

**Richard Kaiser**

*www.rkaiser.de*

# **C++ 11/14/17 Neuerungen in Visual Studio 2017**

# Inhalt

|   |           |
|---|-----------|
| <b>1 Die Entwicklungsumgebung .....</b>   | <b>7</b>  |
| 1.1 Die Sprachversion einstellen .....  | 7         |
| 1.2 Der Quelltexteditor .....   | 7         |
| 1.2.1 Tastenkombinationen .....   | 7         |
| 1.2.2 Intellisense .....  | 8         |
| 1.2.3 Die Formatierung des Quelltexts .....   | 9         |
| 1.2.4 Definitionen einsehen .....   | 10        |
| 1.2.5 Symbole suchen .....  | 10        |
| 1.2.6 Namen umbenennen .....  | 11        |
| 1.2.7 Zeichenfolgen suchen und ersetzen .....   | 12        |
| 1.2.8 Navigieren .....  | 13        |
| 1.2.9 Code-Ausschnitte .....  | 15        |
| 1.2.10 Aufgabenliste .....  | 15        |
| 1.2.11 Klassen und Header-Dateien .....   | 15        |
| 1.2.12 Einstellungen für den Editor $\Theta$ .....  | 16        |
| 1.3 Projektmappen und der Projektmappen-Explorer .....  | 16        |
| 1.4 Der Debugger .....  | 17        |
| 1.4.1 Der Debugger .....  | 17        |
| 1.4.2 Der Debugger – Weitere Möglichkeiten $\Theta$ .....                                       | 20        |
| 1.5 Dokumentationskommentare und Intellisense .....   | 23        |
| <br>  |           |
| <b>2 C++11/C++14/C++17 „core features“ in Visual Studio 2017.....</b>                           | <b>25</b> |
| 2.1 <code>__cplusplus</code> und Makros für die verschiedenen Compiler .....                    | 25        |
| 2.2 Weitere Ganzzahldatentypen in C++11 .....   | 25        |
| 2.2.1 Die Datentypen <code>long long</code> und <code>unsigned long long</code> .....           | 25        |
| 2.2.2 Ganzzahldatentypen fester Breite .....  | 26        |
| 2.3 Trennzeichen bei Zahlliteralen und binäre Literale .....                                    | 26        |
| 2.4 Das Nullzeiger-Literal <code>nullptr</code> .....   | 26        |
| 2.5 <code>&gt;&gt;</code> bei Templates .....   | 27        |
| 2.6 Stark typisierte Aufzählungstypen .....   | 27        |
| 2.7 Typ-Inferenz: Implizite Typzuweisungen mit <code>auto</code> .....                          | 29        |
| 2.8 Initialisiererlisten .....  | 33        |
| 2.8.1 Sichere Konversionen mit Initialisiererlisten .....                                       | 33        |
| 2.8.2 Initialisiererlisten als Parameterlisten .....  | 35        |
| 2.8.3 Einheitliche Initialisierungen: Inkonsistenzen in C++11 und ihre Behebung in C++17 .....  | 36        |
| 2.9 Die bereichsbasierte (range-based) <code>for</code> -Schleife .....                         | 37        |
| 2.10 Synonyme für Datentypen .....  | 39        |
| 2.10.1 Einfache <code>typedef</code> -Deklarationen .....                                       | 39        |
| 2.10.2 Synonyme für Datentypen mit <code>using</code> .....                                     | 39        |
| 2.11 <code>inline</code> Variablen, insbesondere <code>static inline</code> Datenelemente ..... | 40        |
| 2.12 <code>constexpr</code> .....   | 42        |
| 2.12.1 Compilezeit-Konstanten mit <code>constexpr</code> .....                                  | 42        |
| 2.12.2 <code>constexpr</code> Funktionen $\Theta$ .....   | 42        |
| 2.12.3 Unittests zur Compilezeit: <code>static_assert</code> mit <code>constexpr</code> .....   | 43        |
| 2.13 Erweiterungen für die <code>if</code> - und <code>switch</code> -Anweisung .....           | 44        |
| 2.13.1 Init statement für <code>if/switch</code> .....  | 44        |

