-Paket einbinden

Führen Sie nun folgenden Code ebenfalls in der Developer-PowerShell aus:

```
1 dotnet add package Microsoft.EntityFrameworkCore.Design
2 dotnet add package Microsoft.EntityFrameworkCore.SqlServer
```

Listing 19: Einbindung der benötigten NuGet-Pakete

Damit binden Sie die erforderlichen NuGet-Pakete in ihr Projekt mit ein.

## 8.5 Scaffolding der bestehenden Tabellen in das C# Projekt

Über die CLI Tools können Sie auf Grundlage der bestehenden Datenbank automatisch die nötigen Klassen erstellen lassen, die zur Kommunikation mit der Datenbank notwendig sind. Diesen Vorgang nennt man *Scaffolding* (Gerüstbau).

Geben Sie dazu folgendes in die Developer-PowerShell ein:

```
1 dotnet ef dbcontext scaffold "Server=(localdb)\mssqllocaldb;
    Database=AutowerkstattDb;Trusted_Connection=True;" Microsoft.
    EntityFrameworkCore.SqlServer -o Models
```

Listing 20: Scaffolding

Setzen Sie dabei bei *Database*= den Namen Ihrer Datenbank ein.

**Hinweis:** Falls Sie bei diesem Schritt eine Fehlermeldung erhalten, befinden Sie sich möglicherweise nicht in dem Ordner, in dem sich auch ihr Projekt befindet. Der Pfad, an dem Sie sich befinden, wird Ihnen in der PowerShell angezeigt. Über den Befehl `ls` können Sie die Dateien und Verzeichnisse in diesem Pfad anzeigen. In ein untergeordnetes Verzeichnis wechseln Sie mit dem Befehl `cd` gefolgt von dem Namen des Verzeichnisses. Wechseln Sie in Ihr Projektverzeichnis und führen Sie dort das Scaffolding erneut aus.

```
PS C:\Users\bened\source\repos\TestDatenbanken> ls

Verzeichnis: C:\Users\bened\source\repos\TestDatenbanken

Mode                LastWriteTime         Length Name
----                -
d-----          20.02.2022    12:27             TestDatenbanken
-a----          20.02.2022    12:25         1151 TestDatenbanken.sln

PS C:\Users\bened\source\repos\TestDatenbanken> cd .\TestDatenbanken\
PS C:\Users\bened\source\repos\TestDatenbanken\TestDatenbanken> ls

Verzeichnis: C:\Users\bened\source\repos\TestDatenbanken\TestDatenbanken

Mode                LastWriteTime         Length Name
----                -
d-----          20.02.2022    12:27             bin
d-----          20.02.2022    12:43             obj
-a----          20.02.2022    12:25         166 Form1.cs
-a----          20.02.2022    12:25        1245 Form1.Designer.cs
-----          20.02.2022    12:27        5817 Form1.resx
-a----          20.02.2022    12:25         512 Program.cs
-a----          20.02.2022    12:25         306 TestDatenbanken.csproj
-a----          20.02.2022    12:25         278 TestDatenbanken.csproj.user
```

Abbildung 11: Wechsel vom Projektmappenverzeichnis in das Projektverzeichnis

In Ihrem Projekt wurde nun der Ordner **Models** erzeugt. In diesem Finden Sie das *Kontext-Objekt* sowie die Klassen, die Ihre Tabellen der Datenbank abbilden. Mit diesen können Sie nun arbeiten.

## 8.6 Datenbindung an Windows Forms Elemente

Zur Anzeige einer Tabelle in *WinForms* verwendet man das **Data Grid View**. Zum Verwalten der Daten wird eine **Binding Source** verwendet. Anzeige und Bearbeitung/Verwaltung der Daten geschehen also in zwei verschiedenen Objekten. Um das

**Data Grid View** an eine Datenquelle (**Binding Source**) zu binden, muss diese zuerst angelegt werden. Klicken Sie dazu auf das kleine Dreieck in der oberen linken Ecke des Steuerelements.

