DOTNETBLACKBOX

Ein Flugschreiber für .NET-Projekte

„Post mortem“-Runtime-Analyse für C#- und VB.NET-Projekte.

Wer kennt das nicht: Der Kunde ruft verzweifelt an und beklagt, dass sein C#- oder VB.NET-Programm abstürzt. Es wird keine oder nur eine unzureichende Fehlermeldung ausgegeben. Die Log-Einträge der Ereignisanzeige sind auch nicht besonders aussagekräftig. Der Absturz tritt nur sporadisch auf und ist in der Visual-Studio-Umgebung des Entwicklers nicht nachvollziehbar.

Dieser Stresssituation war wohl schon jeder Softwareentwickler ausgesetzt. Jetzt gilt es, möglichst schnell „die Kuh vom Eis“ zu holen und dem Kunden zu helfen. Aber wie? Man kann nicht einfach Visual Studio beim Kunden installieren, um zu debuggen.

Oft kann man nur vermuten, woran es liegt. Scheitert eine Typumwandlung? Ist es eine Division durch null? Oder ein unzulässiger Zugriff auf ein Array? Klappt denn zumindest die Verbindung zur Datenbank? Oder ist etwa die referenzielle Integrität der Datenbank verletzt?

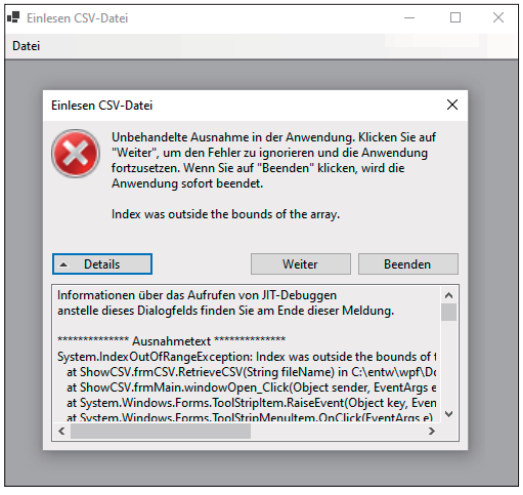
Die Idee

Großartig wäre es doch, wenn das Programm jeden Befehl, den es ausführt, oder Variablen-Inhalte von sich aus protokollieren würde. Dann könnte man dieses Protokoll auswerten und so dem Fehler auf die Schliche kommen. Wie bei einem Flugzeug, dessen Blackbox Spezialisten nach einem Unglück auswerten können. Genau dies macht DotNetBlackbox: Es sorgt dafür, dass Ihr Programm jeden einzelnen Schritt protokolliert. Bei Bedarf sogar mit Variablen-Inhalten. Beim Kunden vor Ort.

Wie das funktionieren soll? Indem Ihr Quellcode automatisch so erweitert wird, dass er eben genau dies tut.

```
frmCSV.cs
ShowCSV                                   ShowCSV.frmCSV
1   using System;
2   using System.Windows.Forms;
3   using System.IO;
4   namespace ShowCSV
5   {
        4 Verweise
6       public partial class frmCSV : Form
7       {
            1 Verweis
8           public frmCSV()
9           {
10              InitializeComponent();
11          }
            1 Verweis
12          public void RetrieveCSV(String fileName)
13          {
14              Text = fileName;
15              using (StreamReader sr = new StreamReader(fileName))
16              {
17                  String[] headerLines = sr.ReadLine().Split(',');
18                  foreach (String header in headerLines)
19                  {
20                      DataGridViewTextBoxColumn col = new DataGridViewTextBoxColumn();
21                      col.HeaderText = header;
22                      DGV.Columns.Add(col);
23                  }
24                  while (!sr.EndOfStream)
25                  {
26                      String line = sr.ReadLine();
27                      int row = DGV.Rows.Add();
28                      for(int i=0;i<headerLines.Length;i++)
29                      {
30                          String[] values = line.Split(',');
31                          DGV.Rows[row].Cells[i].Value = values[i];
32                      }
33                  }
34                  sr.Close();
35              }
36          }
37      }
38  }
```

Der aktuelle Quellcode (Bild 3)



Programmabsturz beim Kunden (Bild 1)

```
EuropeanCapitals.csv
1   Land,Hauptstand,Einwohner
2   Ålandinseln,Mariehamn,11709
3   Albanien,Tirana,418495
4   Andorra,Andorra la Vella,22205
5   Belarus,Minsk,1982444
6   Belgien,Brüssel,1175173
7   Bosnien und Herzegowina,Sarajevo,275524
8   Bulgarien,Sofia,1241675
9   Dänemark,Kopenhagen,616098
10  Deutschland,Berlin,3644826
11  Estland,Tallinn,430805
12  Färöer-Inseln,Thorshaven,13408
13  Finnland,Helsinki,639227
14  Frankreich,Paris,2206488
15  Gibraltar,Gibraltar,34003
16  Griechenland,Athen,664046
17  Guernsey,Saint Peter Port,18958
18  Insel Man,Douglas,27938
19  Irland,Dublin,553165
20  Island,Reykjavík,124644
21  Italien,Rom,2873147
22  Jersey,Saint Helier,33500
```

