

Richard Kaiser

www.rkaiser.de

C# Aufbaukurs

mit

Microsoft Visual Studio 2015/2017

Inhalt

1 Weiterführende Möglichkeiten der Entwicklungsumgebung.....	1
1.1 Der Quelltexteditor	1
1.1.1 Tastenkombinationen	1
1.1.2 Intellisense.....	2
1.1.3 Die Formatierung des Quelltexts.....	3
1.1.4 Definitionen einsehen.....	4
1.1.5 Symbole suchen	4
1.1.6 Namen umbenennen.....	5
1.1.7 Zeichenfolgen suchen und ersetzen.....	6
1.1.8 Navigieren und Definitionen einsehen	7
1.1.9 Code-Ausschnitte	8
1.1.10 Aufgabenliste	9
1.1.11 Die Fenster von Visual Studio anordnen	9
1.1.12 Einstellungen für den Editor	10
1.1.13 Ein C#-Projekt mit verschiedenen VS-Versionen bearbeiten	11
1.1.14 Projektvorlagen Θ	12
1.1.15 Standardeinstellungen für die Entwicklungsumgebung Θ	14
1.2 Konsolen-Anwendungen und die Windows-Konsole Θ	15
1.3 Debugger, Profiler und Codeanalyse	16
1.3.1 Der Debugger – elementare Möglichkeiten	16
1.3.2 Weitere Möglichkeiten des Debuggers Θ	18
1.3.3 Der integrierte Profiler	23
1.3.4 Codeanalyse.....	24
2 Generische Programmierung	29
2.1 Generische Methoden	29
2.1.1 Abgeleitete und explizit angegebene Typ-Argumente.....	30
2.1.2 Die Rolle des Compilers und des Just-in-time Compilers	32
2.2 Generische Klassen.....	33
2.3 Typparameter-Einschränkungen (Constraints)	35
3 Interface-Klassen	40
3.1 Die Definition und Implementation von Interface-Klassen	40
3.2 Typische Anwendungsfälle und generische Interfaces.....	41
3.3 Interface-Klassen und Vererbung	44
3.4 Die Interface-Klassen <i>IComparable</i> und <i>IComparable<T></i>	45
3.5 Iteratoren, <i>IEnumerator</i> und die <i>foreach</i> -Anweisung	47
3.5.1 Iteratoren und die <i>yield</i> -Anweisung	47
3.5.2 Die interne Implementation von Iteratoren und <i>foreach</i> Θ	49
3.5.3 Namenskonflikte und explizit implementierte Elemente Θ	50
3.5.4 Generische Interface-Klassen: <i>IEnumerable<T></i>	52
3.6 Kovariante und Kontravariante Typ-Parameter	53
3.7 <i>IFormattable</i> : Die Formatierung selbstdefinierter Klassen	55
3.8 Entwurfsmuster (Software Design Patterns).....	56

