Microsoft

Visual Studio

Visual Studio Tipps & Tricks Vol. 1

50 Tipps & Tricks zum schnelleren und produktiveren Arbeiten mit Visual Studio

Daniel Meixner
Developer Evangelist, Microsoft Deutschland

http://aka.ms/VS-Tipps-eBook

1. Auflage 2015

Microsoft Deutschland GmbH · Konrad-Zuse-Str. 1 · 85716 Unterschleißheim
Tel.: 089 31760 · www.microsoft.de

Redaktion und Covergestaltung: Wunderman GmbH

Inhaltsverzeichnis

Inhaltsverzeichnis

Vorwort

Millions of developers around the world spend a significant portion of their work week using their preferred developer environment. From writing code and managing tasks and work items to debugging and deploying applications, developers are using tools, like Visual Studio, every day to get the job done. With that much of our livelihood dependent on the tools that we use, it only makes sense that we invest in learning these tools and reaching the highest level of expertise and mastery.

Over my nearly decade and a half of designing and shipping various parts of Visual Studio at Microsoft, I have come to learn a great deal about the intricacies of Visual Studio as a development tool. I have also witnessed firsthand how some developers are able to use the various tools provided by Visual Studio to increase their productivity and spend less time than their colleagues to get the same job done. Knowing the right keyboard shortcuts while editing code makes them more productive by saving minutes in operations that get repeated hundreds of times per day. Utilizing the right diagnostic tools while debugging applications enables them to find bugs faster and improve performance to create higher quality applications and components. Invariably, developers who know their tool inside and out deliver better results.

I have known Daniel as one of our best evangelists and a great developer advocate. On one of my recent trips to Germany, Daniel was part of the team that made sure we visit as many local developers as possible to get their feedback and educate them about the ins and outs of Visual Studio. With this passion for developers, it is no surprise to me that he invested a great deal of time at creating this compilation of tips to make developers' lives better.

In this book, Daniel details dozens of tips and tricks for using Visual Studio that will instantly boost your productivity. With in-depth illustrations, the book makes it easy to understand when a tip is applicable and how and where to use it. Daniel uses his own deep knowledge of Visual Studio to provide you with the tips that you will most frequently want to reference and use.

With the tips in this book, you will learn about aspects of Visual Studio that you had not known about before. In addition to giving you elite ninja status among your work colleagues, mastering Visual Studio usage can save you hours of your work week. This books will help you do just that.

Tarek Madkour
Principal Group Program Manager
Visual Studio IDE
Microsoft Corporation
Redmond, WA
June 8th 2015

Ein paar Zeilen zum Buch

Ich erinnere mich noch an meinen ersten Vortrag zum Thema „Visual Studio Tipps & Tricks“ – damals war ich allerdings einer der Zuhörer. Dieser Vortrag hat mir gezeigt, was alles in Visual Studio steckt, was es unter der Oberfläche zu entdecken gibt und welche witzigen und unerwarteten Funktionen in versteckten Winkeln schlummern – auch deshalb bin ich zu einem regelrechten Fan der IDE geworden. Dass ich Visual Studio seitdem effizienter bediene und alltägliche Aufgaben einfach schneller erledigen kann als zuvor, ist eine Folge des Ganzen.

Dieses Wissen weiterzugeben ist die grundsätzliche Idee, die hinter diesem Buch steckt: Es soll das nötige Know-how im Umgang mit der IDE vermitteln, das in der Praxis relevant ist, Kleinigkeiten vorstellen, die man jeden Tag benötigt, und Helfer erläutern, die im entscheidenden Moment die Arbeit erleichtern.

Darüber hinaus bin ich davon überzeugt, dass man bei der Arbeit ein besseres Gefühl hat, wenn man ein paar Tricks kennt, die eben nicht absolut offensichtlich sind. Es fühlt sich einfach gut an und bestätigt mich selbst in meiner Expertise rund um mein wichtigstes Werkzeug – Visual Studio. Wenn ich Aufrufe und Funktionalitäten kenne, deren Bedeutung und Anwendung sich nur fortgeschrittenen Nutzern erschließen.

Am interessantesten finde ich in diesem Zusammenhang die Tatsache, dass diverse Visual Studio-Tipps schon seit Jahren in unterschiedlichen Foren, Blogs, Vorträgen und Interessengemeinschaften ausgetauscht und diskutiert werden – und dennoch kein Ende der Nachfrage in Sicht ist. Ich denke, es liegt daran, dass die Menge der Tipps einfach viel zu groß ist, um wirklich alle zu kennen. Außerdem gerät manch ein Tipp bisweilen auch wieder in Vergessenheit und gewinnt zu einem anderen Zeitpunkt dann ebenso wieder an Bedeutung. Und natürlich gibt es auch immer Personen, die noch nicht so lange mit Visual Studio arbeiten und davon zehren, wenn andere ihr Wissen weitergeben.

Letzteres ist ein Aspekt, von dem ich selbst profitiert habe, heute noch gerne profitiere und der die Entwickler-Community für mich so einzigartig macht: Die Bereitschaft – in vielen Fällen völlig bedingungslos – Wissen weiterzugeben und damit anderen, teils völlig unbekannten und fremden Entwicklern auf die Sprünge zu helfen und ihnen dadurch einen leichten Einstieg in neue Technologien zu ermöglichen.

Insofern ist die Veröffentlichung dieser Tipps in Buchform auch einfach ein „Dankeschön” an die großartige Entwickler-Community in Deutschland.

Viel Spaß beim Programmieren mit Visual Studio!

Ihr
Daniel Meixner

Über den Autor

Daniel Meixner ist Developer Evangelist bei Microsoft Deutschland und beschäftigt sich mit Windows App Entwicklung und wann immer es möglich ist mit der Integration der Kinect für PC. Er ist bekennender Fanboy von Visual Studio und hat das „Gute, Schlechte und Hässliche" in der Software-Entwicklung zu genüge und aus unterschiedlichsten Blickwinkeln kennengelernt. Vor seiner Zeit bei Microsoft war er Consultant und Architekt für Application Lifecycle Managemen-Lösungen im Enterprise-Umfeld.

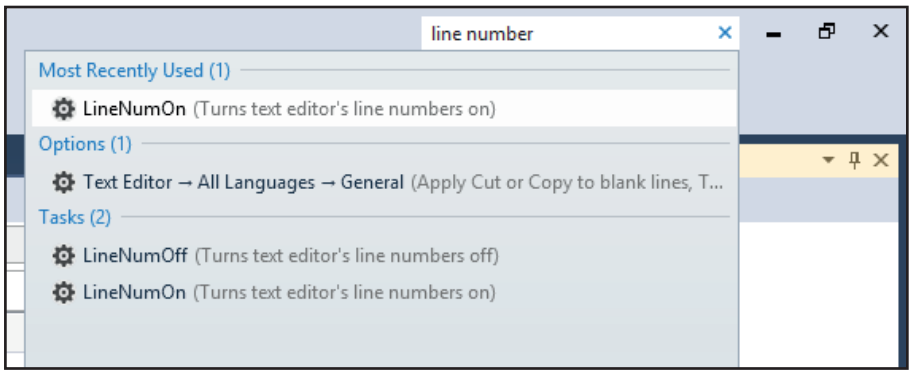
Daniel Meixner bloggt unter www.DevelopersDevelopersDevelopersDevelopers.NET und ist auf Twitter als @DanielMeixner unterwegs.

1 Quick Launch (Ctrl+Q)

Visual Studio verfügt über eine riesige Menge an Funktionen und Einstellungsmöglichkeiten. Das ist nicht zuletzt der Grund für diese Blogpost-Serie. Ab und an findet man sich als Entwickler in der Situation wieder, dass man zwar ganz genau weiß, dass man ein bestimmtes Feature schon genutzt oder eine bestimmte Einstellung modifiziert hat, man findet aber den Weg dorthin nicht mehr. Mit Visual Studio 2012 wurde genau dafür eine Lösung geschaffen: Quick Launch.

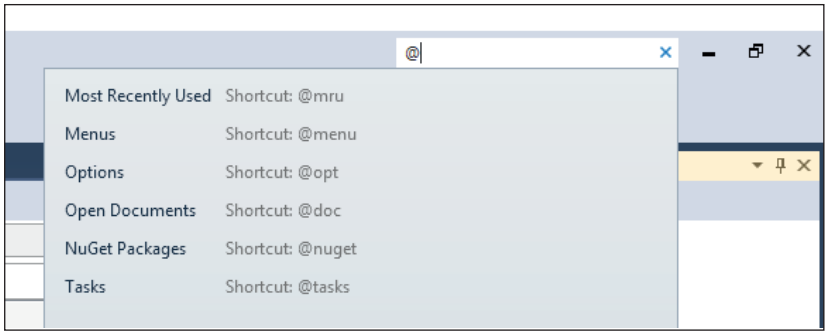
Über das kleine Eingabefeld oben rechts in Visual Studio könnt Ihr einen Suchbegriff eingeben, und Visual Studio wird Euch im Rahmen der Suchergebnisse nicht nur den Pfad zur entsprechenden Einstellung präsentieren, sondern zusätzlich auch noch die Suchergebnisse verlinken, so dass Ihr sofort über einen Klick zum Ziel kommt.

Hier ein kleines Beispiel: Wenn ich über Quick Launch die Stelle in Visual Studio suche, an der ich Zeilennummern einschalten kann (was ohnehin eine gute Idee ist), dann tippe ich einfach den Shortcut Ctrl+Q, setze dadurch den Cursor ins Quicklaunch-Eingabefeld und tippe dann „Line Number".

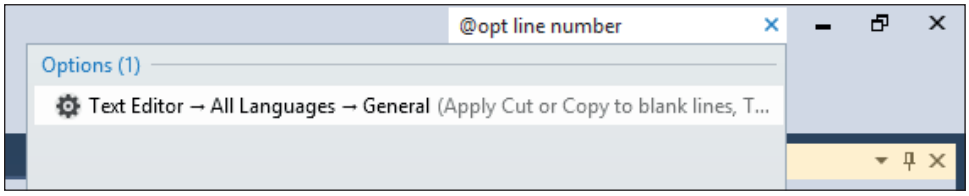


Als Ergebnis bekomme ich eine sortierte Liste, deren Einträge anklickbar sind. Gleichzeitig zeigen mir die Einträge, wie ich mich „zu Fuß" zur entsprechenden Einstellung durchhangeln könnte: Das Zahnrad steht dabei stellvertretend für „Tools->Options", danach müsste ich auf den Unterpunkt „Text Editor" und so weiter.

Wenn ich die Ergebnismenge einschränken möchte, kann ich über die Eingabe eines „@" unterschiedliche Filter anzeigen lassen.



So kann ich dann über den Suchstring „@opt line number" die Ergebnisse auf den Optionsdialog einschränken.



tl;dr

Über „Ctrl+Q" startet Quick Launch, über „@" kann man filtern.

2 Visual Studio via Shortcut starten

Ok, genau genommen bezieht sich dieser Trick nicht auf Visual Studio, sondern auf Windows. Seit Windows Vista ist es möglich, Programme, die auf der Taskbar verlinkt sind, über einen Tastatur-Shortcut zu starten, ohne dazu einen extra Shortcut zu konfigurieren. In Windows einen Shortcut anzulegen geht schon seit geraumer Zeit, wenn man sich die Mühe macht, auf der entsprechenden Verknüpfung einen Rechtsklick zu machen und dann die entsprechende Tastenkombination anzugeben.

Für die Taskleiste gibt es automatisch Shortcuts, und zwar Win+1, Win+2, Win+3, ... bis Win+0. Gestartet werden dadurch die Programme, die auf der Taskleiste verknüpft sind, wobei Win+1 das Programm auf der ganz linken Position (also unmittelbar neben dem Startknopf, falls vorhanden) startet, Win+2 das auf der zweiten und so weiter.

Unterm Strich eine schöne und einfache Möglichkeit, schnell Programme (und auch Visual Studio) zu starten. Bei mir nimmt Visual Studio übrigens Platz 2 und 3 ein, einmal in Version 2013 Preview, einmal als 2012.

Aaaaber: Wie verhält sich das, wenn schon eine Instanz des Programms offen ist? Windows schaltet auf die bereits laufende Instanz um. Um eine weitere Instanz eines Programms zu öffnen, benötigen wir zusätzlich noch die Shift-Taste, also Win+Shift+Zahl.

Ab und an ist es vielleicht notwendig, Visual Studio als Admin zu starten. Das kann man natürlich über einen Rechtsklick bewerkstelligen, aber auch dieser Shortcut unterstützt das Starten als Admin: Wir brauchen zusätzlich noch die Ctrl-Taste, also Win+Shift+Ctrl+Zahl. Ok, das ist jetzt wirklich lang. Im Alltag beschränke ich mich auf Win+Zahl.

tl;dr

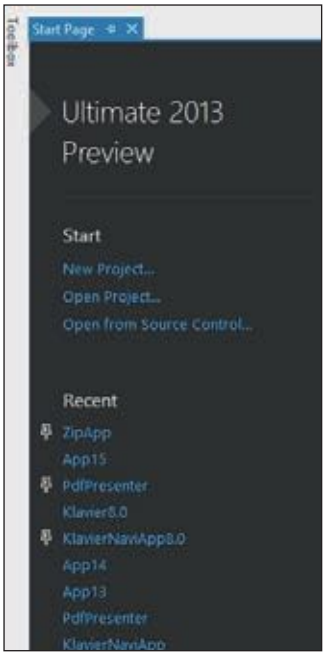
Visual Studio hat einen Platz in der Taskleiste ganz links verdient.
Und dann einfach mal Win+1 drücken.

3 Startpage, Pin Projects, Videos

Die Visual Studio Startpage ist vielen Entwicklern ein Dorn im Auge – weil sie die Startzeit von Visual Studio in der Vergangenheit spürbar verlängert hat und noch dazu überflüssig war. Meinem persönlichen Empfinden nach hat sich mit der Version 2013 die Sache mit der Startzeit erledigt. Ich habe allerdings keine umfangreichen Tests gemacht, vielleicht ist einfach auch mein Rechner schneller. Auf jeden Fall erledigt hat sich aber das Thema, ob sie überflüssig ist.

Für diejenigen, die die Startpage dennoch abschalten möchten, empfiehlt sich ein Blick in die untere linke Ecke derselben. Dort findet man zwei Häkchen, eines davon schaltet die Startpage beim Start von Visual Studio aus. Um sie wieder einzuschalten, kann man Quick Launch nutzen, um die entsprechende Einstellung im Visual Studio wiederzufinden (oder man navigiert über Tools->Options->Environment->Startup).

Aber vielleicht sollte man die Startpage erst mal eingeschaltet lassen; hier finden sich ganz nützliche Sachen. Neben ein paar kurzen Videos zu ausgewählten Features, die durchaus sehenswert (und vor allem wirklich kurz) sind, besteht nämlich die Möglichkeit, hier einen Blick auf die letzten Projekte zu werfen, mit denen man gearbeitet hat.



Wenn man mit der Maus über ein Projekt in dieser Liste fährt, dann erscheint ein kleiner Pin vor dem Projekt – damit kann man das Projekt anheften und dadurch sicherstellen, dass es in dieser Liste bleibt, auch wenn man es längere Zeit nicht mehr öffnet. Über einen Rechtsklick auf das Projekt kann ich das Projekt aus der Liste entfernen oder – und das finde ich besonders spannend – den Ordner des Projekts öffnen. Das ist besonders dann hilfreich, wenn ich ein anderes Projekt suche, dessen Namen ich nicht mehr weiß, bei dem ich aber sicher bin, dass es „ungefähr" da liegt, wo auch das Projekt in der Liste liegt. Da ich sehr viele Projekte zu Demozwecken anlege und bedauerlicherweise nicht immer sinnvolle Namen vergebe, geht es mir recht häufig so …

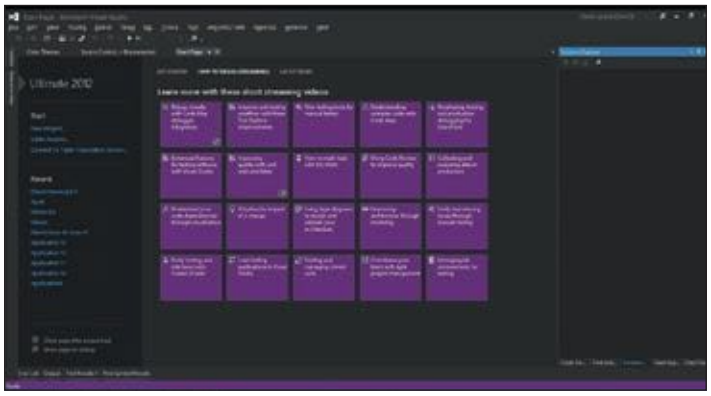
tl;dr

Die Startpage kann man auf der selbigen abschalten. Vielleicht sollte man das aber gar nicht, weil die gepinnten Projekte, der Zugriff auf Projektordner und die kurzen Videos einen echten Mehrwert bringen. Und dann einfach mal Win+1 drücken.

4 Color Themes & Theme Editor

Mit dem Release von Visual Studio 2012 hat Microsoft das Farbschema überarbeitet. Vielleicht habt Ihr mitbekommen, dass vom ersten RC von VS 2012 bis zur finalen Version noch jede Menge Anpassungen gemacht wurden, sicherlich nicht zuletzt, weil viele von Euch über Uservoice wertvolles Feedback gegeben haben. Eigentlich ein schönes Beispiel für die Adaption von Customer Feedback, oder?

Auch wenn ich persönlich sehr gut mit den Default-Einstellungen arbeiten kann, gibt es häufiger den Wunsch, die Farbe des Studios an den Blend Editor anzupassen. Wie Ihr vielleicht wisst, ist es in VS 2012 jetzt möglich, ein „Dark Color Theme“ zu wählen. Ihr findet die entsprechende Einstellung über den Options-Dialog.



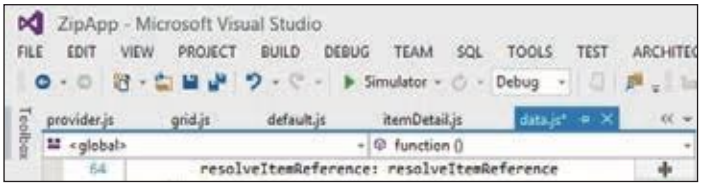
Vielleicht möchtet Ihr aber gerne noch mehr Auswahl haben? Ein Theme wie früher vielleicht? Wie der Screenshot oben zeigt, ist das möglich. Alles, was Ihr benötigt, ist die Extension „Visual Studio 2012 Color Theme Editor“, die Ihr über die Visual Studio Gallery kostenlos herunterladen könnt. Danach habt Ihr jede Menge Templates, wo wirklich für jeden Geschmack was dabei sein sollte. Und falls nicht: Ihr könnt Euch Euer eigenes Theme konfigurieren.

tl;dr

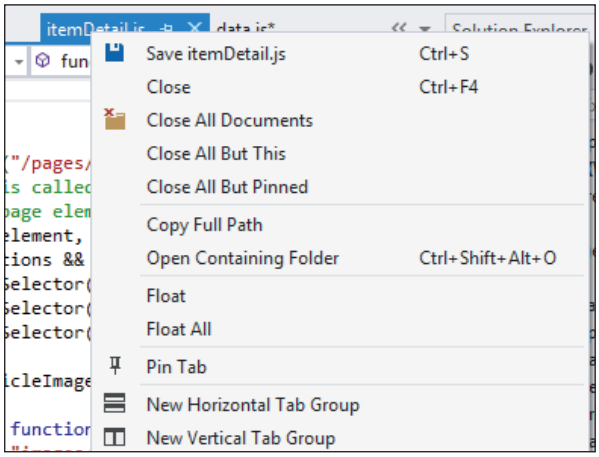
Wenn Ihr wollt, könnt Ihr die Farben für Visual Studio 2012 anpassen. Entweder Ihr nutzt eines von den eingebauten Templates oder erstellt Euch eines über den Theme Editor selbst. Keine Ausreden mehr. :-)

5 Dokument Tabs

Kaum ist Visual Studio gestartet und man hat ein bisschen gearbeitet, schon sind jede Menge Tabs offen und man darf sehen, wie man die Übersicht behält. Der Bildschirmplatz ist begrenzt, wer viel editiert, hat schnell den Überblick verloren. Wenn man die Tabs mal etwas genauer unter die Lupe nimmt, stellt man fest, dass es hier – ähnlich wie auf der Start-page – die Möglichkeit gibt, Einträge anzupinnen.



Gepinnte Tabs rutschen immer ganz nach links und werden nicht in den Hintergrund rutschen – bleiben also immer sichtbar. So kann man besonders wichtige Dokumente problemlos im Blickfeld halten. Über das kleine „X“ lassen sich die Dokumente schließen – spannend wird es aber insbesondere, wenn man auf so ein Tab mal mit der rechten Maustaste klickt.



Hier hat man ein paar nette weitere Möglichkeiten. Ich kann dafür sorgen, dass alle außer dem angeklickten Dokument geschlossen werden. Oder, dass alle außer den gepinnten Dokumenten geschlossen werden. Außerdem gibt es hier die Möglichkeit, direkt zum Ordner des Dokuments im Explorer zu navigieren. Besonders interessant: Ich kann über den Float bzw. Float-All-Befehl dafür sorgen, dass nur das Dokument oder eben alle momentan geöffneten in einem separaten Fenster geöffnet werden. Über die Windows Bordmittel (Win+Left oder Win+Rechts) kann ich das dann wiederum auf dem Bildschirm verschieben, zum Beispiel auf die komplette linke Bildschirmhälfte. Zu guter Letzt habe ich die Möglichkeit, neue Tab-Gruppen zu erstellen, um beispielsweise logisch zueinander gehörende Dokumente horizontal oder vertikal zu gruppieren.

Meine persönlichen Highlights sind hier ganz klar „Open Containing Folder“ und „Close all but this“.

tl;dr

Es lohnt sich, auf den Dokumentenreiter mal mit der rechten Maustaste zu klicken.

6 Preview Tabs ein- und ausschalten

Visual Studio öffnet Dateien, die man im Solution Explorer lediglich markiert und nicht doppelklickt, in einem sogenannten „Preview Tab“. Das Preview Tab befindet sich in der Reihe der Tabs der offenen Dokumente immer ganz rechts. Es gibt immer nur ein Preview Tab, dessen Inhalt dynamisch getauscht wird.

Wie Ihr am Screenshot seht, wird – soweit genug Platz ist – automatisch ein Abstand zwischen normalen Tabs und dem Preview Tab gehalten. Das Preview Tab ist grundsätzlich ein recht nettes Feature: Bei Debugging Sessions hatte man früher oftmals das Problem, sehr viel offene Dateien zu haben. Das ist nicht mehr der Fall, da alle Dateien, die der Debugger durchläuft und die nicht explizit durch den Anwender geöffnet wurden, nur temporär im Preview Tab geöffnet werden. Das hält die IDE sauber.

Das gleiche gilt für die Arbeit im Solution Explorer. Um sich einen Eindruck davon machen zu können, was in einer bestimmten Datei steht, ist es nicht mehr notwendig, die Datei durch Doppelklick zu öffnen, sondern es reicht, diese zu markieren, und sie wird automatisch ins Preview Tab geladen.

Wenn man eine Datei im Preview-Tab offenhalten will, dann kann man das durch einen Klick auf das Icon neben dem Kreuzchen tun – anschließend hat man ein ganz normales Tab, das aber nach links, zu den anderen Tabs, rutscht. Die Funktion der anderen Tabs habe ich hier bereits beschrieben.

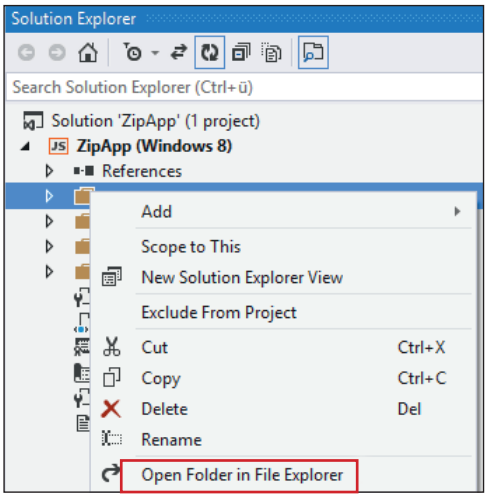
Es mag aber Situationen geben, in denen man die Preview Tabs im Zusammenhang mit dem Browsen des Solution Explorers nicht will. Vielleicht aus Performancegründen, vielleicht, weil man durch die wechselnden Inhalte irritiert ist oder warum auch immer. Gut zu wissen, dass die Preview-Funktion im Solution Explorer abschaltbar ist. Über folgendes Icon könnt Ihr die Preview ein- und ausschalten:

tl;dr

Wenn bei Euch das Preview Tab beim Browsen im Solution Explorer nicht geöffnet wird, schaltet es ein. Wenn Ihr die Funktion temporär nicht braucht, wisst Ihr jetzt, dass Ihr sie auch ausschalten könnt.

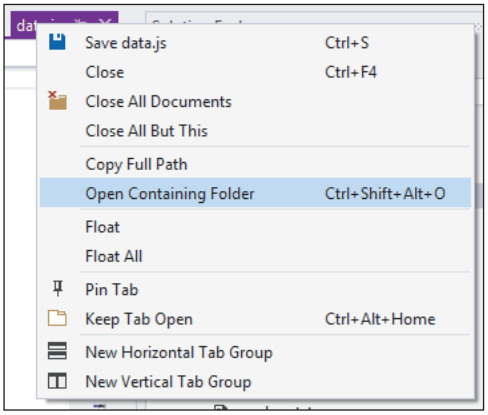
7 Open Containing Folder

Über den Solution Explorer hatte ich es ja schon beim letzten Mal. Ein Feature, das ich fast täglich nutze, ist die Möglichkeit, aus dem Solution Explorer heraus bestimmte Ordner im Windows Explorer zu öffnen. Dazu kann man im Solution Explorer auf einen beliebigen Ordner einfach mal einen Rechtsklick machen und dann „Open Folder in Explorer“ wählen. Daraufhin wird der entsprechende Ordner im Windows Explorer geöffnet.



Der häufigste Anwendungsfall für mich ist der, dass ich den „Images“-Ordner auf der Platte finden will, weil ich hier schnell ein paar Bilder hinzufügen oder einfach austauschen möchte. Nicht vergessen: Bei Dateien, die man in den Ordner kopiert hat und die im Projekt integriert werden sollen, muss man anschließend über „Add existing Item“ die Datei noch dem Projekt hinzufügen.

Wenn man nun versucht, diesen Trick auch auf Dateien anzuwenden, wird man feststellen, dass genau dieses Feature für Dateien nicht angeboten wird. Es gibt aber die Möglichkeit, die Datei im Preview Tab zu öffnen und dann einen Rechtsklick auf das Preview Tab zu machen. Hier findet sich dann der entsprechende Befehl wieder.

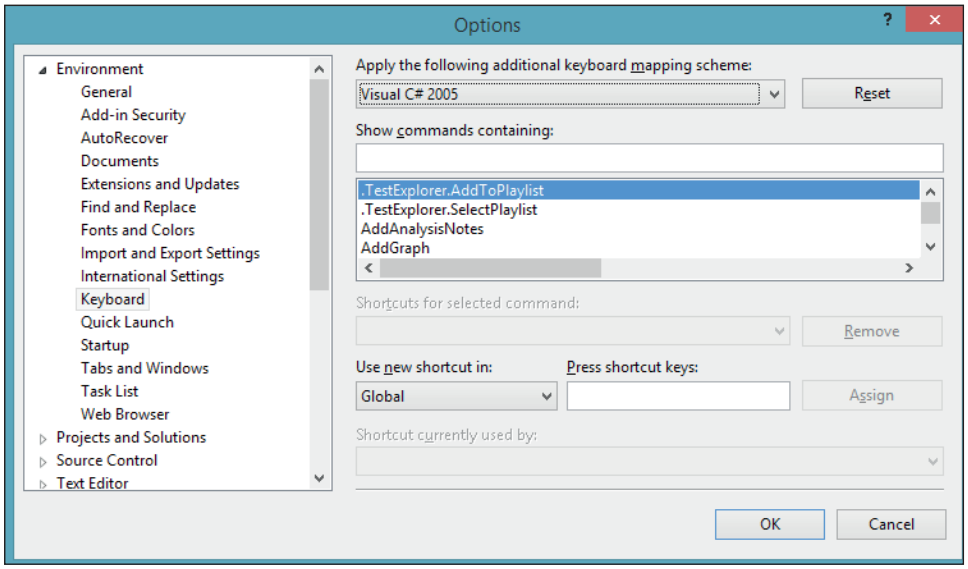


Eine Alternative besteht darin, sich einfach einen eigenen Visual Studio Shortcut zu erstellen. Das folgt im nächsten Tipp.

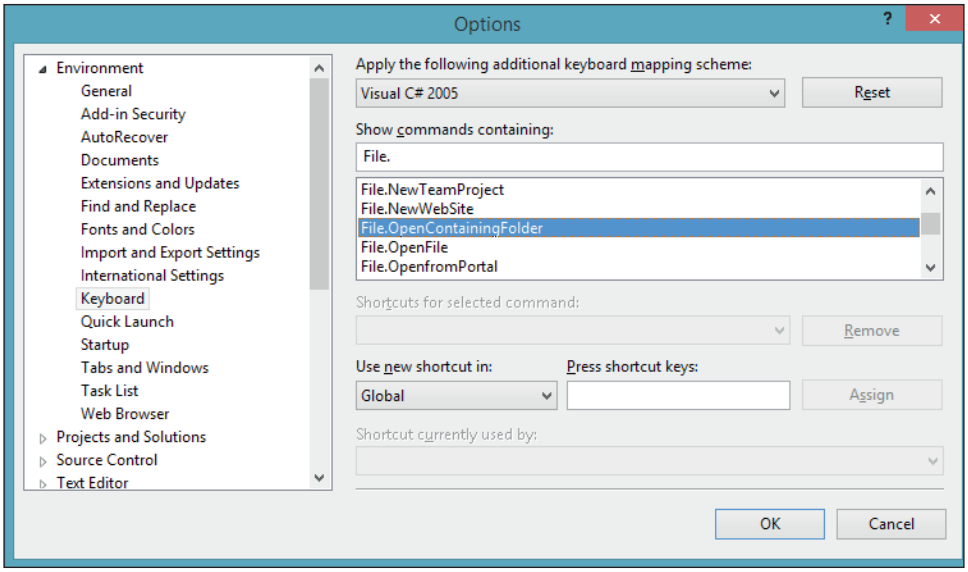
8 Eigene Shortcuts

Im vorherigen Tipp habe ich Euch gezeigt, dass man für Dateien im Solution Explorer von Visual Studio 2012 leider nicht die Funktion „Open Containing Folder“ aufrufen kann. Man muss dazu auf das Preview Tab oder eben auf das Dokument Tab klicken.

Wenn Ihr – wie ich – diese Funktion häufig verwendet und Euch mehr Komfort wünscht, dann ist es an der Zeit, sich einen eigenen Visual Studio Shortcut anzulegen. Das geht denkbar einfach. Alles, was Ihr tun müsst, ist, in Quick Launch „Keyboard“ einzugeben. Der erste Treffer sollte Euch zu folgendem Dialog führen.



Im Suchfeld könnt Ihr jetzt nach dem Befehl suchen, den Ihr über Keyboard Shortcuts erreichen möchtet. In unserem Fall hat das sicher was mit „File“ zu tun (schließlich wollen wir eine Datei öffnen). Wenn wir also „File.“ eingeben, sehen wir jede Menge Kommandos – wir müssen nur unser gewünschtes wählen. Insgesamt sind es doch recht viele, wir haben hier jede Menge Möglichkeiten, Visual Studio anzupassen.