Wählen Sie nun *Objektdatenquelle hinzufügen..*

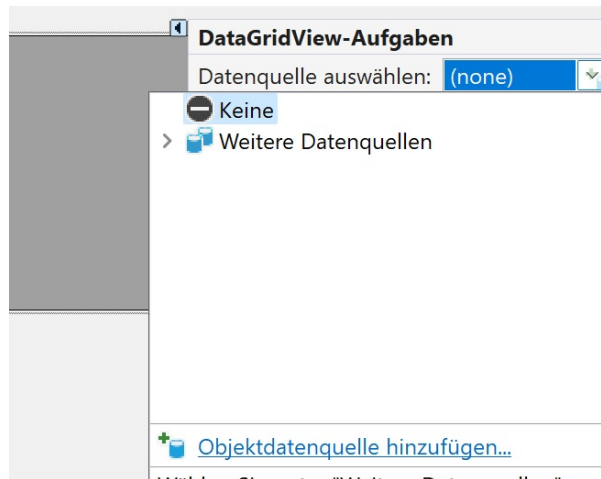


Abbildung 12: Objektdatenquelle hinzufügen.

Wählen Sie in dem Auswahlfenster nun die Tabelle aus der Datenbank, die Sie mit diesem **Data Grid View** verknüpfen wollen.

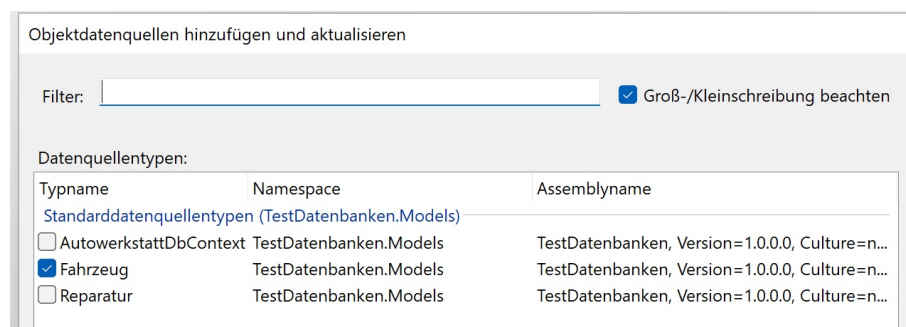


Abbildung 13: Tabelle der Datenbank auswählen.

Bestätigen Sie mit **Ok**. Sie können nun diese Datenquelle auswählen:

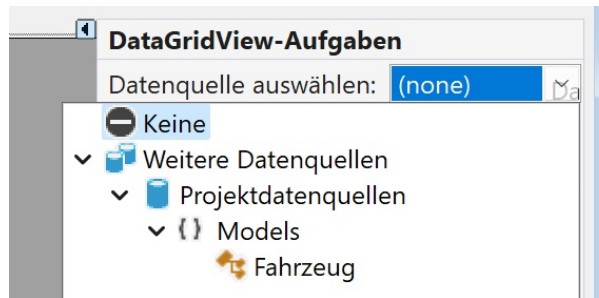


Abbildung 14: Datenquelle auswählen.

Oft ist in dem **Data Grid View** auch eine Spalte enthalten, die für den Benutzer keinen Sinn ergibt. Diese Spalte wird im Hintergrund benötigt, um in relationalen Datenbanken Beziehungen zwischen Tabellen abzubilden. Sie können diese Spalte aus dem **Data Grid View** entfernen, indem Sie erneut auf das kleine Dreieck klicken und die Spalten der Tabelle anpassen.