Die CSV-Datei, mit der wir entwickelt haben, verursacht keinerlei Probleme (Bild 2)

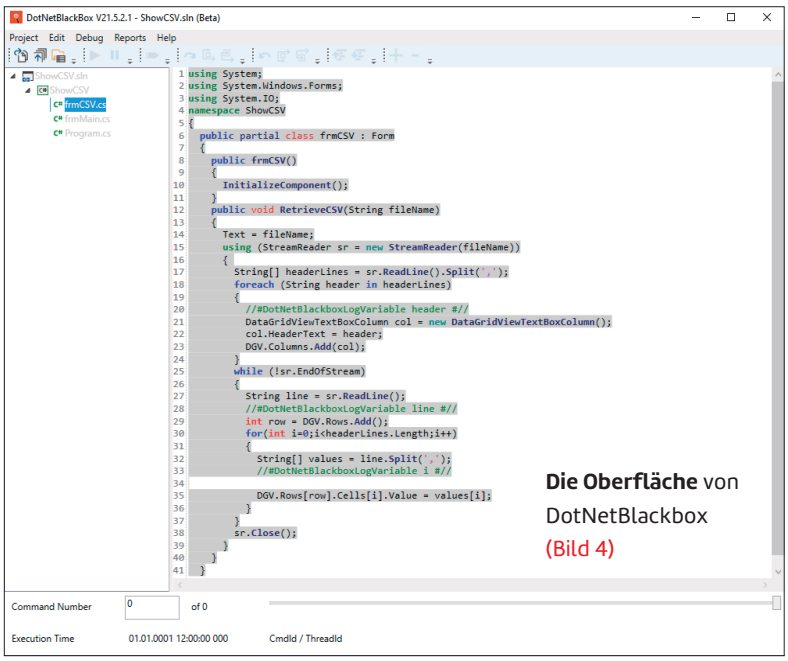
In zehn Schritten zur Fehlerursache

Der Ablauf ist folgender:

1. Der Kunde berichtet von einem mysteriösen Programmabsturz.
2. Öffnen der Visual Studio Solution (*.sln*) in DotNetBlackbox.
3. Bestimmen der zu analysierenden Klassen, Module, Codezeilen und Variablen.
4. Generieren einer Kopie Ihrer Original-Solution mit einer „angereicherten“ Protokollfunktion.

5. Kompilieren der aufbereiteten Solution.
6. Publish der erzeugten Binaries im Kundensystem.
7. Ausführen der Anwendung (EXE oder DLL) im Kundensystem bis zum erwarteten Absturz.
8. Zurückübertragung der Log-Dateien (*.rif* und *.riv*) in die ursprüngliche Entwicklungsumgebung.
9. Öffnen der Log-Dateien mit DotNetBlackbox.
10. Analysieren der Laufzeit bis hin zum „letzten Lebenszeichen“ vor dem Absturz.

Das folgende Projekt soll die Problemstellung verdeutlichen.



Die Oberfläche von DotNetBlackbox (Bild 4)

1. Programmabsturz

Wir haben für unseren Kunden ein Programm entwickelt, das CSV-Dateien anzeigt. Seit Jahren hat dies auch immer gut funktioniert, aber nun liefert das Programm plötzlich einen Fehler (Bild 1).

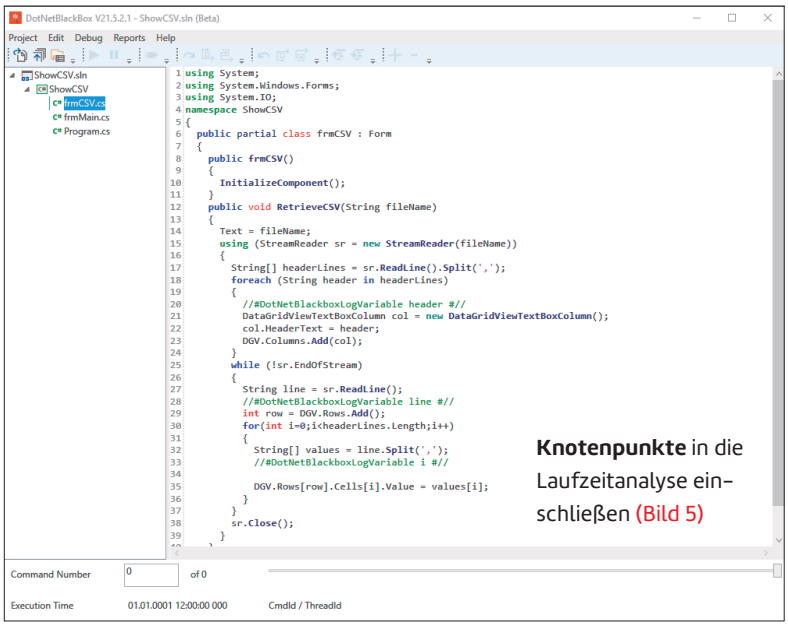
Unsere einzulesende CSV-Datei in der Entwicklungsumgebung (europäische Hauptstädte samt Einwohnerzahl) funktioniert einwandfrei, wie Bild 2 zeigt.