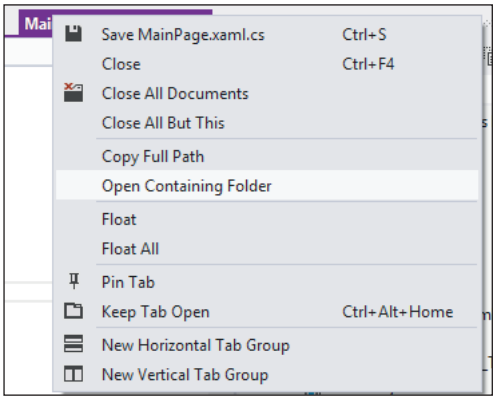


Der gesuchte Befehl lautet „File.OpenContainingFolder“. Jetzt stellen wir unseren Cursor in das „Press shortcut keys“-Feld und müssen dann einmalig den Shortcut unserer Wahl festlegen – indem wir ihn ausführen und anschließend über Assign zuweisen. Wir können uns hier tatsächlich einen beliebigen überlegen – sollten wir einen anderen bestehenden überschreiben, werden wir gewarnt. Um auf Nummer sicher zu gehen, bietet es sich an, den Shortcut nach dem Schema

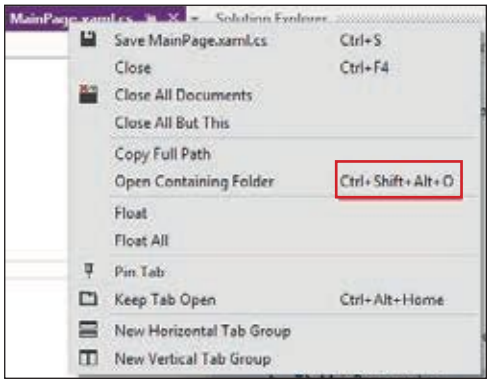
***Ctrl*+*Alt*+*Shift*+Buchstabentaste**

anzulegen. Ctrl+Alt+Shift wird per Default in Visual Studio nicht verwendet – wir überschreiben also nicht. In meinem Fall wähle ich Ctrl+Alt+Shift+O. Der Shortcut wird dann in Zukunft auch im Kontextmenü des Tabs neben dem Befehl angezeigt.

Vorher:



Nachher:



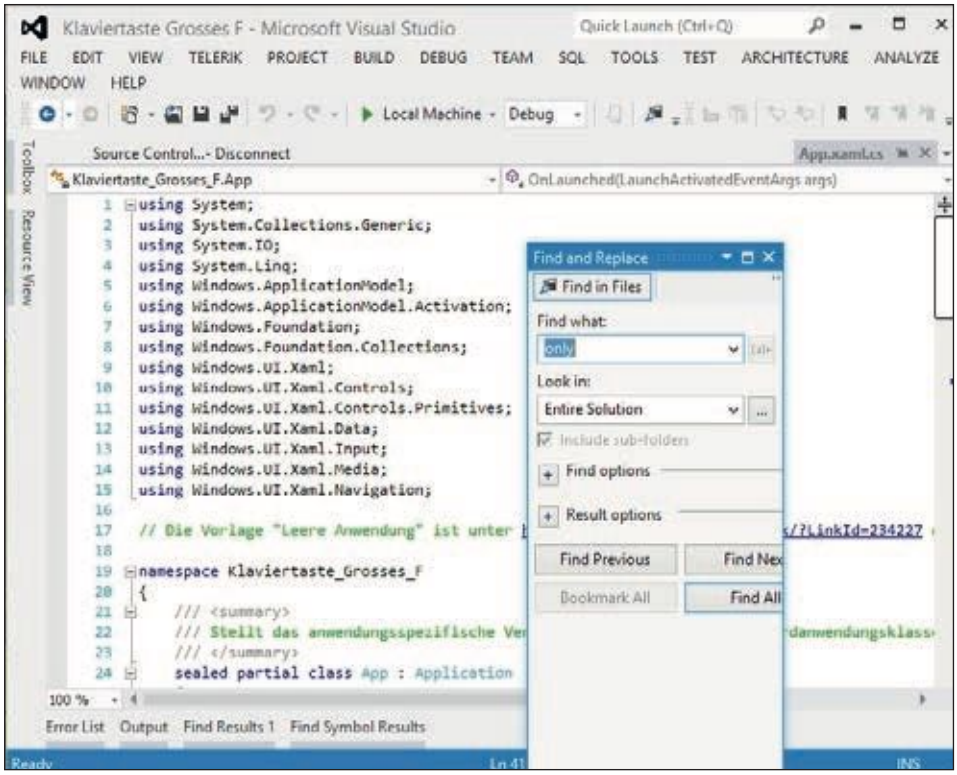
Ergebnis: Der Shortcut funktioniert jetzt auch aus dem Solution Explorer heraus. Einfach Datei anwählen und Ctrl+Alt+Shift+O klicken, schon geht der Explorer auf.

tl;dr

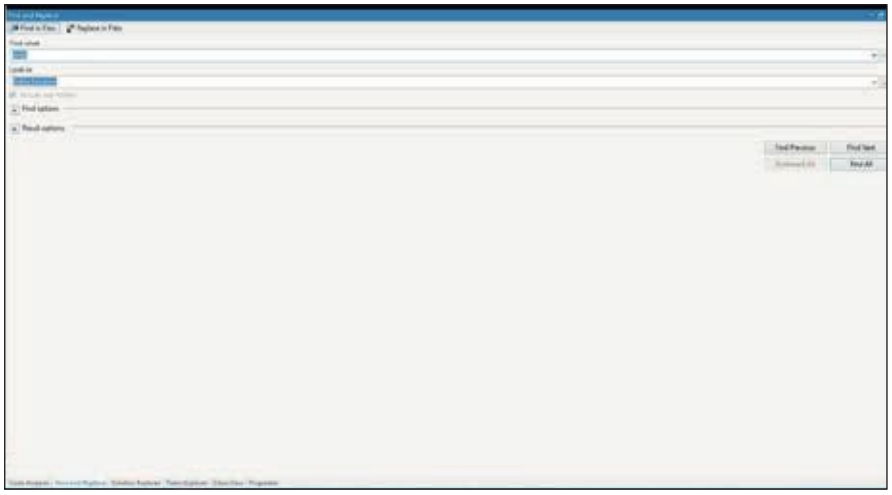
Über den Shortcut-Assistenten kann man ganz einfach eigene Keyboard Shortcuts erstellen. Dazu einfach „Keyboard“ in Quick Launch eingeben und den Anweisungen folgen.

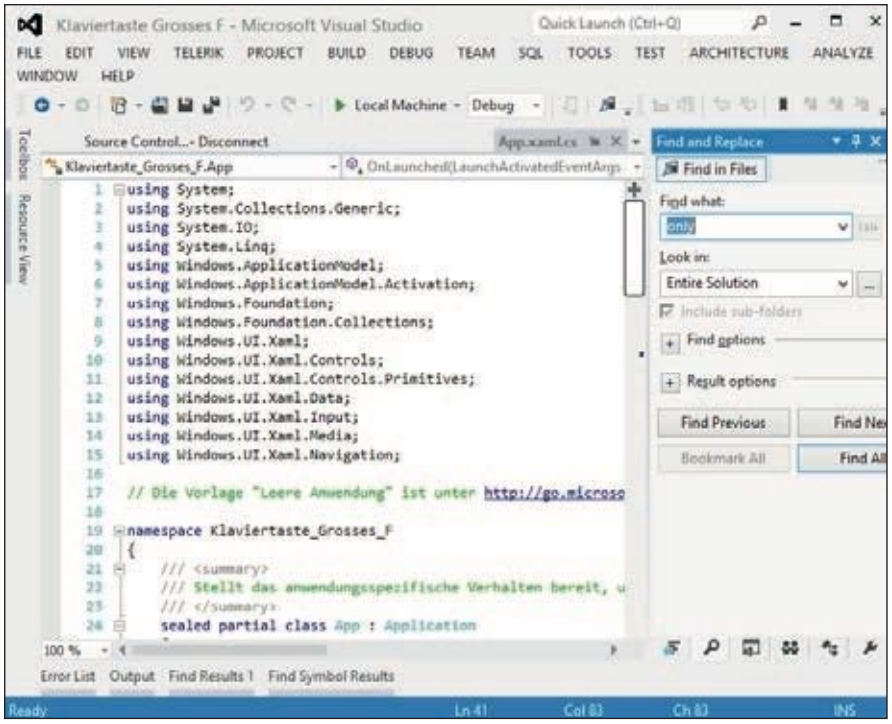
9 Verschobene Tool Windows zurücksetzen

Wer kennt das nicht: Man will eigentlich nur ein Tool Window (z. B. den Solution Explorer) kleiner oder größer ziehen, plötzlich hat man es schon in der „Hand“, bzw. man hat das Fenster verschoben und es hängt irgendwo im luftleeren Raum, unaufgeräumt und unordentlich, und man will es sofort wieder auf den angestammten Platz schieben. Wie geht das? Nun, die Versuchung liegt nahe, das vollmanuell zu tun: Also Fenster oben am Rahmen packen und wieder dahinziehen, wo es hingehört. Das funktioniert zuverlässig, dauert aber ein wenig – und mir damit zu lange.



Möglichkeit 2 wäre ein Doppelklick auf die Kopfzeile des Fensters. Hmmm .... das hat mal funktioniert, funktioniert aber nicht mehr. Scheidet also aus. Stattdessen wird das Fenster auf Vollbildmodus vergrößert – das ist nicht das, was wir wollten. Um das Tool Window wieder anzudocken, ist der einfachste Weg ein Doppelklick auf die Kopfzeile **mit gehaltener Ctrl-Taste**. Dann – schwupps – ist das Fenster wieder da, wo es hingehört.





tl;dr

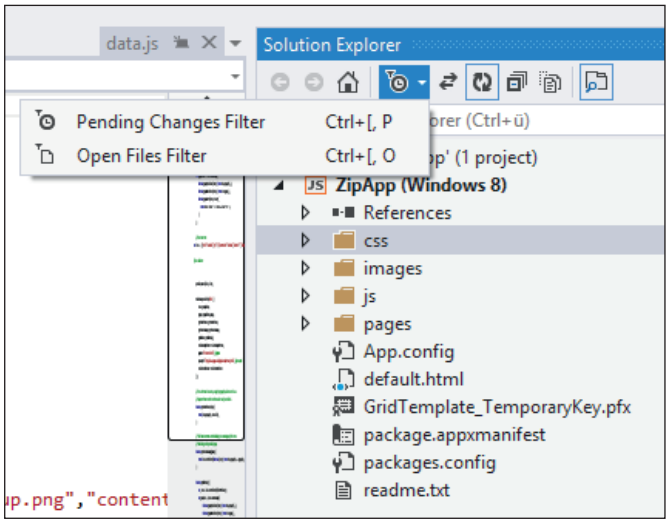
Doppelklick auf den oberen Fensterrahmen+Ctrl-Taste bringt das Tool-Window wieder an die angestammte Position.

10 Tool Windows Menüs per Tastatur steuern (Shift+Ctrl)

Ich finde es persönlich als sehr störend, zwischen all der Tipperei im Visual Studio ab und an doch die Maus bedienen zu müssen. Vielleicht habe ich deshalb irgendwann angefangen, mir für ein paar wichtige Funktionen Keyboard Shortcuts zu merken oder gar selbst anzulegen. Vermutlich ist das auch der Grund dafür, wieso ich Laptops mit dem kleinen Knubbel in der Mitte (aka Trackpoint) seit Jahren gegenüber Laptops ohne diesen Knubbel bevorzuge. Früher oder später kommt nämlich der Zeitpunkt, an dem ich in den meisten Programmen doch mal auf die Maus zugreifen muss.

Mein Bestreben ist es, möglichst große Teile meiner täglichen Arbeit am PC tatsächlich über Keyboard Shortcuts zu tätigen. Das hat zwei Vorteile: Erstens, man muss sich weniger bewegen, zweitens man ist wesentlich schneller.

Ein Bereich, in dem die Bedienung per Tastatur noch nicht flächendeckend verbreitet ist, ist das Anwählen von Menüs im Visual Studio. Normalerweise nimmt man die Maus, klickt auf ein Item und gut ists. Das funktioniert aber auch mit der Tastatur.



Befindet man sich beispielsweise im Solution Explorer und möchte auf den kleinen Schraubenschlüssel klicken, um die Properties zu öffen, dann kann man über Shift + Alt auf das Menü des Solution Explorers zugreifen. Um dann zwischen den einzelnen Menüeinträgen zu wechseln, nutzt man die Links/Rechts Pfeiltasten. Um die Einträge in Dropdown-Boxen durchzuwählen greift man auf “Shift + Pfeil nach unten” zu.

Das Hauptmenü von Visual Studio erreichen wir übrigens – wie in den meisten Windows Programmen – über die Alt Taste.

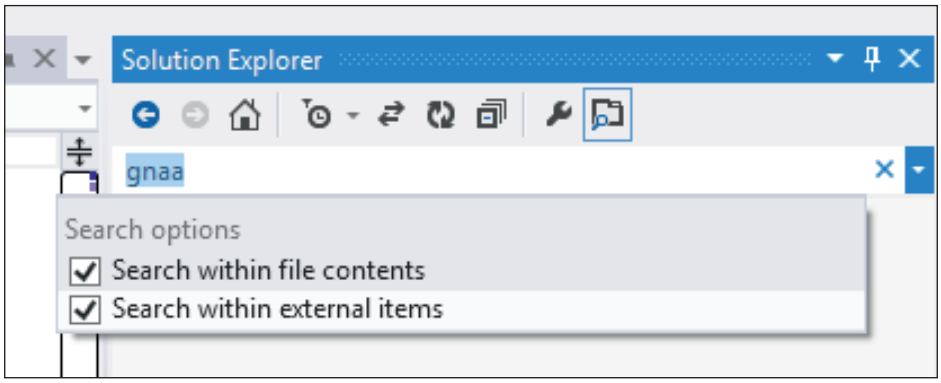
tl;dr

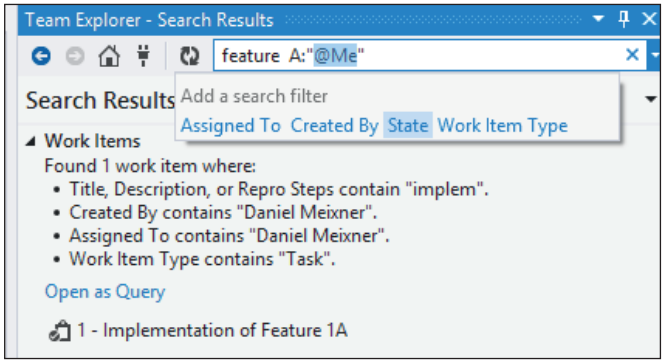
Menüs von Tool Windows lassen sich über Shift+Alt annavigieren und über die Pfeiltasten durchlaufen.

11 Tool Window Suche

Der Solution Explorer hat inzwischen – wie viele andere Tool Fenster auch – ein Suchfeld spendiert bekommen. Dieses Suchfeld können wir über einen Shortcut (Ctrl+ü) direkt anspringen und dann anfangen, unseren Suchbegriff zu tippen.

Besondere Beachtung verdient dabei allerdings auch das kleine Pfeilchen neben dem Suchfeld. Hier können wir noch genauer definieren, wo gesucht werden soll – auch in externen Dateien, auch im Dateiinhalt oder nur im Dateinamen.





Eine vergleichbare Möglichkeit gibt es auch im Team Explorer. Hier bringt uns Ctrl+ä ins Suchfeld. Besonders gut gefällt mir hier die Möglichkeit, nicht nur einen Suchbegriff einzugeben, sondern auch abhängig von bestimmten Work Item-Feldern die Ergebnisse einzuschränken: Über A:"@Me" wird festgelegt, dass ich nur Work Items sehe, die mir zugewiesen sind. Probiert es doch mal aus! Wenn Ihr Euch übrigens wundert, wie man auf die Idee kommt, Ctrl+ü oder Ctrl+ä als Keyboard Shortcut zu verwenden: Das hat mit dem US Keyboard Layout zu tun. Das deutsche Layout hat einfach ein paar Umlaute mehr.

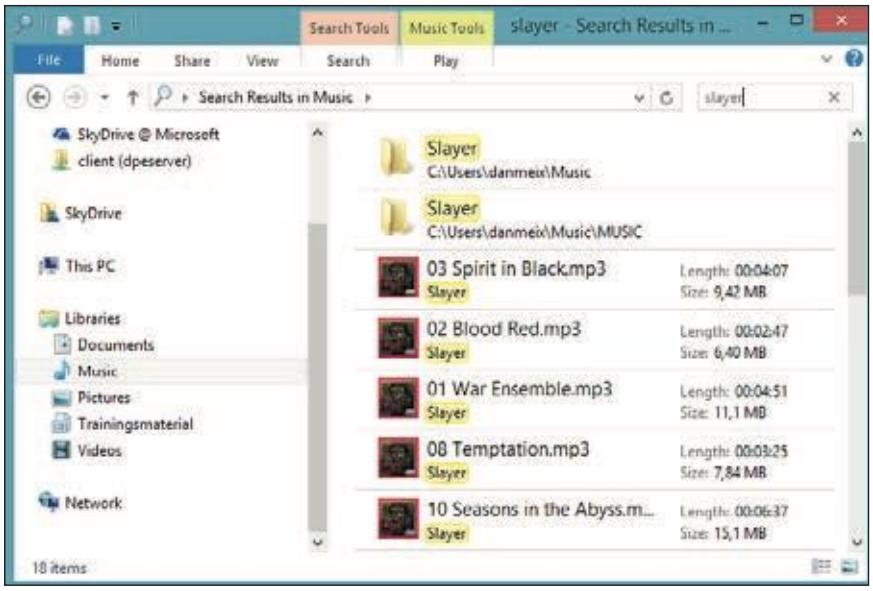
tl;dr

Viele der Tool Fenster haben Suchfelder bekommen, die über einen bestimmten Shortcut zu erreichen sind. Ctrl+ü dürfte einer der wichtigsten sein.

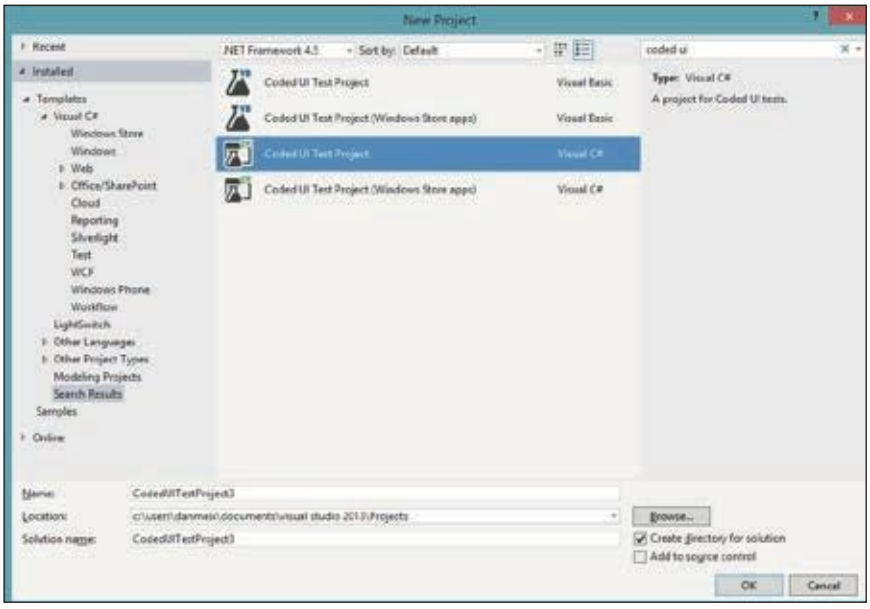
12 Suche in modalen Dialogen (Ctrl + e)

Im letzten Teil habe ich die Suche in Tool Window vorgestellt. Ein Tool Window ist eines der Fenster, die sich im Visual Studio andocken lassen – so kann jeder Entwickler seine Arbeitsumgebung so gestalten, wie er möchte. Es gibt allerdings auch Fenster in Visual Studio, die das nicht zulassen: die modalen Fenster. Teilweise verfügen auch diese Fenster über Sucheingaben. Schön zu wissen, dass es hierfür einen Shortcut gibt, der konsistent in all diesen modalen Fenstern funktioniert – der gleiche Shortcut, der auch in Windows selbst, beispielsweise im Explorer, die Suche öffnet: Ctrl+E.

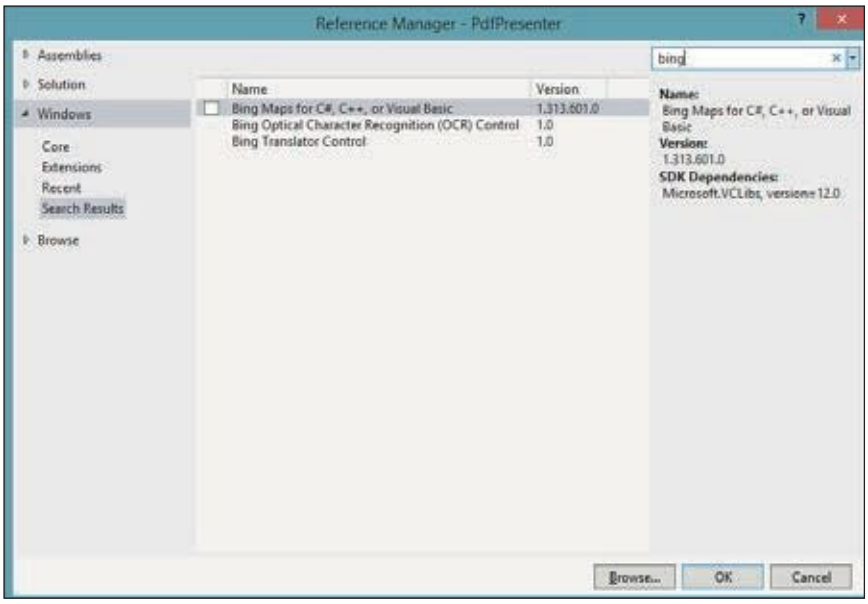
Das funktioniert im Windows Explorer …



… ebenso wie im New Project-Dialog …



... oder aber im Add References-Dialog.



Gut zu wissen: Dieser Shortcut funktioniert auch in vielen anderen Anwendungen für Windows, bspw. auch in Outlook.

tl;dr

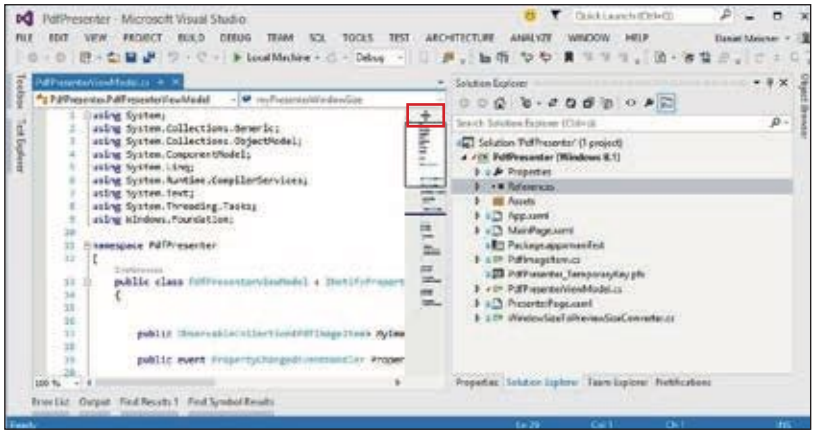
Ctrl+E triggert konsistent die Suche in modalen Visual Studio-Fenstern.

13 Split Windows

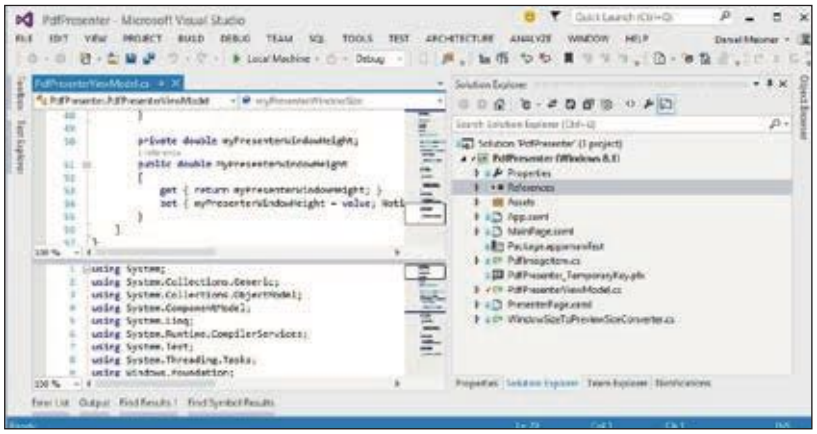
Wer kennt nicht den Fall: Man möchte im Visual Studio einen Codeschnipsel editieren, möchte aber eigentlich gleichzeitig einen anderen Bereich der gleichen Datei im Auge behalten. Vielleicht, weil man von dort ein paar Membernamen ablesen möchte, weil man an zwei Stellen gleichzeitig editieren muss, damit der Code lauffähig bleibt, oder weil man sich einfach vom bereits geschriebenen Code inspirieren lassen will.

Das Problem: Bei großen Dateien kriegt man mitunter nicht alles, was man gerne sehen würde, auf den Bildschirm – man scrollt ständig hoch und runter, ziemlich nervig. Die Lösung: Das Fenster teilen (Split Window).

Über ein kleines, gar nicht mal so sehr verstecktes Icon, ist es möglich, die gleiche Datei innerhalb eines Fensters zweimal anzuzeigen – oben und unten. Wo das Icon sich befindet, zeigt folgender Screenshot:



Wenn man mit gedrückter linker Maustaste dieses Icon nach unten zieht, fängt das Fenster auf magische Weise an, sich zu teilen. Ich kann in beiden Fenstern unabhängig voneinander scrollen und so mehrere Positionen im Code gleichzeitig im Auge behalten. Das Ergebnis kann dann zum Beispiel so aussehen:



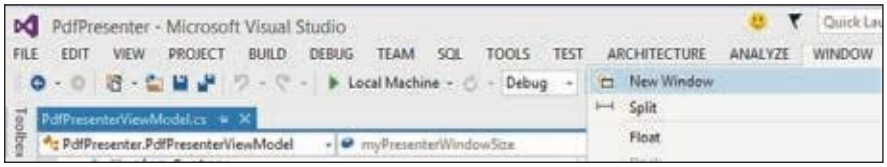
tl;dr

Es ist möglich, innerhalb eines Fensters mehrere Stellen einer Datei gleichzeitig im Auge zu behalten.

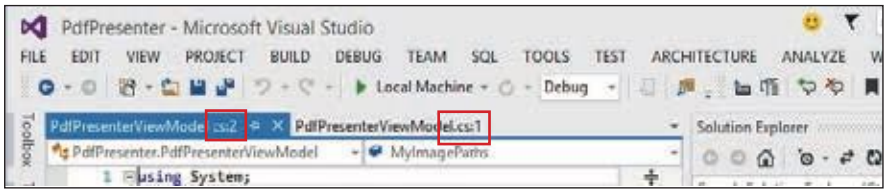
14 Eine Datei in mehreren Fenstern

Im letzten Teil habe ich das Split Window-Feature vorgestellt. Dabei wurde eine Datei innerhalb eines Fensters an unterschiedlichen Scrollpositionen dargestellt. Es ist aber auch möglich, ein komplett neues Fenster für die Datei zu öffnen und eine Datei in zwei (oder mehr) Fenstern darzustellen.

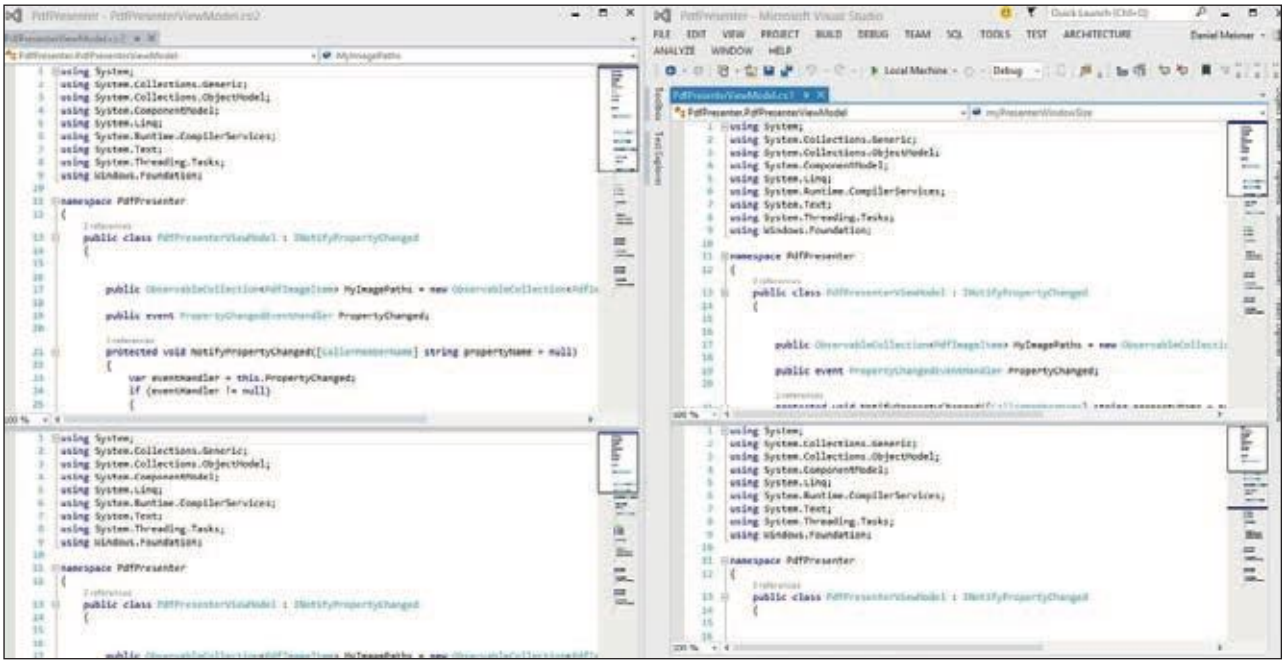
Hierzu wählt man einfach Window –> New Window aus.



Das Ergebnis sieht folgendermaßen aus:



Besonders interessant: Die unterschiedlichen Fenster bekommen unterschiedliche Namen – die Datei ist natürlich die gleiche. Wenn man im einen Fenster editiert, kann man die Änderungen sehen. Natürlich liegt es auf der Hand, das ganze mit dem Split Window zu kombinieren – das funktioniert auch, wie folgender Screenshot zeigt.



Wie funktioniert das?

Ich habe über New Window ein neues Fenster für die Datei erstellt, dieses dann per Drag & Drop abgedockt und anschließend den Bildschirm zwischen meiner eigentlichen Visual Studio-Instanz und dem neuen Fenster 1:1 aufgeteilt. Anschließend habe ich über das Teil Fenster-Feature die Dateidarstellung innerhalb beider Fenster geteilt.