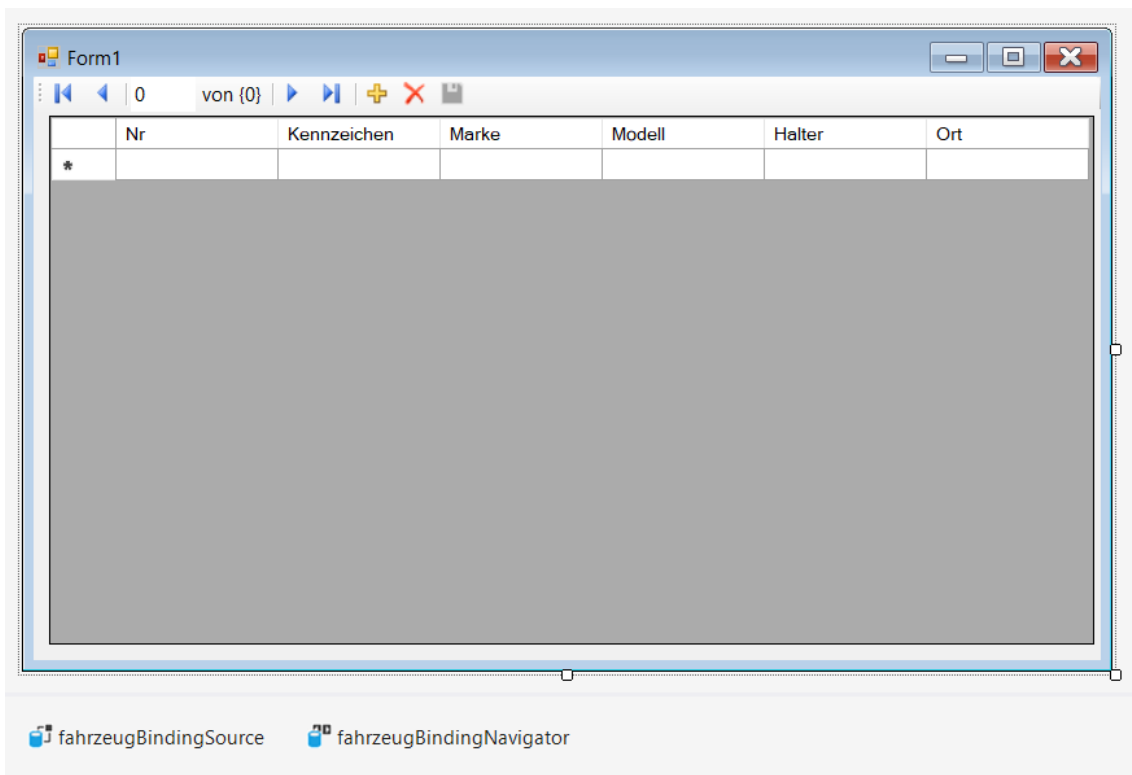


Abbildung 15: Datenquellen im Formular.

**Hinweis** Das **DataGridView**-Control ist gebunden an die **fahrzeugBindingSource**.

Speichern Sie und erstellen Sie die Projektmappe neu, bevor Sie fortfahren.

Weiterhin können die Benutzeroptionen HINZUFÜGEN, BEARBEITEN und LÖSCHEN deaktiviert werden. Dann können die Daten nicht in der Tabelle selbst bearbeitet werden. Dies ist meistens nicht gewünscht.

Ein wichtiges Ereignis des **Data Grid View** ist das **CellDoubleClick**-Event, das sie in den Eigenschaften finden, wenn Sie den Button EREIGNISSE (Symbol: gelber Blitz) drücken. Führen Sie nun einen Doppelklick in die Zelle neben dem **CellDoubleClick** aus, so wird eine Methode erstellt, die aufgerufen wird, sobald der Nutzer auf einer Zeile des **Data Grid View** doppelklickt.

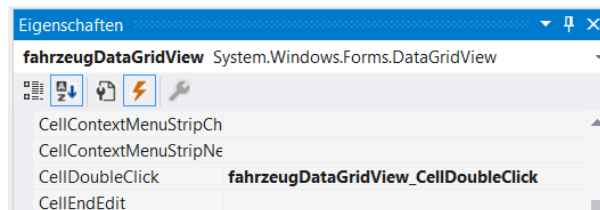


Abbildung 16: Doppelklick-Ereignis des Data Grid View.

### 8.6.1 Context-Instanz erzeugen und Datenquelle binden

Um Daten aus der Datenbank abzurufen, muss im Code eine Instanz des **AutoWerkstattContext** erzeugt werden. Die Daten werden als Datenquelle der **fahrzeugBindingSource** gesetzt.

```
1 public partial class frmWerkstatt : Form
2 {
3     // Eigenschaft deklarieren
4     AutowerkstattContext ctx;
5
6     public frmWerkstatt()
7     {
8         InitializeComponent();
9     }
10
11     private void frmWerkstatt_Load(object sender, EventArgs e)
12     {
13         // Neue Instanz erzeugen
14         ctx = new AutowerkstattContext();
15         // Datenquelle zuweisen
16         fahrzeugBindingSource.DataSource = ctx.Fahrzeugs.ToList();
17     }
18 }
```

Listing 21: Zuweisung der Datenquelle

### 8.6.2 PositionChanged-Ereignis der Datenquelle

Wenn im `DataGridView` der Cursor auf eine andere Zeile gesetzt wird, so wird in der dazugehörigen `DataSource` das `PositionChanged`-Ereignis aufgerufen. Möchten Sie nun immer wenn sich die Auswahl im `DataGridView` ändert eine Funktion ausführen, so wählen Sie in ihrem Formular die gewünschte `BindingSource` aus und klicken in deren Eigenschaftsfenster auf `METHODEN` (gelber Blitz). Dort führen Sie in der Zeile `PositionChanged` einen Doppelklick aus. In der nun automatisch erzeugten Funktion können Sie implementieren, wie ihr Programm reagieren soll, wenn die Auswahl im `DataGridView` verändert wurde.

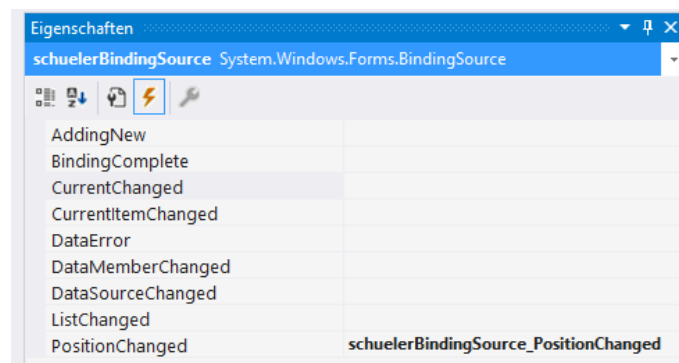


Abbildung 17: `PositionChanged`-Ereignis der `BindingSource`.

### 8.6.3 Label oder Textbox

Bei den Steuerelementen `Label` und `TextBox` kann die Eigenschaft `Text` an eine Datenquelle gebunden werden. Stellen Sie dazu in den Eigenschaften des Steuerelements unter `(DATABINDINGS)` · `TEXT` die Datenquelle ein.

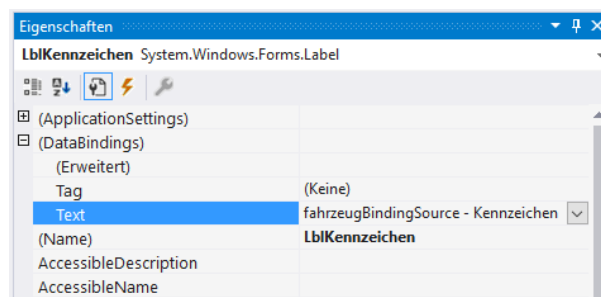


Abbildung 18: Datenbindung der Eigenschaft `Text` an eine `BindingSource`.



### 8.6.4 Combobox

Um eine ComboBox an Daten zu binden, sind vor allem drei Eigenschaften interessant:

<i>Eigenschaft</i>	<i>Erläuterung</i>
<code>DataSource</code>	Datenquelle.
<code>DataMember</code>	Gibt an, welches Merkmal in der ComboBox angezeigt werden soll.
<code>ValueMember</code>	Gibt an, welcher Wert tatsächlich für dieses Element verwendet werden soll.

Tabelle 6: Datenbindung einer ComboBox

Als `ValueMember` wird meist der Primärschlüssel verwendet, als `DataMember` gibt man eine für den Benutzer aussagekräftige Eigenschaft an.

Das Standard-Ereignis einer ComboBox ist das `SelectedIndexChanged`-Ereignis. Über die Eigenschaft `ComboBox.SelectedValue` ruft man den Wert ab, der in der Eigenschaft gespeichert ist, die man unter `ValueMember` angegeben hat.

```

1 private void comboBox1_SelectedIndexChanged(object sender,
    EventArgs e)
2 {
3     // Aktuelles Fahrzeug aus der Binding-Source
4     int fahrzeugNummer = Convert.ToInt32(comboBox1.SelectedValue);
5
6 }

```

Listing 22: Beispiel: Abruf des Primärschlüssels aus einer ComboBox

## 8.7 Code um Änderungen in der Datenbank zu speichern

### 8.7.1 Neuen Datensatz anlegen

```

1 // Neues Objekt anlegen
2 Fahrzeug fzg = new Fahrzeug();
3 fzg.Kennzeichen = "R-AP-209";
4 fzg.Marke = "VW";
5 fzg.Modell = "Golf";
6 fzg.Halter = "Maier, Anton";
7 fzg.Ort = "Regensburg";
8
9 // Objekt zu Modell hinzufügen
10 ctx.Fahrzeugs.Add(fzg);
11 // Änderungen in der Datenbank speichern
12 ctx.SaveChanges();

```

Listing 23: Neuen Datensatz anlegen

### 8.7.2 Datensatz löschen

```
1 // Ausgewähltes Objekt
2 Fahrzeug fzg = fahrzeugBindingSource.Current as Fahrzeug;
3
4 // Prüfen, ob Objekt existiert
5 if (fzg == null) return;
6
7 // Objekt aus Modell löschen
8 ctx.Fahrzeugs.Remove(fzg);
9
10 // Änderungen in der Datenbank speichern
11 ctx.SaveChanges();
```

Listing 24: Datensatz löschen

### 8.7.3 Datensatz ändern

```
1 // Ausgewähltes Objekt
2 Fahrzeug fzg = fahrzeugBindingSource.Current as Fahrzeug;
3
4 // Prüfen, ob Objekt existiert
5 if (fzg == null) return;
6
7 // Datensatz ändern
8 fzg.Kennzeichen = "SAD-EM-123";
9
10 // Änderungen in der Datenbank speichern
11 ctx.SaveChanges();
```

Listing 25: Datensatz ändern

### 8.7.4 Datensatz suchen

```
1 // Aktuellen Datensatz ermitteln
2 Fahrzeug fzg = fahrzeugBindingSource.Current as Fahrzeug;
3
4 // Oder Fahrzeug suchen
5 int nr = 10;
6 Fahrzeug fzg = ctx.Fahrzeugs
7     .Where(k => k.Nr == nr)
8     .First();
9
10 // Datensatz-Nr. ermitteln
11 int pos = fahrzeugBindingSource.IndexOf(fzg);
12
13 // Datensatz auswählen
14 fahrzeugBindingSource.Position = pos;
```

Listing 26: Datensatz suchen

## 9 LINQ-Abfragen

Id	Bezeichnung	Preis	Typ
1	Visual Studio	600,0000	Software
2	Microsoft Office Professional	500,0000	Software
3	Bluetooth Mouse	50,0000	Zubehör
4	Wireless Keyboard	40,0000	Zubehör

Abbildung 19: Artikel-Tabelle.

```
1 ArtikelDbContext ctx = new ArtikelDbContext();
```

Listing 27: Datenkontext erstellen

### 9.1 Filtern

```
1 ctx.Artikels  
2 .Where(a => a.Typ == "Zubehör")  
3 .ToList();
```

Listing 28: Nach Eigenschaft filtern

Id	Bezeichnung	Preis	Typ
3	Bluetooth Mouse	50,0000	Zubehör
4	Wireless Keyboard	40,0000	Zubehör

Abbildung 20: Ergebnis der Filterung.

Es kann auch nur nach Teilen einer Zeichenkette gefiltert werden:

```
1 ctx.Artikels  
2 .Where(a => a.Typ.Contains("Zube"))  
3 .ToList();
```

Listing 29: Nach Teilstrings filtern

Id	Bezeichnung	Preis	Typ
3	Bluetooth Mouse	50,0000	Zubehör
4	Wireless Keyboard	40,0000	Zubehör

Abbildung 21: Ergebnis der Filterung.

## 9.2 Sortieren

### 9.2.1 Aufsteigend

```
1 ctx.Artikels
2   .OrderBy(a => a.Bezeichnung)
3   .ToList();
```

Listing 30: Ergebnis sortieren (aufsteigend)

Id	Bezeichnung	Preis	Typ
3	Bluetooth Mouse	50,0000	Zubehör
2	Microsoft Office ...	500,0000	Software
1	Visual Studio	600,0000	Software
4	Wireless Keyboard	40,0000	Zubehör

Abbildung 22: Ergebnis der Sortierung (aufsteigend).

### 9.2.2 Absteigend

```
1 ctx.Artikels
2   .OrderByDescending(a => a.Bezeichnung)
3   .ToList();
```

Listing 31: Ergebnis sortieren (absteigend)

Id	Bezeichnung	Preis	Typ
1	Visual Studio	600,0000	Software
2	Microsoft Office ...	500,0000	Software
3	Bluetooth Mouse	50,0000	Zubehör
4	Wireless Keyboard	40,0000	Zubehör

Abbildung 23: Ergebnis der Sortierung (absteigend).

## 9.3 Erster Datensatz (Genau ein Datensatz)

```
1 ctx.Artikels
2   .First();
```

Listing 32: Ersten Datensatz abrufen

Id	Bezeichnung	Preis	Typ
1	Visual Studio	600,0000	Software

Abbildung 24: Ergebnis: Abruf erster Datensatz.

## 9.4 Anzahl Datensätze

```
1 ctx.Artikels.Count();
```

Listing 33: Anzahl der Datensätze abrufen

Ergebnis 4.

## 9.5 Summe bilden

```
1 ctx.Artikels  
2 .Sum(x => x.Preis);
```

Listing 34: Summe bilden

Ergebnis 1190.0.

## 9.6 Maximum

```
1 ctx.Artikels  
2 .Max(x => x.Preis);
```

Listing 35: Maximum

Ergebnis 600.0.

## 9.7 Minimum

```
1 ctx.Artikels  
2 .Min(x => x.Preis);
```

Listing 36: Minimum

Ergebnis 40.0.

## Notizen