|        |   |    |
|--------|---|----|
| 2.13.2 | Bedingte Kompilation mit <i>if constexpr</i> .....                | 45 |
| 2.14   | Erweiterungen für Klassen .....                                   | 46 |
| 2.14.1 | Virtuelle Funktionen mit <i>override</i> .....                    | 46 |
| 2.14.2 | <i>final</i> .....  | 49 |
| 2.14.3 | Erweiterte <i>friend</i> -Deklarationen $\Theta$ .....            | 49 |
| 2.14.4 | Die Angaben <i>=delete</i> und <i>=default</i> .....              | 50 |
| 2.14.5 | Delegierende Konstruktoren $\Theta$ .....                         | 52 |
| 2.14.6 | Initialisierer für nicht statische Datenelemente .....            | 52 |
| 2.14.7 | Konvertierende und explizite Konstruktoren .....                  | 53 |
| 2.14.8 | Konversionsfunktionen mit und ohne <i>explicit</i> $\Theta$ ..... | 56 |
| 2.15   | R-Wert Referenzen und Move-Semantik .....                         | 57 |
| 2.15.1 | R-Werte und R-Wert Referenzen .....                               | 57 |
| 2.15.2 | move-Semantik und <i>std::move</i> .....                          | 59 |
| 2.15.3 | Move-Semantik in der C++11 Standardbibliothek .....               | 63 |
| 2.15.4 | Move-Semantik für eigene Klassen .....                            | 64 |
| 2.15.5 | Benchmarks .....  | 66 |
| 2.16   | Attribute .....   | 67 |
| 2.17   | Compiler Feature Tests .....                                      | 68 |
| 2.17.1 | Der Präprozessorkonstanten-Ausdruck <i>__has_include</i> .....    | 68 |
| 2.17.2 | Die <i>__cpp_xxx</i> -Makros .....                                | 68 |
| 2.17.3 | <i>__has_cpp_attribute</i> .....                                  | 69 |
| 2.18   | Einige kleinere Erweiterungen.....                                | 69 |
| 2.18.1 | Array-Referenzen und <i>-Zeiger</i> .....                         | 69 |
| 2.18.2 | Garantierte Copy elision .....                                    | 70 |
| 2.18.3 | Deprecated Features .....   | 71 |
| 2.18.4 | Makros mit ... (Variadic Macros) $\Theta$ .....                   | 71 |

### 3 Funktoren, Funktionsobjekte und Lambda-Ausdrücke..... 73

|       |   |    |
|-------|---|----|
| 3.1   | Funktionen als Objekte und Parameter mit <i>std::function</i> ..... | 73 |
| 3.2   | Der Aufrufoperator ().....  | 76 |
| 3.3   | Prädikate und Vergleichsfunktionen .....                            | 78 |
| 3.4   | Lambda-Ausdrücke.....   | 82 |
| 3.5   | Lambda-Ausdrücke – Weitere Konzepte $\Theta$ .....                  | 88 |
| 3.5.1 | Lambda-Ausdrücke werden zu Funktionsobjekten .....                  | 88 |
| 3.5.2 | Nachstehende Rückgabetypen .....                                    | 89 |
| 3.5.3 | Generische Lambda-Ausdrücke .....                                   | 89 |
| 3.5.4 | Lambda-Ausdrücke höherer Ordnung $\Theta$ .....                     | 90 |

### 4 Templates und die STL..... 91

|       |  |     |
|-------|--|-----|
| 4.1   | Generische Funktionen: Funktions-Templates .....                 | 92  |
| 4.1.1 | Die Deklaration von Funktions-Templates mit Typ-Parametern ..... | 92  |
| 4.1.2 | Spezialisierungen von Funktions-Templates .....                  | 93  |
| 4.1.3 | Funktions-Templates mit Nicht-Typ-Parametern.....                | 98  |
| 4.1.4 | Explizit instanziierte Funktions-Templates $\Theta$ .....        | 100 |
| 4.1.5 | Explizit spezialisierte und überladene Templates .....           | 100 |
| 4.1.6 | Rekursive Funktions-Templates $\Theta$ .....                     | 103 |
| 4.1.7 | Variadische Templates .....                                      | 104 |
| 4.1.8 | fold expressions .....   | 105 |
| 4.2   | Generische Klassen: Klassen-Templates.....                       | 105 |
| 4.2.1 | Die Deklaration von Klassen-Templates mit Typ-Parametern.....    | 106 |
| 4.2.2 | Spezialisierungen von Klassen-Templates.....                     | 106 |
| 4.2.3 | Klassen-Templates mit Nicht-Typ-Parametern.....                  | 111 |
| 4.2.4 | Explizit instanziierte Klassen-Templates $\Theta$ .....          | 112 |
| 4.2.5 | Partielle und vollständige Spezialisierungen $\Theta$ .....      | 113 |
| 4.2.6 | Vererbung mit Klassen-Templates $\Theta$ .....                   | 117 |
| 4.2.7 | Tupel mit <i>&lt;tuple&gt;</i> $\Theta$ .....                    | 118 |
| 4.2.8 | Alias Templates $\Theta$ .....                                   | 119 |
| 4.3   | Type Traits .....  | 119 |
| 4.3.1 | Prüfungen bei der Kompilation: <i>static_assert</i> .....        | 119 |