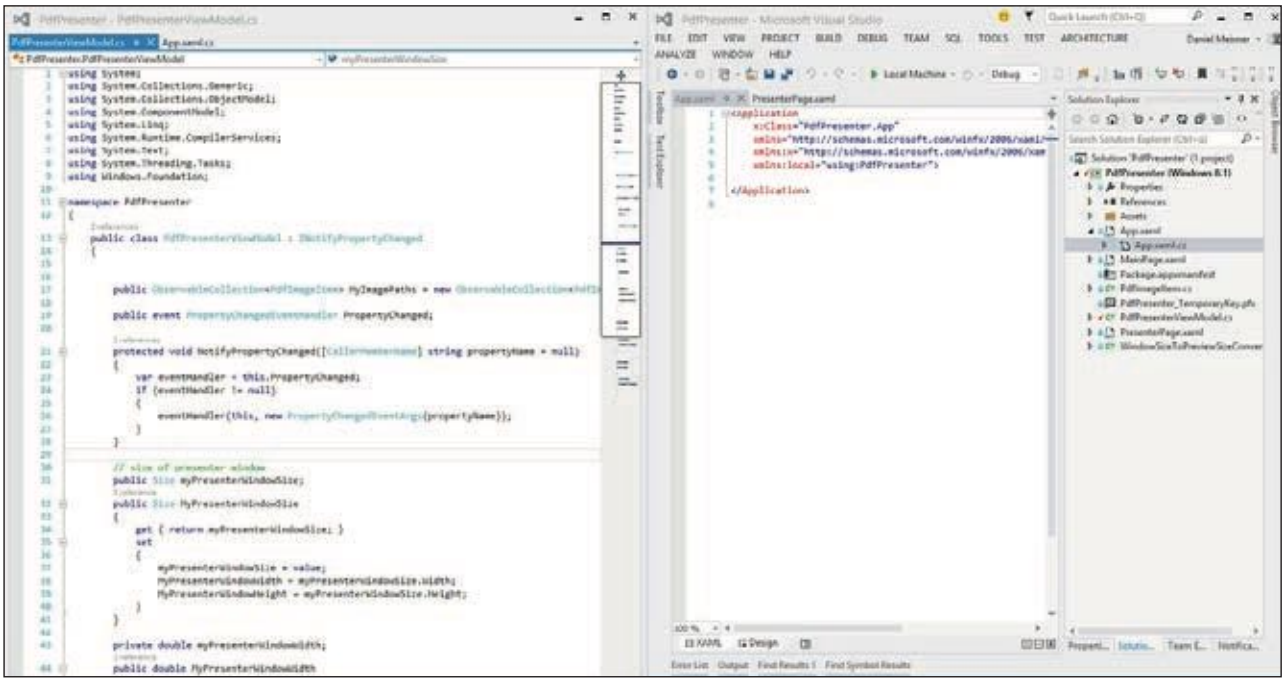
tl;dr

Es kann mehrere unterschiedliche Instanzen des Solution Explorers mit einem vom Anwender festgelegtem Scope geben.

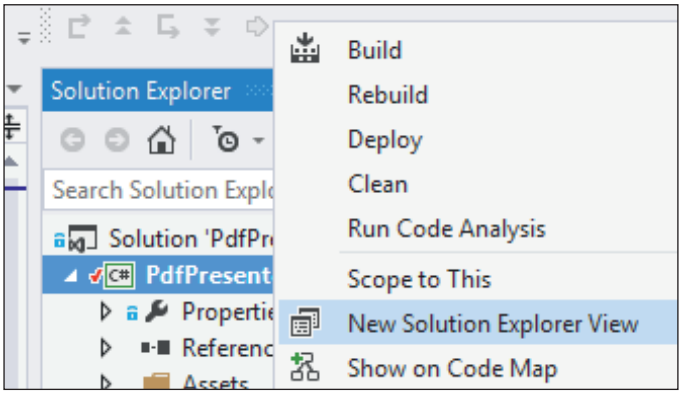
15 Mehrere Solution Explorer Views

Im letzten Trick haben wir eine Datei in mehreren Fenstern dargestellt. Diese neuen Fenster, die wir erstellt haben, können auch mit Tabs arbeiten, das heißt, ich kann innerhalb eines losgelösten Fensters mehrere Dokumente darstellen. Oder, um es von der anderen Richtung zu beschreiben, Visual Studio kann mit mehreren vollwertigen Dokumentfenstern umgehen.

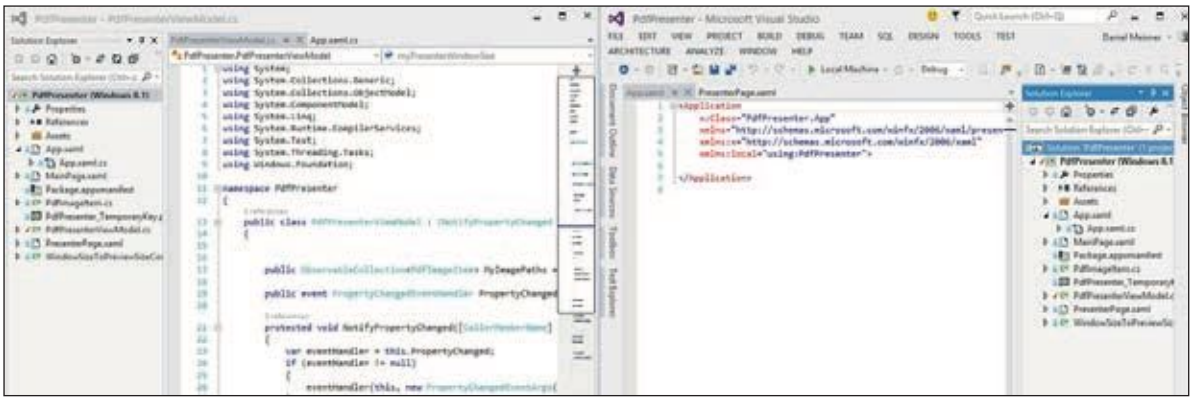
Dadurch werden folgende Darstellungen möglich:



Man kann sich gut vorstellen, dass dieses Layout gerade im Multimonitor-Betrieb echten Mehrwert bringt. Allerdings gibt es einen kleinen Nachteil: Man stelle sich vor, der Cursor befindet sich im linken Fenster, das in voller Größe auf dem linken Monitor zu sehen ist. Man möchte aber ein Element im Solution Explorer anklicken … der ist ganz rechts. Wenn man ein System mit zwei 24" Monitoren betreibt, kann da locker ein Meter Mausweg dazwischen liegen. Den können wir uns sparen, wenn wir einen neuen Solution Explorer öffnen und diesen links andocken. Das geht ganz leicht: Einfach ein Element im Solution Explorer anwählen, Rechtsklick und „New Solution Explorer View" wählen.



Daraufhin öffnet sich ein neues Floating Window – eine neue Instanz des Solution Explorers. Der darin dargestellte Inhalt ist der von mir ausgewählte mit allen Kindelementen. Das Fenster ist dockable, kann also wie erwartet auch beim linken Visual Studio-Fenster angedockt werden.



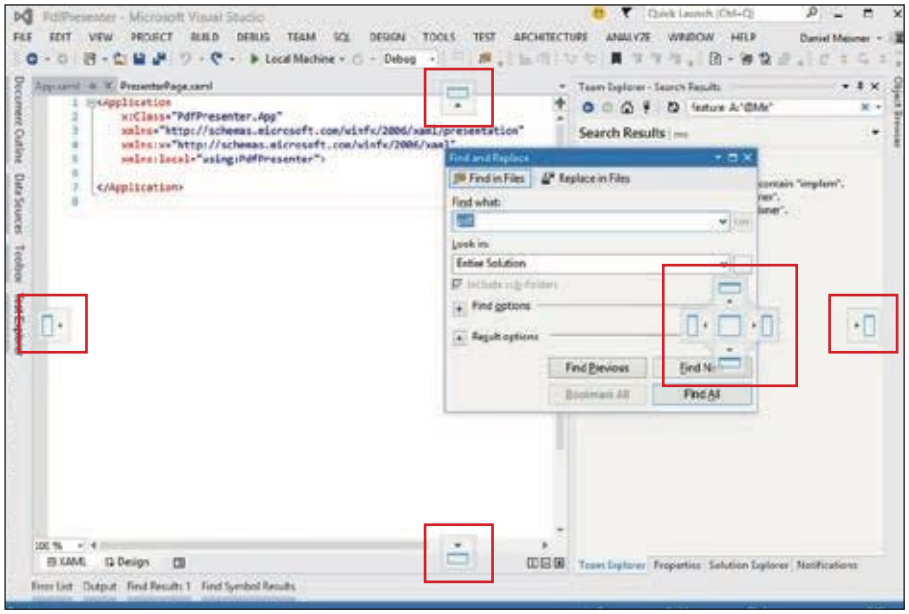
tl;dr

Es kann mehrere unterschiedliche Instanzen des Solution Explorers mit einem vom Anwender festgelegtem Scope geben.

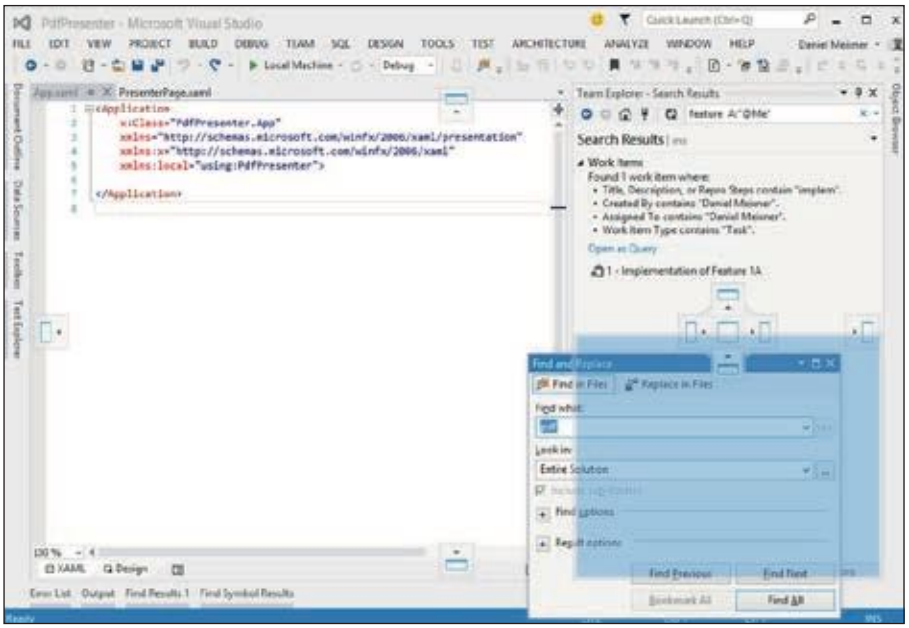
16 Window Docking

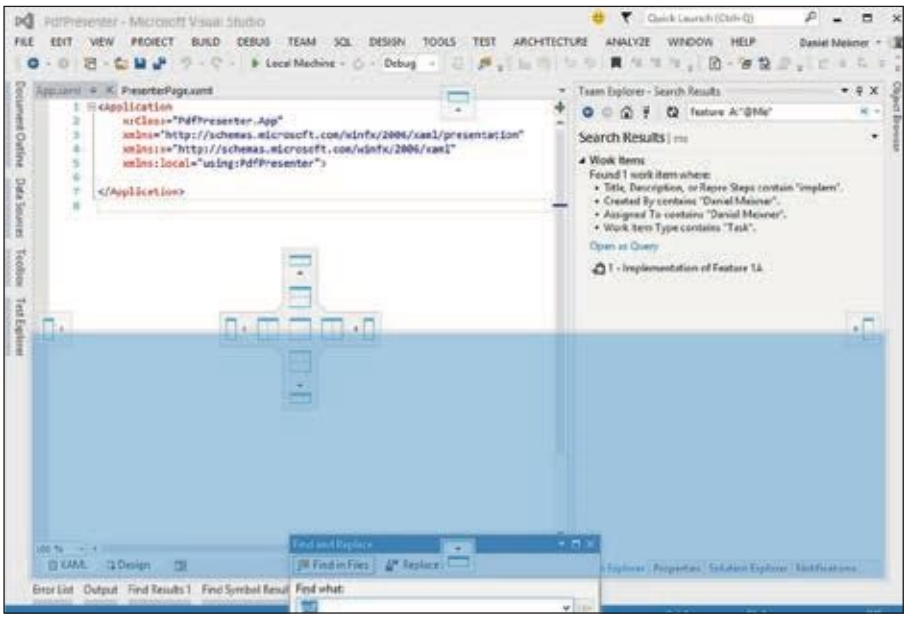
Das Docking-Verhalten der Fenster im Visual Studio ist eigentlich recht logisch. Wenn man allerdings so gut wie nie Anpassungen gemacht hat und dann versehentlich doch mal die Fenster verschiebt, stellt sich das Zurückbefördern oder Umpositionieren von Fenstern im Visual Studio durchaus als Herausforderung dar. Hier also eine Erklärung.

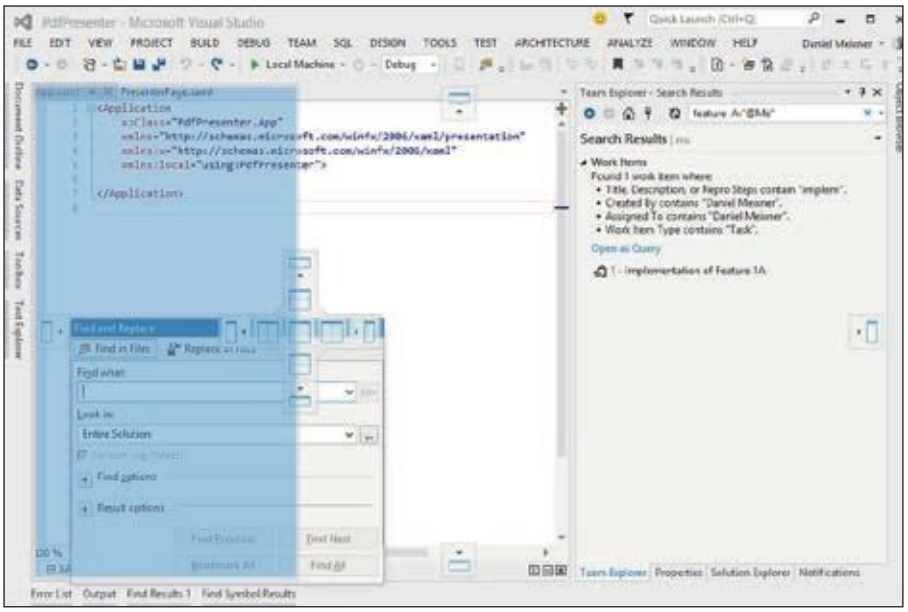
Die Tool Windows im Visual Studio dockt man typischerweise irgendwo am Rand an. Man ist in der Positionierung absolut frei – es gibt kein richtig oder falsch, höchstens ein üblich und ungewöhnlich. Wenn man eines dieser Tool-Fenster verschiebt, erscheint auf dem darunterliegenden Fenster eine Art Fadenkreuz aus Dockingflächen und im Visual Studio-Hauptfenster oben, unten, links, rechts auch.



Verschiebt man nun (mit gedrückter Maustaste) ein Tool Window und fährt dabei mit dem Mauszeiger über eine dieser Dockingflächen, wird angezeigt, wo das Fenster landen würde, wenn man den Mauszeiger loslassen würde.







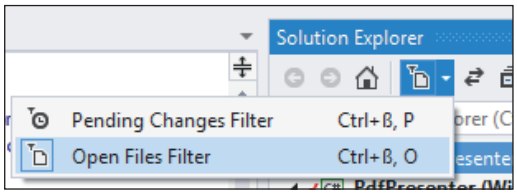
Diese Vorschaufunktion finde ich eigentlich recht angenehm. Sobald man die Maustaste loslässt, wird das Fenster angedockt. Ganz einfach, oder?

tl;dr

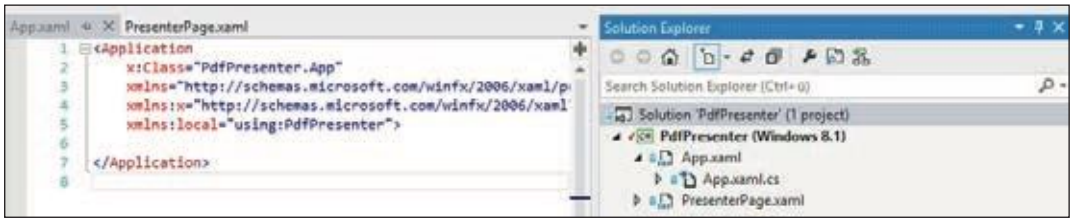
Um Fenster an die richtige Position anzudocken, empfiehlt es sich, einfach mit der Maus über die Dockingflächen zu fahren, um die Vorschau zu sehen.

17 Filter im Solution Explorer

Der Visual Studio Solution Explorer wurde bereits in der Version 2012 ordentlich überarbeitet und hat jede Menge Zusatzfunktionen bekommen. Besonders schön finde ich hier die Möglichkeit, die dargestellten Inhalte zu filtern. Über den entsprechenden Eintrag in der Toolbar ist es möglich, die dargestellten Elemente im Solution Explorer auf die gerade geöffneten Dateien oder die Dateien, an denen Änderungen durchgeführt wurden, zu reduzieren.

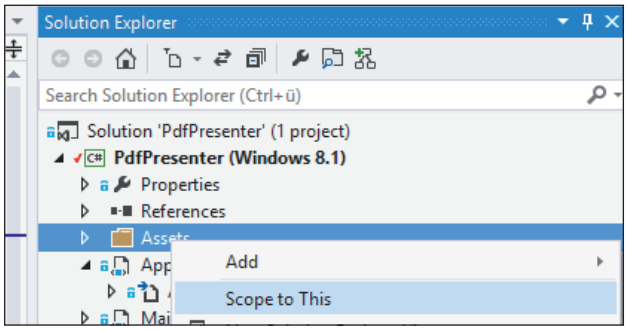


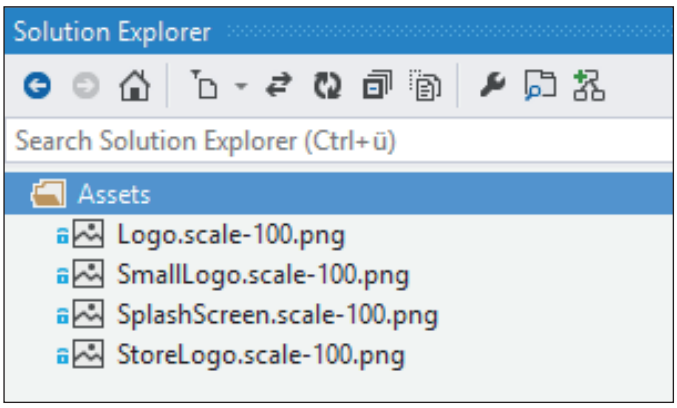
Das ist nicht zu verwechseln mit der Möglichkeit, Projekte aus einer Solution zu entladen oder gar auszuschließen – im Fall des Filters wird nur eine optische Änderung an der Darstellung durchgeführt, das Buildverhalten der Solution bleibt also gleich, lediglich die angezeigten Dateien werden reduziert.



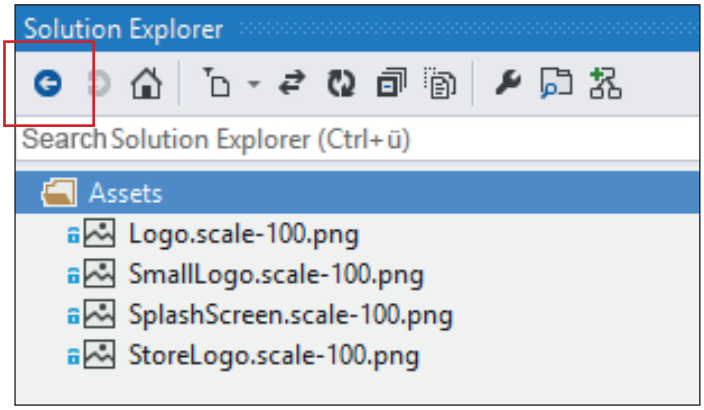
Wenn Ihr also in Zukunft Dateien in Eurer Solution vermisst, schaut doch mal nach, ob diese nicht aufgrund eines aktivierten Filters nicht zu sehen sind.

Die Funktionen im Solution Explorer gehen aber noch etwas weiter. Ich kann im Solution Explorer über einen Rechtsklick den Umfang der dargestellten Inhalte bestimmen. Beispielsweise kann ich auf ein bestimmtes Projekt oder einen bestimmten Ordner klicken und dann „Scope to this" anwählen.



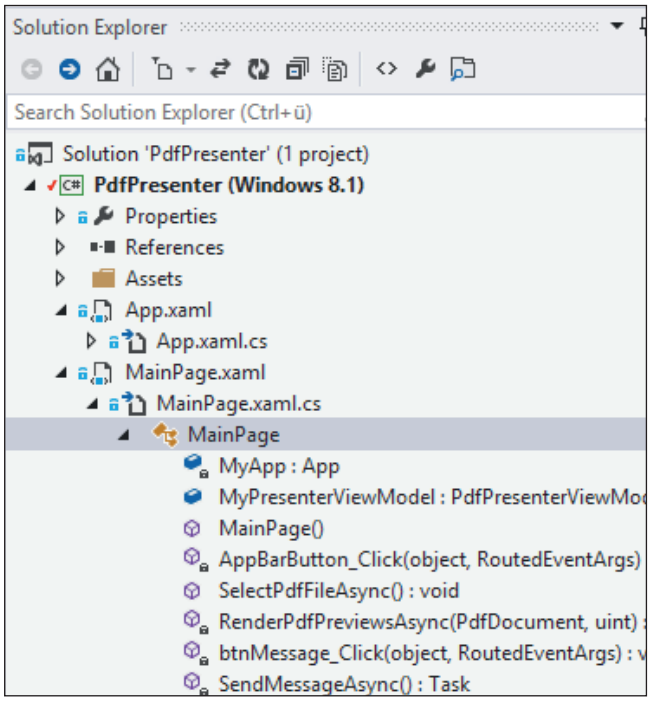


Mein Solution Explorer zeigt mir dann nur noch, was ich auch sehen will – den von mir gewählten Ordner.

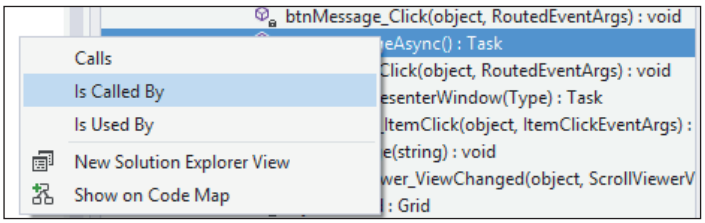


Wie kommt man aus dieser Ansicht wieder zurück zur normalen Ansicht? Ganz einfach: über den Rückwärts-Pfeil, ähnlich wie im Internet Explorer oder in Windows Store Apps.

Es gibt aber noch weitergehende Filtermöglichkeiten, die sogenannten Pivot-Filter. Visual Studio 2012 zeigt im Solution Explorer ja nicht mehr nur physikalische Dateien an, sondern auch logische Elemente – Klassen und Methoden.



Ich kann mir nun über die Pivot-Filter, basierend auf logischen Abhängigkeiten, weitere Dateien anzeigen lassen. Dazu macht man einen Rechtsklick bspw. auf eine Methode und wählt zum Beispiel „is Called by", um anschließend nur die Elemente dargestellt zu bekommen, die die entsprechende Methode aufrufen. Wie Ihr seht, ist es auch möglich, basierend auf diesem Pivot-Filter, gleich eine neue Solution Explorer-Instanz anlegen zu lassen.



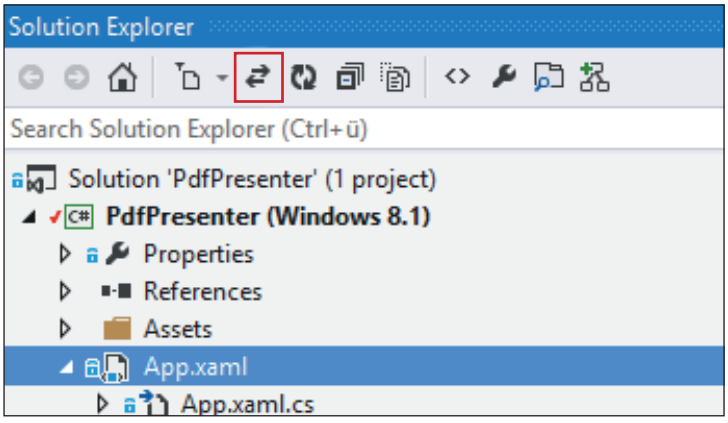
tl;dr

Im Solution Explorer kann man filtern, bis der Arzt kommt.

18 Sync Solution Explorer (Ctrl+ß, Ctrl+S)

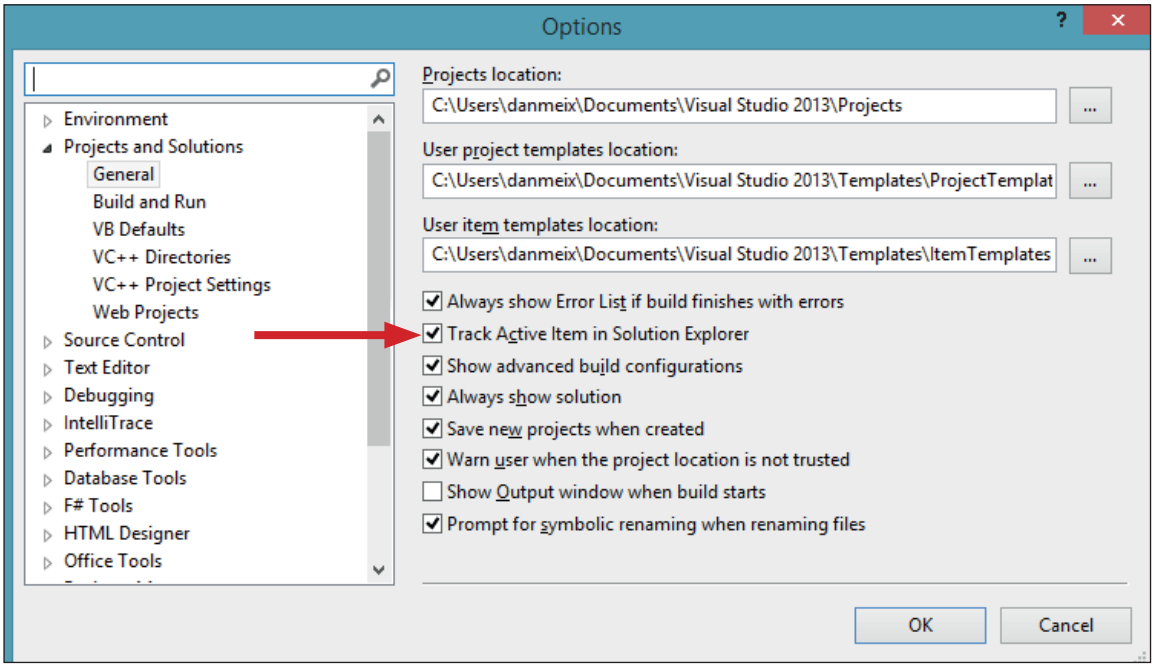
Es kommt bei mir recht häufig vor, dass ich jede Menge Dokumenten-Tabs offen habe und, dass Solutions jede Menge Projekte, Unterverzeichnisse und Dateien beinhalten. Wir haben schon ein paar Mechanismen kennengelernt, um hier aufzuräumen bzw. auszublenden.

Manchmal möchte man aber gar nicht die Darstellung im Solution Explorer verändern – man möchte nur die gerade relevante Stelle angezeigt bekommen, also im Idealfall die Position der Datei, in der man gerade arbeitet. Mit Visual Studio 2012 wurde hierfür ein neues Feature eingeführt, das es erlaubt, genau diesen Sync hinzukriegen.



Über das oben markierte Icon kann ich dafür sorgen, dass im Solution Explorer die Datei markiert wird, die ich ohnehin gerade bearbeite. Sehr praktisch. Es gibt auch für alle Tastaturfreunde einen vordefinierten Shortcut: Ctrl+ß, Ctrl + S.

Leider muss man das für jede Datei explizit tun, es geht standardmäßig nicht automatisch. Wer es doch automatisch haben möchte, der kann die Funktion „Track active Item in Solution Explorer" aktivieren. Dann funktioniert das auch ohne unser Zutun. Und wie finden wir die Einstellung „Track active ..."? Richtig: Am schnellsten über Quick Launch, aber auch über den normalen Weg, Tools->Options....



tl;dr

Der Solution Explorer kann über den Sync Button schnell zur gerade geöffneten Datei navigieren. Wenn man will, kann man das auch automatisieren.

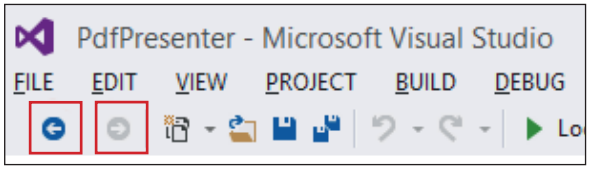
19 Wo war ich gerade? (Ctrl+Minus)

Ich bin sicher, Ihr kennt die Situation: Eben noch war man in Datei A am Programmieren, dann öffnet man Datei B, dann geht man in die bereits geöffnete Datei C und scrollt dort ganz runter, programmiert weiter, geht in Datei A zurück, dann wieder in B, um da zu merken, dass man irgendwo zwischenzeitlich eventuell noch mal was nachsehen müsste.

Und dann? Dann fängt man an sich zu erinnern, wo man eigentlich gerade war, und navigiert genau da wieder hin. Das heißt, man fängt an sich zu überlegen, wie die Dateien heißen, in denen man sich befand, und scrollt wieder munter drauflos – in die andere Richtung.

Je größer die Projekte sind, umso mehr Dateien hat man, die man potentiell bearbeitet, und umso schwieriger wird es, sich darin zurechtzufinden. Der folgende Tipp ist da vielleicht echt ein Lebensretter:

Visual Studio merkt sich, an welcher Stelle man sich im Code befunden hat, und hilft uns dabei, sich auf dem zurückgelegten Pfad rückwärts zu bewegen. Dafür gibt es zwei Buttons im Visual Studio. Lustigerweise habe ich persönlich lange Zeit diese Buttons nicht wahrgenommen, da ich irgendwo einmal den Shortcut aufgeschnappt und deshalb nie nach Buttons für diese Funktion gesucht habe. Wie auch immer: Es gibt Buttons dafür und die findet Ihr hier:



Hier könnt Ihr wie im Internet Explorer vor- und rückwärts navigieren. Neben dem Rückwärtsbutton ist auch ein kleines Dropdown-Menü, über das Ihr direkt eine Cursorposition in der Vergangenheit anspringen könnt. Um ehrlich zu sein: Ich benutze diese Buttons nie. Wie ich ja schon angedeutet habe, kannte ich die Shortcuts vor den Buttons und diese sind bei mir in Fleisch und Blut übergegangen:

Ctrl+Minus navigiert rückwärts. Bleibt natürlich die Frage, wie man wieder vorwärts navigiert. Man mag glauben, es wäre Ctrl+Plus. Aber das ist nicht der Fall. Das Invertieren von Shortcut-Funktionen in Visual Studio funktioniert typischerweise über ein zusätzliches Shift. (Es ist auch sinnvoll sich daran zu halten, wenn man eigene Shortcuts anlegt.)

So ist es auch hier:

Ctrl+Shift+Minus navigiert vorwärts.

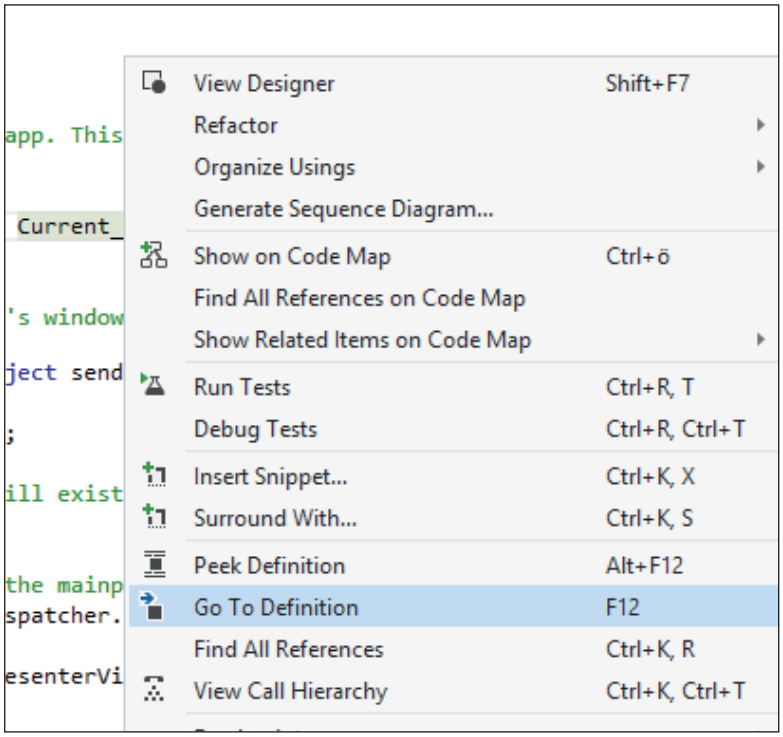
Es vergeht kaum eine Coding Session, in der ich diese Shortcuts nicht nutze. Ich hoffe, sie helfen Euch ebenso wie mir.

tl;dr

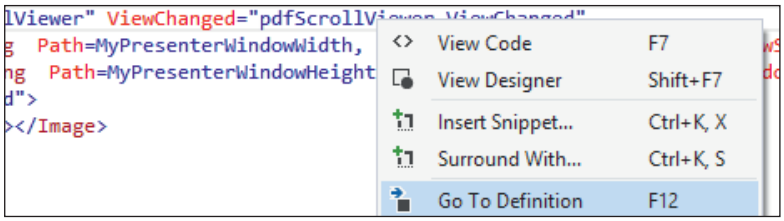
Um sich im Code-Navigationsverlauf rückwärts durch die Zeit zu bewegen, ist Ctrl+Minus der passende Shortcut. Wieder nach vorn geht es mit einem zusätzlichen Shift. Wer mag, findet auch die Buttons dazu.

