Plugins für FastView entwickeln

Einleitung

In diesem Tutorial will ich beschreiben, wie man für mein Programm „FastView“ Plugins mit dem Visual Studio entwickeln kann. Solche Plugins ermöglichen es FastView mehr Dateitypen anzuzeigen.

Alle Plugins sind DLL-Dateien, die beim Start von FastView geladen werden. Damit dies geschieht müssen sich diese Dateien bloß im „plugins“-Ordner von FastView befinden. Dort sind auch schon einige Plugins zu finden, denn **alle** Dateitypen sind bei FastView als Plugin implementiert. In diesem Tutorial werde ich exemplarisch ein Plugin entwickeln, das den Inhalt von Zip-Dateien anzeigen kann. Dazu benutze ich die „SharpZipLib“-Bibliothek.

Los geht's!

Als ersten müssen wir ein neues Projekt anlegen. Dazu klicken wir auf „Datei/Neu…/Projekt“. Das neue Projekt ist vom Typ „Klassenbibliothek“ (s. Abb. 1).

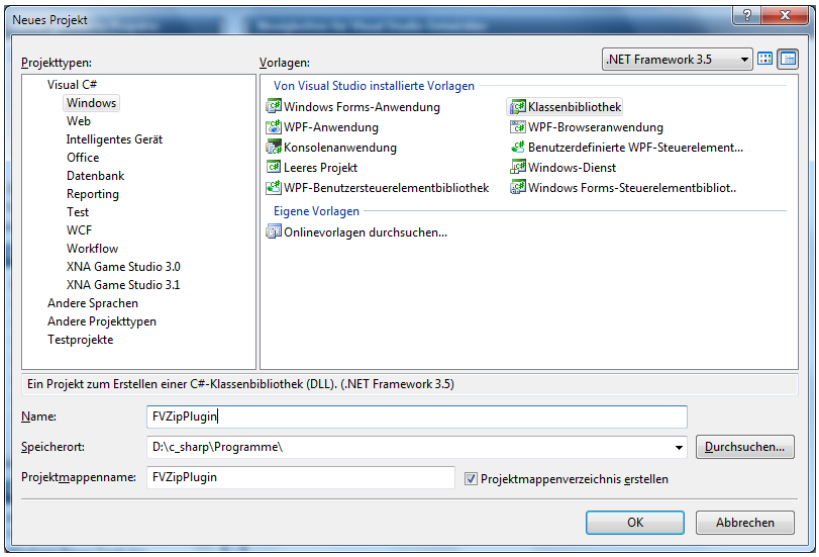


Abbildung 1: Neues Projekt

Damit haben wir ein leeres neues Projekt erstellt. Damit FastView dieses Plugin auch laden kann, muss es ein bestimmtes Interface implementieren, d.h. es muss sich an ein bestimmtes „Muster“ halten. Dieses „Muster“ befindet sich in der Datei „PluginInterface.dll“ im FastView Verzeichnis. Diese Datei muss jetzt dem Projekt als Verweis hinzugefügt werden. Dazu klicken wir mit der rechten Maustaste auf den Ordner „Verweise“ im Projektmappen-Explorer, und wählen „Verweis hinzufügen“ aus. In dem sich nun öffnendem Fenster klicken wir auf den Reiter „Durchsuchen“ und wählen die oben erwähnte Datei aus (s. Abb. 2).

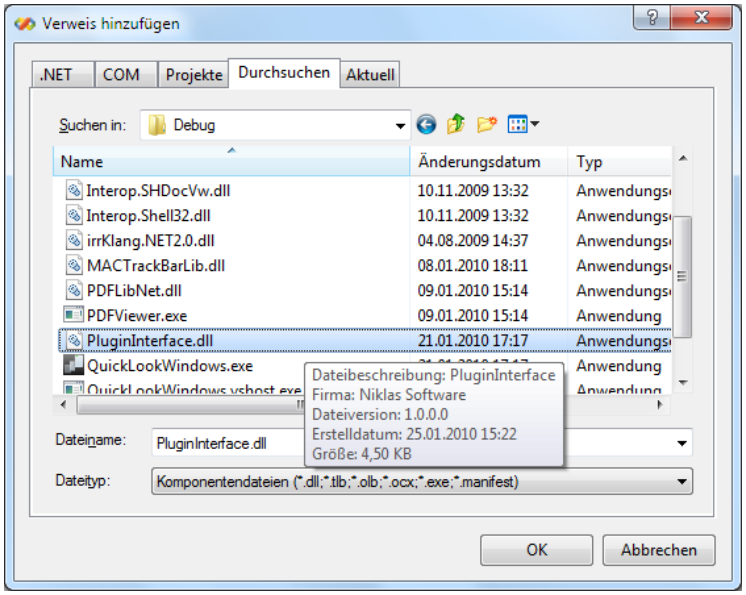


Abbildung 2: Verweis hinzufügen

Da wir ja die SharpZipLib benutzen wollen, müssen wir diese ertmal von dieser Seite downloaden: http://www.icsharpcode.net/OpenSource/SharpZipLib/. In diesem Download ist auch eine DLL Datei enthalten, die wir auch wieder dem Projekt hinzufügen. Das ganze sollte jetzt so aussehen, wie in Abbildung 3:

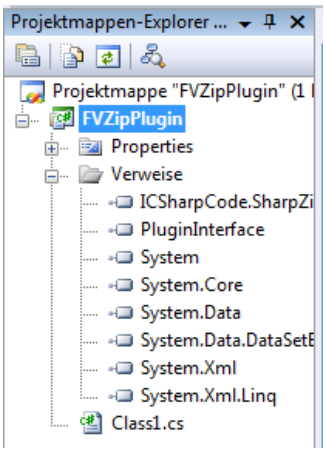


Abbildung 3: So sollst aussehen

Ein Verweis fehlt noch, nämlich der auf „System.Windows.Forms“. Den finden wir aber nicht mit „Durchsuchen“, sonder unter „.NET“.

Der Code

Da jetzt alle „Formalitäten“ erfüllt sind, können wir jetzt mit dem wirklichen programmieren Anfangen. Momentan sieht unser Code ja noch etwas dünn aus:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FVZipPlugin
{
    public class Class1
    {
    }
}
```

Das wird sich jetzt aber ändern. Zuerst einmal müssen wir zwei using-Zeilen einfügen:

```
using ICSharpCode.SharpZipLib.Zip;
using PluginInterface;
using System.Windows.Forms;
```

Damit können wir jetzt auch auf die beiden DLLs und System.Windows.Forms zugreifen. Als nächstes wird unsere „Class1“ vom dem Interface PluginInterface abgeleitet. Dazu ändern wir die zehnte Zeile wie folget:

```
public class Class1 : IFVPlugin
```

Ein kleiner Pfeil unter dem „IFVPlugin“ bietet uns an, die Member, die die Schnittstelle verlangt, auch gleich zu implementieren. Dieses Angebot nehmen wir gerne an! (s. Abb. 4)

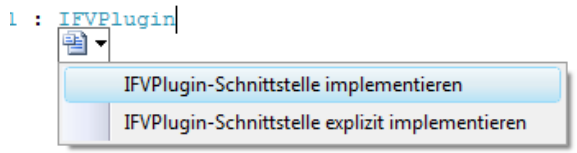


Abbildung 4: Schnittstelle implementieren

Nun hat sich unser Code gewaltig geändert:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using ICSharpCode.SharpZipLib;
using PluginInterface;
using System.Windows.Forms;

namespace FVZipPlugin
{
    public class Class1 : IFVPlugin
    {
        #region IFVPlugin Member

        public void ChangedWindowSize()
        {
            throw new NotImplementedException();
        }

        public void DisposeHandle()
        {
            throw new NotImplementedException();
        }

        public List<string> GetFileextensions()
        {
            throw new NotImplementedException();
        }

        public void InitContainer(Panel ContentPanel, Panel ControlPanel)
        {
            throw new NotImplementedException();
        }

        public bool IsInit
        {
            get
            {
                throw new NotImplementedException();
            }
            set
            {
                throw new NotImplementedException();
            }
        }

        public bool ShowFile(string path)
        {
            throw new NotImplementedException();
        }

        #endregion
    }
}
```

Das Visual Studio hat nun alle benötigten Methoden automatisch angelegt. Diese Methoden machen aber noch nicht mehr, als eine Ausnahme zu erzeugen. Das wollen wir nun ändern! Ich werde jetzt jede der Methoden durchgehen, und erklären wann sie aufgerufen wird, und was sie machen sollte.

Eines davon ist aber gar keine Methode:

```
  public bool IsInit
        {
            get
            {
                throw new NotImplementedException();
            }
            set
            {
                throw new NotImplementedException();
            }
        }
```

„IsInit“ ist eine Variable die angibt, ob das Plugin schon initialisiert wurde. Die Implementierung ist trivial:

```
public bool IsInit
         { get; set; }
```

Damit sorgt der Compiler für alles nötige.