|          |   |            |
|----------|---|------------|
| 4.3.2    | type traits und <i>static_assert</i> .....  | 120        |
| 4.3.3    | Eine Konstruktion von type traits.....  | 122        |
| 4.3.4    | Die type traits Kategorien.....   | 123        |
| 4.3.5    | type traits zur Steuerung der Übersetzung und Optimierung .....                             | 124        |
| 4.4      | Typ-Inferenz: Mit <i>decltype</i> den Datentyp eines Ausdrucks bestimmen .....              | 125        |
| 4.5      | Kovarianz und Kontravarianz.....  | 127        |
| <b>5</b> | <b>STL-Algorithmen und Lambda-Ausdrücke.....</b>  | <b>129</b> |
| 5.1      | Iteratoren .....  | 129        |
| 5.1.1    | Die verschiedenen Arten von Iteratoren .....  | 130        |
| 5.1.2    | Umkehriteratoren .....  | 131        |
| 5.1.3    | Einfügefunktionen und Einfügeiteratoren.....  | 132        |
| 5.1.4    | Stream-Iteratoren .....   | 133        |
| 5.1.5    | Container-Konstruktoren mit Iteratoren .....  | 134        |
| 5.2      | Globale Iterator-Funktionen $\Theta$ .....  | 135        |
| 5.3      | Lineares Suchen.....  | 136        |
| 5.4      | Suche nach Teilfolgen.....  | 137        |
| 5.5      | Minimum und Maximum .....   | 138        |
| 5.6      | Kopieren und Verschieben von Bereichen .....  | 139        |
| 5.7      | Elemente transformieren und ersetzen .....  | 140        |
| 5.8      | Bereiche sortieren .....  | 141        |
| 5.9      | Numerische Berechnungen.....  | 143        |
| 5.9.1    | Verallgemeinerte numerische Algorithmen .....   | 143        |
| 5.9.2    | Zufallszahlen mit $\langle \text{random} \rangle \Theta$ .....                              | 145        |
| <b>6</b> | <b>Erweiterungen der Standardbibliothek (STL).....</b>                                      | <b>149</b> |
| 6.1      | Move-Semantik und Optimierung auf der Basis von type traits .....                           | 149        |
| 6.2      | Neue Algorithmen in C++11/14: <i>all_of</i> , <i>any_of</i> , <i>none_of</i> $\Theta$ ..... | 149        |
| 6.3      | Erweiterungen für Strings .....   | 151        |
| 6.3.1    | Raw-String-Literale (Rohzeichenfolgen).....   | 151        |
| 6.3.2    | Konversionen zwischen <i>string/wstring</i> und elementaren Datentypen .....                | 153        |
| 6.3.3    | Unicode-Strings .....   | 154        |
| 6.3.4    | Reguläre Ausdrücke $\Theta$ .....   | 155        |
| 6.4      | <i>string_view</i> .....  | 162        |
| 6.5      | Erweiterungen für Container-Klassen .....   | 163        |
| 6.5.1    | Iteratoren .....  | 163        |
| 6.5.2    | Die Speicherverwaltung bei Vektoren $\Theta$ .....  | 165        |
| 6.5.3    | <i>emplace</i> für Container .....  | 166        |
| 6.6      | Erweiterungen für Stream-Klassen .....  | 166        |
| 6.7      | Ungeordnete Assoziative Container (Hash-Container) .....                                    | 167        |
| 6.8      | <i>std::array</i> - Array Container fester Größe $\Theta$ .....                             | 169        |
| 6.9      | Filesystem .....  | 171        |
| 6.10     | Parallele Algorithmen .....   | 174        |
| 6.10.1   | Parallel for mit <i>for_each</i> .....  | 175        |
| 6.10.2   | <i>std::sort</i> .....  | 177        |
| 6.10.3   | <i>std::search</i> .....  | 177        |
| 6.10.4   | <i>std::reduce</i> und <i>std::transform_reduce</i> .....                                   | 177        |
| 6.11     | ??? <i>std::search</i> mit Boyer-Moore .....  | 179        |
| 6.12     | Datentypen mit Varianten .....  | 179        |
| 6.12.1   | <i>std::any</i> – Eine Klasse für Werte beliebiger Typen $\Theta$ .....                     | 179        |
| 6.12.2   | <i>std::optional</i> – Eine Klasse für einen oder keinen Wert.....                          | 181        |
| 6.12.3   | <i>std::variant</i> – Eine Klasse für Werte bestimmter Typen .....                          | 183        |
| 6.12.4   | <i>std::visit</i> und Polymorphie mit <i>std::variant</i> .....                             | 184        |
| <b>7</b> | <b>Exception-Handling .....</b>   | <b>185</b> |
| 7.1      | <i>noexcept</i> .....   | 185        |
| 7.2      | Die Exception-Klasse <i>system_error</i> $\Theta$ .....                                     | 186        |