20 Go To Definition (F12) und zurück

Ein absolut essentieller Shortcut ist mit Sicherheit F12 in Visual Studio. Vielleicht hat er tatsächlich aufgrund seiner Wichtigkeit einen Shortcut aus nur einer einzigen F-Taste bekommen. F12 bringt uns zur Definition einer Methode, wenn unser Cursor auf dem Methodenaufruf steht. Oder zur Definition einer Klasse, wenn der Cursor auf dem entsprechenden Classname steht. Jeder setzt das jeden Tag vermutlich mehrmals ein. Es gibt auch hier einen Weg mit der Maus, wie der Screenshot unten zeigt, dazu einfach Rechtsklick auf den Methodenaufruf machen und „Go To Definition" wählen.



Inzwischen (VS 2013) funktioniert das auch im XAML. „Endlich" wird manch leidgeplagter Zeitgenosse sagen.



Mein persönlicher Favorit ist hier allerdings die Bedienung per Tastatur. F12 kann sich wahrscheinlich jeder merken. Die Tastaturbedienung hat außerdem einen geringfügigen Vorteil: Wir können mal sehen, was passiert, wenn wir gleichzeitig Shift drücken. Wie wir bereits erfahren haben, invertiert das ja im Idealfall den Befehl – wenn wir also über F12 von den Referenzen zur Definition kommen, dann sollten wir über Shift+F12 eigentlich von der Definition zu allen Referenzen kommen, richtig? Und siehe da: Shift+F12 auf eine Methode zeigt uns alle Referenzen in einem kleinen Results-Fensterchen an:

So ein Zwischenschritt ist logischerweise notwendig – es gibt ja keine eindeutige Stelle, an die der Cursor jetzt springen könnte, da es potentiell mehrere Referenzen geben kann. Über die Pfeiltasten Up & Down können wir im Results-Fenster direkt die Stelle anwählen, die uns interessiert – es wird automatisch eine Vorschau im Editor angezeigt, und über Return können wir die Stelle anspringen.

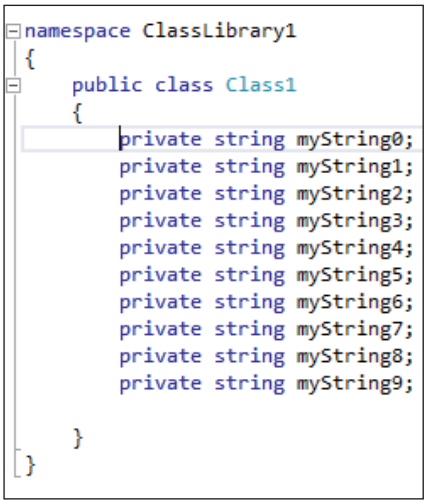
tl;dr

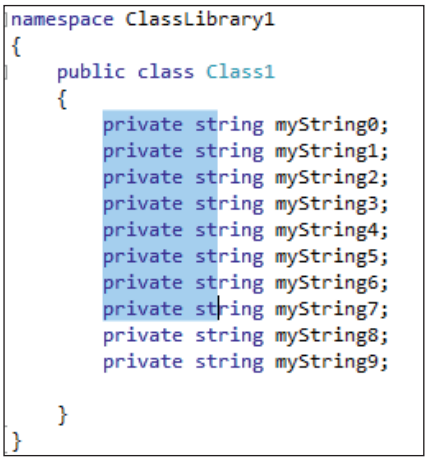
F12 bringt uns zur Definition. Shift+F12 bringt uns zurück.

21 Mehrere Zeilen gleichzeitig editieren

Es folgt der letzte Tipp vor Weihnachten – und gleichzeitig einer, der einen ebenso langen Bart hat wie der Weihnachtsmann, aber dennoch immer wieder auf ebenso große Begeisterung stößt wie ein vollgedeckter Gabentisch (und hiermit lass ich es auch mit den jahreszeitabhängigen Vergleichen).

Man kann in Visual Studio seit geraumer Zeit mehrere Zeilen gleichzeitig editieren. Man mag sich allerdings berechtigterweise fragen, wozu das gut sein soll. Stellt Euch einfach vor, Ihr legt mehrere Variablen untereinander an und markiert sie als „private". Und irgendwann später fällt Euch ein, dass Ihr dieses „private" in ein „public" umwandeln wollt. Ihr könntet über Copy & Paste vermutlich recht schnell zum Ziel kommen. Schneller geht es, wenn Ihr alle Zeilen gleichzeitig markiert und editiert.





Wie geht das? Ganz einfach: Stellt einfach den Cursor vor das erste „p" und drückt dann Alt und haltet diese Taste gedrückt. Dann könnt Ihr wahlweise mit dem Mauszeiger oder mit den Pfeiltasten eine Position annavigieren. Wohlgemerkt bei gedrückter Alt-Taste!

Auf diese Weise könnt Ihr – wie im obigen Screenshot zu sehen, absolut sinnlose Bereich markieren oder durchaus sinnvolle Bereiche wie im Screenshot unten:

```
namespace ClassLibrary1
{
    public class Class1
    {
        private string myString0;
        private string myString1;
        private string myString2;
        private string myString3;
        private string myString4;
        private string myString5;
        private string myString6;
        private string myString7;
        private string myString8;
        private string myString9;

    }
}
```

Wenn Ihr dann anfangt zu tippen, wird automatisch in allen Zeilen der Inhalt editiert – ganz so, wie man es gewohnt ist, wird der markierte Bereich mit neuem Inhalt überschrieben:

```
namespace ClassLibrary1
{
    public class Class1
    {
        public string myString0;
        public string myString1;
        public string myString2;
        public string myString3;
        public string myString4;
        public string myString5;
        public string myString6;
        public string myString7;
        public string myString8;
        public string myString9;

    }
}
```

```
namespace ClassLibrary1
{
    public class Class1
    {
        public string myString0 = String.Emp;
        public string myString1 = String.Emp;
        public string myString2 = String.Emp;
        public string myString3 = String.Emp;
        public string myString4 = String.Emp;
        public string myString5 = String.Emp;
        public string myString6 = String.Emp;
        public string myString7 = String.Emp;
        public string myString8 = String.Emp;
        public string myString9 = String.Emp;

    }
}
```

Natürlich kann man das nicht nur nutzen, um die Access Modifier zu ändern, sondern auch für Wertzuweisung und so weiter und so weiter. Wie man im Beispiel links erkennen kann, steht der Cursor gerade hinter dem „p" in Empty.

Damit verabschiede ich mich in die Feiertage.

Erholt Euch gut, schaltet den Rechner auch mal aus, und wir sprechen uns im neuen Jahr – natürlich mit weiteren Tipps & Tricks.

tl;dr

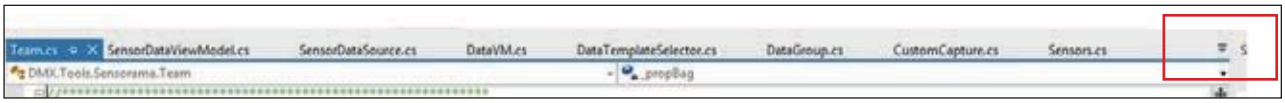
Zeichenbereiche über mehrere Zeilen lassen sich mit gedrückter Alt-Taste über die Pfeiltasten oder mit der Maus markieren.

22 Offene Dateien anzeigen

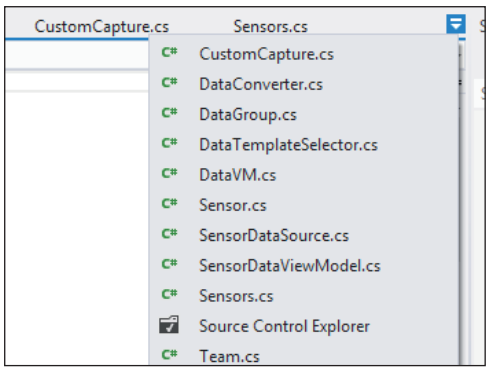
Seid Ihr auch so gestrickt, dass Ihr einen Tipp erst ausprobiert, bevor Ihr einen beschreibenden Text zu Ende lest? Ja? Gut, ich auch. In diesem Fall rate ich Euch allerdings davon ab. Lest also erst noch ein paar Zeilen, bevor Ihr Euch ins Visual Studio stürzt.

Wenn man im Studio mehrere Dateien geöffnet hat, verschwinden manche von ihnen mit der Zeit aus der Tab-Leiste, weil diese nur begrenzten Platz hat. Um sie dennoch zu sehen, kann man mit der Maus ein kleines Icon klicken und erhält dann eine Übersicht über die geöffneten Dateien.

Hier das Icon:



Hier die Übersicht:



Für die Navigation zu dieser Übersicht gibt es auch einen Shortcut – dieser lautet: Ctrl+Alt+Down. Falls Ihr es jetzt schon ausprobiert habt, kann es sein, dass Euer Bildschirm auf dem Kopf steht. Manche Grafikkartentreiber überschreiben diesen Shortcut nämlich und drehen das Bild dadurch auf den Kopf. Ich habe Euch gewarnt. Das Gemeine an der Sache ist, dass das Rückgängigmachen des kopfstehenden Bildes über einen anderen Shortcut funktioniert: Ctrl+Alt+Up.

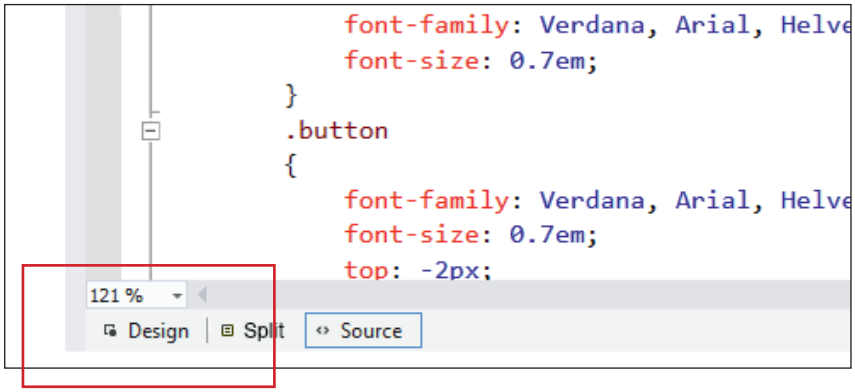
Aber vielleicht ist bei Euch ja gar nichts passiert. Dann kennt Ihr jetzt den Shortcut zur Liste der offenen Dateien: Ctrl+Alt+Down.

tl;dr

Ctrl+Alt+Down zeigt die Liste der offenen Dateien. Falls stattdessen der Bildschirm Kopf steht, kann man diesen mit Ctrl+Alt+Up wieder richtig drehen.

23 Zoom In & Out (Ctrl+Shift+, / Ctrl+Shift+.)

Zoomen im Visual Studio Editor war lange Zeit ein etwas hakeliges Thema. Man hat zwar schon seit geraumer Zeit die Möglichkeit, den Zoomfaktor per Maus über eine Dropdownbox einzustellen oder den Faktor einzugeben, das empfinde ich aber immer als nicht schnell genug. Schließlich muss ich dazu die Maus bemühen, die meist an anderer Stelle positioniert ist.

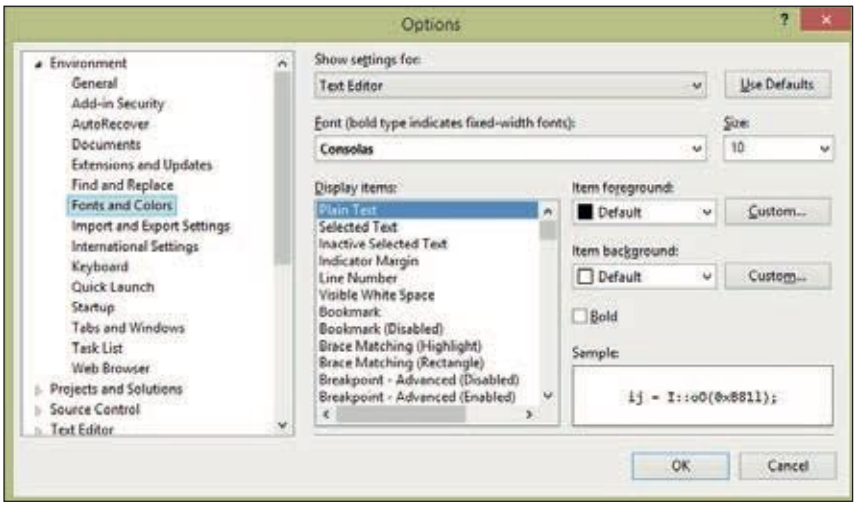


Alternativ kann ich – was mir gerade in Vorträgen entgegenkommt – über das Mausrad bei gehaltener Steuerungstaste ein- und auszoomen. Das ist meine bevorzugte Methode, weil sie relativ konsistent über verschiedene Anwendungen hinweg funktioniert.

Wenn Ihr einen Referenten seht, der das nicht tut, dann hat er vermutlich keine Maus angeschlossen und arbeitet nur mit dem Trackpad. Ihr könnt das mal beobachten – es entsteht dann regelmäßig eine relativ lange Pause, vermutlich weil er die dritte Möglichkeit nicht kennt: Die Tastatur-Shortcuts Ctrl+Shift+, und Ctrl+Shift+.

Gerade für den Fall, dass man in Vortragssituationen ist und keine Maus zur Hand hat oder mal ein paar Kollegen über den eigenen Code sehen lässt und der Bildschirm zu klein oder zu weit weg ist, ist das recht angenehm und vor allem noch schneller als das Zoomen per Mausrad.

Natürlich könnte man auch die Schriftgröße verstellen. Das geht am schnellsten, indem wir über Quick Launch nach „Font Size" suchen und dann die Schriftgröße einstellen. Um ehrlich zu sein, mache ich davon aber nie Gebrauch.



tl;dr

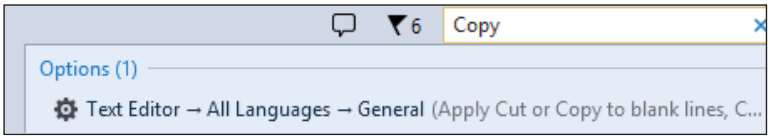
Wenn keine Maus vorhanden ist, zoomt es sich über Shortcut am schnellsten: Ctrl+Shift+, und Ctrl+Shift+.

24 Copy & Paste von Leerzeilen

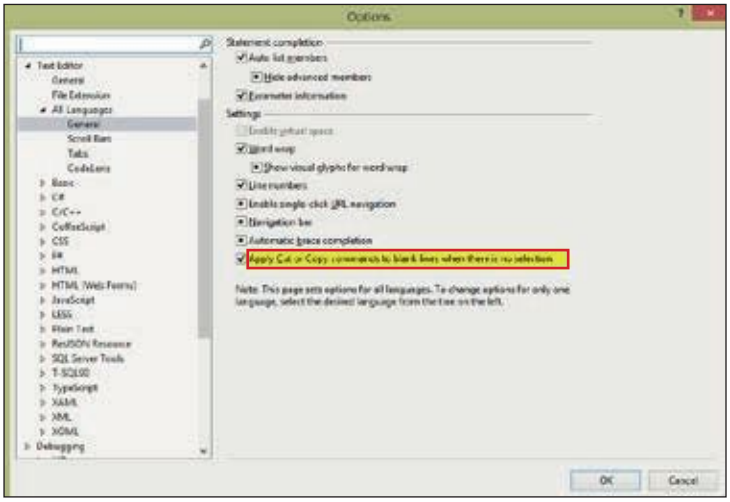
In Visual Studio gibt es seit geraumer Zeit eine Art Copy & Paste für Fortgeschrittene. Copy & Paste funktioniert ja ganz normal mit Ctrl+C, Ctrl+V. Vielleicht kennt Ihr das Problem, dass Ihr versehentlich mal eine Zeile kopiert, in der gar nix drin steht. Ihr habt dann in Eurer Zwischenablage eine leere Zeile – na super! Gleichzeitig habt Ihr vielleicht unmittelbar vorher einen wichtigen Schnipsel Code mit Ctrl+X ausgeschnitten. Wenn Ihr den dann mit einer leeren Zeile Code überbügelt habt, dann ist das natürlich doppelt ärgerlich.

Aber keine Sorge, es gibt einen Weg aus der Misere: Man kann Visual Studio dazu überreden, dass es keine leeren Codezeilen mehr kopiert.

Ich denke, die meisten werden diese Einstellung beibehalten, sobald sie mal ausgewählt ist. Wo stellt man das ein? Nun, am einfachsten gibt man in der Quick-Launch-Zeile (Ctrl+Q) einfach mal „Copy" ein.



Der erste Treffer bringt einen direkt zum Einstellungsmenü, wo die entsprechende Seite auch schon geöffnet ist.



Hier kann man ein Häkchen wegnehmen bei „Apply Cut or Copy commands to blank lines when there's no selection". Das wird das unbeabsichtigte Kopieren einer leeren Zeile verhindern. Wichtig: Das funktioniert nur, wenn man wirklich nichts selektiert hat. Wenn man mehrere leere Zeilen selektiert, dann werden diese dennoch in die Zwischenablage kopiert. Umgekehrt führt dieses Verhalten natürlich dazu, dass man jetzt manchmal Ctrl+C, Ctrl+V nutzen kann, ohne, dass was passiert – dann bleibt der „alte" Wert in der Zwischenablage. Gemeinhin sollte das aber kein Problem sein.

tl;dr

Über das Settingsmenü kann man das Kopieren von Leerzeilen verhindern.

25 Copy & Paste Zwischen-ablage-Ring

Dieser Tipp ist ein „Anhang" an den vorherigen Tipp 24 zum Thema Copy & Paste. Er ist aber zu gut, um nicht alleine gelistet zu werden: Man kann in Visual Studio auf „alte" Werte in der Zwischenablage zugreifen.

Wenn man in Visual Studio beim Einfügen von Werten aus der Zwischenablage einfach mal zusätzlich die Shift-Taste drückt und dann mehrmals Ctrl+V, wird man feststellen, dass "alte" Werte aus der Zwischenablage auftauchen. Das heißt, ich kann mehrere Werte über Ctrl+C oder Ctrl+X in die Zwischenablage verfrachten und diese dann nacheinander durchgehen. Immer wenn ich einen Wert einfüge, ist dieser komplett markiert, wird also, falls ich erneut Ctrl+V+Shift drücke, mit dem nächsten Wert aus der Zwischenablage ersetzt. Das ist wirklich sehr komfortabel, probiert es einfach aus.

Wenn ich also in dieser Reihenfolge Abschnitte markiere und in die Zwischenlage kopiere ...

```
36          // initialize
37          myGame.InitGame();
```

Ctrl+C

```
39          // set binding data context
40          myGrid.DataContext = myGame;
```

Ctrl+C

```
45          myGame.StartGame();
```

Ctrl+C

... würde ich über Ctrl+V Folgendes erhalten:

```
47
48                  myGame.StartGame();
```

Über Ctr+V+Shift erhalte ich beim ersten Mal das gleiche, beim zweiten Mal Folgendes:

```
47  // set binding data context
48          myGrid.DataContext = myGame;
```

Beim dritten Mal das:

```
47  // initialize
48          myGame.InitGame();
```

tl;dr

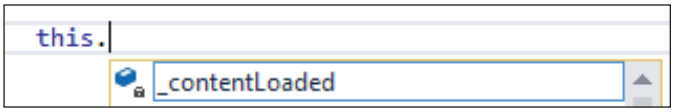
Beim Einfügen von Text aus der Zwischenablage kann man mit gehaltener Shift-Taste auf „alte" Werte zugreifen.

26 IntelliSense aufrufen (Ctrl+J)

IntelliSense ist großartig, ich denke soweit sind wir uns einig. (Ich erinnere mich noch an die Tage, als jegliche Unterstützung der IDE beim Entwickeln als Bevormundung wahrgenommen wurde. Zumindest für mich ist diese Zeit vorbei.)

IntelliSense arbeitet ja basierend auf den von uns eingegebenen Zeichen und findet dann auf Basis des vorhandenen Codes heraus, was „möglich" ist, im Sinne von „welchen Code kannst Du noch schreiben, damit der Code noch kompiliert". Was macht man aber, wenn man IntelliSense braucht, ohne schon ein paar Zeichen eingegeben zu haben, z. B. wenn man innerhalb eines neu angelegten Konstruktors auf vorhandene Properties und Methoden der Klasse zugreifen möchte und hierfür jetzt IntelliSense-Unterstützung braucht?

Der einfachste Weg ist es sicherlich, einfach „this." zu tippen. IntelliSense springt dann an und sagt mir, was möglich ist.



Selbst schon gemacht? Klar, 1000 Mal. Aber eigentlich braucht man „this." ja gar nicht, oder? Also tippt man „this.", man nutzt IntelliSense und lässt sich Vorschläge machen, wählt eine Methode aus, drückt Return und – drückt die Pos1-Taste (oder navigiert manuell an den Zeilenanfang) und löscht „this." wieder. Auch schon gemacht? Irgendwie sinnlos, oder? Gut zu wissen, dass wir nicht zwingend „this." brauchen, um IntelliSense aufzurufen: Ctrl+J macht das auch – dann braucht man danach nichts mehr zu löschen.

tl;dr

IntelliSense lässt sich über Ctrl+J manuell aufrufen.

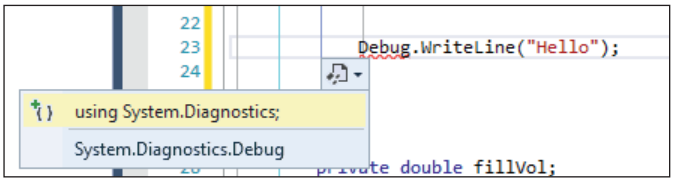
27 Die automatische Fehlerbehebungs-Wunderwaffe Ctrl+.

In vielen Fällen macht man in Visual Studio Änderungen, die dazu führen, dass der Quellcode erst mal nicht mehr lauffähig ist, weil wir noch weitere Anpassungen machen müssen. So verwendet man Methoden aus Klassen, die noch nicht über Usings importiert wurden. Oder man macht Umbenennungen, die dazu führen, dass Referenzen nicht mehr stimmen. Oder, oder, oder.

In vielen dieser Fälle gehört nicht viel Know-how dazu, um die Fehler zu beheben – letztlich würden wir das als Entwickler sowieso „jetzt dann gleich" machen, aber unsere Vorgehensweise war gewissermaßen in der falschen Reihenfolge, so dass der Code erstmal nicht kompiliert. Und nichts ist höher zu bewerten als kompilierender Code, oder?

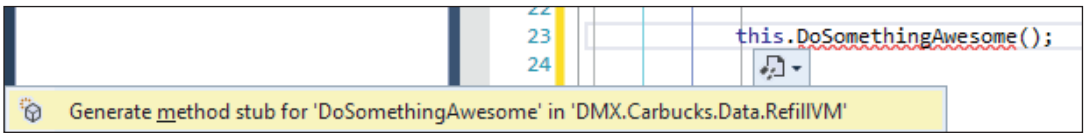
Visual Studio kann in vielen Fällen unterstützen – es ahnt quasi schon, was noch fehlt. Wenn wir in Zukunft durch eine rote Unterkringelung auf Fehler aufmerksam gemacht werden, lohnt es sich auf jeden Fall, einfach mal die entsprechende Stelle mit dem Cursor anzunavigieren und einfach mal „Ctrl+." zu klicken. Im Idealfall kriegen wir dann automatisch ein paar Korrekturvorschläge gemacht, die wir auswählen können, und der Code kompiliert wieder ohne größeren Aufwand.

Beispiele gefällig?



Verwenden von Debug.Writeline ohne den entsprechenden Namespace eingebunden zu haben – der Namespace wird automatisch eingebunden.

Verwenden einer Methode, die es gar nicht gibt: Ctrl+. bietet an, eine entsprechende Methode zu erzeugen:



tl;dr

Wenn Visual Studio irgendwas rot unterkringelt, lohnt es sich, einfach mal Ctrl+. zu klicken. Vielleicht ist das der schnellste Weg zur Fehlerbehebung.

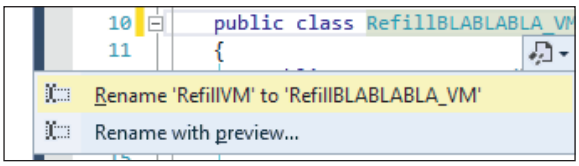
28 Klassenname und Dateiname gleichzeitig umbenennen

Zum Umbenennen von Klassen reicht es in der Regel, wenn man den Cursor auf den Klassennamen setzt und einfach den Namen ändert.

```
public class RefillVM : NotifyPropertyChangedBase
```

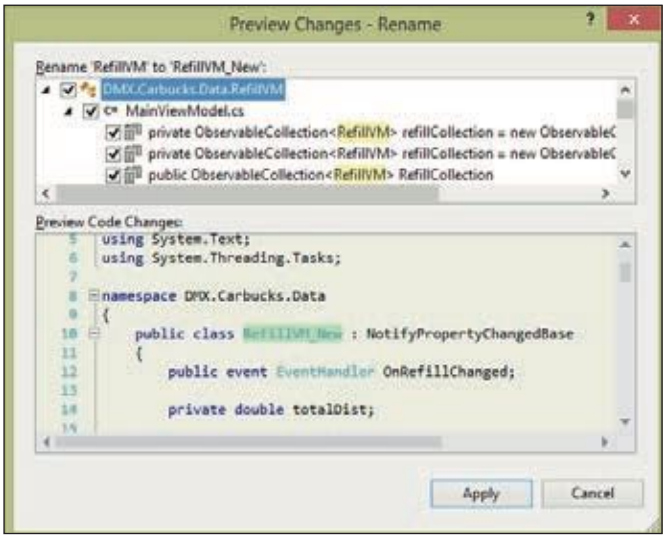
Dabei ergibt sich das Problem, dass eventuell vorhandene Referenzen auf den Typen nicht mitangepasst wurden – deshalb wird uns auch eine kleine rote Markierung angezeigt.

Am leichtesten ist dann, einfach Ctrl+. zu drücken – unsere Wunderwaffe zur automatischen Fehlerkorrektur in Visual Studio.



Ein Kontextmenü zeigt uns dann die Möglichkeit, eine saubere Umbenennung durch VS durchführen zu lassen, alle Referenzen werden automatisch mitangepasst.

Alternativ können wir auch den Namen der Klasse mit dem Cursor ansteuern und dann Ctrl+R, Ctrl+R drücken. Das ruft den Rename-Dialog auf, wir können den neuen Namen wählen und bekommen dann eine Vorschau auf die angepassten Referenzen, mit der Möglichkeit, für manche Referenzen das Renaming nicht durchzuführen.



In beiden Fällen bleibt allerdings etwas auf der Strecke: der Dateiname. In C# ist es ja relativ weit verbreitet, als Dateinamen den Klassennamen zu nutzen und jeweils eine Klasse pro Datei zu implementieren. Wer diesem Schema folgt, wird bei der Umbenennung der Klassennamen in C# auch die Dateinamen nachziehen wollen. Bei der bisherigen Vorgehensweise ist das ein manueller zusätzlicher Schritt.

Schneller geht's anders herum: Wenn man im Solution Explorer zunächst die Datei umbenennt, kommt anschließend die Abfrage, ob man auch den Klassennamen anpassen will – klar, wollen wir das.

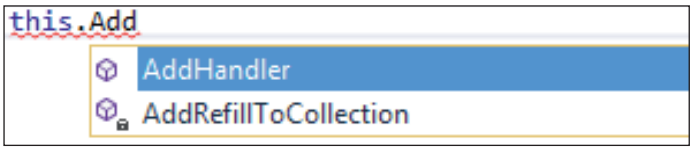
Referenzen werden automatisch mitangepasst. Wichtig: Das funktioniert nur, wenn der Klassenname bereits dem Dateinamen entspricht!

tl;dr

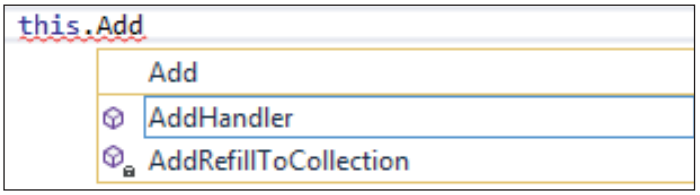
Umbenennung einer .cs Datei im Solution Explorer sorgt auch für eine automatische Umbenennung der darin implementierten Klasse, wenn diese den gleichen Namen trägt.

29 IntelliSense bändigen (Ctrl+Alt+Space)

In einem meiner vorherigen Blogposts habe ich ja schon beschrieben, wie man IntelliSense aufruft. Und insgesamt ist IntelliSense natürlich großartig. Manchmal treibt es einen aber auch in den Wahnsinn:
Man stelle sich vor, man fängt an zu tippen und verwendet dabei einen Methodennamen auf einer Klasseninstanz, wohlwissend, dass es diesen Methodennamen nicht gibt. Vielleicht, um anschließend mit der Ctrl+. Wunderwaffe automatisch einen Method-Stub zu erzeugen. Dann kann es passieren, dass einem IntelliSense in die Quere kommt, mit der Absicht, eigentlich zu helfen. IntelliSense weiß ja, welche Namespaces, Klassen & Methoden verfügbar sind und will korrigierend eingreifen und macht im schlimmsten Fall aus einer Eingabe wie dieser ...



... also dem Aufruf der Methode „Add" auf „this" per Autovervollständigung ein „this.AddHandler", sobald ich die Return-taste drücke – aber nicht nur bei der Returntaste, auch das Betätigen der Leertaste führt zu diesem Verhalten. Nicht, weil IntelliSense es böse mit uns meint, sondern weil IntelliSense es nicht besser weiß und uns doch nur helfen will.
In diesem Fall ist es gut zu wissen, dass wir IntelliSense vom vorliegenden Completion-Mode in den Suggestion-Mode schalten können. Dann listet IntelliSense auch alle Member, wir können aber tippen, was wir wollen. Wir bekommen aber auch keine Auto-Korrektur mehr. Das Umschalten geht sehr schnell über einen Shortcut – alles andere wäre auch nicht praktikabel. Der Shortcut dazu lautet: Ctrl+Alt+Space.