Unsere aktuelle C#-Programmquelle hierzu ist in Bild 3 zu sehen.

Dass wir es mit einem Indexüberlauf zu tun haben, scheint zunächst klar. Aber bei welchem Datensatz ist dies passiert? Bei größeren Projekten wird es oft schwierig, das Problem eindeutig einzugrenzen.

Insbesondere gilt das, da wir oft aus datenschutzrechtlichen Gründen nicht an die Original-Kundendatenbank oder die zu verarbeitenden Dateien gelangen. Auch sind die Infrastrukturelemente wie Pfade, Verzeichnisse, Zugriffsrechte, Einstellungen in Active Directory und so weiter immer anders.

Dies ist aber kein Grund zur Verzweiflung, denn nun gibt es DotNetBlackbox [1].



Knotenpunkte in die Laufzeitanalyse einschließen (Bild 5)

2. Öffnen der Solution in DotNetBlackbox

Öffnen wir zunächst das Projekt über *Project | Open Visual Studio Solution (SLN)* (Bild 4).

Sie sehen, dass im Baum links die Menüpunkte und der Quelltext rechts ausgegraut sind. Dies bedeutet, dass die einzelnen Code-Dateien beziehungsweiseAbschnitte noch nicht zum Analysieren vorbereitet sind.

3. Bestimmen der Quellcode-Dateien, Zeilen und Variablen

Ändern wir dies, indem wir mit der rechten Maustaste auf den ersten Knotenpunkt (*ShowCSV.sln*) klicken und im daraufhin erscheinenden Kontextmenü *Analyze all lines* auswählen. Auf diese Weise werden alle Dateien innerhalb der Solution zum Analysieren vorgesehen. Von der Laufzeitanalyse ausschließen lassen sich Knotenpunkte im Solution-Baum analog dazu durch Auswahl von *Analyze no lines*.

Nun sieht das Ganze so aus wie in Bild 5. Ab jetzt ist Farbe im Spiel: Alle nicht ausgegrauten Knotenpunkte werden analysiert.

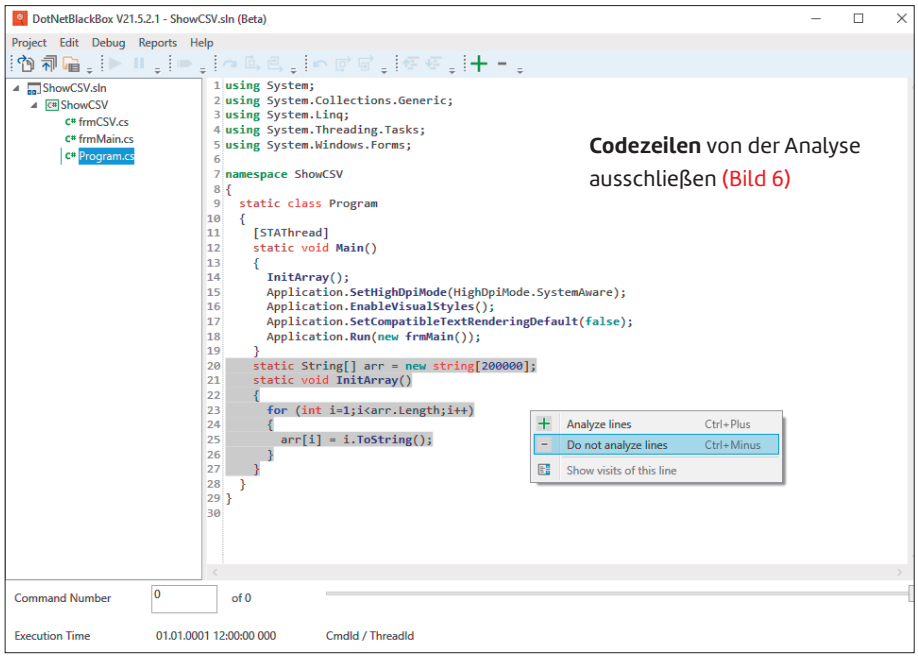
Selbstverständlich können wir nun wieder einzelne Codedateien im Baum, bei denen wir sicher sind, dass sie nicht für den Fehler verantwortlich sein können, von der Verarbeitung ausschließen.