3.8.1	Das Factory Pattern	56
3.8.2	Abstract Factory	58
3.8.3	Dependency Injection mit Interfaces	60
3.8.4	Dependency Injection mit Templates	62
3.8.5	Dependency Injection und Mocking	62
3.8.6	Das Strategy-Pattern	63
3.8.7	Das Singleton-Pattern	64
4	Exception-Handling	65
4.1	Die <i>try</i> -Anweisung	66
4.2	Die Basisklasse <i>Exception</i>	69
4.3	Einige vordefinierte Exceptions	70
4.4	<i>throw</i> -Anweisungen	72
4.5	Exceptions und IntelliSense	74
4.6	Fehler und Exceptions	74
4.7	Selbstdefinierte Exception-Klassen	76
4.8	Exceptions in einem Exception-Handler auslösen	77
4.9	Die Freigabe von Ressourcen	78
4.9.1	Die <i>try-finally</i> Anweisung	79
4.9.2	Die <i>using</i> -Anweisung	80
4.10	Einige spezielle Themen Θ	81
4.10.1	Nicht behandelte Exceptions in Windows Forms-Anwendungen Θ	81
4.10.2	Die Protokollierung von Exceptions in einem EventLog Θ	82
4.10.3	Exceptions in Konstruktoren und Destruktoren Θ	83
5	Delegat-Typen, Lambda-Ausdrücke und Ereignisse	86
5.1	Delegat-Typen und -Instanzen	86
5.2	Elemente der Delegat-Klassen Θ	90
5.3	Generische Delegat-Typen: <i>Action</i> und <i>Func</i>	91
5.4	Operationen zur Verwaltung von Aufruflisten	93
5.5	Anonyme Methoden	95
5.6	Lambda-Ausdrücke	97
5.7	Abfragen mit einfachen LINQ-Ausdrücken	100
5.7.1	Die Elemente und die Syntax von LINQ-Ausdrücken	100
5.7.2	Die Anzeige von LINQ-Ausdrücken	102
5.7.3	Verzögerte Ausführung	102
5.7.4	<i>Select</i> -Klauseln und anonyme Datentypen	103
5.7.5	Sortieren mit <i>orderby</i>	104
5.7.6	Gruppieren mit <i>group by</i>	105
5.7.7	<i>let</i> -Klauseln	106
5.7.8	Verknüpfen mit <i>Join</i>	107
5.7.9	Abfragen in XML-Daten mit einfachen LINQ-Ausdrücken	108
5.7.10	LINQ-Methoden	110
5.7.11	Paralleles LINQ (PLINQ)	111
5.7.12	Debuggen von LINQ-Abfrageausdrücken	112
5.8	LINQ-Abfrageausdrücke und ihre Methodenaufrufe	113
5.8.1	Methodenaufrufe aus einfachen Abfrageausdrücken	114
5.8.2	LINQ-Methoden, die nicht zu Abfrageausdrücken gehören	115
5.9	Ereignisse (events)	115
5.9.1	Ereignisse bei Windows Forms-Anwendungen	117
6	Laufzeit-Typinformationen und Reflektion	119
6.1	Laufzeit-Typinformationen der Klasse <i>Type</i>	119
6.2	Reflektion mit der Klasse <i>Assembly</i>	121
6.3	Dynamisch erzeugte Datentypen und Plugins	121
7	Attribute	125

7.1	Vordefinierte Attribute.....	126
7.2	Selbstdefinierte Laufzeitattribute.....	127
8	Multithreading und die Task Parallel Library	131
8.1	Threads und Tasks	132
8.2	Multithreading mit der Task Parallel Library (TPL).....	133
8.2.1	Funktionen als Threads starten: <i>Task.Factory.StartNew</i> und <i>Task.Run</i>	134
8.2.2	Parameter ohne Lambda-Ausdrücke übergeben Θ	139
8.2.3	Threads in einer Schleife starten	140
8.2.4	Exceptions	141
8.2.5	Fortsetzungen (Continuations): Tasks nacheinander ausführen	143
8.2.6	Reaktionsfähige Oberflächen: Auf Steuerelemente in Tasks zugreifen.....	145
8.2.7	Verschachtelte und untergeordnete Tasks.....	147
8.2.8	Tasks abbrechen.....	148
8.2.9	Threads Debuggen und parallele Leistungsanalyse.....	149
8.3	Asynchrone Programmierung in C# 5.0: <i>async</i> und <i>await</i>	150
8.3.1	Die Syntax von asynchronen Methoden	151
8.3.2	Der Programmablauf mit <i>await</i>	152
8.3.3	Der Zugriff auf Steuerelemente in asynchronen Funktionen	157
8.3.4	Vordefinierte asynchrone Methoden	158
8.3.5	Selbstdefinierte asynchrone Methoden.....	159
8.3.6	Auf die Ergebnisse von asynchronen Tasks warten	160
8.3.7	Exceptions	161
8.4	Kritische Abschnitte und die Synchronisation von Threads.....	161
8.4.1	<i>lock</i> und die Klasse <i>Monitor</i>	163
8.4.2	Thread-sichere Collections	164
8.4.3	ReaderWriter Locks	166
8.4.4	Interlocked-Operationen und Laufzeit-Vergleiche.....	166
8.4.5	Thread-sichere Methoden	167
8.4.6	Die Klassen <i>Mutex</i> und <i>Semaphore</i> Θ	167
8.5	Actions mit <i>Parallel</i> ausführen: <i>Invoke</i> , <i>For</i> und <i>ForEach</i>	169
8.6	Einige ältere Konzepte Θ	171
8.6.1	Reaktionsfähige Oberflächen mit <i>BackgroundWorker</i> Θ	171
8.6.2	Ereignisbasierte asynchrone Programmierung Θ	174
8.6.3	<i>AsyncResult</i> -basierte asynchrone Programmierung Θ	175
8.6.4	Asynchrone Aufrufe von beliebigen Methoden Θ	175
8.6.5	Multithreading mit der Klasse <i>Thread</i> Θ	177
8.6.6	Threads für Methoden ohne Parameter und ohne Rückgabewert.....	177
8.6.7	Weitere Elemente der Thread-Klasse.....	177
8.6.8	ThreadPool	178
9	Uhrzeiten, Kalenderdaten und Timer	179
9.1	Zeitpunkte mit der Struktur <i>DateTime</i>	179
9.2	Zeitspannen mit der Struktur <i>TimeSpan</i>	180
9.3	Datumsarithmetik	181
9.4	Steuerelemente zur Eingabe von Kalenderdaten und Zeiten	181
9.5	Timer und zeitgesteuerte Ereignisse	182
9.6	Multithreadfähige Timer	182
9.7	Hochauflösende Zeitmessung mit der Klasse <i>Stopwatch</i>	184
9.8	Kulturspezifische Datumsformate und Kalender	185
9.9	Kalenderklassen.....	187
10	TCP-Clients und Server	189
10.1	TCP/IP: IP-Adressen und Port-Nummern	189
10.2	Ein einfacher synchroner Server: <i>TcpListener</i> Grundlagen.....	190
10.3	Ein einfacher synchroner Client: <i>TcpClient</i> Grundlagen.....	192
10.4	Synchrones Ping Pong mit <i>NetStream.Read</i> und <i>-Write</i>	193
10.5	Mit <i>StreamReader</i> und <i>StreamWriter</i> senden und empfangen Θ	194

