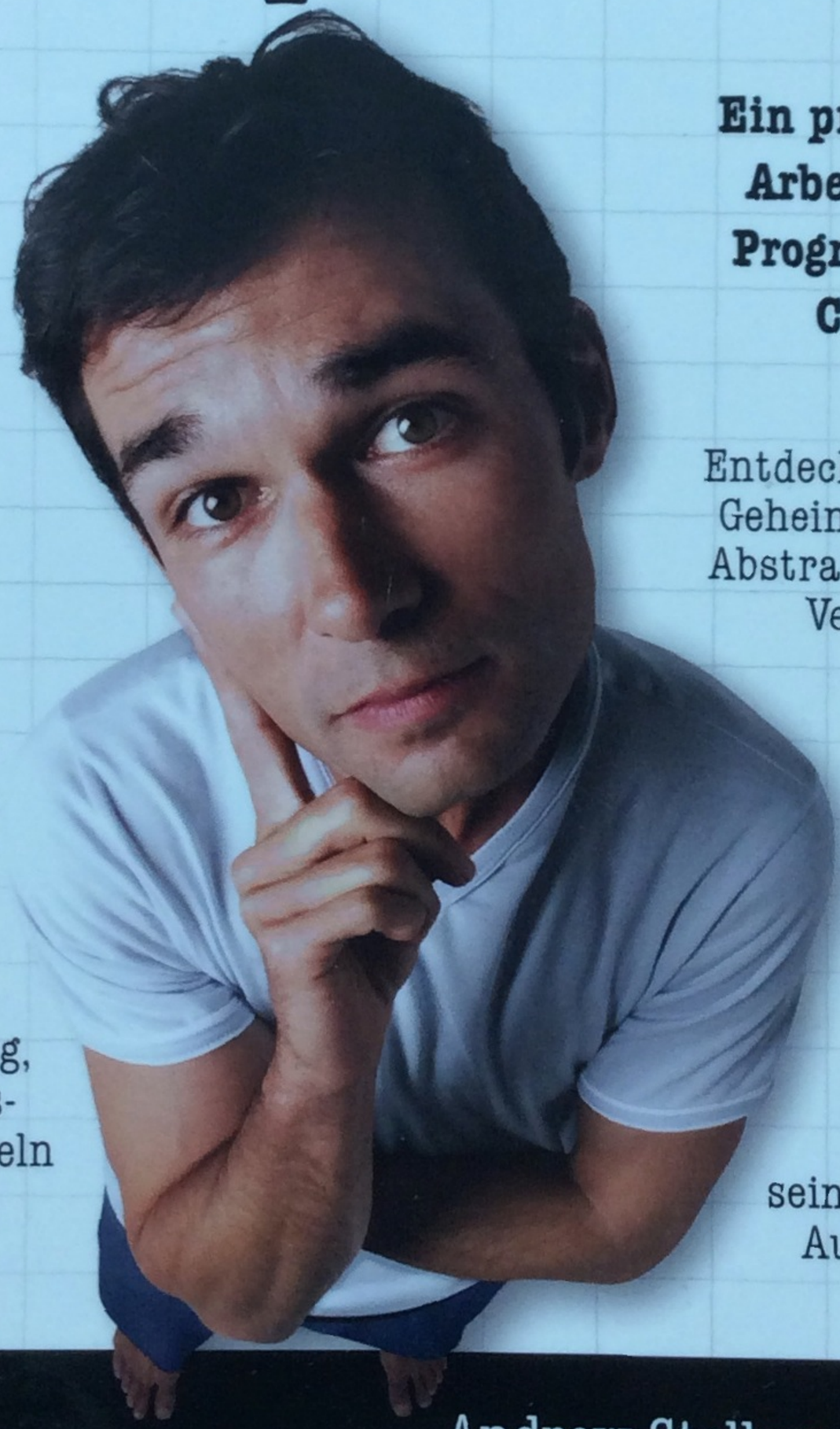
Ein Buch zum Mitmachen und Verstehen

2. Auflage
Behandelt C# 4.0 und Visual Studio 2010

C#

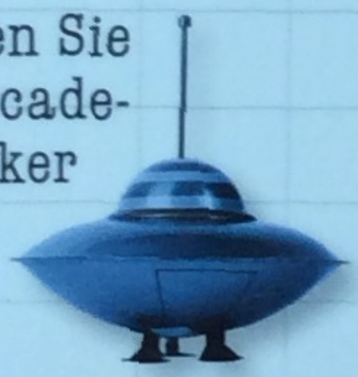
von Kopf bis Fuß



Ein praxisorientiertes Arbeitsbuch für die Programmierung mit C# und .NET

Zeigen Sie Ihren Daten mit LINQ, wer der Boss ist

Reanimieren Sie einen alten Arcade-Klassiker



Entdecken Sie die Geheimnisse von Abstraktion und Vererbung

Erfahren Sie, wie es Susi gelang, mit Erweiterungsmethoden die Regeln von Objekthausen zu beugen