GetFileextensions

Diese Methode wir schon beim Laden des Plugins aufgerufen, und sollte eine Liste von Dateiendungen (mit Punkt) zurückgeben, die dieses Plugin unterstützt. Wer die Anzeige von Ordnern verbessern will (die ist momentan fest in FastView integriert) muss als Dateiendung einen Leerstring zurückgeben. Da unser Plugin nur Zip-Dateien behandeln soll, lautet der Inhalt der Methode wie folgt:

```
List<string> ret = new List<string>();
ret.Add(".zip");
return ret;
```

InitContainer

Dies ist die erste Methode, die während der typischen Benutzung einer Plugins aufgerufen wird. Diese Methode soll, wie der Name schon sagt, das Plugin initialisieren. (Sie wird vor jeder Verwendung aufgerufen, wenn „IsInit“ „false“ ist; Damit kann das Plugin auch selbst steuern, wenn es noch mal initialisiert werden möchte)

Die Methode hat zwei Parameter, beides Panels. Das ContentPanel ist das wichtigste: In disem Panel soll die Datei später angezeigt werden. Das ControlPanel sitzt weiter unter im FastView Fenster, hier sollen z.B. Play/Pause Buttons oder ähnliche (einfache) Steuerungs-Möglichkeiten platziert werden.

Wenn es geht, sollten sich diese im Design gut in das restliche Interface einfügen.

Unser Plugin braucht nun aber keinerlei Steuereinheiten, darum lassen wir das Panel einfach leer. In das ContetPanel muss aber etwas rein: Eine „TextBox“, in der später die Liste der Dateinamen und Ordner angezeigt wird.

Für diese Textbox legen wir erst mal eine globale Variable an:

```
TextBox textBox1 = new TextBox();
```

Nun ändern wir den Inhalt der „InitContainer“-Methode in folgenden Code:

```
ContentPanel.Controls.Add(textBox1);

textBox1.Dock = DockStyle.Fill;
textBox1.Multiline = true;
textBox1.ReadOnly = true;
textBox1.AutoSize = true;
textBox1.ScrollBars = ScrollBars.Both;

IsInit = true;
```

Damit wird die TextBox dem ContetPanel hinzugefügt, und ein wenig angepasst. Danach wird IsInit auf „true“ gesetzt, da das Plugin ja nun initialisiert ist. Natürlich könnte man hier das Erscheinungsbild der TextBox noch etwas anpassen. Wer hier eine kompliziert Ansammlung von Steuerelemente platzieren will, sollte diese evtl. in einem UserControl zusammen fassen, den bei dessen Erstellung kann man den Designer vom Visual Studio benutzten. So muss man das Design direkt im Code vornehmen. Zudem ist darauf zu achten, das sich das Panel (bzw. das Fenster) in der Größer ändern kann. Nach Möglichkeit sollte das Interface damit umgehen können (z.B. durch benutzen von Dock und Anchor).

ShowFile

Diese Methode bekommt als Parameter den Pfad der anzuzeigenden Datei, und übernimmt das wichtigste: Die Datei zu laden und irgendwie in den zuvor angelegten Steuerelementen anzuzeigen!

Ich werde jetzt nicht weiter auf den Code eingehen, da dieses Tutorial mehr zeigen soll wie man **generell** Plugins erstellt. Wir fügen also folgenden Code in die ShowFile-Methode ein:

```
ZipFile file = new ZipFile(path);

List<string> dirs = new List<string>();
List<string> files = new List<string>();

foreach(ZipEntry entry in file)
{
        if (entry.IsDirectory)
            dirs.Add(entry.Name);
        else if (entry.IsFile)
            files.Add(entry.Name);
}

StringBuilder sb = new StringBuilder();
sb.AppendLine("Ordner:\n");

foreach (string dir in dirs)
   sb.AppendLine(dir);

sb.AppendLine("Dateien:\n");

foreach (string filenmame in files)
   sb.AppendLine(filename);

textBox1.Text = sb.ToString();
textBox1.BringToFront();

return true;
```

Dieser Code ist jetzt nicht besonders ausgereift, aber er tut was er soll. Wenn hier Fehler auftreten, sollte sie natürlich abgefangen werden, und dann statt der Datei eine Fehlermeldung in dem Ausgabe Fenster angezeigt werden. Wenn eine Ausnahme das Plugin verlässt, wird sie von FastView abgefangen, und eine generelle Fehlermeldung angezeigt.

Wichtig ist auch die vorletzte Zeile: Da das ContetPanel auch alle anderen Steuerelemente der anderen Plugins enthält, ist es sehr wichtig, das/die eigenen nach vorne zu bringen. Das erledigt praktischer Weise die Methode „BringToFront“ die alle Controls erben.

DisposeHandle & ChangedWindowSize

Diese beiden Methoden brauchen wir nicht, hier muss nur der automatisch generierte Fehler entfernt werden. Trotzdem einige Erklärungen:

ChangedWindowSize wird aufgerufen, wenn das FastView Fenster zwischen Vollbild und normaler Größe wechselt. Hier sollten die Steuerelemente, wenn nötig, auf die neue Größe geändert werden. Da unsere TextBox aber an das ContetPanel angedockt ist, ist hier keine Aktion nötig.