Ausschließen von Codeblöcken

Dies gilt auch für einzelne Codezeilen, für die es unnötig ist, sie zu beobachten, da sie nichts Fehleranfälliges tun. In ▶

diesem Fall ist es eine große Initialisierungsschleife, die auf dem Kundensystem das Programm nur unnötig langsam machen würde und nur überflüssigen Speicherplatz für die erstellten Log-Dateien generieren würde (Bild 6).

Solche großen, harmlosen Schleifen sollten nicht analysiert werden, denn dies verlangsamt die Laufzeit erheblich und bläht die Log-Dateien nur unnötig auf.



Codezeilen von der Analyse ausschließen (Bild 6)

Beobachten von Variablen-Inhalten

Sie haben auch die Möglichkeit, den Inhalt von Variablen zu protokollieren. Dies muss jedoch vor dem Öffnen mit DotNetBlackbox in der Original-Quelle mit Visual Studio geschehen. Fügen Sie hierzu folgenden harmlosen Kommentar in Ihren C#-Quellcode ein (Bild 7):

```
//#DotNetBlackboxLogVariable variable #//
```

Für VB.NET lautet das Format dieses Kommentars:

```
''DotNetBlackboxLogVariable variable #''
```

Intern wird für jedes zu analysierende Objekt die *.ToString()*-Methode aufgerufen. So ist es möglich, auch Objekte, Datumsfelder oder Variablen vom Typ GUID bis zu einer Länge von 2048 Bytes zu protokollieren. Arrays serialisieren Sie am besten vorher und loggen dann diesen String.

Achtung: Beachten Sie beim Protokollieren von Variablen-Inhalten (Adressdaten, Kreditkartennummern, Passwörter und so weiter) jedoch, dass Sie dadurch möglicherweise gegen datenschutzrechtliche Vorschriften verstoßen. Klären Sie bitte deshalb vorher den Umfang der Protokollierung mit Ihrem Kunden ab.

```
12    public void RetrieveCSV(String fileName)
13    {
14      Text = fileName;
15      using (StreamReader sr = new StreamReader(fileName))
16      {
17        String[] headerLines = sr.ReadLine().Split(',');
18        foreach (String header in headerLines)
19        {
20          //#DotNetBlackboxLogVariable header #//
21          DataGridViewTextBoxColumn col = new DataGridViewTextBoxColumn();
22          col.HeaderText = header;
23          DGV.Columns.Add(col);
24        }
25        while (!sr.EndOfStream)
26        {
27          String line = sr.ReadLine();
28          //#DotNetBlackboxLogVariable line #//
29          int row = DGV.Rows.Add();
30          for(int i=0;i<headerLines.Length;i++)
31          {
32            String[] values = line.Split(',');
33            //#DotNetBlackboxLogVariable i #//
34
35            DGV.Rows[row].Cells[i].Value = values[i];
36          }
37        }
38        sr.Close();
39      }
40    }
```

Inhalte von Variablen mitprotokollieren (Bild 7)

4. Generieren der „angereicherten" Solution

Über den Menüpunkt *Project | Generate SLN with Runtime Log* wird nun eine neue Solution generiert (Bild 8). Diese finden Sie anschließend in Ihrem persönlichen Dokumenten-Ordner im Verzeichnis *DotNetBlackbox Solution Files*.

Legen Sie hier fest, wie oft die Log-Dateien oben abgeschnitten werden sollen, um die Festplatte des Zielrechners nicht zum Überlaufen zu bringen.

Geben Sie hier außerdem den Ort der Log-Dateien auf dem Zielrechner an.

Achtung: Der ausführende Prozess auf dem Zielrechner braucht unbedingt eine Schreibberechtigung für dieses Verzeichnis. Nach dem Generieren finden Sie die neue Solution in Ihrem persönlichen Dokumenten-Ordner.