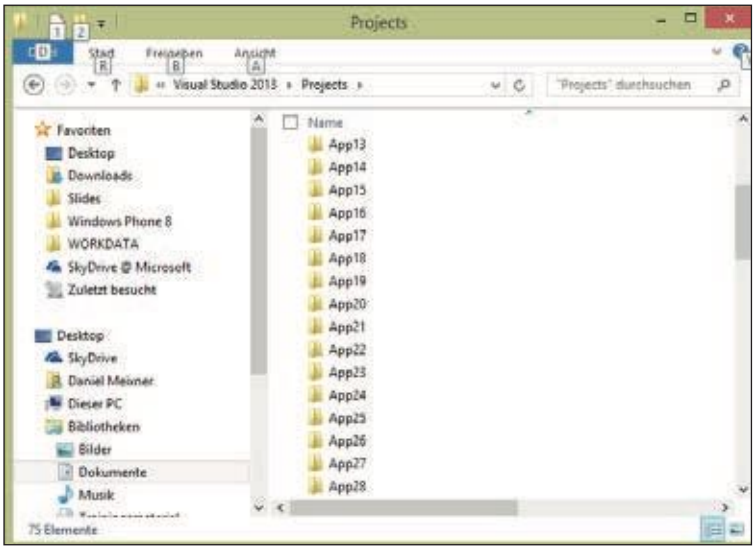
Wie man sieht, ändert sich optisch nicht allzu viel. Die gewünschte Wirkung bleibt aber nicht aus: Ich kann tippen, was ich will. Das Rückschalten in den Completion-Mode funktioniert mit dem gleichen Shortcut.

tl;dr

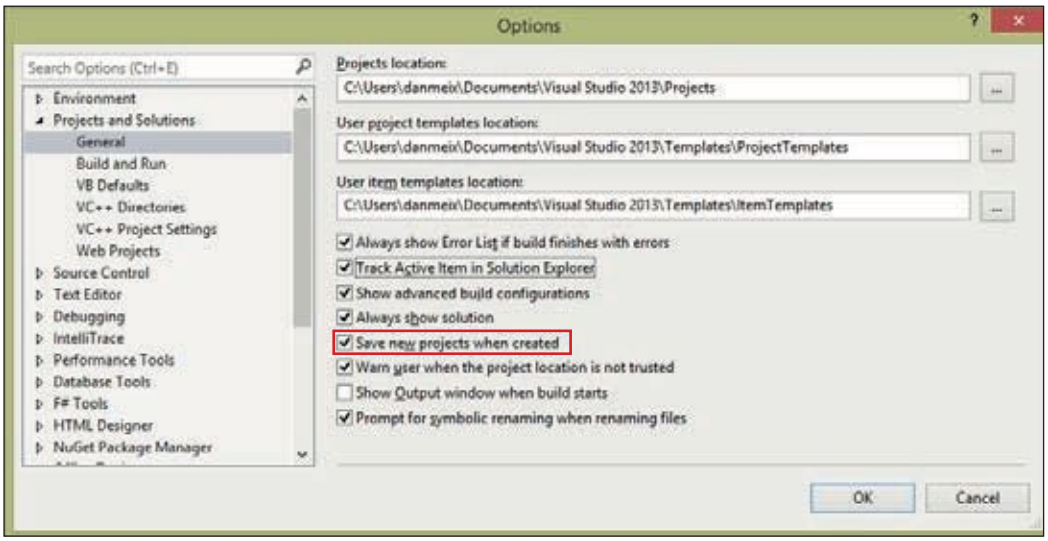
Wenn IntelliSense beim Entwickeln per Autovervollständigung ungewollte Neben-wirkungen verursacht, hilft mit hoher Wahrscheinlichkeit ein Umschalten in den Suggestion-Mode per Ctrl + Alt + Space.

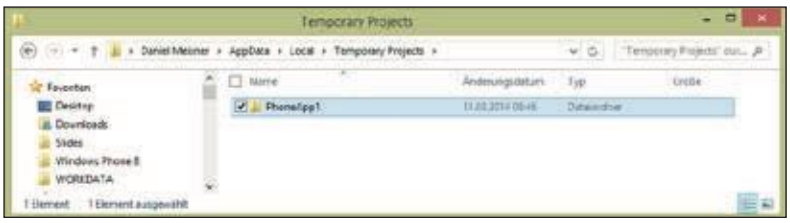
30 Projekte nicht speichern

Wer in Visual Studio viel herumexperimentiert und einfach mal ein paar Dinge ausprobiert, der wird feststellen, dass mit jedem Testprojekt, das angelegt wird, dieses Testprojekt auch auf die Festplatte geschrieben wird und dort sein Dasein fristet, bis man es manuell wieder entfernt. Letzteres findet bei mir relativ selten statt, in der Regel dauert es etwas, bis mich der Ordnungswahn packt und ich nicht benötigte Projekte aufräumen will. Das resultiert in einer Vielzahl sinnloser Projektfolder, wie Euch dieser Screenshot zeigt.

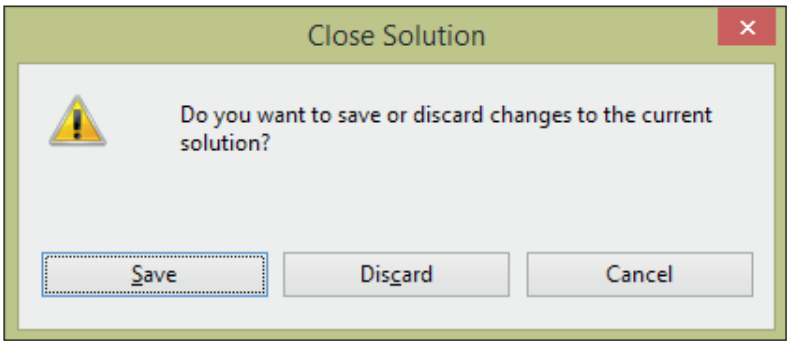


Kommt Euch bekannt vor? Ok. Es gibt Abhilfe. Man kann Visual Studio dazu überreden, Testprojekte erst mal nicht zu speichern, bis man dies explizit selbst tut. Dazu kann man in den Optionen das „Save new Projects when created" Häkchen abwählen.



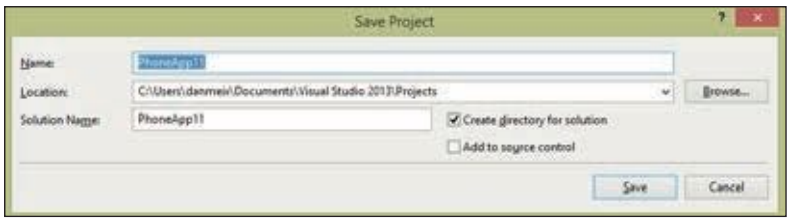


Neu angelegte Projekte liegen dann zunächst mal in C:\Users\<user>\AppData\Local\Temporary Projects.

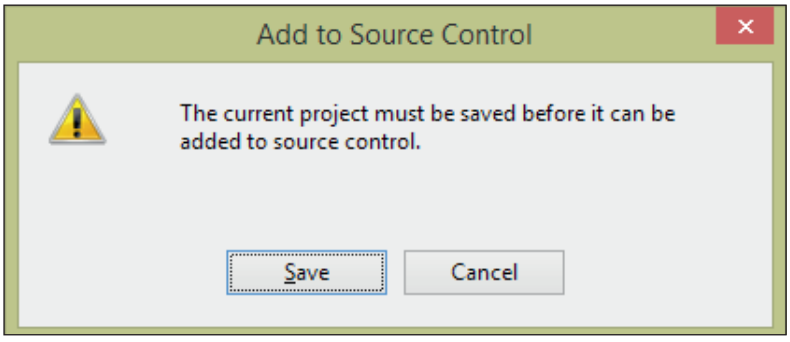


Beim Schließen des Visual Studios kommt noch eine letzte Möglichkeit, das Projekt zu speichern:

Wer hier auf „Discard" drückt, der sorgt dafür, dass das Projekt sofort wieder aus dem temporären Folder gelöscht wird.



Natürlich hat man aber jederzeit die Möglichkeit, über Ctrl+Shift+S (Save All) das Projekt zu speichern und hier einen anständigen Namen zu vergeben und einen Zielspeicherort auszuwählen.



Gute Sache also – solange man das Speichern wichtiger Projekte nicht versehentlich vergisst. Aber die sollten ohnehin in die Quellcodeverwaltung aufgenommen werden. Wer das mit einem temporären Projekt versucht, bekommt den Hinweis, dass genau das nicht geht, da es erst dauerhaft gespeichert werden muss, mit der Möglichkeit, das dauerhafte Speichern gleich vorzunehmen. So soll es sein!

tl;dr

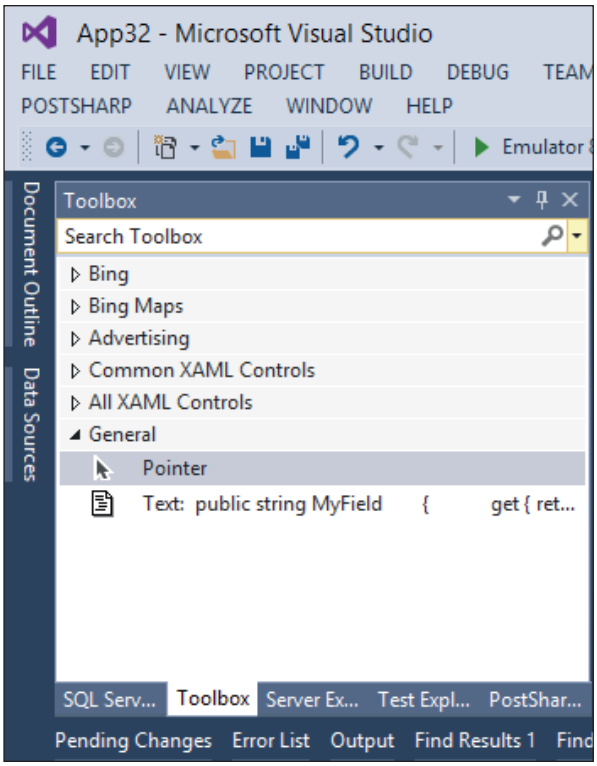
Über die Visual Studio Settings kann man dafür sorgen, dass Projekte nur temporär gespeichert werden.

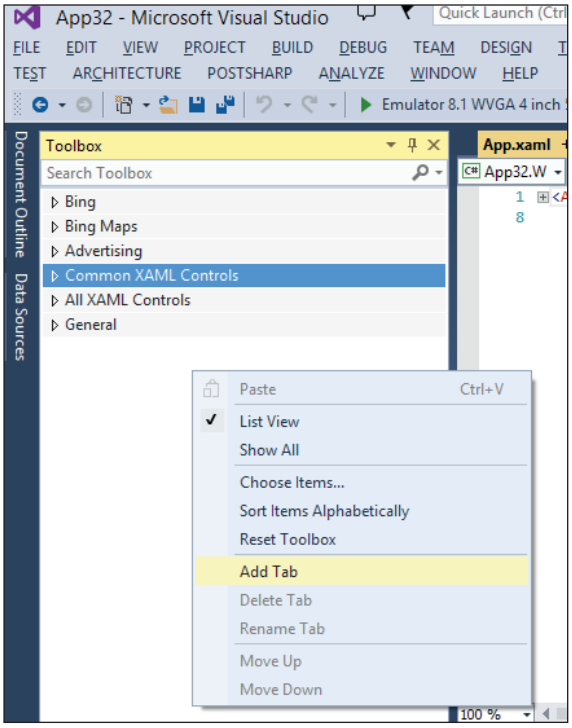
31 Code in der Toolbox ablegen

Bereits in den Tipps 24 und 25 habe ich Methoden beschrieben, das Copy & Paste-Verhalten in Visual Studio optimal auszunutzen. Manchmal will man aber nicht Copy & Paste verwenden – schließlich ist Copy & Paste „flüchtig", überlebt also keinen Visual Studio-Neustart.

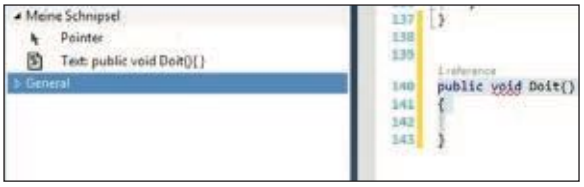
Wer einen Weg sucht, Codeschnipsel für die spätere Verwendung aufzuheben, dem bietet Visual Studio eine sehr komfortable Möglichkeit, die ich sehr gerne selbst häufig verwende – insbesondere für Demos auf Konferenzen. Bevor ich hier erläutere, was ich meine: Natürlich ist die beschriebene Methode nicht als Ersatz für eine Quellcodeverwaltung zu sehen. Auch das (unglaublich coole) Shelving im Team Foundation Server soll damit nicht an Bedeutung verlieren. Die hier vorgestellte Methode ist eher was für den Fall, dass man sich ein paar Programmschnipsel zur schnellen Verfügbarkeit zurechtlegen will – ohne irgendeinen Overhead.

So geht's: Wie Ihr wisst, gibt es in Visual Studio das Toolbox-Window:





Abhängig davon, welche Datei gerade geöffnet ist, seht Ihr hier Einträge, die Ihr per Doppelklick im Code integrieren könnt. Diese Toolbox ist auch in der Lage, Euren eigenen Code entgegenzunehmen. Dazu könnt Ihr einfach einen Schnipsel Code markieren und per Drag & Drop in die Toolbox ziehen und dort fallenlassen.



Innerhalb der Toolbox könnt Ihr auch für etwas Struktur sorgen, indem Ihr zusätzliche Sektionen erstellt.

Wenn Ihr jetzt Visual Studio schließt und wieder öffnet, werdet Ihr Eure Schnipsel wiederfinden. Zum Einfügen reicht ein Doppelklick auf den entsprechenden Schnipsel. Beachtet dabei zwei Dinge:

1. Die letzte Visual Studio-Instanz gewinnt: Wenn Ihr mehr als eine Instanz geöffnet habt, dann überschreibt die zuletzt geöffnete Visual Studio-Instanz die Information der anderen.

2. Die Inhalte der Toolbox sind sprachspezifisch. Wenn Ihr einen Schnipsel in einer Javascipt-Datei in die Toolbox legt, werdet Ihr ihn in einer C# Datei nicht angezeigt bekommen.

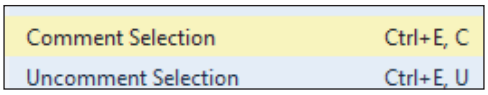
tl;dr

Die Toolbox kann über Drag & Drop mit eigenen Inhalten befüllt werden.

32 Aus- und Einkommentieren

Einer der für mich am häufigsten verwendeten Shortcuts (vermutlich gleich nach Copy & Paste) ist das Auskommentieren. Ich weiß nicht, wie lange es her ist, dass ich zum letzten Mal selbst „//" oder „/* … */" eintippt habe. Ganz spontan fällt mir da ein: Wie ist eigentlich die Syntax, um in CSS-Dateien einen Kommentar zu schreiben? Und damit man sich das nicht merken muss und um überhaupt viel schneller zu sein, gibt es natürlich wieder einen Shortcut.

Auch, wenn mir Visual Studio selbst als entsprechenden Shortcut Ctrl+E, C und Ctrl+E, U empfiehlt, hat sich bei mir komischerweise Ctrl+K, C und Ctrl+K, U festgesetzt. Vielleicht war das in der Vergangenheit Standard und funktioniert jetzt noch aus Kompatibilitätsgründen. Oder ich habe wieder irgendwas beim Herumexperimentieren verstellt … Wie auch immer, natürlich funktioniert auch, was VS mir vorgibt.




```
//public ObservableDictionary DefaultViewMode
//{
//    get { return this.defaultViewModel; }
//}
```

Natürlich funktionert das in C#…

```
////public ObservableDictionary DefaultViewModel
////{
////    get { return this.defaultViewModel; }
////}
```

… bei wiederholtem Auskommentieren des gleichen Blocks kommen einfach neue „//" dazu.

```
<!--<Grid.RowDefinitions>
    <RowDefinition Height="140"/>
    <RowDefinition Height="*"/>
</Grid.RowDefinitions>-->
```

Gleiches gilt, wenn man einen Block in XAML auskommentiert. Dann sieht das ungefähr so aus:

In der Vergangenheit gab es bisweilen Probleme, wenn man einen bereits auskommentierten XAML-Block erneut auskommentiert hat. Das ist nicht länger der Fall. Mit der neuesten Version von Visual Studio hat VS hier dazugelernt und kommentiert sinnvoll aus, baut also <!- - und - - !> clever vor und hinter bereits auskommentierte Bereiche.

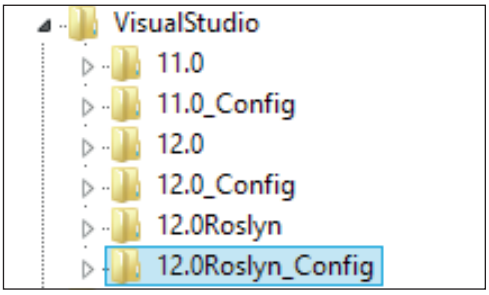
Gleiches gilt für XAMARIN basierte Android-Projekte und deren AXML-Dateien. Und – olé – seit VS 2012 können wir auch in CSS-Dateien via Shortcut auskommentieren.

tl;dr

Auskommentieren und Einkommentieren geht am schnellsten via Ctrl+E, C und Ctrl+E, U.

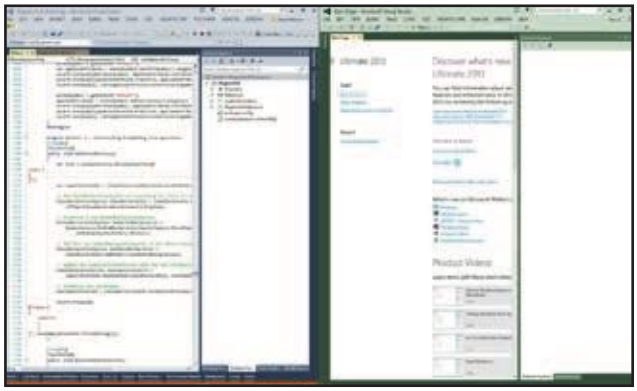
33 Verwirrung vermeiden bei mehreren Visual Studio-Hives

Wer Visual Studio-Extensions schreibt oder sich mit Roslyn beschäftigt und Compiler-Erweiterungen schreibt, der wird eher früher als später die Extension testen wollen. Die Extension liegt – wie man das eben erwartet - im Output-Verzeichnis des Projekts und endet mit .vsix. Um das .vsix zu testen, ist es nötig, es in Visual Studio zu installieren. Würde man während der Entwicklung des .vsix das .vsix im gleichen Visual Studio installieren, das auch zum Entwickeln verwendet wird, dann wird es gefährlich – es könnte passieren, dass die Extension fehlerhaft ist und deshalb Visual Studio durcheinanderbringt oder gar nicht mehr lauffähig macht.



Aus diesem Grund werden .vsix in der Regel in „anderen" Visual Studio-Instanzen getestet – bisweilen nennt man diese auch „experimentelle Instanzen" . Wenn man innerhalb des .vsix-Projekts auf F5 drückt, wird eine solche neue Instanz von Visual Studio gestartet. Die neue Instanz von Visual Studio bezieht ihre Settings, Infos, Plugins aber nicht von der Hauptinstanz, sondern aus anderen Quellen – sie hat auch einen eigenen Registryhive, der neben dem Registryhive des „echten" Visual Studio liegt, den sogenannten „experimentellen Hive".

Insofern kann da eigentlich gar nicht viel schiefgehen. Allerdings wird es bisweilen in der Praxis kompliziert: Wenn man mehrere Visual Studio-Instanzen offen hat, ein paar mal auf F5 drückt und dann auch noch die experimentellen Instanzen laufen, verliert man schnell die Übersicht, da nicht mehr auf den ersten Blick erkennbar ist, was jetzt eigentlich eine experimentelle Instanz ist und was nicht. Die experimentellen Instanzen sind ja voll lauffähig. Man kann da also auch komplett neue Projekte anlegen usw. Davon würde ich allerdings abraten, weil das in der Regel nur zu Verwirrung führt – aber genau darum geht's ja: Wie vermeidet man die Verwirrung?



Ich selbst habe dafür eine einfache Lösung gefunden: Die experimentellen Instanzen von Visual Studio sind voll lauffähig und unterstützen somit auch Plugins und Extensions. Ich kann also in der experimentellen Instanz von Visual Studio einfach das Color Theme Plugin herunterladen und dann in der experimentellen Instanz die Farbe z. B. auf grün setzen. Das bezieht sich dann, wie gesagt, ausschließlich auf die experimentelle Instanz und nicht auf meine Haupt-Instanz. Durch die farbliche Markierung sehe ich auf den ersten Blick, was experimentell ist und was nicht. Das Plugin überlebt auch Neustarts und kann jederzeit wieder deinstalliert werden.

tl;dr

Unterschiedliche Visual Studio-Hives unterstützen separate Plugins und Settings – darüber kann man unter anderem unterschiedliche Farbschemas festlegen.

34 Interfaces generieren lassen (Ctrl+R, Ctrl+I)

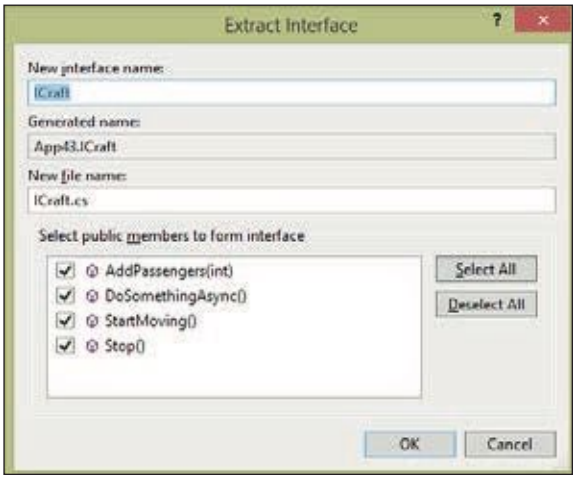
Es wird wieder einmal Zeit für einen Tipp. Und weil ich gerade dabei bin, eine Visual Studio Productivity Session vorzubereiten, ist mir folgender Tipp über den Weg gelaufen, den ich gerne mit Euch teile: Man kann in Visual Studio Interfaces, basierend auf Klassendefinitionen, automatisch generieren zu lassen. Dabei werden typische Namenskonventionen (ein großes „i" vor dem Klassennamen) eingehalten, Files automatisch generiert und eben die (möglichen) Interfaces auf Basis der öffentlichen Methoden automatisch erkannt.

So geht's:

```
using System.Threading.Tasks;

namespace App43
{
    public class Craft
    {
        private double speed;
        private int wheels;
        private int passengers;

        public void StartMoving()
        {
            // ...
        }
        public void Stop()
        {
            // ...
        }
        public int AddPassengers(int p)
        {
            //...
            return 42;
        }
        private void DoSomethingPrivate()
        {
            //...
        }
        async public Task<int> DoSomethingAsync()
        {
            //
            await Task.Delay(100);
            return 42;
        }
    }
}
```

Wenn man in Visual Studio eine Klasse definiert hat, nennen wir sie exemplarisch mal „Craft", dann fängt man früher oder später typischerweise an, Methoden und Properties dafür zu definieren. Das sieht dann z. B. so aus:

Wenn man hier auf den Klassennamen (Craft) den Cursor setzt und dann Ctrl+R, Ctrl+I aufruft, dann erscheint folgender Dialog:



Wie man sieht, schlägt VS vor, ein Interface für diese Klasse ICraft zu benennen. Hierfür wird eine Datei ICraft.cs erstellt. Die öffentlichen Methoden der Klasse werden erkannt und können automatisch im Interface definiert werden. Falls nicht gewünscht, kann man hier auch abhaken.

Der erzeugte Inhalt sieht dann aus wie folgt:

```
using System.Threading.Tasks;

namespace App43
{
    1 reference
    interface ICraft
    {
        1 reference
        int AddPassengers(int p);
        1 reference
        Task<int> DoSomethingAsync();
        1 reference
        void StartMoving();
        1 reference
        void Stop();
    }
}
```

Der Code der Klasse Craft wird automatisch angepasst, so dass Craft ICraft implementiert.

```
0 references
public class Craft : ICraft
{
    private double speed;
    private int wheels;
```

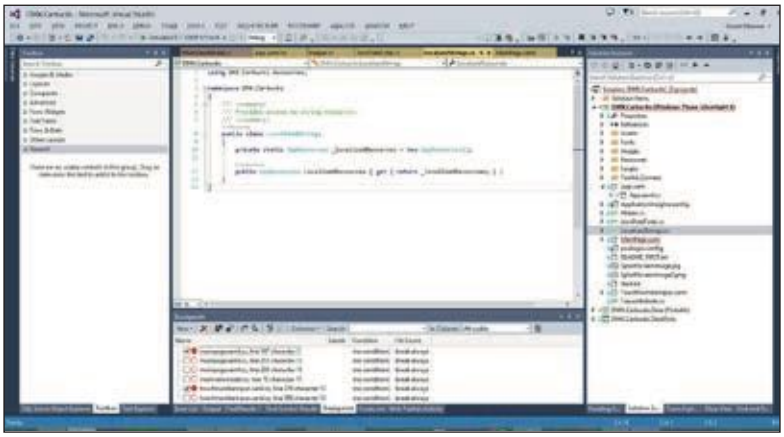
Das ist nur eines von vielen eingebauten Features, die bei der Code-Umstrukturierung und beim Refactoring helfen. In Kürze mehr.

tl;dr

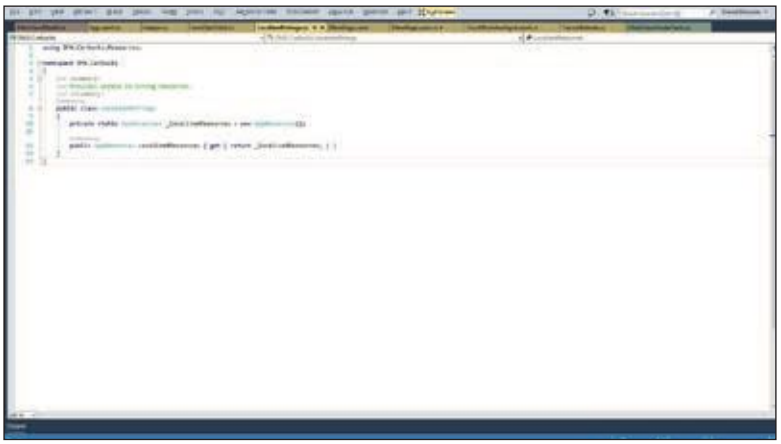
Über Ctrl+R, Ctrl+I kann man Interfaces automatisch generieren lassen.

35 Mehr Code sehen (Alt+Shift+Return)

Kurz vor dem Urlaub ein Tipp für alle, die sich in Visual Studio in erster Linie für den Code interessieren. Wer will, kann Visual Studio dazu überreden, das Editor-Fenster in den (fast) Fullscreen-Modus zu bringen. Das ist nützlich, wenn man den Code präsentiert, mit langen Codezeilen arbeitet oder sich einfach auf den Code und nicht auf das Drumherum konzentrieren möchte.



Standardmäßig sieht Visual Studio ungefähr so aus – abhängig von Eurer persönlichen Anpassung seht Ihr vielleicht andere Tool-Fenster, aber ich denke der Punkt wird klar: Der Quellcode nimmt nur einen Teil der zur Verfügung stehenden Fläche ein.



Was man bisweilen will, ist eine Möglichkeit, um schnell umzuschalten zu einer Ansicht, die sich ausschließlich auf den Code konzentriert. Die Tastenkombination Alt+Shift+Return macht genau das. Eine Fullscreen-Ansicht wird sichtbar, in der das Codefenster maximiert wird. Das Umschalten dauert einen kurzen Augenblick, ist aber um Welten schneller als das manuelle Abschalten oder Ausblenden aller Tool Fenster. Die Ansicht, die wir erhalten, ist folgende: Code soweit das Auge reicht.

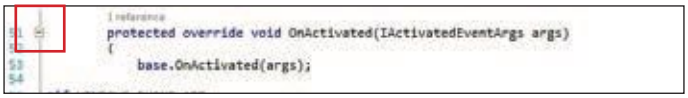
Das Rückschalten geht übrigens genauso: Erneutes Tippen von Alt+Shift+Return bringt uns zur ursprünglichen Ansicht zurück. Ich benutze die Ansicht sehr gerne bei Präsentationen, wenn es um viel Quellcode geht. Natürlich sollte man hier gegebenenfalls die Schriftart noch etwas hochdrehen, damit auch die Zuhörer in der letzten Reihe noch was erkennen können. Aber das habe ich ja schon mal beschrieben.

tl;dr

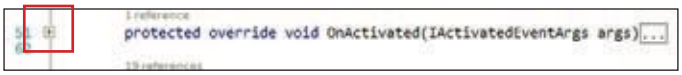
Alt+Shift+Return maximiert das Codefenster.

36 Code aufklappen und zuklappen

Der letzte Tipp hat sich damit beschäftigt, mehr Code zu sehen. Heute geht es unterm Strich darum, weniger Code zu sehen. Wer mit großen Dateien arbeitet, der will sich vielleicht erst einmal eine Übersicht darüber verschaffen, wie der Code aufgebaut ist. Man möchte also wissen: welche Methoden gibt es? Dabei interessiert mich die Implementierung der Methoden gar nicht so sehr; wenn die Namen der Methoden aussagekräftig sind, dann ist mir damit genug geholfen. Das gleiche gilt auf allen weiteren Ebenen – vielleicht interessieren mich die Klassen innerhalb einer Datei nicht im Detail, sondern ich möchte nur wissen, welche es überhaupt gibt.

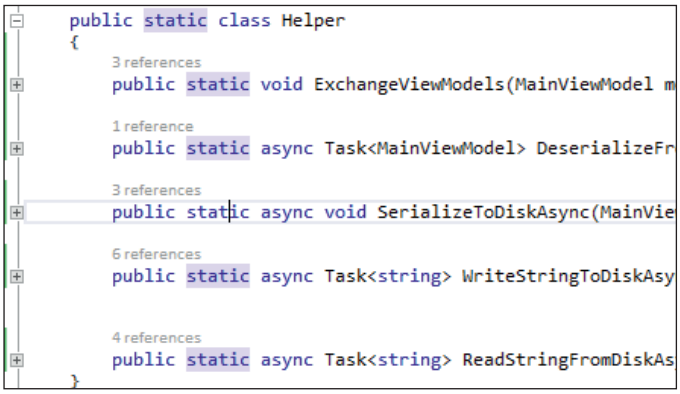


Wie Ihr in VS sicher schon entdeckt habt, gibt es ein kleines Minuszeichen vor allen Blöcken, die mit geschweiften Klammern voneinander getrennt sind:



Per Mausklick auf dieses Minus verschwindet der zugehörige Code und wird hinter einer einzelnen Zeile Code versteckt:

Das funktioniert im Übrigen auch für die geliebten, gehassten, in jedem Falle umstrittenen Regions in C#:



Um sich wirklich einen schnellen Überblick zu verschaffen, muss man nur genügend Blöcke zuklappen. Das sieht dann so aus und ist, wie ich finde, doch recht angenehm für einen ersten Eindruck:

Wenn es viele Methoden gibt, gibt es viel zu klicken – Zeit für einen Shortcut:

Einen einzelnen Block kann man über Ctrl+M, Ctl+M auf- und zuklappen. Über Ctrl+M, Ctrl+O kann man erreichen, dass alles unterhalb von Class Level zusammengeklappt wird – perfekt für mich geeignet, so spart man sich jede Menge Klickerei. Über Ctrl+M, Ctrl+L klappt man wieder alles auf.

tl;dr

Über das kleine Plus-/Minus-Symbol am Rande des Codefensters kann man Codeblöcke auf- und zuklappen. Dafür gibt es auch Shortcuts, die einem das Leben noch leichter machen.

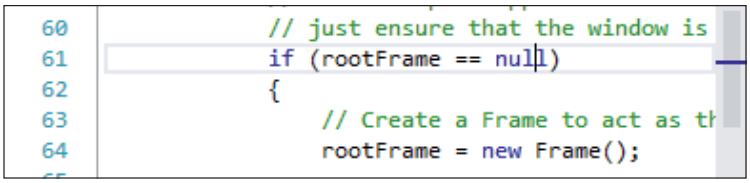
37 Eine Scrollbar mit Mehrwert

Was kann man schon Spannendes über eine Scrollbar berichten? Irgendwie hilft sie uns ja doch nur dabei, den Bildschirminhalt nach oben und unten zu schieben, oder?

