Der lange Weg

Die Aufgabe war einfach: Ein Bild aus der Zwischenablage von Windows soll auf eine bestimmte Größe skaliert werden.

Anschließend soll die Anwendung das skalierte Bild in einem bestimmten Ordner ablegen. Ha, piece of cake, straight forward, drei Zeilen Code und fertig. Mittels *Clipboard.GetImage()* holt man die Daten als *BitmapSource* aus der Zwischenablage, übergibt sie der Bibliothek ImageFactory, die das Bild dann skaliert und anschließend speichert.

Fragt sich nur noch, was zum Henker eine *BitmapSource* ist und wie man daraus einen *MemoryStream* macht, den ImageFactory als Eingabeformat erwartet.

Glücklicherweise gibt es das Internet, das für solche Fragen tonnenweise Hilfen bereithält. Also ein paar Routinen kopiert, und schon kompilierte das Programm. Der Börner, der Champ. Give me five.

Allein – das Bild, das auf der Festplatte ankam, bestand vollflächig und ausschließlich aus der wunderschönen Farbe Weiß. Mensch Börner, nicht mal die einfachsten Dinge kriegst du hin.

Und schon fing das wilde Rumprobieren an, unterbrochen nur durch das selektive Lesen unbedingt notwendiger Dokumentation. Ich will fertig werden.

Dabei stellte sich heraus, dass es sich beim .NET Framework in puncto Verarbeitung von Bitmaps mal wieder um ein typisches Microsoft'sches Meisterwerk handelt. Schon in der Vergangenheit hatte Microsoft die Frameworks unglaublich flexibel gebaut. Man konnte alles damit anstellen – wirklich alles. Nur der Fall, den 99 Prozent der Entwickler benötigen, erfordert 50 Zeilen Code und wird erst auf massive Nachfragen in einem Blogpost dokumentiert. Man denke nur, wie lange es gedauert hat, bis es so etwas wie *File.ReadAllText* gab.

Das .NET Framework enthält auf jeden Fall ein wildes Rudel an *Bitmap*, *BitmapSource*, *Image*, *ImageSource* und vieles mehr. Aber damit nicht genug. In einem Blogpost [1] erklärt Rick Strahl, Entwickler von MarkdownMonster, dass die harmlos aussehende Methode *Clipboard.GetImage()* fehlerhaft arbeite, weshalb man sie meiden solle. Der Börner, vom Looser zum Microsoft-Opfer.

Mithilfe des Blogposts ist es schließlich gelungen, die Anforderungen zu erfüllen. Aber von wegen schnell erledigt: Aus drei Zeilen Code sind 200 Zeilen geworden, aus einer halben Stunde vier. Aber egal: Das Ding funktioniert. Der Börner, geknickt und doch glücklich. Softwareentwicklung kann das.

Viel Spaß mit der dotnetpro

Tilman Börner
Chefredakteur dotnetpro

[1] www.dotnetpro.de/SL2101Edi1

Fabian Zankl

saugt über eine sichere Ende-zu-Ende-Verbindung Daten in den Azure IoT Hub (S. 8)

Bernhard Sageder

setzt einen Monolithen per SOLID in eine flexible Lösung um (S. 30)

Michael Sperber

erweitert die Sprache F# um domänenspezifische Konstrukte (S. 56)