DisposeHandle wird aufgerufen, wenn das FastView Fenster geschlossen wird. Hier sollten alle evtl. noch geöffneten Datei Handles wieder freigegeben werden, damit man die Datei danach auch noch bearbeiten kann.

In unserem Fall geschieht die Freigabe automatisch mit dem Verlassen der ShowFile Methode, da dort das ZipFile-Objekt vom GarbageCollector aufgelöst wird.

Der Test

Bevor wir das Plugin kompilieren können sollten wir noch eine Sache ändern: Momentan wird das Plugin auf einem 64-Bit System auch als 64-Bit Anwendung ausgeführt. Dadurch können aber keine 32-Bit DLLs geladen werden. Das ist bei uns zwar auch nicht der Fall, aber bei anderen könnte das ja durchaus nötig sein.

Um dies nun zu ändern, öffnen wir nun die Projekt-Eigenschaften, und wählen unter „Erstellen“ als Zielplattform „x86“ aus (s. Abb. 5).

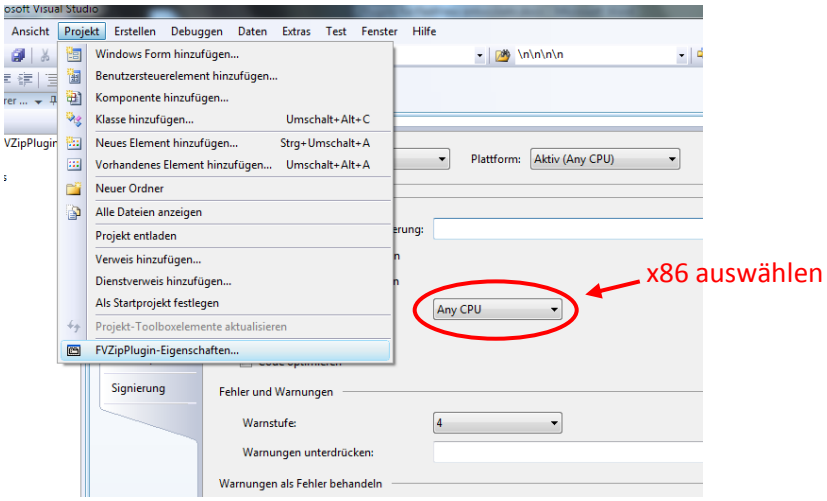
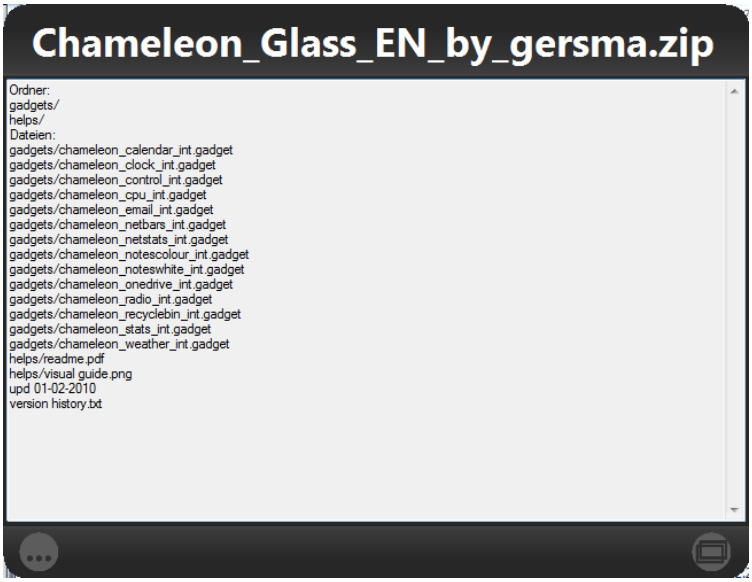


Abbildung 5: Zielpattform auswählen

Nun kann das Programm durch einen Druck auf F6 kompiliert werden. Dadurch erhalten wir im Unterordner „bin/Debug“ eine .dll Datei, die unser Plugin enthält.

Nun muss FastView nur noch dieses Plugin auch laden! Dazu wird die Plugin-DLL in den Unterordner „plugins“ von FastView geladen, und, sehr wichtig, die SharpZipLib.dll Datei in den Hauptordner von FastView. (NICHT in den Plugins-Ordner!)

Wenn alle geklappt hat, sollte man jetzt auch Zip-Dateien mit FastView betrachten können! (s. Abb. 6)



Das komplette Plugins, inkl. Sourcecode ist auch auf meinem Blog zu finden: http://niklas-rother.de/artikel/fastview-plugins-entwicklen

Vielen Dank für das durcharbeiten dieses Tutorials! Anmerkungen, Fragen, usw. bitte direkt auf meinem Blog, unter dem oben angegeben Link.