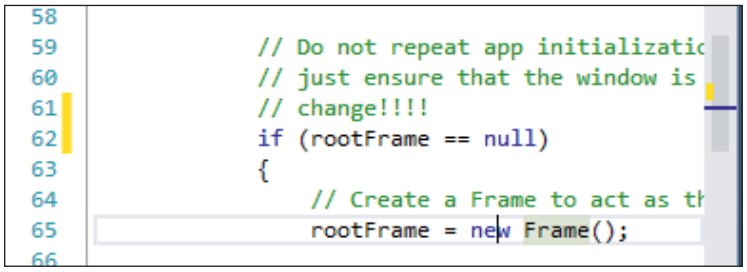
Vielleicht sollten wir im Falle von Visual Studio etwas genauer hinsehen. Die Scrollbar ist nämlich ziemlich clever. Neben der eigentlichen Funktion des Scrollens bietet sie uns auch Informationen darüber, welcher Inhalt sich an welchen Stellen im Code befindet. Und das ist schon ziemlich cool.

Mit unterschiedlichen Farbcodes werden folgende Stellen im Code markiert:

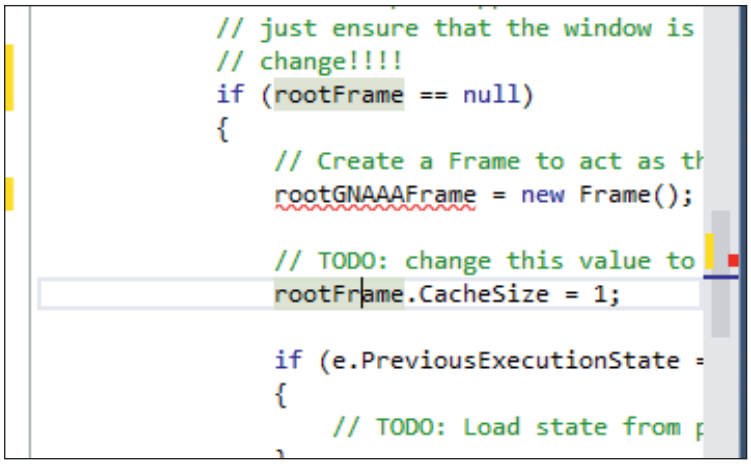
Ein blauer Strich bedeutet: Hier befindet sich der Cursor.



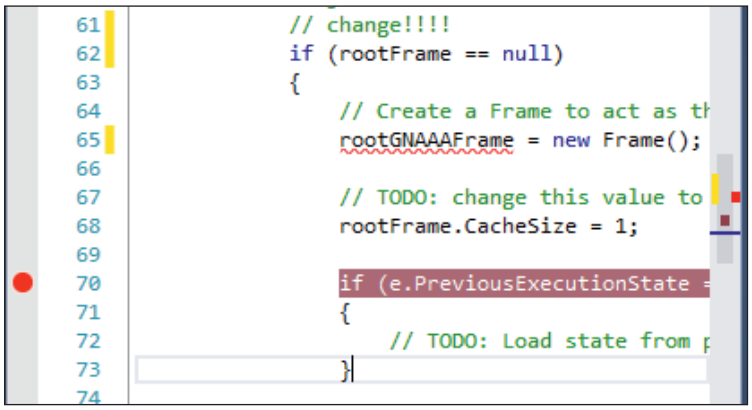
Eine gelbe Markierung bedeutet: Hier wurde eine Änderung gemacht.



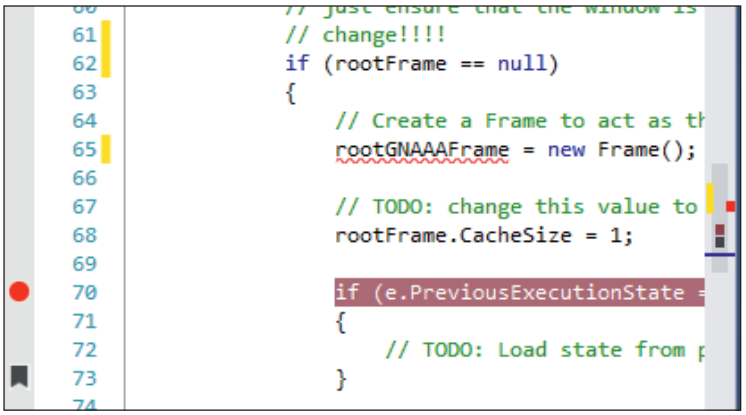
Ein rotes Quadrat bedeutet: Hier ist ein Fehler im Code, den Visual Studio entdeckt hat (und der uns daran hindern kann zu kompilieren).

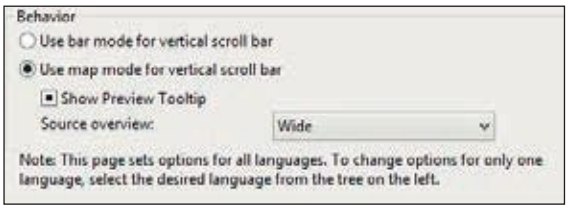


Ein dunkelroter Punkt informiert uns darüber, dass hier ein Breakpoint gesetzt wurde – sehr hilfreich bei Debuggingsessions!



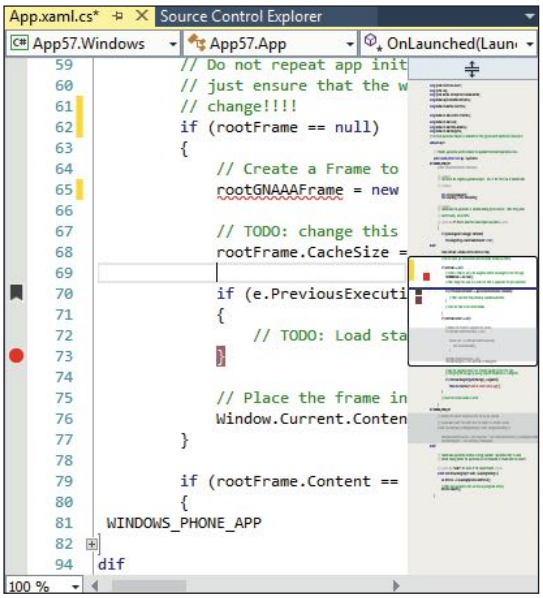
Ein schwarzes Lesezeichen-Symbol informiert uns über Lesezeichen:





Alles in allem also nützliche Hinweise, um das Navigieren im Code zu erleichtern. Das ist aber noch nicht alles. Wem die Darstellung zu klein ist, der kann die Scrollbar verbreitern – das geht über einen Rechtsklick auf die Scrollbar und das Aufrufen der Scorllbaroptions.

Hier kann man vom „Bar Mode" in den „Map Mode" umschalten und bswp. die breite Darstellung wählen.



Die Darstellung der Scrollbar ist dann wirklich richtig breit und bietet eine Art Vorschau auf die Codestruktur – neben den bereits erwähnten farblichen Markierungen. Warum ist das hilfreich? Weil man dann zum einen genau weiß, wo man sich innerhalb der Datei befindet. Bei langen Codefiles kann das sehr hilfreich sein. Des Weiteren bietet es mir eine farbliche Vorschau auf die Syntax – das heißt, ich erkenne hier den Codeaufbau wieder und bsp. Kommentare etc. stechen deutlich heraus. Sehr häufig ist es ja so, dass man als Entwickler Code nicht über Suchfunktionen sucht, sondern manuell – weil man nicht genau weiß, was da steht, aber sich am Aufbau des Codes orientiert. Zuletzt bietet einem diese Ansicht auch eine bessere Einschätzung darüber, in welchem Gültigkeitsbereich man sich befindet, da ja die Einrückungen analog zu Code angezeigt werden. Schöne Sache also, vorausgesetzt, man hat genug Platz auf dem Bildschirm.

Für Leute mit Touchscreen, die das Touch-Scrolling in Visual Studio vermissen, bietet die Scrollleiste natürlich noch den zusätzlichen Vorteil, dass man sich leichter tut zu scrollen – schließlich trifft man hier besser. Aber das ist ja nur eine temporäre Lösung, das Scrollen per Touch wartet ja schon in VS 2014 auf uns.

Wer jetzt mit der Maus über die Scrollbar fährt, wird bemerken, dass eine Vorschau des Codes an dieser Stelle angezeigt wird. Wer also mal eben sehen will, was an anderer Stelle im Code geschrieben wurde, kann die genannten Hinweise (farbliche Markierungen und Codestruktur in der Scrollbar) nutzen, um die Stelle von Interesse aufzufinden, und bekommt dann bei Positionierung des Mauszeigers an dieser Stelle den Code angezeigt – ohne die Stelle im Code zu verlassen, an der er sich befunden hat. Für alle, denen diese Funktion zu abgefahren ist, lässt sich das auch deaktivieren.

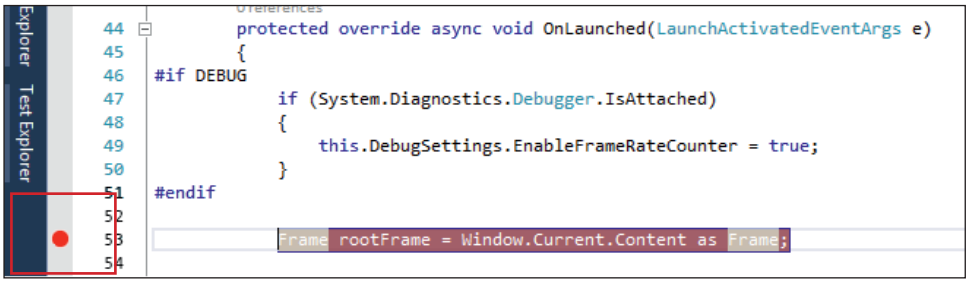
tl;dr

Die Scrollbar zeigt farbige Markierungen zur besseren Orientierung im Code – um mehr Komfort zu genießen, kann man sie auch verbreitern.

38 Break ohne Breakpoint (Ctrl+F10)

Ich gebe zu, dieser Tipp hat ein klein wenig magisches Flair – taugt also gegebenenfalls auch zum Beeindrucken von Kollegen. Und aus gegebenem Anlass möchte ich auch kurz darauf hinweisen, dass die Übersetzung von „Breakpoint" nicht „Brechpunkt" ist, sondern „Haltepunkt". In freier Wildbahn höre ich aber eigentlich nur „Breakpoint".

Einen Breakpoint setzt man ja gewöhnlich mit F9 oder eventuell mit der Maus per Klick in die linke Randleiste des Codeeditorfensters. Dann schubst man die Ausführung des Codes mit F5 an.



Die Ausführung des Codes stoppt dann genau an der markierten Zeile Code, dafür sind Breakpoints ja da. Nach dem Anhalten kann ich mit F5 die Ausführung des Codes wieder anstoßen – der Code läuft dann solange, bis er nichts mehr zu tun hat oder eben der nächste Breakpoint im Weg steht.

Sehr häufig passiert es mir, dass ich einen Breakpoint zu früh setze – sicher ist sicher, man will ja den entscheidenden Moment im Code nicht verpassen. Ich lasse den Code dann anhalten und entscheide dann – vielleicht auch basierend auf Werten von Variablen – wo der nächste wichtige Zwischenstopp wäre. Dort wird dann der nächste Breakpoint gesetzt.

Das führt dazu, dass man irgendwann viele Breakpoints hat, die man eigentlich alle gar nicht benötigt. Man darf sie dann alle wieder löschen … wer will das schon?

Jetzt kommt der Kniff: Man kann Visual Studio anweisen, an einer bestimmten Stelle anzuhalten, ohne einen Breakpoint zu setzen – indem man Visual Studio mitteilt, einfach den Code bis zur aktuellen Position des Cursors laufen zu lassen. Der zugehörige Shortcut ist CTRL+F10. Das heißt auch, dass ich eine neue Debugging Session starten kann, indem ich den Cursor in eine Zeile Code stelle und dann einfach Ctrl+F10 nutze – die App wird kompiliert, ausgeführt, und in der Zeile des Cursors bleibt der Debugger stehen.

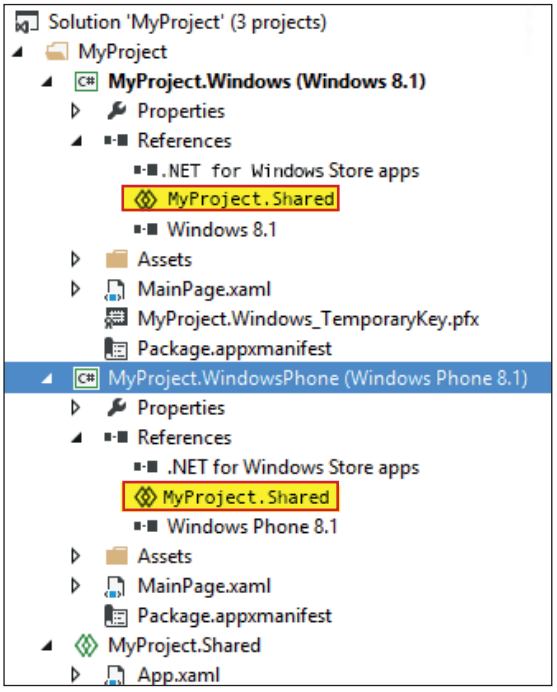
Das hält die Anzahl meiner Breakpoints übersichtlich und erlaubt es, spontan und „spurlos" zu debuggen, ohne am Ende wieder aufräumen zu müssen.

tl;dr

Mit Ctrl+F10 wird die Ausführung der Applikation im Debugger an der Stelle angehalten, an der gerade der Cursor steht.

39 Code Reuse mit Shared Projects – auch außerhalb von Apps

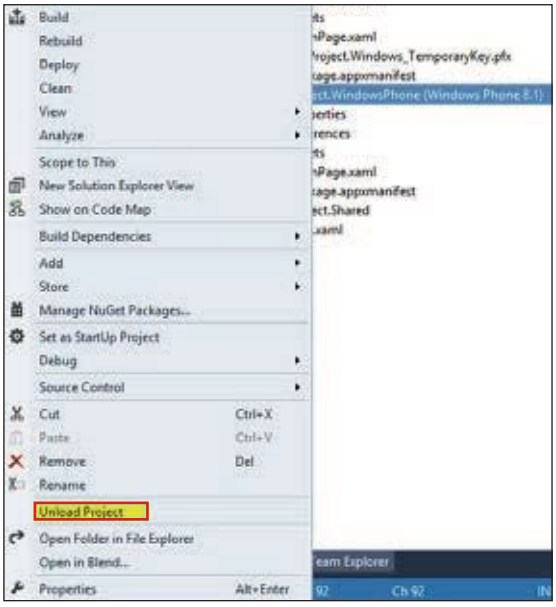
Heute ein etwas längerer Tipp. Aber er ist es allemal wert:



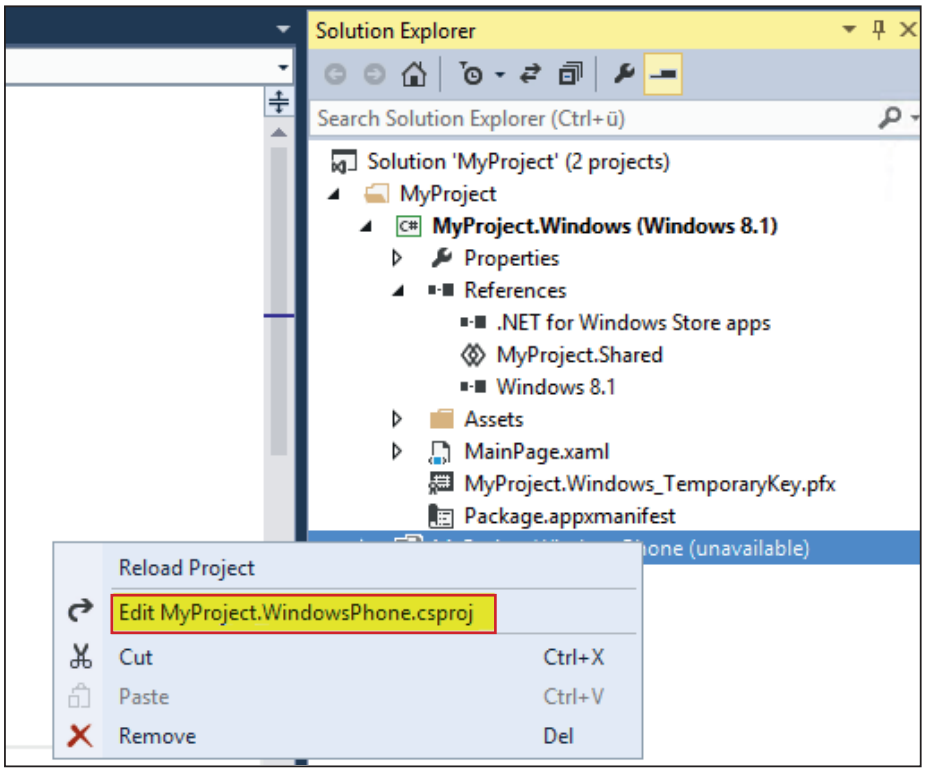
Mit Visual Studio 2013 Update 2 wurden ja die Universal Windows Apps eingeführt, die das „Shared Project" eingeführt haben. Mittels Shared Project ist es möglich, den 2 Quellcode auf einfache Weise zwischen mehreren Projekten zu teilen – alles, was im Shared Project liegt, ist in allen Projekten verfügbar, die das Shared Project referenzieren.

Standardmäßig gibt es diese Funktion nur für Universal Apps – alle anderen Projekte bleiben außen vor. Wäre es nicht schön, genau diese Möglichkeit auch für herkömmliche Desktopapplikationen zur Verfügung zu haben? Logo!

Wenn wir nun das Windows Phone-Projekt entladen, können wir uns den Aufbau der dahinterliegenden XML-Datei anschauen und werden sehen, dass es sich dabei technisch um ein einfaches Import-Statement handelt. Entladen:



Edit klicken, um das XML zu sehen:

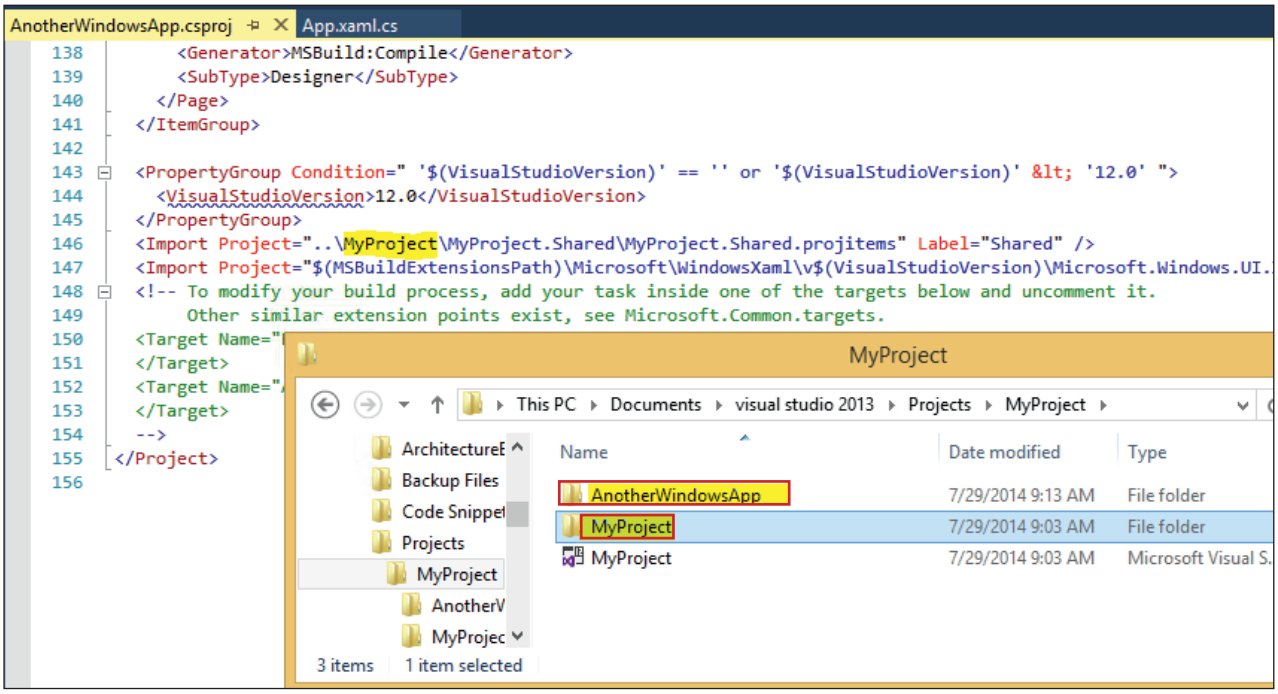


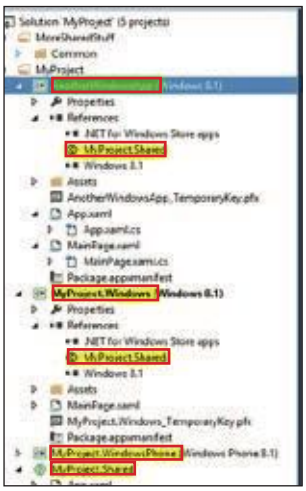
Hier ist das Import-Statement:

Hier steht dann zum Beispiel:

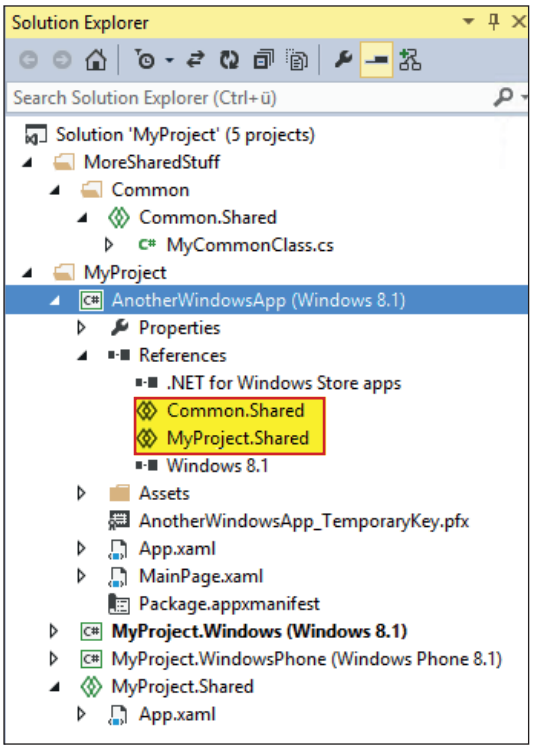
<Import Project=„..\MyProject.Shared\MyProject.Shared.projitems“ Label=„Shared“ />
Wenn wir nun ein weiteres Projekt – zum Beispiel eine weitere Windows 8 App – zur Solution hinzufügen, dann können wir diesen Schnipselcode auch dort im Projekt-XML einfügen und erhalten ebenso eine Referenz auf das Shared Project. Aber Achtung! Beim Hinzufügen eines neuen Projektes liegt dieses vermutlich eine Ebene höher im Dateisystem – außer Ihr habt das schon beim Anlegen berücksichtigt.

Den Pfad müsst Ihr also entsprechend anpassen. Der Screenshot unten sollte das verdeutlichen. Ich habe hier die Windows App „AnotherWindowsApp“ mit in die Solution aufgenommen und eine entsprechende Referenz eingefügt.



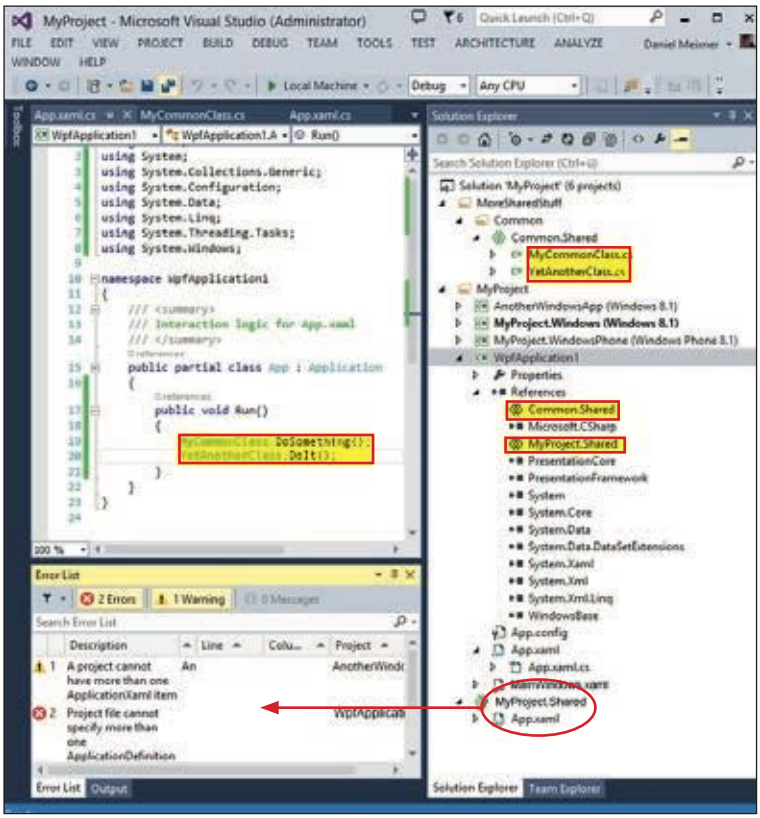


Das Ergebnis: Wir haben jetzt 3 Projekte, die sich ein „Shared Project“ teilen. Cool!

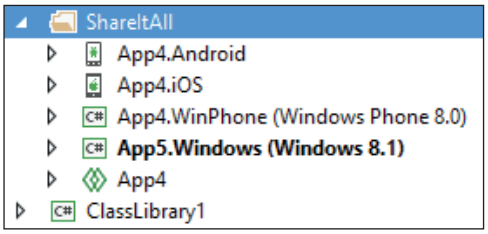


Man kann das jetzt natürlich auf die Spitze treiben … und könnte auf die Idee kommen, mehrere Shared Projects zu referenzieren – auch das geht. Logischerweise muss man ein bisschen darauf Rücksicht nehmen, was man wo liegen hat, und dass man nichts doppelt teilt.

Exemplarisch habe ich jetzt noch eine Desktop WPF-Anwendung hinzugefügt. Das geht technisch auch – man muss allerdings aufpassen, was geteilt wird, da das Desktop WPF XAML und das Windows Store App XAML nicht vollständig kompatibel sind. Hinzukommt, dass man natürlich keine 2 App.xaml in einem Projekt haben darf. Hier helfen die Fehlermeldungen weiter.



Richtig abgefahren wird es, wenn man sich XAMARIN installiert und dann auch noch die Möglichkeit hat, Code zu teilen zwischen Windows Apps, Windows Phone, Desktop, Android und iOS. Siehe Screenshot.



Schön ist, dass das ganze, sobald es einmal angelegt ist, auch Drag & Drop unterstützt – das heißt, es ist wirklich leicht, neue Dateien zum geteilten Projekt hinzuzufügen.

tl;dr

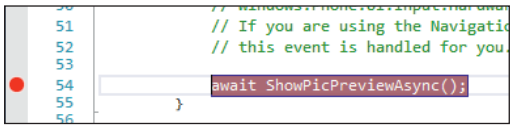
Das „Shared Project" funktioniert auch außerhalb der App-Welt, auch wenn es dafür kein leeres Template gibt.

40 Tracepoints! Tracepoints? Tracepoints!

Der 40. Tipp der Serie widmet sich einem der best kept secrets in Visual Studio. Nicht, dass es irgendjemand absichtlich versteckt hätte oder geheimhalten würde. Aber es ist immer wieder so: Tracepoints erfreuen sich nicht gerade großer Bekanntheit.

Ich denke, jeder von uns kennt Breakpoints. Diese roten Kugeln am linken Rand. Falls nicht, möchte ich hiermit zusätzlich Verwirrung stiften und auf den Blogpost hinweisen, der erklärt, wie man auch ohne Breakpoints breaken kann.

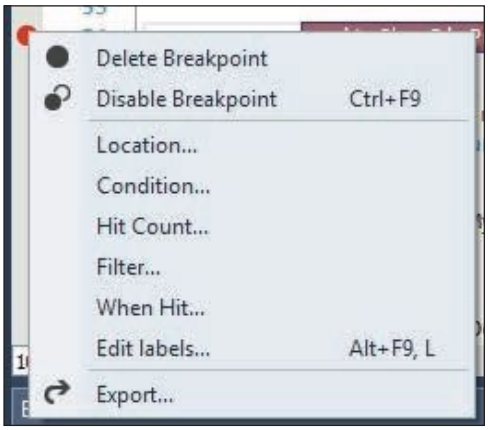
Aber zurück zum Thema: Breakpoints sind also bekannt und sehen ungefähr so aus:



Man setzt sie mit F9 oder mit Mausklick in die dunkelgraue Seitenleiste. Wenn wir den Code jetzt ausführen, wird die Ausführung hier anhalten, wir können Werte von lokalen Variablen checken etc., etc. Alles wie gewohnt.

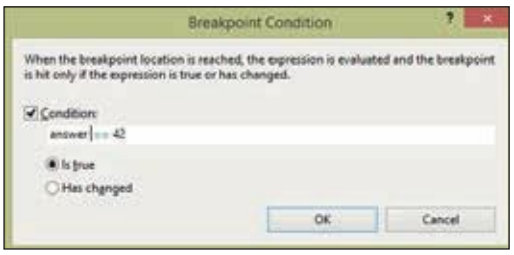
Nun geschieht ein kleines Wunder. Wir befinden uns an einem Punkt in der Benutzung von Visual Studio, wo ausnahmsweise die Mausnutzer einen kleinen Vorteil gegenüber den Tastatur-Shortcut-Junkies haben. Denn – mal ehrlich – wer würde schon auf die Idee kommen, diesen komischen roten Knubbel einfach mal mit der rechten Maustaste zu klicken?

Let's do it!

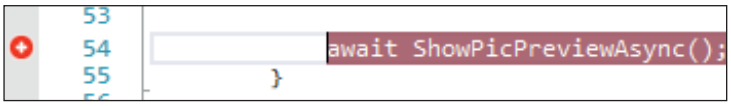


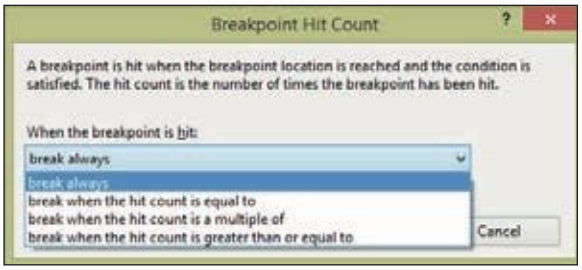
Was sehen unsere müden Augen? Ein Kontextmenü, das es wert ist, erforscht zu werden. Ich will in diesem Blogpost nur auf die für mich relevantesten Möglichkeiten eingehen.

Bedingungen
Wir können über „Condition.." eine Bedingung angeben, unter der hier tatsächlich angehalten wird. Das ist angenehm, wenn wir in längeren Debuggingsessions immer wieder auf den Breakpoint treffen, der eigentlich nur unter ganz bestimmten Bedingungen von Bedeutung ist. Wir können hier auf lokale Variablen zugreifen und einfach Werte vergleichen.



Der Breakpoint ändert dann seine Darstellung:



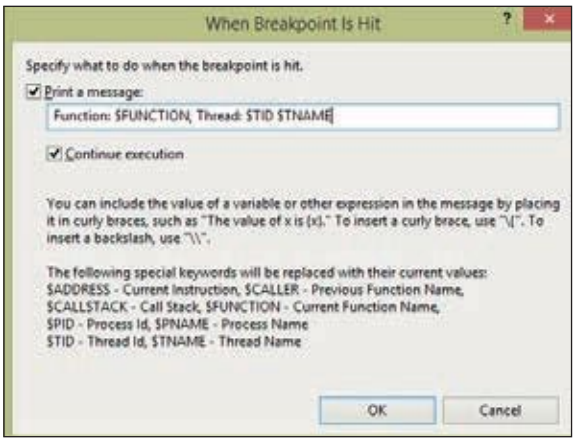


Zähler

Wir können über „Hit Count …" eine Zählfunktion einbauen und dafür sorgen, dass der Breakpoint eben nicht immer, sondern nur beim x-ten Mal anspringt. Schon mal eine Schleife über 100.000 Datensätze gedebugged, die komischerweise immer beim 999.998 Durchlauf abschmiert? Da wird das Debuggen zur Qual, wenn man immer einzeln weiterklicken muss. Hier ist die Lösung – einfach den Hit Count entsprechend nach oben kurbeln.