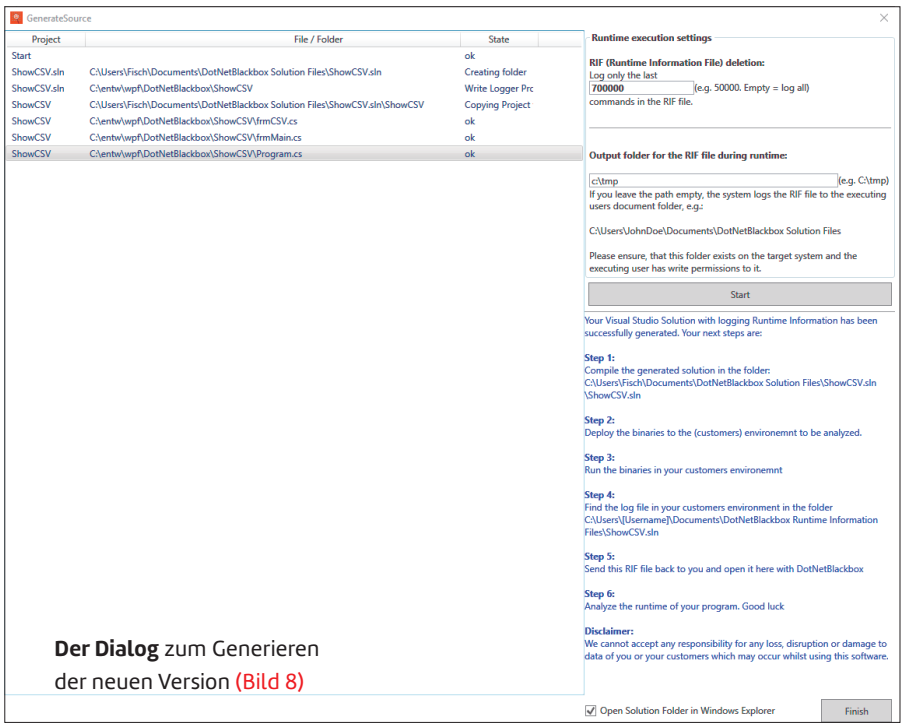
5. Kompilieren der neuen Version

Die Solution wird nun mit Visual Studio geöffnet und kompiliert. Sie sehen am Quellcode, dass dieser mit sehr vielen *DotNetBlackboxLogger.Log.RIF*-Aufrufen angereichert wurde (Bild 9).

Wichtig hierbei: Die Grundfunktionalität Ihres Quellcodes wurde nicht verändert. Jener Quellcode, welchen Sie nicht zur Beobachtung markiert haben, erhält auch hier keine vorangehenden Log-Anweisungen.

6. Publish der erzeugten Binaries im Kundensystem

An dieser Stelle erfahren Sie von mir nichts Neues. Sie wissen besser als jeder andere, wie Sie die neue Version auf dem Kundensystem installieren. Ob dies überhaupt in der echten Produktivumgebung geschehen darf, entscheidet ohnehin meist Ihr Kunde. Aber vielleicht ist der Fehler in der Test-



Der Dialog zum Generieren der neuen Version (Bild 8)

protokollieren gedacht haben, auch wirklich protokollieren muss. Von daher ist es wichtig, dass Sie vorher Bereiche, von denen Sie sich sicher sein können, dass diese für den Absturz nicht verantwortlich sind, von der Analyse ausschließen (siehe Punkt 3). Oder schließen Sie nur diejenige Klasse in die Verarbeitung mit ein, von der Sie ganz sicher sind, dass in ihr der Fehler passiert.

Ihr Programm erstellt nun zwei Log-Dateien auf dem Rechner, auf welchem es läuft. Haben Sie keine explizite Pfad-Angabe für die Log-Dateien angegeben (siehe Punkt 4), werden die beiden Dateien im persönlichen Dokumenten-Ordner unter *DotNetBlackbox Runtime Information Files* erzeugt. Diese beiden Dateien können abhängig von Ihrer Konfiguration und der Laufzeitlänge schon mal einige Hundert Megabyte groß werden:

- **RIF-Datei:** Diese Datei enthält Datum, Uhrzeit und die Positionen der Statements, welche ausgeführt wurden.
- **RIV-Datei:** In dieser Datei sind die aufgezeichneten Variablen-Inhalte gespeichert.

Damit Ihr Programm diese beiden Log-Dateien erstellen kann, müssen Sie sicherstellen, dass es auch Schreibzugriff auf dieses Verzeichnis hat. Dies sollte bereits während der Planung in Punkt 4 mit berücksichtigt worden sein.