|   |            |
|---|------------|
| <b>8 C++11 Smart Pointer: <i>shared_ptr</i>, <i>unique_ptr</i> und <i>weak_ptr</i>.....</b> | <b>187</b> |
| 8.1 Gemeinsamkeiten von <i>unique_ptr</i> und <i>shared_ptr</i> .....                       | 187        |
| 8.2 <i>unique_ptr</i> .....   | 192        |
| 8.3 <i>shared_ptr</i> .....   | 193        |
| 8.4 <i>weak_ptr</i> $\Theta$ .....  | 194        |
| <br>  |            |
| <b>9 Zeiten und Kalenderdaten mit <i>chrono</i> .....</b>                                   | <b>197</b> |
| 9.1 Brüche als Datentypen: Das Klassen-Template <i>ratio</i> .....                          | 197        |
| 9.2 Ein Datentyp für Zeiteinheiten: <i>duration</i> .....                                   | 198        |
| 9.3 Datentypen für Zeitpunkte: <i>time_point</i> .....                                      | 201        |
| 9.4 Uhren: <i>system_clock</i> und <i>steady_clock</i> .....                                | 202        |
| <br>  |            |
| <b>10 Multithreading .....</b>  | <b>205</b> |
| 10.1 Funktionen als Threads starten .....   | 206        |
| 10.1.1Funktionen mit <i>async</i> als Threads starten .....                                 | 206        |
| 10.1.2Funktionen mit <i>thread</i> als Threads starten.....                                 | 209        |
| 10.1.3Lambda-Ausdrücke als Threads starten.....   | 211        |
| 10.1.4Zuweisungen und <i>move</i> für Threads.....  | 214        |
| 10.1.5Die Klassen <i>future</i> und <i>promise</i> .....                                    | 215        |
| 10.1.6Exceptions in Threads und ihre Weitergabe mit <i>promise</i> .....                    | 217        |
| 10.1.7Der Programmablauf mit <i>async</i> .....   | 220        |
| 10.1.8Informationen über Threads.....   | 224        |
| 10.1.9 <i>Sleep</i> -Funktionen .....   | 226        |
| 10.1.10Threads im Debugger .....  | 227        |
| 10.2 Kritische Abschnitte.....  | 228        |
| 10.2.1Atomare Datentypen.....   | 230        |
| 10.2.2Kritische Bereiche mit <i>mutex</i> und <i>lock_guard</i> sperren .....               | 231        |
| 10.2.3Weitere Lock-Klassen: <i>unique_lock</i> und <i>shared_lock</i> .....                 | 235        |
| 10.2.4Weitere Mutex-Klassen .....   | 237        |
| 10.2.5Deadlocks .....   | 238        |
| 10.2.6 <i>call_once</i> zur Initialisierung von Daten .....                                 | 240        |
| 10.2.7Thread-lokale Daten.....  | 241        |
| 10.3 Bedingungsvariablen zur Synchronisation von Threads.....                               | 242        |
| <br>  |            |
| <b>11 Der Visual Studio C++ Core Guidelines Checker .....</b>                               | <b>245</b> |
| <br>  |            |
| <b>12 Literatur.....</b>  | <b>249</b> |