10.6 Ping Pong asynchron: Client und Server mit *async* und *await* 196


11 Reguläre Ausdrücke 199

1 Weiterführende Möglichkeiten der Entwicklungsumgebung

Die in diesem Abschnitt vorgestellten Möglichkeiten sind für kleinere Projekte wie die Übungen in diesem Buch nicht unbedingt notwendig. Anfänger können diese zunächst übergehen. Für größere Projekte können sie aber doch sehr nützlich sein.

1.1 Der Quelltexteditor



Der Quelltexteditor (kurz: **Editor**) ist das Werkzeug, mit dem die Quelltexte geschrieben werden. Er ist in die Entwicklungsumgebung integriert und kann auf verschiedene Arten aufgerufen werden, wie z.B.



- durch Anklicken seines Registers 
 - über *Ansicht/Code* oder *F7*
 - von einem Formular aus über das Kontextmenü „Code anzeigen“
 - durch einen Doppelklick auf die rechte Spalte eines Ereignisses im Eigenschaftenfenster. Der Cursor befindet sich dann in der Methode, die zu dem angeklickten Ereignis gehört.
 - durch einen Doppelklick auf ein Steuerelement in einem Formular. Der Cursor befindet sich dann in einer Methode, die zu einem bestimmten Ereignis des Steuerelements gehört.
- Auf diese Weise kann man auch eine Methode finden, ohne sie lange im Editor suchen zu müssen.

1.1.1 Tastenkombinationen

Der Editor enthält über Tastenkombinationen zahlreiche Funktionen, mit denen sich nahezu alle Aufgaben effektiv durchführen lassen, die beim Schreiben von Programmen auftreten.

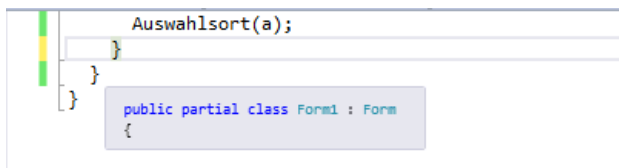
Die nächste Tabelle enthält Funktionen, die vor allem beim Programmieren nützlich sind, und die man in einer allgemeinen Textverarbeitung nur selten findet. Die meisten dieser Optionen werden auch unter *BEARBEITEN/Erweitert* sowie auf der *Text Editor* Symbolleiste (unter *ANSICHT/Symbolleisten*) angezeigt:

Tastenkürzel	Aktion oder Befehl
<i>F5</i> bzw. 	kompilieren und starten, wie <i>Debuggen/Debugging Starten</i>
<i>Umschalt+F5</i>	Laufendes Programm beenden, wie <i>Debuggen/Debugging beenden</i> . Damit können auch Programme beendet werden, die mit  nicht beendet werden können. Versuchen Sie immer zuerst diese Option wenn Sie meinen, Sie müssten Visual Studio mit dem Windows Task Manager beenden.
<i>F6</i>	kompilieren, aber nicht starten
<i>F7</i>	wie <i>Ansicht/Code</i>
<i>Alt/Enter</i>	wie <i>Ansicht/Eigenschaftenfenster</i>
<i>Umschalt+F7</i>	wie <i>Ansicht/Designer</i>

Tastenkürzel	Aktion oder Befehl
<i>F1</i>	kontextsensitive Hilfe
<i>Strg + ´</i> (´ ist das Zeichen links von der Rücktaste) <i>Umschalt+Strg + ´</i>	setzt den Cursor vor die zugehörige Klammer, wenn er vor einer Klammer (z.B. (), {}, [] oder <>) steht markiert den Bereich zwischen den Klammern außerdem noch
<i>Strg+M+M</i> bzw. unter <i>Bearbeiten/Gliedern</i>	ganze Funktionen, Klassen usw. auf- oder zuklappen
<i>Alt+Maus bewegen</i> bzw. <i>Alt+Umschalt+Pfeiltaste</i> (←, →, ↑ oder ↓)	zum Markieren von Spalten, z.B. <pre> /// <summary> /// Clean up any /// </summary> /// <param name= </pre>
<i>Strg+K+C</i> oder <i>Strg+K+U</i> bzw.  oder 	einen markierten Block auskommentieren bzw. die Auskommentierung entfernen

Eine ausführliche Liste der Tastenkombinationen findet man in der MSDN-Dokumentation mit dem Suchbegriff „Visual Studio Tastenkombinationen“.

Bei größeren Programmen ist oft nicht sofort klar, ob eine schließende geschweifte Klammer eine Klasse, einen Namensbereich, eine Funktion oder eine Anweisung (*if*, *while* usw.) beendet. Seit Visual Studio 2015 wird der Anfang zu einer solchen Klammer im Editor eingblendet, wenn man mit dem Mauszeiger darüber fährt:



Weitere Möglichkeiten des Editors, die gelegentlich nützlich sind:

- Die Schriftgröße kann man mit *Strg+Mausrad* vergrößern und verkleinern.
- Wenn man verschiedene Teile eines Textes in verschiedenen Fenstern anschauen (und z.B. vergleichen will), kann man die aktuelle Datei in verschiedenen Fenstern öffnen:



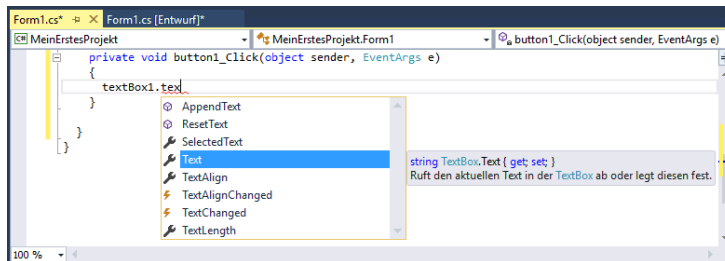
1.1.2 Intellisense

Die folgenden Programmierhilfen beruhen auf einer Analyse des aktuellen Programms. Sie werden zusammen mit einigen weiteren (siehe *Bearbeiten/IntelliSense*) unter dem Oberbegriff **IntelliSense** zusammengefasst:

- Der Editor bietet fast jederzeit eine Auswahl von **zulässigen Sprachelementen** an. Wenn ein Formular z.B. eine *textBox1* enthält, wird in einer *ButtonClick*-Funktion des Formulars nach dem Eintippen von „tex“ *textBox1* angeboten. Außerhalb des Formulars (z.B. am Ende von *Form1.cs*) ist der Name *textBox1* nicht bekannt und wird auch nicht angeboten.
- **Elemente anzeigen und auswählen:** Nachdem man den Namen eines Steuerelements (genauer: einer Klasseninstanz, eines Namensbereichs usw.) und einen Punkt „.“ eingetippt hat, wird eine Liste mit allen Elementen der Klasse oder des Namensbereichs angezeigt. Aus dieser Liste kann man mit der *Enter*-Taste ein Element auswählen.
- **Parameter Info:** Zeigt nach dem Eintippen eines Funktionsnamens und einer öffnenden Klammer die Parameter der Funktion an
- **Quick Info:** Wenn man mit der Maus über einen Namen für ein zuvor deklariertes Symbol fährt (bzw. mit *Strg+K+I*), wird die Deklaration angezeigt. Bei einer vordefinierten Funktion wird außerdem eine Beschreibung angezeigt.