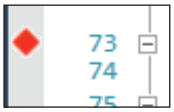
Filter

Die Filterfunktion erlaubt es uns, den Breakpoint nur zu zünden, wenn bestimmte Umgebungsparameter zutreffend sind. Das ist insbesondere bei Remotedebugging-Szenarios cool, wenn man z. B. nur auf bestimmten Maschinen debuggen möchte.



Tracepoints

Ja, und dann gibt's noch das Tracen: Wir können festlegen, was passieren soll, wenn der Tracepoint erreicht wird, indem wir eine „When Hit …"-Aktion angeben. Was soll passieren? Nun, wir könnten ja z. B. eine Debug-Message ausgeben, inklusive einem aktuellen Variablenwert. Schön ist dabei, dass wir die Ausführung weiterlaufen lassen können (wenn wir wollen).



Tracepoints erkennen wir optisch an der rautenförmigen Darstellung.

Insgesamt viele Möglichkeiten, ohne Eingriff in den Quellcode die Debuggingsessions deutlich zu vereinfachen.

tl;dr

Ein Rechtsklick auf einen Breakpoint eröffnet neue Horizonte!

41 Bing Power für Entwickler

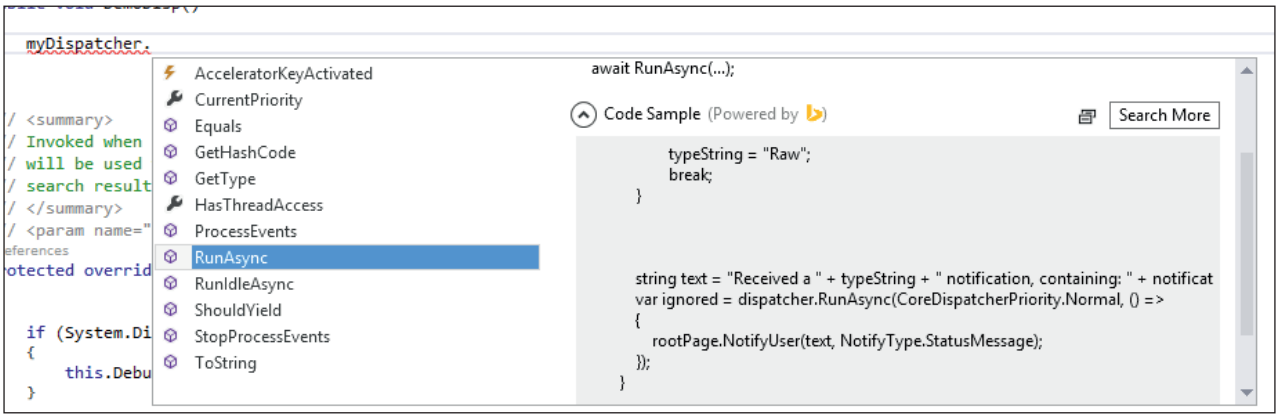
Wenn ich in die Runde fragen würde: „Wer kann mir sagen, wie man die Proximity Sensoren von Windows ansprechen kann?“ – Wer wüsste dann die Antwort und könnte das sofort runtercoden? Ich denke, die Frameworks und Plattformen, die wir heute anprogrammieren, können inzwischen einfach viel zu viel, als dass man alles auswendig wissen müsste oder könnte. Getreu dem Motto „Man muss nicht alles wissen, solange man weiß, wo es steht“ suchen wir Entwickler relativ häufig einfach mal nach einem Sample, das die Basisfunktionen bestimmter APIs zeigt. Alternativ könnte man natürlich auch ellenlange Dokus lesen (sofern vorhanden und aktuell), aber in der Regel ist man schneller, wenn man ein paar Snippets als Grundlage nutzen kann.

Was tun wir also, wenn wir vor einer neuen Aufgabe stehen? Die Suchmaschine unserer Wahl anwerfen oder in einschlägigen Foren oder MSDN-Websites wie dev.windows.com nach Codeschnipseln suchen. In der Regel noch deutlich bevor wir ein echtes Problem haben, einfach um einen gewissen Startschub zu bekommen.

Wäre es nicht schön, wenn uns jemand genau diese Arbeit abnimmt? Man stelle sich vor, man fängt an zu tippen und bekäme direkt im Studio die richtigen, relevanten Codeschnipsel … so ähnlich wie IntelliSense funktioniert …
Good News: Genau das liefert der Bing Developer Assistant, den man über die Visual Studio Gallery oder direkt hier beziehen kann.

Ich muss gestehen, dass ich diese Extension noch nicht intensiv genutzt (sie ist ja noch relativ neu), aber ein paar Anwendungsszenarien durchgespielt habe, die mich vollends überzeugen. Momentan ist die Extension noch eine Beta-Version, aber folgende Punkte liefert sie schon recht anständig ab:

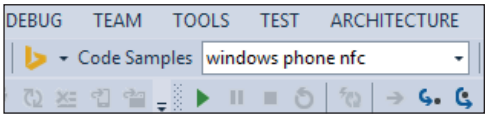
Inline Code Snippets
Das oben beschriebene Szenario wird erfüllt, indem die Code Snippets tatsächlich inline angezeigt werden. Sobald was getippt wurde, wird relevanter Code angezeigt – die Quelle für den Demo-Code ist dabei online. Das sieht dann z. B. folgendermaßen aus.



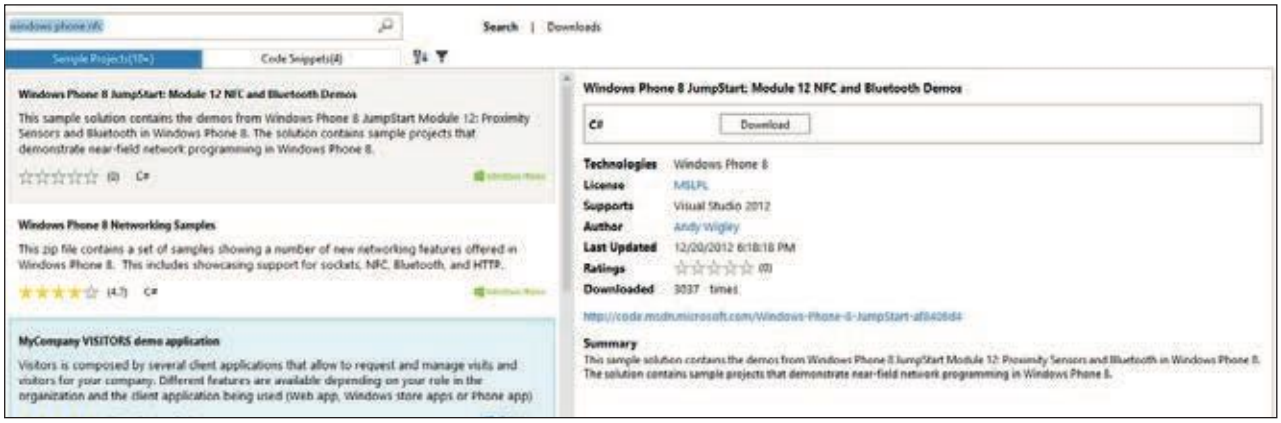
Wie Ihr seht, wurde im Prinzip einfach die IntelliSense Preview erweitert, und wir finden auf der rechten Seite einen Schnipselcode, den wir ganz einfach als Vorlage nutzen können und, wenn wir wollen, auch kopieren können.

Sample-Suche

Manchmal ist ein Schnipselcode aber auch zu wenig und man möchte ein ganzes, lauffähiges Beispiel direkt zum Download. Dann hilft die neue Toolbar, die ein Suchfeld mitbringt, über das man direkt nach Samples suchen kann:

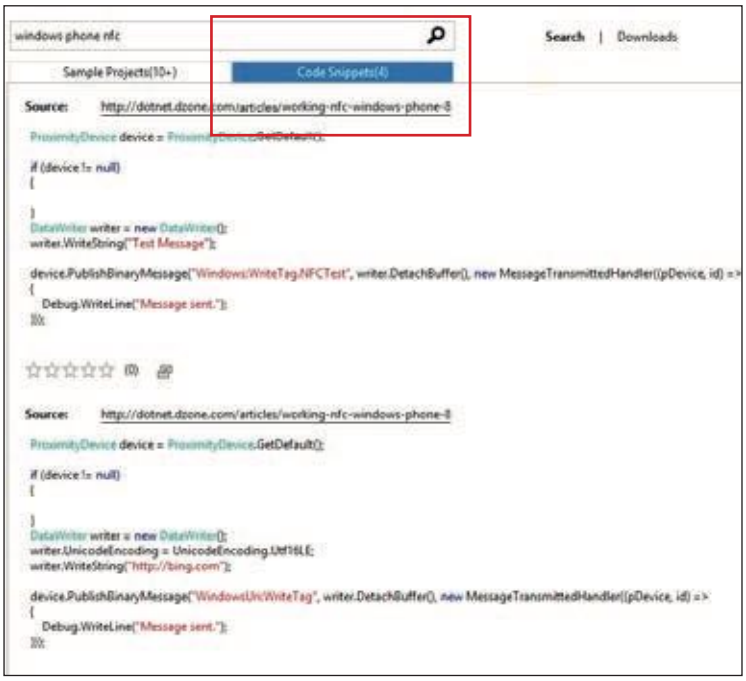


Das Ergebnis erscheint direkt in Visual Studio, und wir müssen uns nicht mit neuen Browserfenstern herumschlagen.



Die Samples können direkt heruntergeladen werden.

Schön, dass es hier nicht nur Samples gibt, sondern auch gleich noch mehr Codesnippets:



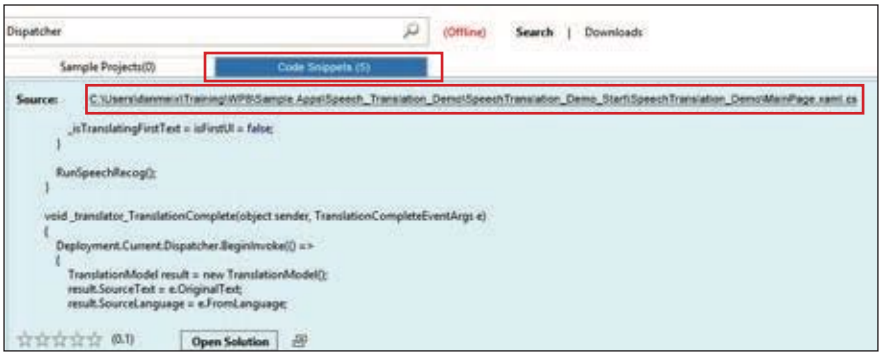
Offline Search

Das bisher genannte hat natürlich einen Haken: Es funktioniert nur, wenn man online ist. Wenn man im Flieger oder in der Bahn ist, hat man gute Chancen, genau im falschen Moment keine Verbindung zu haben. Aber ist es nicht so, dass man als Entwickler ohnehin das Gefühl hat, eigentlich alles irgendwann schon einmal programmiert zu haben? Man weiß nur gerade nicht, in welchem Projekt zu welcher Zeit und welche Solution … Aber irgendwo müsste man den entscheidenden Schnipsel Code doch noch auf der Festplatte haben!

Gerade für mich und meine Demos trifft genau das häufig zu. Klar, Ordnung ist das halbe Leben, aber manchmal probiert man eben Dinge aus, speichert sie dann nicht ordnungsgemäß in ein Projekt mit aussagekräftigem Namen und muss Monate später das gleiche Problem erneut lösen.

Hier hilft die Offline Search – eine Suche, die meinen von mir selbst geschriebenen Code auf meiner Festplatte durchsucht und mir meinen eigenen Code als Orientierung zur Verfügung stellt. Das ist doch der Knaller, oder? Wo gesucht werden soll, kann man natürlich in den Optionen konfigurieren.

Das Ergebnis der Offline-Suche sieht dann aus wie folgt. Über den Link kann ich die Datei auch gleich im Visual Studio öffnen.

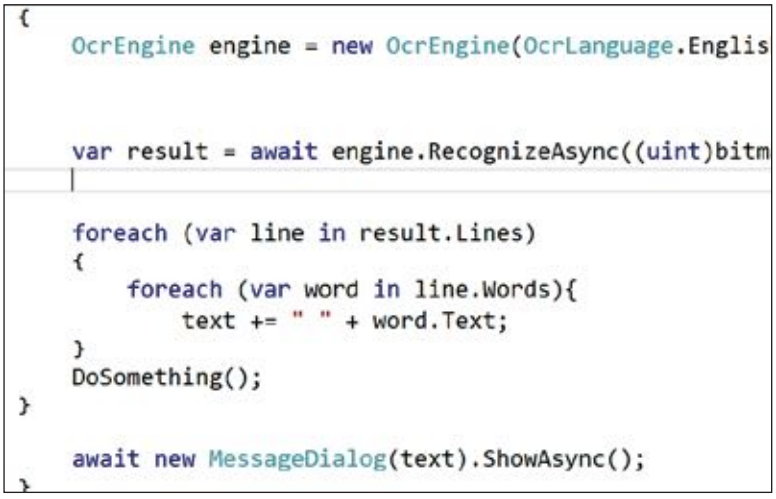


tl;dr

Mit dem Bing Developer Assistant findet man relevante Codebeispiele schneller.

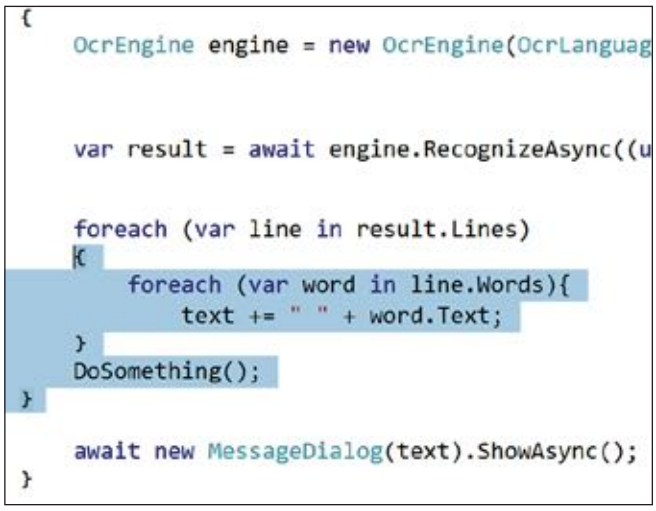
42 Klammer sucht Klammer

Tipp 42 muss natürlich besonders sinnvoll sein. Ist er auch, und dahinter verbirgt sich ein Shortcut, den ich selbst so oft und automatisch nutze, dass ich schon fast vergessen habe, dass es sich dabei um einen Tipp handelt – und deshalb fast vergessen hätte, ihn weiterzugeben.



Es handelt sich dabei um eine Hilfestellung, wenn wir als Entwickler auf der Suche nach dem Gegenstück für eine öffnende oder schließende Klammer suchen. Durch die Einrückungen im Code ist das zwar vereinfacht und häufig ohnehin zu erkennen, aber nicht immer ist der Code korrekt eingerückt. Dann sieht eine falsch gesetzte Klammer schnell so aus, als wäre sie auf einer anderen Verschachtelungstiefe, als sie eigentlich ist, was zu unangenehmen Bugs führen kann, wenn Code im falschen Klammerscope ausgeführt wird – wie im Bild zu sehen.

Der Shortcut Ctrl+` schafft Abhilfe. Damit wird automatisch zu einer Klammer das passende Gegenstück gefunden. Der Cursor muss einfach vor oder hinter einer Klammer stehen, dann springt der Cursor bei Verwendung des Shortcuts automatisch zur „Gegenklammer". Die Markierung der Gegenklammer geschieht automatisch, auch ohne Shortcut – allerdings ist ja nicht immer der komplette Inhalt einer Klammerung auf dem Bildschirm zu sehen, so dass bisweilen der Shortcut sehr hilfreich ist.



Besonders nützlich empfinde ich den Shortcut in Verbindung mit der Shift-Taste. Durch Ctrl+Shift+` wird nicht nur das Gegenstück zur Klammer gefunden, sondern automatisch der gesamte Bereich dazwischen markiert. So wird eine ungünstige Formatierung relativ schnell ersichtlich, und natürlich kann man wunderbar den Inhalt anschließend ausschneiden.

Kleiner Hinweis um Verwirrung zu vermeiden: ` ist das Zeichen links neben der Backspace-Taste.

tl;dr

Zum Auffinden von Gegenstücken zu öffnenden und schließenden Klammern hilft Ctrl+`. Mit zusätzlich gedrückter Shift-Taste wird der Inhalt auch noch markiert.

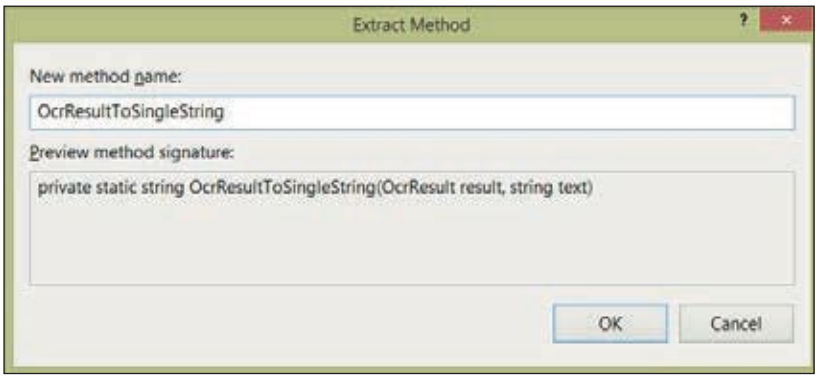
43 Methoden Refactoring (Ctrl+R, Ctrl+M)

Vielleicht seid Ihr ja auch Fans von kurzen, übersichtlichen Methoden. Am liebsten mag ich es eigentlich, wenn Methoden auf einen Blick zu verstehen sind und der Name der Methode eigentlich schon alles sagt. Sicher gibt es hier unterschiedliche Philosophien und nicht immer wird das so einfach funktionieren, so dass man in der Realität auch Methoden finden wird, die „mehr" tun, als erwartet. Gerade, wenn man mit bestehenden Codebasen arbeitet hat man nicht immer die Wahl, und ein Refactoring manuell durchzuführen wäre doch bisweilen etwas mutig.

Visual Studio kann hierbei unterstützen und das – ich nenne es mal „halbautomatische" – Refactoring erleichtern. Dazu suchen wir uns einfach die Codezeilen einer Methode, die wir gerne in eine andere, neue Methode auslagern möchten. Nehmen wir doch einfach mal unser OCR Sample, weil ich den Schnipselcode gerade zur Hand habe.

```
async private void Extract_OCR(object sender, RoutedEventArgs e)
{
    OcrEngine engine = new OcrEngine(OcrLanguage.English);
    var result = await engine.RecognizeAsync((uint)bitmap.PixelHeight, (uint)bitmap.PixelWidth, bitmap.PixelBuffer.ToArray());
    string text = String.Empty;

    foreach (var line in result.Lines)
    {
        foreach (var word in line.Words)
        {
            text += " " + word.Text;
        }
    }

    await new MessageDialog(text).ShowAsync();
}
```

Wir bekommen einen Dialog gezeigt, in dem wir den neuen Methodennamen festlegen können. Außerdem sehen wir eine Vorschau auf die Methodensignatur. Mit dem Klick auf „OK" wird die Methode angelegt, und der Code bleibt – so soll es sein – lauffähig.



Interessant ist dabei, dass der Automatismus relativ clever mit den lokalen Variablen umgeht. So werden aus diesen lokalen Variablen, die wir im Codeschnipsel verwendet haben, automatisch Methodenparameter erzeugt, und wir müssen uns hierum nicht selbst kümmern. Globale Variablen werden allerdings unverändert übernommen – was auch logisch ist. Natürlich wird der aufgerufene Code entsprechend angepasst.

Globale Variablen werden allerdings unverändert übernommen – was auch logisch ist. Natürlich wird der aufrufende Code entsprechend angepasst.

```
async private void Extract_OCR(object sender, RoutedEventArgs e)
{
    OcrEngine engine = new OcrEngine(OcrLanguage.English);
    var result = await engine.RecognizeAsync((uint)bitmap.PixelHeight, (uint)bitmap.PixelWidth, bitmap.PixelBuffer.ToArray());
    string text = String.Empty;

    text = OcrResultToSingleString(result, text);

    await new MessageDialog(text).ShowAsync();
}
```

tl;dr

Ctrl+R, Ctrl+M extrahiert Code aus einer Methode, erzeugt eine neue Methode und sorgt dafür, dass sich das Verhalten der Software nicht ändert!

44 Interfaces extrahieren (Ctrl +R, Ctrl +I)

Nach meinem letzten Blogpost zum Thema „Methoden Refactoring" möchte ich Euch gerne noch ein paar Shortcuts zum schnellen, komfortablen Refactoring nennen. Dieser Blogpost beschäftigt sich mit dem Thema, wie man am schnellsten aus einer Klasse ein Interface extrahiert bekommt.
Damit wir auch feststellen, welche Arbeit uns hier abgenommen wird, müssen wir uns noch einmal vor Augen halten, wie wir eigentlich vorgehen würden:

Manuelles Vorgehen

1. Wir stellen fest, dass wir eine Klasse (z. B. MyClass) implementiert haben, die ein Interface implementieren sollte – die entsprechenden Methoden hat die Klasse schon implementiert, es fehlt nur noch das Interface.
2. Wir legen den Code für das Interface an und schreiben ihn in eine neue Datei, die typischerweise den Namen IMyClass hat.
3. Wir packen die Methodensignaturen in den Rumpf des Interfaces.
4. Wir erweitern die Klasse um den Hinweis, dass sie von IMyClass ableitet.

Gerade der 3. Schritt könnte ein bisschen Geeier sein, abhängig davon, wie viele Methoden wir hier im Interface sehen wollen.

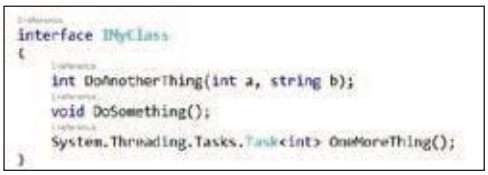
Automatisiertes Vorgehen

Ok, long story short, So geht's schneller: Wer aus einer existierenden Klasse ein Interface extrahieren möchte, dem werden obige Schritte abgenommen, wenn man auf dem Methodennamen Ctrl+R, Ctrl+I drückt.

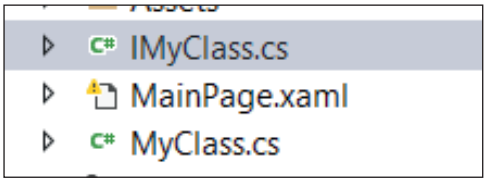
Ausgehend von einer Klasse wie dieser …

```
class MyClass
{
    public void DoSomething()
    {
    }

    public int DoAnotherThing(int a, string b)
    {
        return 42;
    }

    async public Task<int> OneMoreThing()
    {
        await Task.Delay(1000);
        return 42;
    }
}
```

… bringt uns Ctrl+R, Ctr+I diesen Dialog, in dem wir die Methoden sehen, die wir extrahieren können, und wir können wählen, was extrahiert werden soll.



Mit „OK" wird das Interface angelegt …



… wie sich's gehört in einer neuen Datei.

Die Klasse selbst wird angepasst, so dass hier das Interface implementiert wird.

Cool, oder?

tl;dr

Ctrl+R, Ctrl+I extrahiert Interfaces aus bestehenden Klassen.

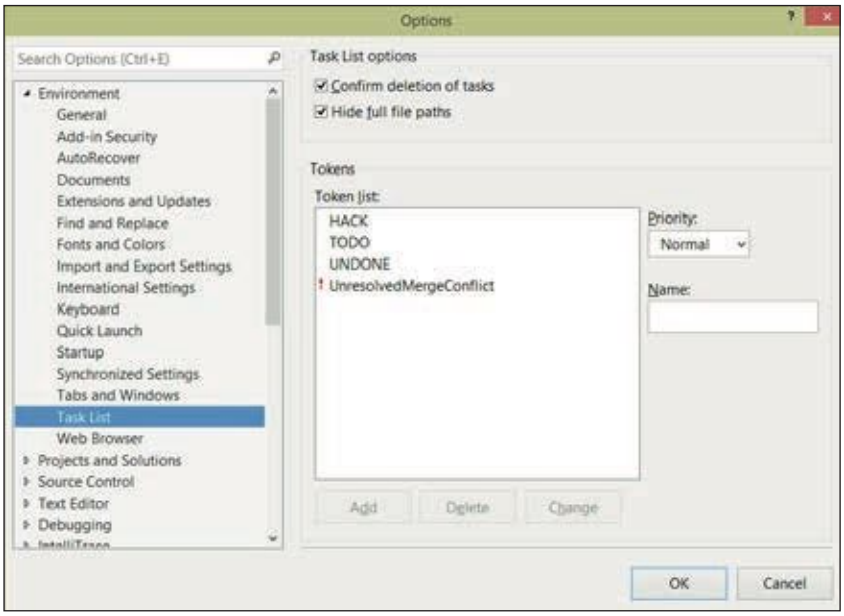
45 Die To-do-Liste für Code, der wirklich fertig ist

Wann ist Code eigentlich fertig? Sicher gibt es unterschiedliche Ansichten und Interpretationen, aber ich denke, die meisten werden sich darauf verständigen können, dass keine To-dos mehr offen sein sollten. Natürlich kann man testhalber einfach mal nach „To-do" im Code suchen. Ein bisschen schicker ist allerdings die Möglichkeit, die To-dos im Code über einen echten Task Viewer anzeigen zu lassen. Das erleichtert mit Sicherheit das Abarbeiten, ist aber für einen selbst vielleicht auch eine bessere Motivation, die Todos im Code tatsächlich loszuwerden.

Geöffnet wird der Task Viewer im Visual Studio über Ctrl+W, T. Es erscheint ein Tool-Fenster, das sauber alle Todos auflistet.

Um die Abarbeitung zu erleichtern, kann man einfach auf ein To-do klicken und wird direkt an die entsprechende Stelle Code navigiert. Sobald man das To-do dort entfernt hat (und natürlich hoffentlich vorher die entsprechende Arbeit erledigt hat), verschwindet der Eintrag in der To-do-Liste.

Man kann die Schlüsselwörter übrigens auch selbst definieren und die Priorität vergeben. Das geschieht im zugehörigen Options-Dialog:



Bei höherer Prio wird dann ein entsprechendes Ausrufezeichen mit angezeigt.

Sicher ersetzt diese Liste von To-dos keine ausgefeilte Arbeitspaketplanung. Für kleine Unteraufgaben ist es aber eine schöne Orientierung und letztlich hilft es auch dabei, wirklich nur fertigen Code am Ende als fertig zu bezeichnen. Wäre ja schließlich peinlich, wenn man Code abgibt, in dem noch jede Menge To-dos zu finden sind, oder?

tl;dr

Die Task View gibt mir Übersicht über offene Baustellen im bestehenden Code.

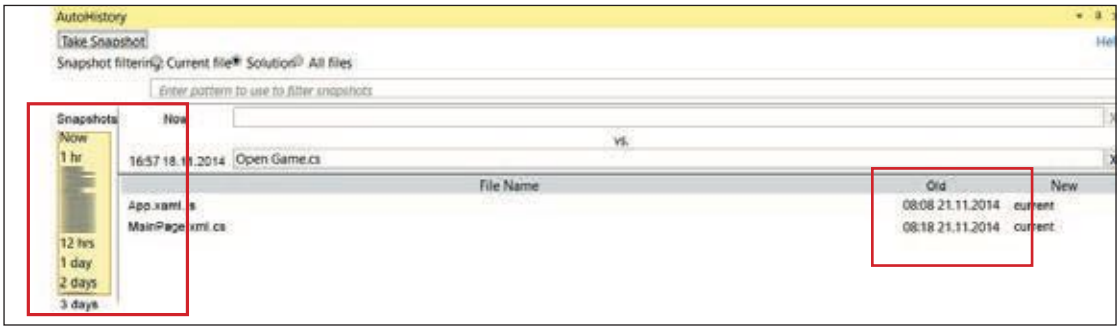
46 Der Zustand meiner Datei vor 15 Minuten mit AutoHistory

In Visual Studio speichert man als Anwender normalerweise immer (mit Ausnahme von Builds) explizit – das heißt, man drückt den Speicherbutton. Im Vergleich dazu arbeitet bspw. Word etwas anders – hier wird alle paar Minuten ein Zwischenstand der Datei zwischengespeichert (in einer anderen Datei). Sollten unvorhergesehene Dinge passieren, z. B. ein Absturz, kann ich dann die zwischengespeicherte Datei wieder öffnen.

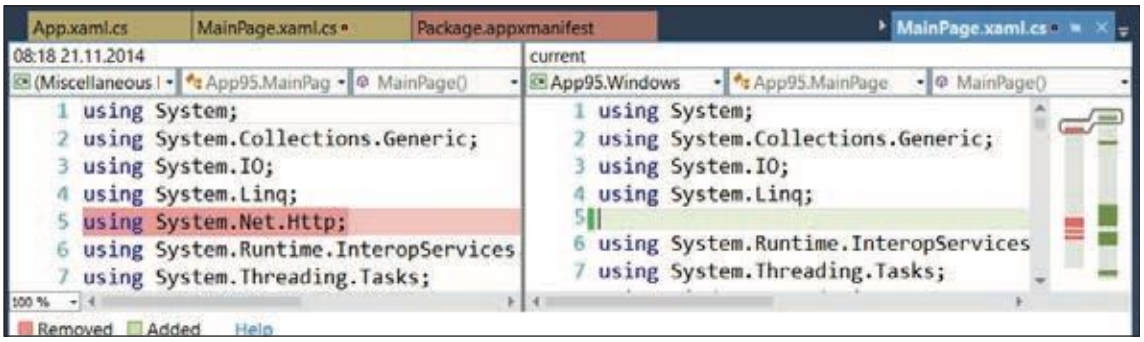
Für Visual Studio bietet sich ein solcher Mechanismus natürlich auch an. Und eigentlich interessiert uns Entwickler in der Regel nicht nur ein einzelner Zwischenstand, sondern potentiell die unterschiedlichen Versionen im Laufe der Zeit. Genau deshalb wurde irgendwann mal eine Versionsverwaltung erfunden. Das Einchecken in eine Versionsverwaltung ist aber auch immer ein expliziter Vorgang. Wäre es nicht schön, wenn ich – einfach so, ohne Zutun – auf die Version von vor 15 Minuten zugreifen könnte, die ich nie der Versionsverwaltung gezeigt habe? Weil ich vielleicht die letzten 40 Minuten codiert habe, ohne einzuchecken, und in den letzten 15 Minuten wieder alles kaputt gemacht hab?

Mit der AutoHistory Extension für Visual Studio wird genau das ermöglicht – unterschiedliche Stände Eurer Dateien werden gesichert, ohne Euer Zutun. So soll's sein. (Das Speichern auf Disk erfolgt aber immer erst nach dem Schließen von VS!) AutoHistory ist über die Visual Studio Gallery verfügbar und zwar hier: http://aka.ms/vsautohistory

Solange Euch die alten Stände nicht interessieren, ist Euch AutoHistory nicht im Weg. Sobald Ihr das AutoHistory Fenster öffnet, seht Ihr die von Euch geänderten Dateien mit Uhrzeit und Schiebebalken für unterschiedliche Versionen:



Den Abgleich mit einer alten Version gegen die aktuelle Version könnt Ihr mit Doppelklick durchführen, dann öffnet – wie gewohnt – das DiffTool:



Natürlich ersetzt das keine Versionsverwaltung! Aber falls man längere Zeit mal nicht eingecheckt hat und auch kein Shelving durchgeführt hat, ist das sicherlich ein schöner Rettungsanker.