Erleben Sie, wie Tim seine Daten mit generischen Auflistungen in den Griff bekommt

O'REILLY®

Andrew Stellman & Jennifer Greene
Deutsche Übersetzung von Lars Schulten

Mit Stream.Read() Bytes aus einem Stream lesen

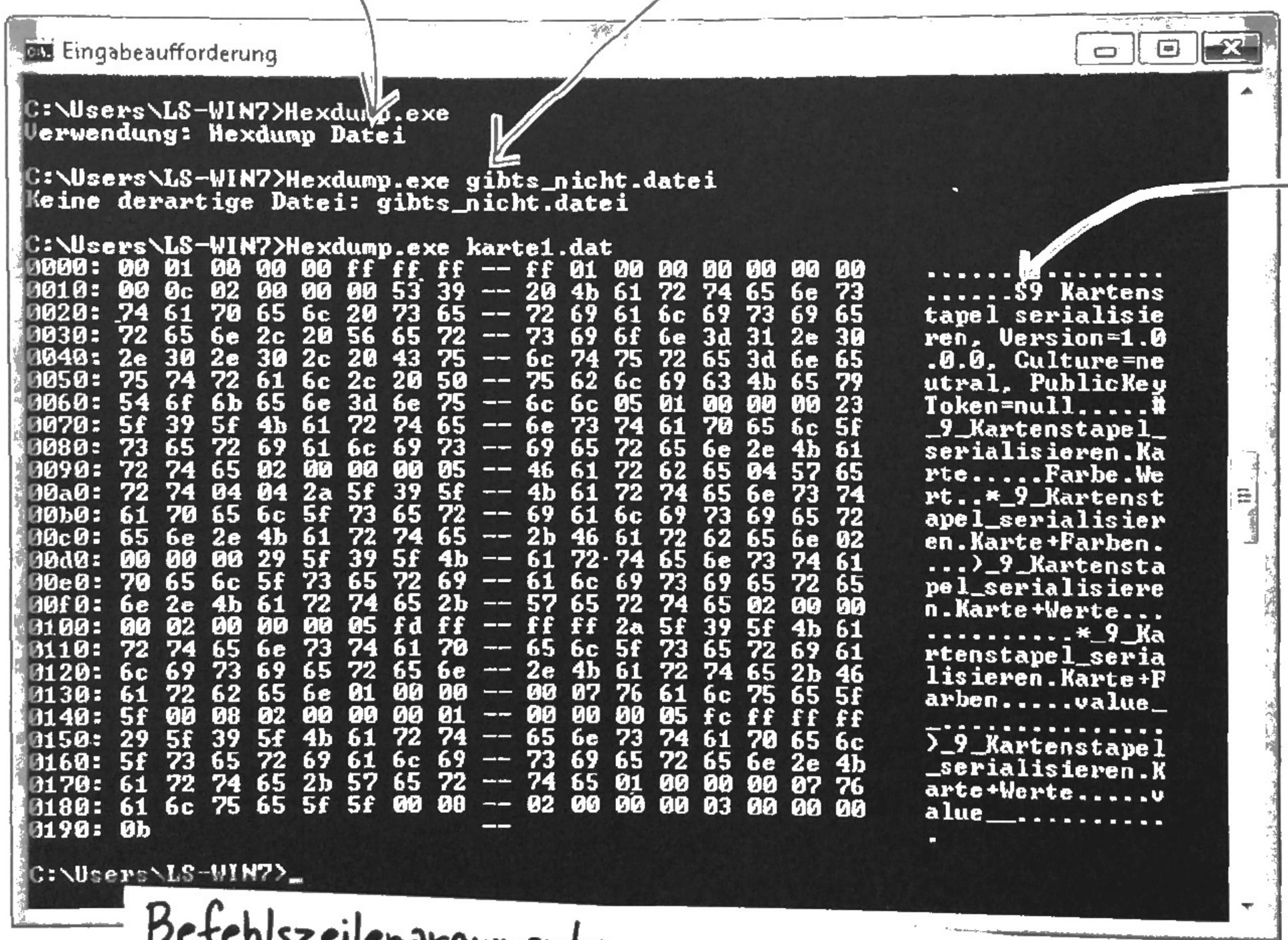
Tun Sie das

Bei Textdateien funktioniert der Hex-Dumper ausgezeichnet. Aber es gibt ein Problem. Versuchen Sie, mit `File.WriteAllBytes()` ein Array mit Bytes, die alle größer als 127 sind, in eine Datei zu schreiben und diese dann mit dem Dumper ausgeben zu lassen. Alle Bytes werden als »fd« eingelesen! Grund dafür ist, dass **`StreamReader` zur Verarbeitung von Textdaten gedacht ist,** diese aber nur Bytes mit Werten unter 128 enthalten können. Lassen Sie uns das beheben – indem wir die Bytes mit der Methode **`Stream.Read()`** direkt aus dem Stream lesen. Und als Schmankerl werden wir die Sache gleich so ausführen wie in einem echten Hex-Dump-Programm: Wir lassen es einen Dateinamen als **Befehlszeilenargument** entgegennehmen.