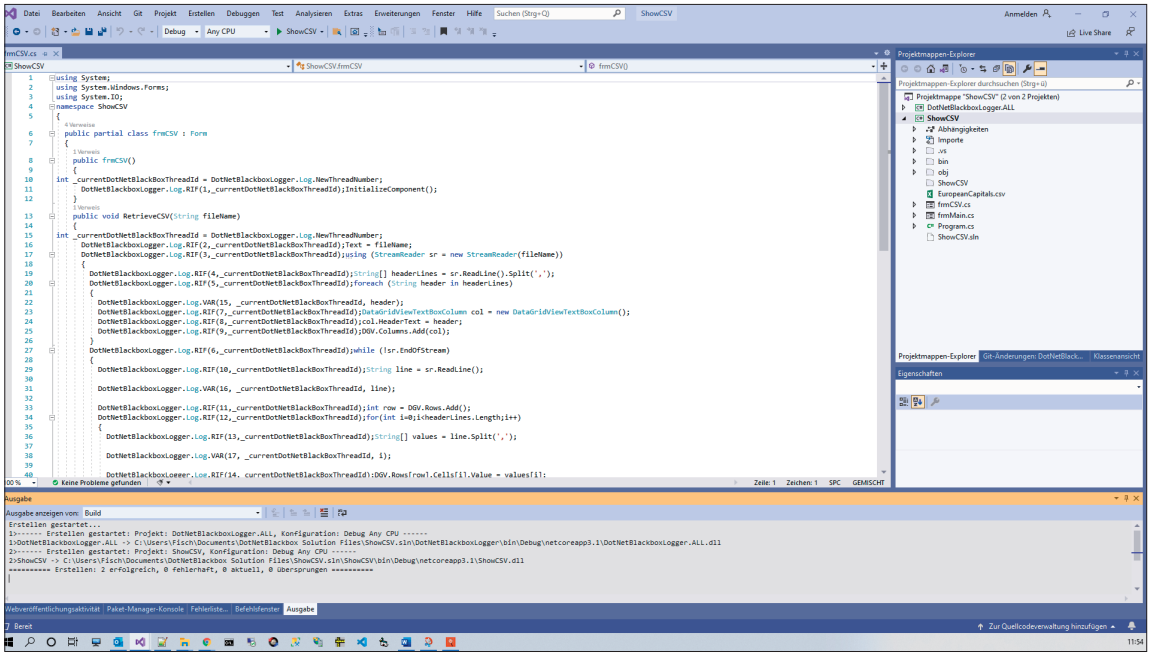
Den kompletten Ordner *DotNetBlackbox Runtime Information Files* mit den dazugehörigen Dateien und dem *tmp*- ▶

oder Staging-Umgebung nachvollziehbar und Sie dürfen diese neue Version dort einsetzen?

7. Ausführen der Anwendung auf dem Zielsystem

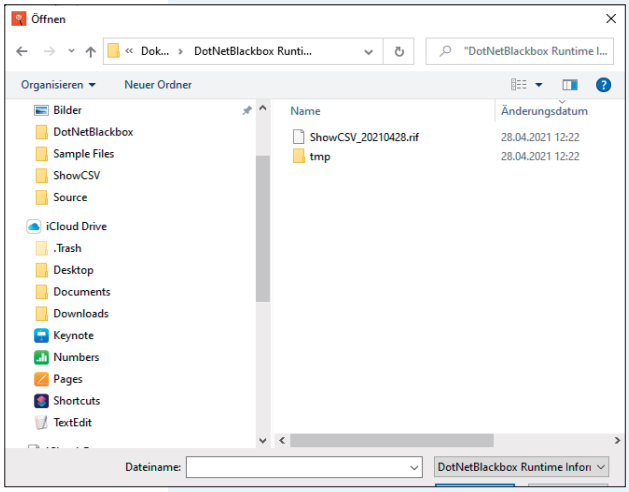
Wenn Sie den Punkt 6 geschafft haben, sind wir kurz vor dem Ziel. Was wir sonst freilich nicht hoffen, wünschen wir uns heute schon: Dass der Fehler möglichst schnell auftritt, damit wir ihn endlich analysieren können.

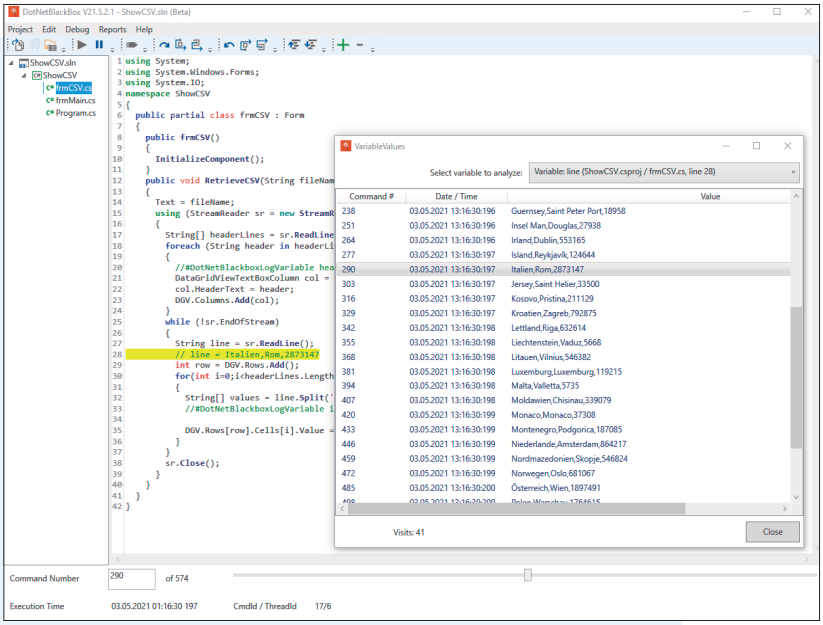
Das um DotNetBlackbox erweiterte Programm wird möglicherweise sehr viel langsamer sein als gewohnt. Dies liegt einfach nur daran, dass es wirklich jeden Schritt, den Sie zu