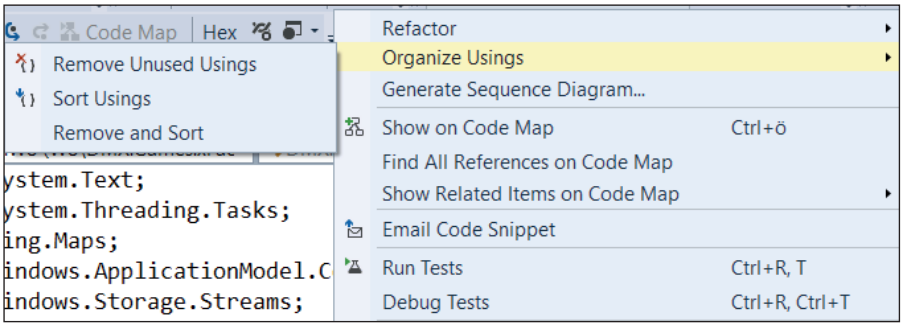
47 Usings aufräumen

Kennt Ihr solche Codeschnipsel?

```
using System.Text;
using System.Threading.Tasks;
using Bing.Maps;
using Windows.ApplicationModel.Core;
using Windows.Storage.Streams;
using Windows.UI.Core;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Media.Imaging;
using System;
using Bing.Maps;
using System.Text;
using System.ComponentModel;
using System.Diagnostics;
```

Ich denke, es ist klar, wie so etwas entsteht: Man kopiert aus anderen Dateien ein paar Zeilen Code und um sicherzugehen, dass alles funktioniert, kopiert man die Using-Statements gleich mit. Schwubdiwub, schon hat man Duplikate … Eigentlich ist das nicht weiter wild. Aber ungeordnete Usings und noch dazu doppelte und ungenutzte Usings sind ein bisschen sinnlos und sorgen dafür, dass man mehr denken muss als nötig, während man entwickelt.

Wieso sollte man sich das Leben selbst schwer machen? Zum Glück kann uns Visual Studio dabei helfen, hier aufzuräumen:

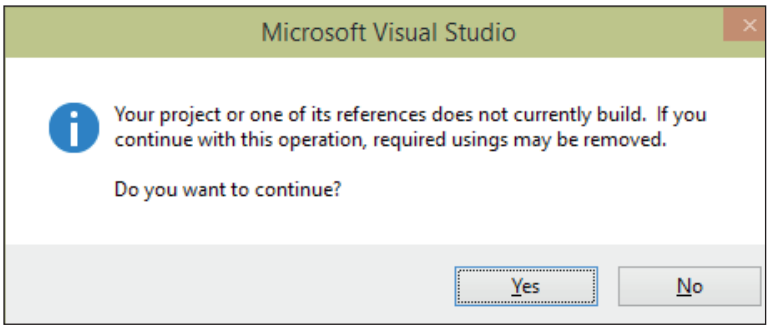


Ein Rechtsklick auf den Code öffnet das Kontextmenü, dort findet sich „Organize Usings". Hier kann ich wahlweise ungenutzte Usings entfernen oder eben die Using-Statements sortieren lassen – oder beides.

So schön sauber kann das danach aussehen:

```
using Bing.Maps;
using System;
using System.Threading.Tasks;
using Windows.ApplicationModel.Core;
using Windows.Storage.Streams;
using Windows.UI.Core;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Media.Imaging;
```

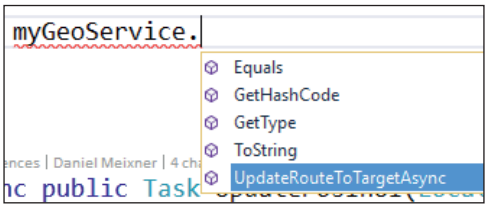
Sollte das Projekt nicht baubar sein, bekomme ich eine Warnung – ich denke diese Art von Refactoring macht man besser, wenn der Code baubar ist:



Per Default ist für diese Aktion kein Shortcut hinterlegt. Wenn Ihr das tun wollt, schaut Euch folgenden Blogpost von mir an: Visual Studio Tipps & Tricks, Teil 8–Eigene Shortcuts.

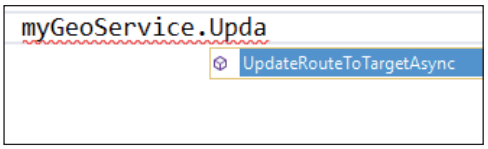
48 IntelliSense Modus umschalten (Ctrl+Alt+Space)

Dass IntelliSense super ist, habe ich ja schon in Tipp 26 erzählt … Manchmal findet man sich aber vielleicht in einer Situation wieder, die einen etwas frustriert – und IntelliSense scheint Schuld zu tragen.

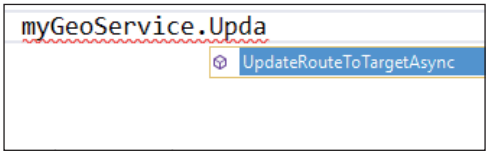


Man nehme an, wir haben ein Objekt und möchten eine Property des Objektes setzen. Die Property selbst haben wir aber noch gar nicht angelegt. Mittels der Visual Studio-Bordmittel, wie z. B. Ctrl+. aus Tipp 27 ist so ein Vorgehen sehr bequem. Man programmiert vor sich hin und alles, was noch fehlt, wird im Nachhinein generiert.

Der Haken: Wenn man ein Property setzen möchte, das es nicht gibt, springt Intellisense trotzdem an.



Wenn man nun eine Property anlegen will, die einen ähnlichen Namen trägt wie eine existierende Property oder Methode (im Beispiel „Update"), dann erkennt IntelliSense hier einen Zusammenhang, den es eigentlich gar nicht geben soll.



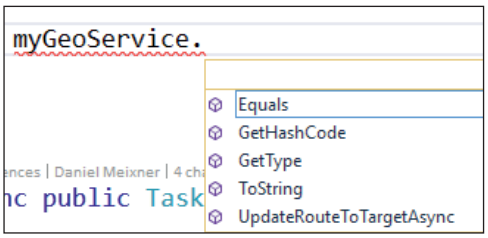
Drückt man nun nach dem Tippen von „myGeoService.Update“ auf „=“, um den Wert zuzuweisen, springt die automatische Vervollständigung an. Dieser Modus (Completion Mode) ist eigentlich super, weil er es uns erlaubt, nur wenige Buchstaben einer Property zu tippen. Das spart normalerweise Arbeit. In diesem Fall, wird aber die neue Property falsch vervollständigt und zu “UpdateRouteToTargetAsync” – das ist die einzige Property, die IntelliSense für dieses Objekt kennt, denn die andere gibt’s ja noch nicht. Autsch!

Das ist leider nicht, was wir wollten. Wer das schon erlebt hat, weiß, wie sehr man genervt ist, wenn man wieder alle Zeichen löschen muss (und im schlimmsten Fall danach nochmal in das gleiche Problem läuft)!

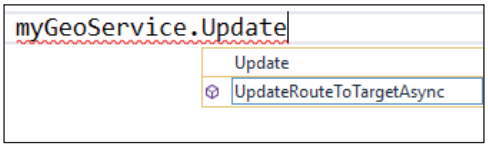
Abhilfe: IntelliSense Suggestion Mode

Zum Glück gibt es den „Suggestion Mode“. Dieser lässt sich über Ctrl+Alt+Space aktivieren und sorgt dafür, dass man frei drauflostippen kann, ohne, dass IntelliSense korrigierend eingreift. Trotzdem kann man über die Cursortasten die Vorschläge von IntelliSense aktivieren, man muss aber in diesem Fall selbst aktiv werden.

Das sieht dann folgendermaßen aus – man hat im IntelliSense Vorschaufenster oben einen kleinen Platzhalter.



Hier kann man ohne Probleme neue Properties eintippen.



Die Vervollständigung greift nicht ein, und man erhält das gewünschte Ergebnis.

So soll es sein. Das Umschalten funktioniert wie beschrieben via Shortcut – alles andere wäre auch etwas sinnlos, da man sich meist vermutlich im Completion-Modus befinden will. Ctrl+Alt+Space ist also eine Tastaturkombination, die es sich zu merken lohnt!

Wer den Bing Developer Assistant installiert hat, der wird feststellen, dass es hier keine optische Änderung gibt – der Suggestion Mode funktioniert zwar, der Platzhalter im Vorschaufenster fehlt aber.

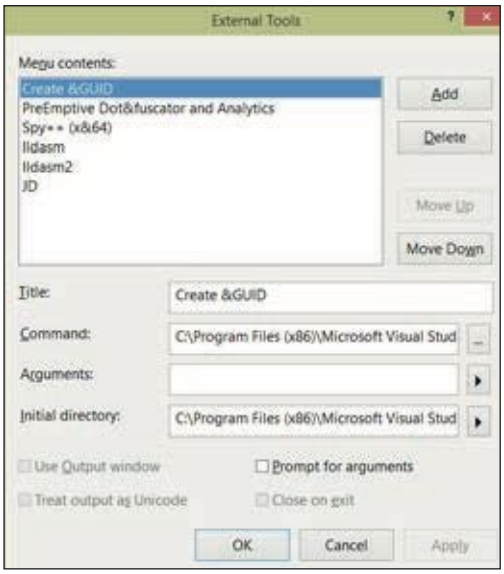
49 Externe Tools einbinden

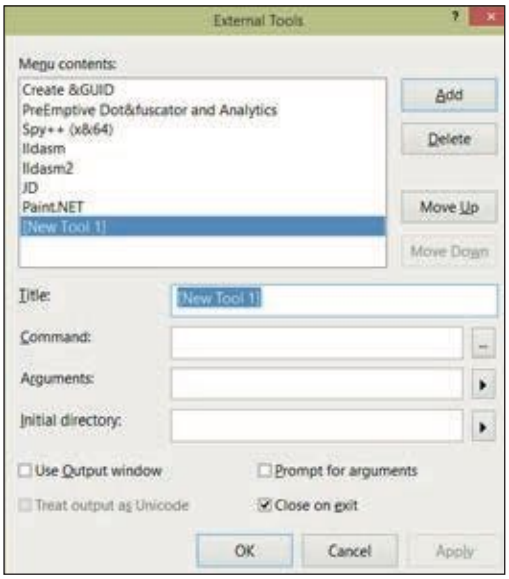
Als Entwicklergemeinde ist natürlich unser Tool der Wahl Visual Studio. Gott sei Dank gibt es aber eine lebendige Community rund um Visual Studio und ein gesundes Ecosystem, wodurch uns eine Vielzahl von weiteren Tools und Programmen zur Verfügung stehen, die wir als Entwickler gerne mal verwenden. Beispiel gefällig?

„Just Decompile" von Telerik, das es uns ermöglicht, .Net Code zu decompilieren! Natürlich kann man einen Schritt weiter gehen und sagen: Ist nicht auch ein Grafikprogramm (wie Photoshop, Paint.Net oder Metro Studio) für Entwickler bisweilen ein wichtiges Werkzeug, um bestimmte Aufgaben wie das Erstellen von Icons zu erledigen?

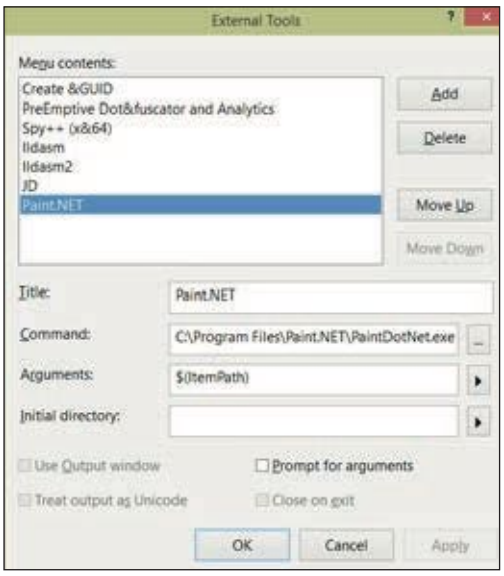
Wäre es nicht wunderbar, wenn wir diese Tools direkt aus Visual Studio aufrufen könnten – ohne Umweg über die Taskleiste, mit direktem Zugriff über eine selbstkonfigurierte Toolbar? Wie das geht, zeigt dieser Tipp.

Über Quick Launch können wir mit der Suche nach „External Tools" das Menü für Externe Tools aufrufen:

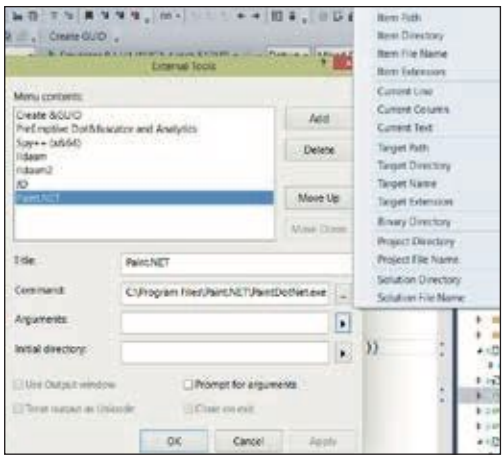




Hier können wir zusätzliche Einträge erstellen, die dann wiederum im Tools-Menü erscheinen werden. Wir klicken auf „Add".



Hier können wir für unser externes Tool einen Namen vergeben (Paint.NET) und können angeben, wo das Tool liegt:

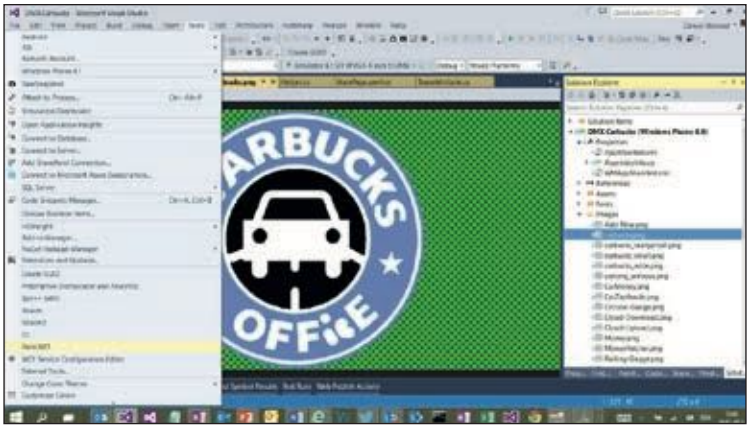


Außerdem haben wir die Möglichkeit, zusätzliche Argumente zu übergeben.

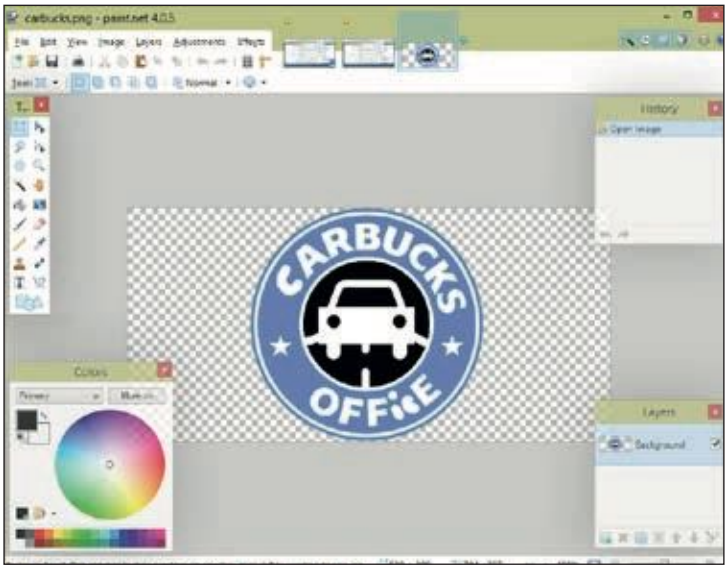
Wir übergeben hiermit den Item Path. Das bedeutet, das von uns geöffnete Item wird automatisch an das externe Tool übergeben.

Wenn ich jetzt in Visual Studio eine Datei öffne und für das Bearbeiten mein externes Tool öffnen möchte, dann genügt ein Klick auf den Eintrag im Tools-Menü, um das externe Tool mit der richtigen Datei zu öffnen.

Eintrag im Tools-Menü finden ...



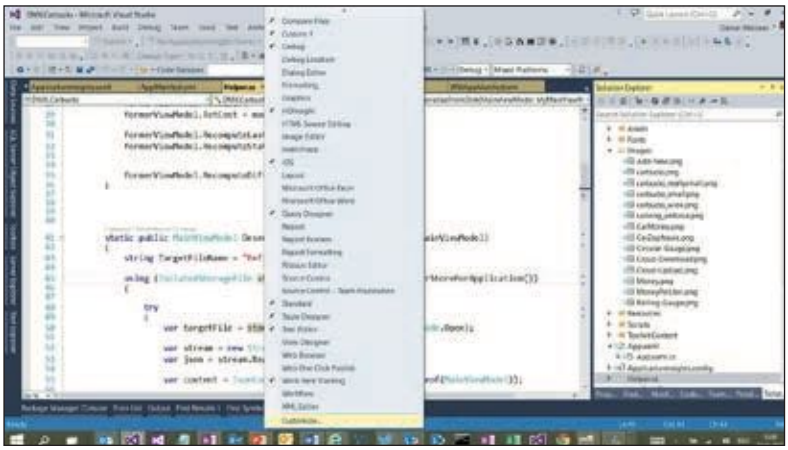
Eintrag im Tools-Menü finden ...

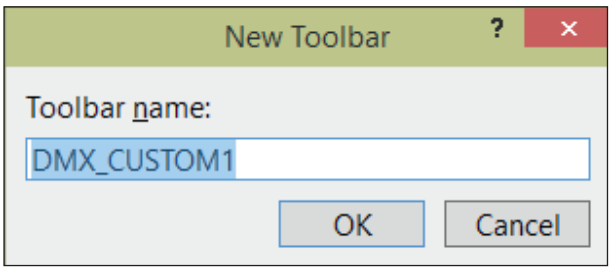


Eigene Toolbar

Wie bekomme ich meinen Button jetzt in eine eigene Toolbar?

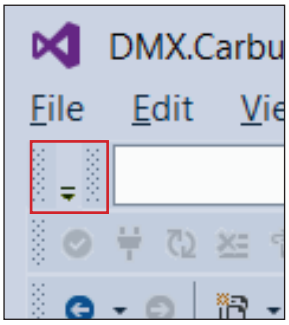
Erster Schritt: Toolbar anlegen. Dazu klicken wir mit der rechten Maustaste auf die Toolbars und wählen (ganz unten) „Customize“:



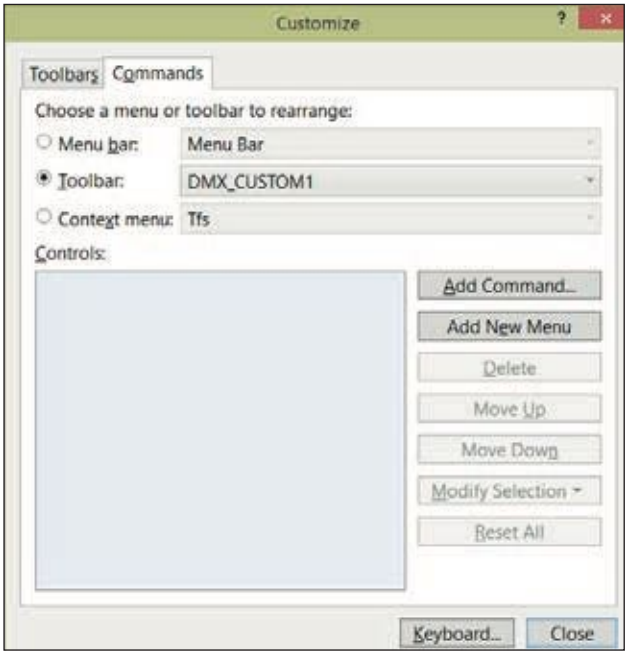


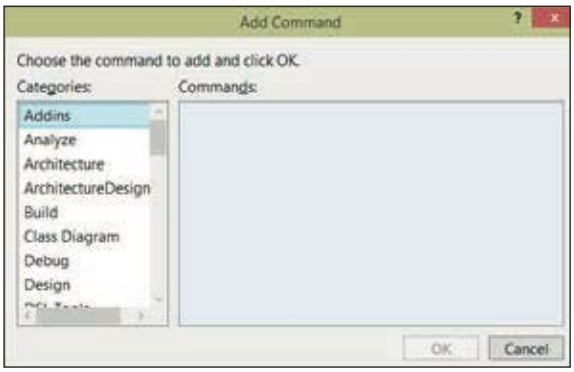
Über „New“ erstellen wir eine neue Toolbar.

Danach schließen wir alle Dialoge und blenden die (noch leere) Toolbar ein. Sie befindet sich standardmäßig ganz oben links, falls Ihr sie nicht findet.



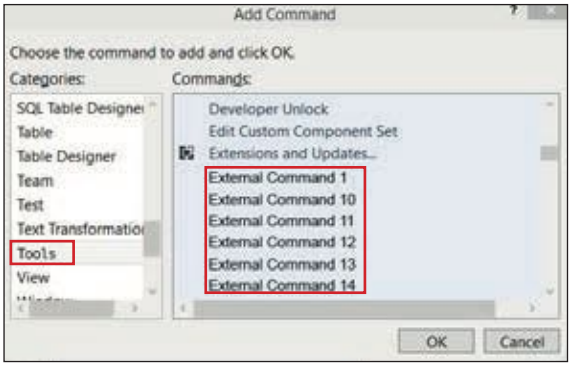
Wir klicken auf die DropDown-Box und wählen „Add or Remove Button“—> „Customize“. Hier sind wir wieder im Dialog von eben, allerdings auf dem anderen Tab gelandet.



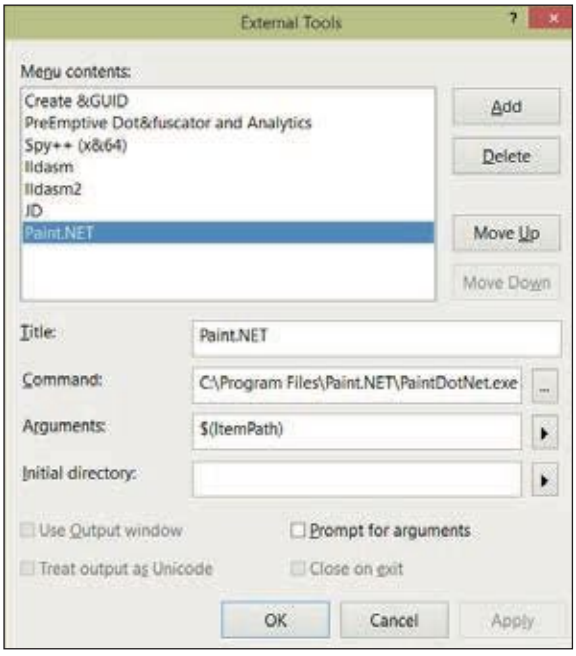


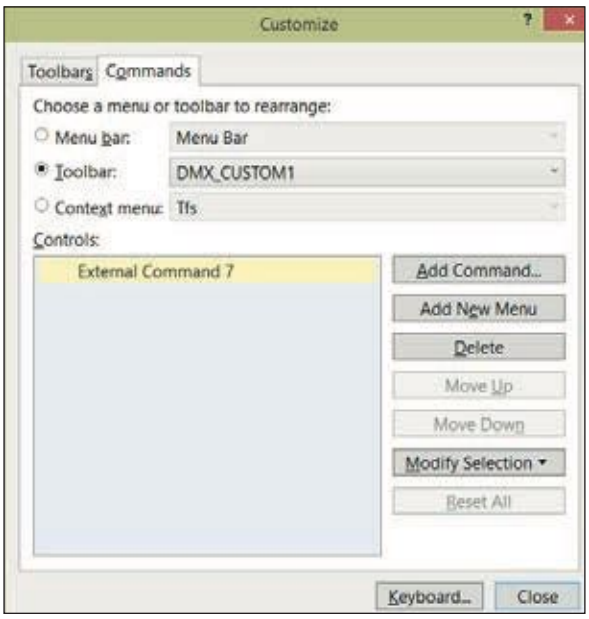
Über „Add Command" können wir jetzt in der Toolbar unsere eigenen Commands hinzufügen.

Aber wo finden wir diese? Natürlich unter „Tools"!



Jetzt wird's etwas kompliziert. Unsere eigenen externen Commands werden auf „External Command 1" etc. gemapped. Abhängig davon, an welcher Position sich unser External Tool im External Tools Dialog befindet! Das ist etwas gewöhnungsbedürftig, da wir hier keinen selbst angelegten Namen sehen können. Wir müssen also zählen.

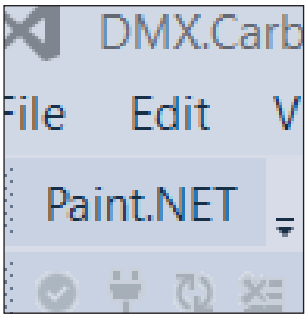




Paint.NET befindet sich an Position 7 (Wir zählen hier ab 1, nicht ab 0, wie man als Entwickler meinen könnte!).

Wir wählen also das External Command 7 – bei Euch kann das natürlich eine andere Position haben.

Und hier ist unser Menü-Eintrag.



Viel Spaß mit Euren eigenen Menüs!

50 Vertikale Hilfslinien

Es ist möglich, in Visual Studio vertikale Hilfslinien anzeigen zu lassen – das kann sinnvoll sein, um den Code sauber zu strukturieren, wenn die automatische Formatierung nicht das tut, was man will, oder aber – was ich für sinnvoller halte – um sich selbst eine Orientierung zu geben, wenn man zu viel in einer einzelnen Zeile unterbringen will.

Gerade bei Methodennamen bin ich persönlich ein großer Fan von „sprechenden" Namen, die gerne auch etwas ausführlicher sein dürfen und im Idealfall schon beschreiben, was die Methode tut, ohne dass das Lesen eines weiteren Kommentars notwendig ist. Das kann dann schon etwas länger werden. Wenn dann noch ein paar Parameter hinzukommen, die natürlich auch aussagekräftig benannt sind, ist ruckzuck die Zeile voll. Wobei sie natürlich nie richtig voll ist, aber sie passt eben nicht mehr auf den Bildschirm. Hier mag es sinnvoll sein, mittels Hilfslinien daran erinnert zu werden, dass der ein oder andere Zeilenumbruch sinnvoll sein könnte.

Wie aktiviert man diese Hilfslinien? Über die Registry – dieser Tipp hat also seinen Namen verdient, denn darauf kommt tatsächlich kein Mensch.

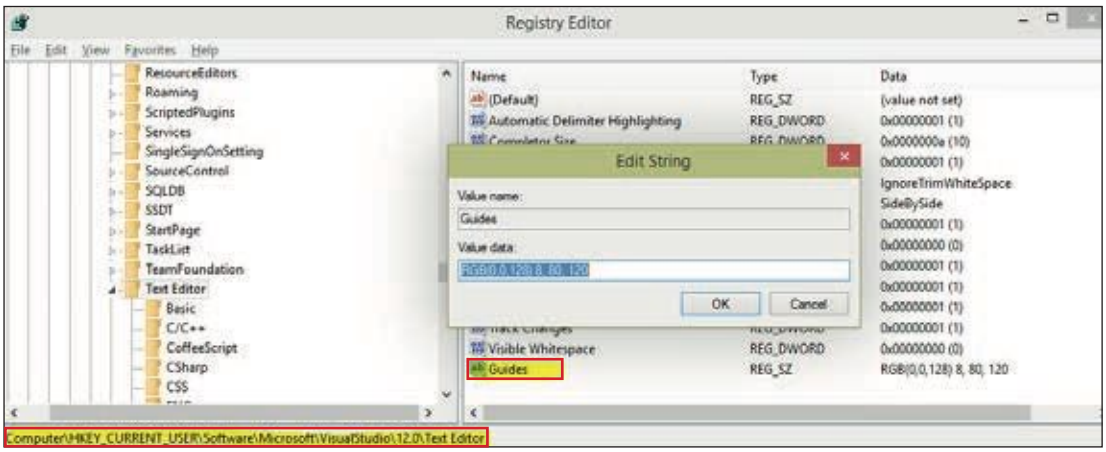
Unter Computer\HKEY_CURRENT_USER\Software\Microsoft\VisualStudio\12.0\Text Editor legt man einfach einen neuen String mit der Bezeichnung „Guides" an und gibt ihm einen Wert, der wie folgt aufgebaut ist:

Er besteht aus einem RGB-Wert, über den Ihr die Farbe der Hilfslinie bestimmt.

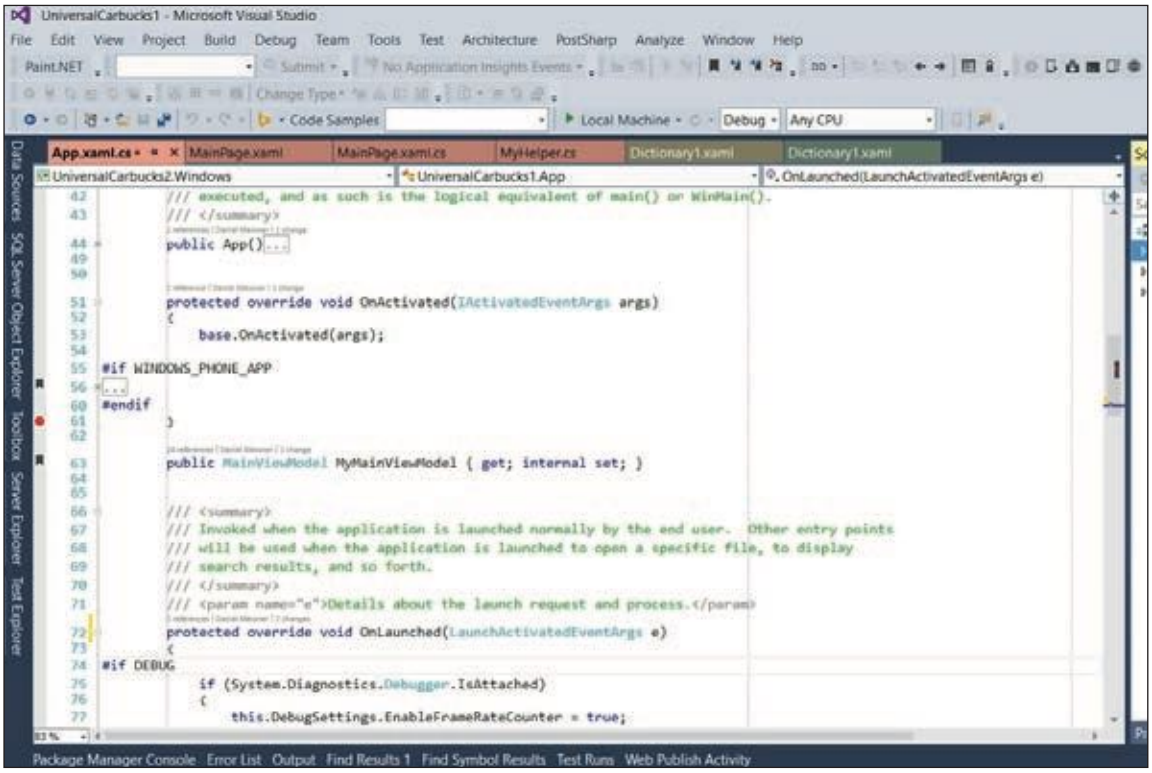
RGB(0,0,128)

Danach gibt man im gleichen Key an, an welcher Stelle die Hilfslinie zu sehen sein soll. Im Editor wird dann an der entsprechenden Zeichenposition (gezählt wird logischerweise von links) eine gepunktete Linie angezeigt:

RGB(0,0,128) 8, 80, 120



In meinem Fall sollen also 3 Hilfslinien in Blau gezeichnet werden – damit erhalte ich Orientierung, wenn meine Zeilen zu lang werden. Das sieht in VS dann so aus:



Mir ist nicht bekannt, dass dieses Feature auch über die Oberfläche in Visual Studio aktiviert werden kann. Demzufolge ist der Weg, um die Hilfslinien zu deaktivieren, auch der, dass der Key in der Registry wieder gelöscht wird. Bitte beachtet, dass das Konfigurieren der Registry potentiell Nebeneffekte haben kann, und geht hier vorsichtig vor. Aber als Entwicklern muss ich Euch das hoffentlich nicht erzählen .

Microsoft TechWiese

News und Ressourcen für Entwickler
www.techwiese.de

Microsoft

Visual Studio