Erstellen Sie eine Konsolenanwendung und **geben Sie ihr den Namen `hexdumper`**. Den Code für das Programm sehen Sie auf der gegenüberliegenden Seite. So sieht es aus, wenn Sie das Programm ausführen:

Geben Sie kein Argument an, liefert das Programm eine Fehlermeldung und bricht die Ausführung ab.

Auch wenn Sie einen Dateinamen angeben, den es nicht gibt, bricht das Programm mit einem Fehler ab.



Übergeben Sie einen gültigen Dateinamen, wird ein Hex-Dump des Inhalts auf der Konsole ausgegeben.

Gewöhnlich schreiben wir mit Console.WriteLine() auf die Konsole. Aber zur Ausgabe von Fehlern nutzen Sie besser Error.WriteLine(), damit Fehlermeldungen nicht umgeleitet werden, wenn wir die Ausgabe mit > oder >> umleiten.

Befehlszeilenargumente

Wenn Sie eine Konsolenanwendung anlegen, erstellt Visual Studio eine Klasse Program mit einem Einstiegspunkt mit folgender Deklaration: **static void Main(string[] args)**. Führen Sie Ihr Programm mit Befehlszeilenargumenten aus, enthält der Parameter **args** diese Argumente. Das gilt nicht nur für Konsolenanwendungen, sondern auch für Windows Forms-Anwendungen.

Sie können dem Programm auch beim Debugging Kommandozeilenargumente übergeben. Wählen Sie dazu, wenn Sie das Programm in der IDE im Debugger ausführen, im Projekt-Menü »Eigenschaften...« und geben Sie die Befehlszeilenargumente dann im Tab Debugging an.

Über diesen Parameter werden Kommandozeilenargumente übergeben.

Wenn args.Length nicht gleich 1 ist, wurde entweder null oder mehr als ein Argument angegeben.

```
static void Main(string[] args)
{
    if (args.Length != 1)
    {
        Console.Error.WriteLine("Verwendung: hexdmper Datei");
        System.Environment.Exit(1);
    }
    if (!File.Exists(args[0]))
    {
        Console.Error.WriteLine("Keine derartige Datei: {0}", args[0]);
        System.Environment.Exit(2);
    }
    using (Stream eingabe = File.OpenRead(args[0]))
    {
        int position = 0;
        byte[] puffer = new byte[16];
        while (position < eingabe.Length)
        {
            int geleseneZeichen = eingabe.Read(puffer, 0, puffer.Length);
            if (geleseneZeichen > 0)
            {
                Console.Write("{0}: ", String.Format("{0:x4}", position));
                position += geleseneZeichen;

                for (int i = 0; i < 16; i++)
                {
                    if (i < geleseneZeichen)
                    {
                        string hex = String.Format("{0:x2}", (byte)puffer[i]);
                        Console.Write(hex + " ");
                    }
                    else
                        Console.Write("   ");

                    if (i == 7)
                        Console.Write("-- ");

                    if (puffer[i] < 32 || puffer[i] > 250) { puffer[i] = (byte)'.'; }
                }
                string pufferInhalt = Encoding.UTF8.GetString(puffer);
                Console.WriteLine("    " + pufferInhalt.Substring(0, geleseneZeichen));
            }
        }
    }
}
```

Beachten Sie die Verwendung von Console.Error.WriteLine().

Exit() beendet das Programm. Übergeben Sie beim Aufruf einen int, wird dieser als Fehlercode geliefert (was bei Befehlsskripten und Batch-Dateien hilfreich ist).

Prüft, ob eine gültige Datei übergeben wurde. Gibt es die Datei nicht, wird eine andere Meldung ausgegeben und mit einem anderen Code die Programmausführung abgebrochen.

Einen StreamReader brauchen wir nicht, weil wir direkt aus dem Stream lesen.

Nutzen Sie Stream.Read(), um Bytes direkt in einen Puffer zu lesen. Diesmal ist dieser Puffer ein byte-Array. Was sinnvoll ist, da wir jetzt Bytes lesen und keine Zeichen aus einer Textdatei.

Dieser Teil des Programms ist identisch, nur enthält der Puffer Bytes, keine Zeichen (aber String.Format() macht in beiden Fällen das Richtige).

Auf diese Weise kann man ein Byte-Array in einen String umwandeln. Es ist Teil von Encoding.UTF8 (oder einer anderen Kodierung), da unterschiedliche Kodierungen die gleichen Byte-Folgen auf unterschiedliche Zeichenfolgen abbilden können.