Beispiel: Wenn das Formular eine TextBox *textBox1* enthält, wird nach dem Eintippen von „textBox1.“ eine Liste mit allen Elementen von *textBox1* angezeigt.

Tippt man weitere Buchstaben ein, werden nur die Elemente angezeigt, die diese Zeichenfolge enthalten. In einem weiteren Fenster werden außerdem der Datentyp und eine kurze Beschreibung der Eigenschaft angezeigt.



Tippt man weitere Buchstaben ein, werden nur die Elemente mit diesen Anfangsbuchstaben angezeigt. In einem weiteren Fenster werden außerdem der Datentyp und eine kurze Beschreibung der Eigenschaft angezeigt.

Unterbricht man die Eingabe einer Zeichenfolge (z.B. weil man die Zeile verlässt und anschließend wieder zurückkehrt), bietet Intellisense keine Vorschläge an. Mit *Strg+Leerzeichen* werden dann wieder Vorschläge angezeigt.

Falls der Editor einen **Syntaxfehler** entdeckt, zeigt er das mit einer rot unterstrichenen Wellenlinie an (wie bei Rechtschreibfehlern in Word).

1.1.3 Die Formatierung des Quelltexts

Wenn der Editor erkennt, dass die Eingabe einer Anweisung, eines Blocks usw. abgeschlossen wurde (z.B. nach einer Anweisung beim Eintippen des Semikolons, bei einer Funktion nach dem Eintippen der schließenden geschweiften Klammer „}“ usw.) „verschönert“ der Editor das Layout. Tippt man z.B.

```
int x=17
```

macht der Editor daraus mit dem Eintippen des „;“,

```
int x = 17;
```

Diese automatische Formatierung erfolgt aber nur bei dem Eintippen eines abschließenden Symbols. Entfernt man die Leerzeichen anschließend wieder, wird der Text nicht erneut formatiert.

Dann kann man den Text mit *Bearbeiten/Erweitert/Dokument formatieren* formatieren lassen. Damit wird z.B. aus



dieser Text

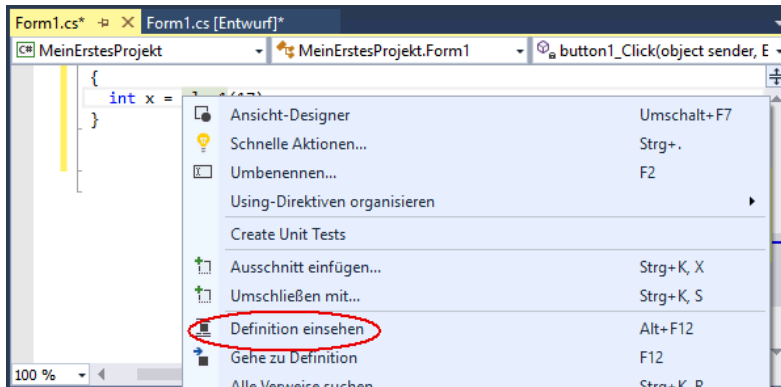


bei dem die Anweisungen einer Funktion in derselben Spalte beginnen.

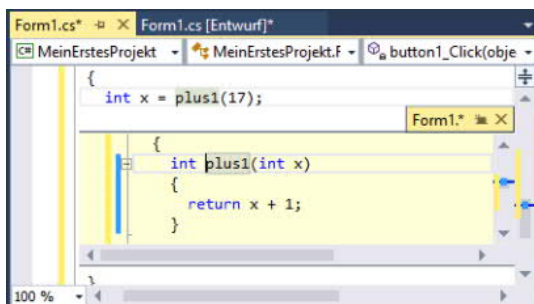
Unter *Extras/Optionen/Text-Editor/C#/Formatierung* gibt es zahlreiche Möglichkeiten, das so erzeugte Layout zu gestalten.