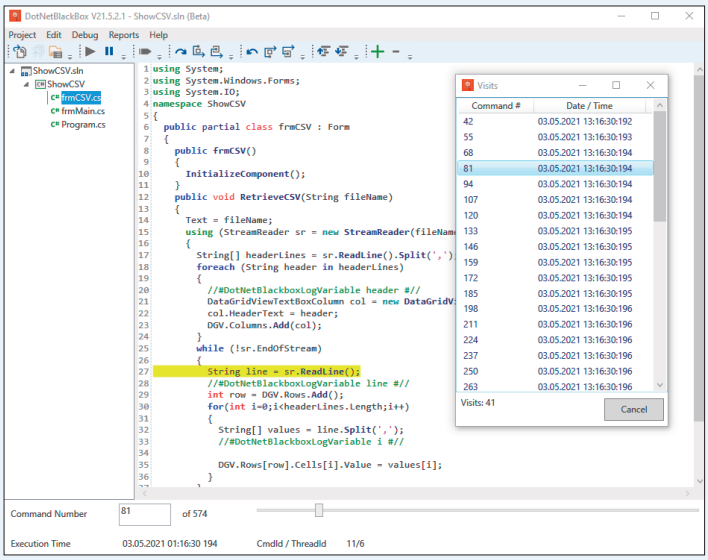
Ihr Quellcode, angereichert mit sehr vielen Log-Aufrufen (Bild 9)

Die RIF-Datei des Kunden. Die dazugehörige RIV-Datei muss sich im selben Ordner befinden (Bild 10)





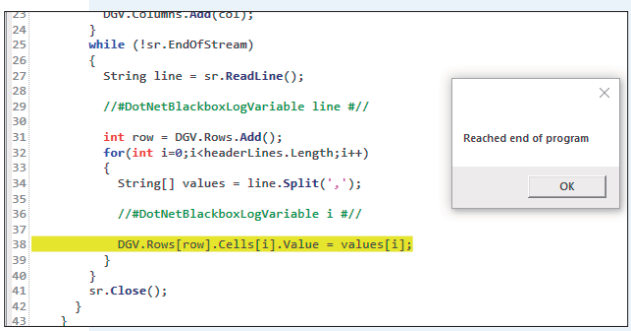
Welche Inhalte hatten wann welche Variablen? (Bild 11)

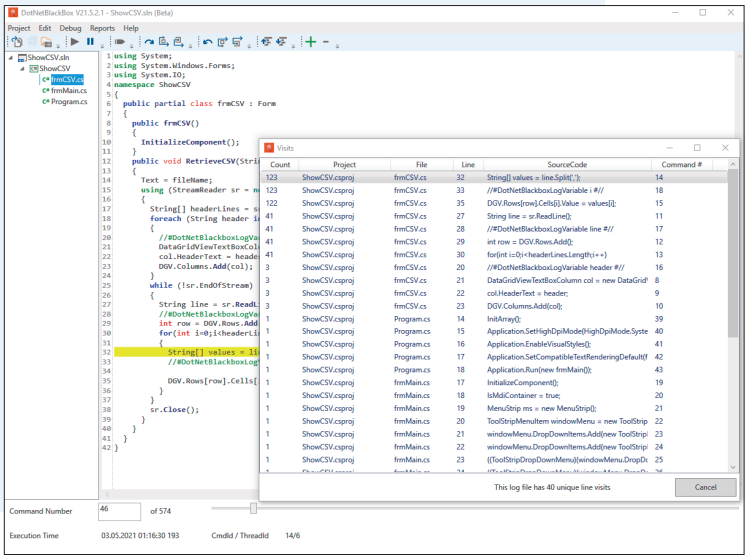


Anzahl der Besuche pro Zeile ermitteln (Bild 12)

Welche Zeilen wurden am häufigsten ausgeführt? (Bild 13)

Das letzte Lebenszeichen vor dem Absturz (Bild 14)





Verzeichnis sollten Sie nach getaner Arbeit sicherheitshalber wieder löschen, da diese eventuell datenschutzrechtlich bedenkliche Inhalte enthalten können.

8. Abholen der Log-Dateien

Falls die Dateien größer geworden sind, können Sie diese einfach in ein ZIP-Archiv legen und zu sich auf den eigenen Entwicklerrechner übertragen. Ob dies über einen File-Server, über FTP, E-Mail oder Remote-Desktop geschieht, bleibt dabei Ihnen überlassen.

Achtung: Beachten Sie jedoch, dass sich in der RIV-Datei eventuell datenschutzrelevante Daten befinden, falls Sie Adressen oder persönliche Daten protokolliert haben. In diesem Fall sollten Sie die beiden Dateien vor dem Übertragen zu Ihrem Rechner unbedingt verschlüsseln, vor allem, wenn Sie öffentliche Filesharing-Server verwenden möchten.

9. Öffnen der RIF- und RIV-Dateien mit DotNetBlackbox

Sie können die beiden Log-Dateien nun auf Ihrem Entwicklerrechner nach Herzenslust über den Menüpunkt *Project | Open Runtime Information File (RIF)* analysieren (Bild 10).

10. Analysieren der Laufzeit

Jetzt wird es spannend. Dennoch bleibt nicht mehr viel zu erklären. Sehen Sie sich bitte die folgenden Screenshots an, sie sprechen für sich:

- Die gelb hinterlegte Zeile zeigt das aktuell ausgeführte Statement an. Am unteren Bildschirmrand sehen Sie die Uhrzeit und um den wievielten Befehl es sich gerade handelt.
- Durch Verschieben des Sliders neben dem Befehlszähler spielen Sie Zeitmaschine: Sie reisen zurück in die Vergangenheit und halten die Zeit an.

Über *Debug* rufen Sie den Debugger auf und können durch Auswahl des entsprechenden Menüpunkts alle Programmschritte Schritte vorwärts und rückwärts analysieren: Sie überspringen Methodenaufrufe vor- oder rückwärts, oder Sie springen wieder zurück zum Aufruf. Mit einem Klick springen Sie zum letzten Befehl vor dem eigentlichen Programmabsturz.

```
25          while (!sr.EndOfStream)
26          {
27            String line = sr.ReadLine();
28
29            //#DotNetBlackboxLogVariable line #//
30
31            int row = DGV.Rows.Add();
32            for(int i=0;i<headerLines.Length;i++)
33            {
34              String[] values = line.Split(',');
35
36              // i = 2
37
38              DGV.Rows[row].Cells[i].Value = values[i];
39            }
40          }
41          sr.Close();
```

Anzeigen der Inhalte von Variablen (Bild 15)

```
25          while (!sr.EndOfStream)
26          {
27            String line = sr.ReadLine();
28
29            // line = Schweden,Stockholm935619
30
31            int row = DGV.Rows.Add();
32            for(int i=0;i<headerLines.Length;i++)
33            {
34              String[] values = line.Split(',');
35
36              //#DotNetBlackboxLogVariable i #//
37
38              DGV.Rows[row].Cells[i].Value = values[i];
39            }
```

Die Absturzursache: ein unzulässiger Inhalt in der CSV-Datei (Bild 16)

Bild 11 zeigt, wie Sie analysieren, wie sich Variableninhalte im Lauf der Zeit verändert haben.

Analysieren Sie nun, wie oft bestimmte Programmzeilen aufgerufen worden sind (Bild 12).

Überprüfen Sie, welche Codezeilen am häufigsten aufgerufen worden sind. So finden Sie möglicherweise Stellen, die fälschlicherweise zu oft aufgerufen werden (Bild 13).

Bild 14 zeigt das letzte Kommando, welches protokolliert wurde, sozusagen das „letzte Lebenszeichen". Und nur ein Schritt zurück führt zur Lösung (Bild 15): Die Variable *i* steht auf *2*.

Und noch einmal zwei Schritte zurück (Bild 16): In der Variablen *line* fehlt das Komma nach *Stockholm*. Der Split in Zeile 34 und das Auslesen des zweiten Indexes in Zeile 38 verursachen den Index-Überlauf.

Fazit

DotNetBlackbox ermöglicht echtes „Post mortem"-Debugging für Ihre C#- und VB.NET-Projekte. Da die Funktionalität Ihres Programms unverändert bleibt, es jedoch akribisch mitprotokolliert, was es denn wann, wo und wie so treibt, haben Sie die Möglichkeit, alles über die Laufzeit Ihres Programms beim Kunden herauszufinden und Programmabstürze zu analysieren und künftig zu vermeiden.

Es werden alle Projekttypen und Frameworks von Visual Studio 2019 unterstützt: Windows Forms, WPF, Bibliotheken, ASP.NET (Code beside), Razor, .NET Framework von 4.5 bis .NET Core.

DotNetBlackbox wurde von mir mit der aktuellen Version von Visual Studio 2019 (Version 16.9.4) in WPF und C# entwickelt. ■

[1] http://fssoft.de/Products/DotNetBlackbox/

Beta-Tester gesucht

Das Programm ist noch sehr neu und daher bestimmt noch nicht ganz fehlerfrei. Deshalb wird es unter [1] zunächst kostenlos als Beta-Version zur Verfügung gestellt. Bitte teilen Sie mir Ihre Erfolge oder Misserfolge bei der Nutzung per E-Mail mit: *DotNetBlackBox@fssoft.de*

Robert Fischbacher
ist freiberuflicher Softwareentwickler seit 1998. Seit den Anfängen von .NET entwickelt er individuell auf seine Kunden zugeschnittene Lösungen auf Basis von ASP.NET, ASP.NET MVC, WPF und WinForms mit CU oder VB.NET.
Robert.Fischbacher@fssoft.de

dnpCode A2110Blackbox