1.1.4 Definitionen einsehen

Beim Aufruf einer Funktion oder der Verwendung einer Variablen besteht immer wieder die Notwendigkeit, die Definition einer Funktion bzw. Variablen anzuschauen. Das ist mit der Option *Definition einsehen* aus dem Kontextmenü zu einer Anweisung besonders einfach, da sie das Suchen und Blättern im Quelltext erspart und man nicht einmal die aktuelle Position im Editor verlassen muss:



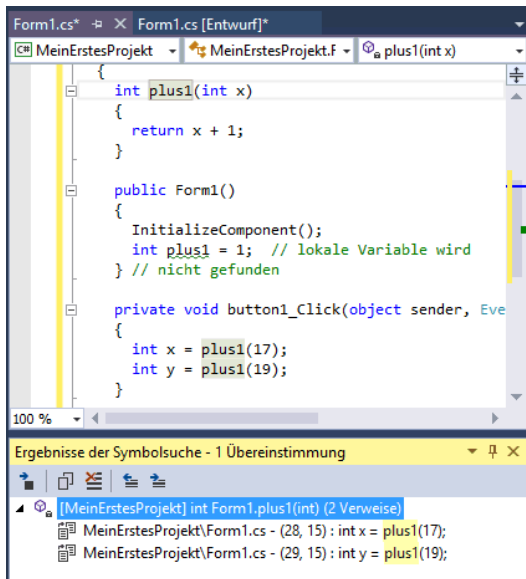
Damit wird ein Fenster mit dem Quelltext eingeblendet, indem man sogar Änderungen vornehmen kann.



Falls man trotzdem noch zur Definition oder Deklaration gehen will, ist das *Gehe zu Definition* bzw. *Gehe zu Deklaration* aus dem Kontextmenü möglich.

1.1.5 Symbole suchen

Alle Aufrufe einer Funktion bzw. alle Stellen, an denen eine Variable verwendet wird, kann man mit der Option „Alle Verweise suchen“ aus dem Kontextmenü im Editor anzeigen lassen:

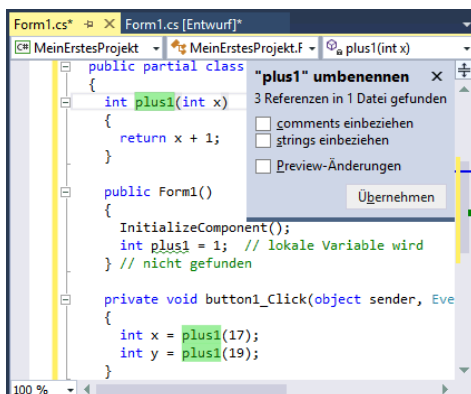


Die Treffer sind hiermit viel präziser als mit *Bearbeiten/Suchen* von Abschnitt 1.1.7.

1.1.6 Namen umbenennen

Namen, die in C# definiert wurden (Namen von Variablen, Funktionen, Klassen usw.) können mit **Umgestalten/Umbenennen** im gesamten Quelltext geändert werden. Da die meisten Zeichenfolgen, die man in einem Programm ersetzen will, solche Namen sind, ist die mit *Bearbeiten/Suchen und Ersetzen* (siehe Abschnitt 1.1.7) verfügbare Option nur noch selten notwendig.

Klickt man im Editor mit der rechten Maustaste den Namen einer Variablen, Methode usw. an, wird im Kontextmenü die Option **Umgestalten/Umbenennen** angeboten, mit der man diese Variable, Methode usw. im gesamten Quelltext umbenennen kann. Wenn der Name anders verwendet wird (z.B. für eine andere Variable mit demselben Namen), wird er nicht geändert. Ändert man hier den Namen, wird die Änderung an allen Stellen angezeigt.



Vor der Durchführung der Änderungen können diese in einer Vorschau angezeigt werden.

Diese Option steht auch nach dem Ändern eines Namens im Editor zur Verfügung. Nach der Änderung wird das geänderte Symbol mit einem gestrichelten Rechteck umrandet und das Glühbirnen-Symbol angezeigt: