

Add-in-fähige Anwendungen mit der Bibliothek ICSharpCode.Core

Programmerweiterung vom Baum pflücken

Die Erweiterbarkeit einer Anwendung gehört zu den wichtigsten Kriterien bei der Planung und Entwicklung von Software. Die Bibliothek ICSharpCode.Core, der Kern der bekannten .NET Entwicklungsumgebung SharpDevelop, bietet hierfür eine höchst flexible Basis.

SharpDevelop gibt es bereits seit den Zeiten von .NET 1.x. Damals war diese IDE die einzige kostenlose Alternative, denn von den Visual-Studio-Express-Versionen war weit und breit noch nichts zu sehen. Aber auch heute ist sie durchaus ein ernst zu nehmender Konkurrent, sogar zu den kostenpflichtigen Versionen der Microsoft-Entwicklungsumgebung.

Der Kern von SharpDevelop ist die Bibliothek *ICSharpCode.Core*. Das wichtigste Ziel bei ihrem Entwurf war die Möglichkeit, dass Add-ins andere Add-ins erweitern können.

Außerdem sollen Add-ins mit der Hauptanwendung verschmelzen, anstatt diese nur an bestimmten Punkten zu erweitern. Zu den weiteren Merkmalen der Bibliothek gehören:

- Laden von Add-ins erst dann, wenn sie zum ersten Mal gebraucht werden (Lazy Loading),
- eingebaute Logging-Funktionen mithilfe des Frameworks log4net [1],
- eingebaute Ressourcenverwaltung und Lokalisierung,
- GUI-Unabhängigkeit.

Da die Bibliothek unter der Lizenz LGPL steht, ist ein Einsatz auch in Anwendungen, die nicht öffentlichen Code enthalten, möglich, sofern sie nicht fest in eine Anwendung eingebunden wird.

Voraussetzungen

Dieser Artikel basiert auf Version 3.0 der *Core*-Bibliothek, die Teil von SharpDevelop 3.0 ist. Als IDE für Anwendungen auf Basis des SharpDevelop-Kerns kommt sowohl Visual Studio 2008 als auch SharpDevelop 3.0 in Frage. SharpDevelop bietet IntelliSense für *addin*-Dateien und kann auch *resources*-Dateien bearbeiten; Visual Studio unterstützt nur *resx*-Dateien und verlangt deshalb ein wenig Zusatzarbeit, wenn Sie *ICSharpCode-Add-ins* entwickeln wollen, siehe Kasten *ICSharpCode.Core & Visual Studio*.

Eine auf dem SharpDevelop-Kern basierende Anwendung besitzt eine grundlegende Projektstruktur, die in Abbildung 1 zu sehen ist. Das Projekt *StartUp* ist das Projekt, das der Benutzer ausführt. Es handelt sich dabei aber nicht um die Hauptanwendung, sondern um eine Art Starthil-

Auf einen Blick

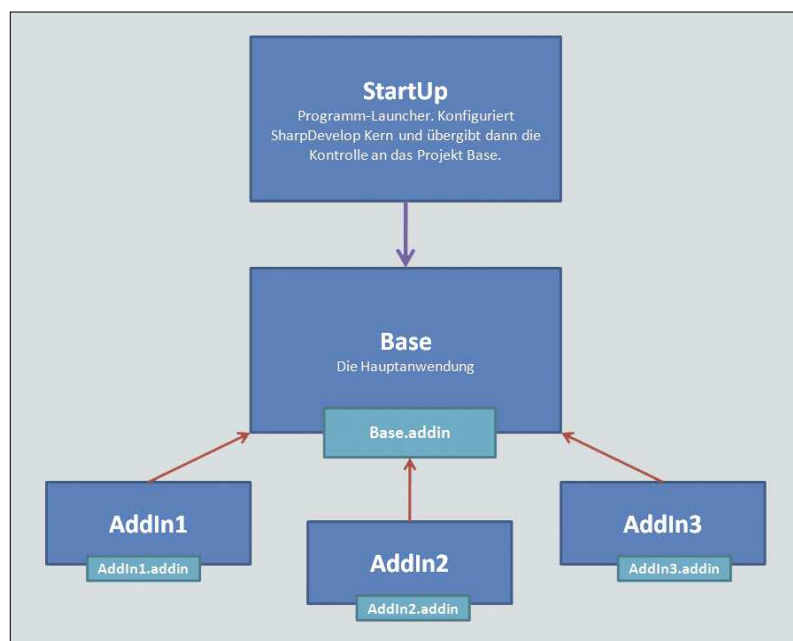


Simon Schweiger lebt in Reutte in Tirol und ist zurzeit Schüler an der Handelsakademie Reutte. Er beschäftigt sich schon seit mehreren Jahren mit der .NET-Technologie und ist über simon.schweiger@gmail.com zu erreichen.

Inhalt

- ➔ Um .NET-Anwendungen Add-in-fähig zu machen, bietet die Bibliothek ICSharpCode.Core aus dem SharpDevelop-Projekt ein komfortables Framework.
- ➔ Die Bibliothek nutzt das RESOURCES-Format und erfordert dafür bei der Arbeit mit Visual Studio einen externen Editor.
- ➔ Die Verknüpfung von Add-in und Hauptprogramm erfolgt über eine XML-Konfigurationsdatei.

dnpCode
A0904ICSharp



[Abb. 1] Die grundlegende Projektstruktur einer auf dem SharpDevelop-Kern aufgebauten Anwendung.

Listing 1

Den Menüpunkt für ein Plug-in in einer XML-Datei definieren.

```
<!-- Diesen Pfad fragen wir in der Hauptanwendung ab. -->
<Path name="/Workbench/MainMenu">
  <!-- Das Label Attribut wird mit lokalisiertem Inhalt aus den Ressourcendateien gefüllt. -->
  <MenuItem
    id="Image"
    type="Menu"
    label="{res:Workbench.MainMenu.Image}"
    insertbefore="Settings">
    <MenuItem
      id="ImageSize"
      label="{res:Workbench.MainMenu.Image.ImageSize}"
      class="ImageTools.ImageSizeCommand" />
    </MenuItem>
  </Path>
```

Listing 2

Das Bild-Menü um einen Eintrag erweitern.

```
<!-- Image Menü um einen Menüeintrag erweitern -->
<Path name="/Workbench/MainMenu/Image">
  <MenuItem
    id="ImageSize"
    label="{res:Workbench.MainMenu.Image.ImageSize}"
  />
</Path>
```

fe, die den SharpDevelop-Kern konfiguriert und danach die Kontrolle an das Hauptprogramm *Base* übergibt, das die eigentliche Anwendung darstellt. Die Trennung zwischen *Startup* und *Base* ist notwendig, da der SharpDevelop-Kern sogar die Hauptanwendung als Add-in betrachtet. Dazu kommen die „richtigen“ Add-ins – in der Abbildung *AddIn1*, *AddIn2* und *AddIn3*.

Das Pfadkonzept

Add-ins bestehen aus mindestens zwei Dateien: dem Programmcode in Form einer *dll*-Datei und einer XML-basierten *addin*-Datei, die den Programmcode mit der Hauptanwendung verknüpft.

Die Verknüpfung des Codes mit der Anwendung erfolgt über sogenannte Pfade, welche die Erweiterungspunkte einer Anwendung darstellen. Listing 1 zeigt einen Ausschnitt aus der Datei *ImageTools.addin*. Sie finden Sie auf der Heft-CD im Beispielprojekt zu diesem Artikel im Ordner *Src\AddIns\ImageTool*. Dieser XML-Abschnitt definiert das *Bild*-Menü und darin einen Menüpunkt. Die *<MenuItem>*-Elemente innerhalb von *<Path>* heißen in der SharpDevelop-Terminologie Codons. Jedes

Codon sollte eine Id in Form eines *id*-Attributs besitzen, damit andere Add-ins darauf zugreifen können. Wie Letzteres funktioniert, zeigt Listing 2. Dort wird das *Bild*-Menü um einen Menüpunkt erweitert.

Die Id *Image* des *Bild*-Menüs aus Listing 1 ist in die Angabe *res:Workbench.MainMenu.Image.ImageSize* gewandert. Die DLL-Dateien mit dem eigentlichen Add-in-Code spielen zu dem Zeitpunkt noch keine Rolle. Sie werden erst geladen, wenn sie zum ersten Mal benötigt werden.

Zu den wichtigsten Konzepten von *ICSharpCode.Core* gehört der Add-in-Tree. Er fasst alle Codons ihrem Pfad entsprechend zu einem Baum zusammen; aus ihm lassen sich die gewünschten Add-ins über den Befehl *AddInTree.Parse()* abrufen. Für den Zugriff auf die Menüleiste stellt der SharpDevelop-Kern einen *MenuService* bereit; so genügt eine Zeile Code, um im Hauptformular die Menüleiste zu erstellen:

```
MenuService.AddItemToMenu(mainMenu.Items,
    this, "/Workbench/MainMenu");
```

XML-Markup erweitern

Um im XML-Markup nicht nur die eingebauten, sondern auch eigene Befehle verwenden zu können, gibt es mehrere Möglichkeiten: Zum einen können Sie innerhalb der *<Path>*-Tags den *Class*-Befehl verwenden. Er ermöglicht es, jede beliebige Klasse im *AddInTree* unterzubringen. Wenn Klassen im Pfad */TestPlugins* das Interface *IPluginTest* implementieren sollen, genügt folgende Zeile Code, um diese auszulisten:

```
foreach (IPluginTest plugin in
    AddInTree.BuildItems<IPluginTest>(
    "/TestPlugins", this)) { }
```

Dabei geschieht Folgendes: Der *AddInTree* durchläuft alle XML-Elemente, die in dem angegebenen Pfad abgelegt wurden, und ruft die dazugehörigen Doozer auf. Ein Doozer ist eine Klasse, die sich um die Umwandlung von Codons in Objekte kümmert. Bei Menüeinträgen erzeugt ein schon eingebauter Doozer GUI-unabhängige *MenuItemDescriptor*-Objekte, bei *Class*-Elementen gibt er eine entsprechende Objektinstanz zurück.

Eine elegantere Methode ist, einen eigenen Doozer zu schreiben. Die Beispielanwendung *AddInSample.sln* zu diesem Artikel implementiert einen Doozer für Toolbox-Einträge (Listing 3).

ICSharpCode.Core und Visual Studio

Wer Anwendungen, die auf dem SharpDevelop-Kern beruhen, mit Visual Studio entwickeln will, stößt schnell auf ein Problem im Zusammenhang mit den Lokalisierungsmerkmalen der Bibliothek: Sie kann nur mit *resources*-Dateien umgehen; diese sind eine binäre Form von *resx*-Dateien. Visual Studio dagegen kann nur *resx*-Dateien bearbeiten.

Aus diesem Grund empfiehlt sich ein externer Ressourcen-Editor wie zum Beispiel Lutz Roeders Resourcer [2]. Dieser lässt sich dann im Projektmappen-Explorer von Visual Studio per Rechtsklick auf eine *resources*-Datei und über den Menüpunkt *Öffnen mit* in die IDE einbinden; mit derselben Methode können Sie auch SharpDevelop einbinden.

Zumindest für die Hauptanwendung genügt es, auch die *resx*-Dateien als eingebettete Resource einzubinden. Für Add-ins ließe sich auch ein Post-Build-Skript verwenden, das mithilfe von *ResGen.exe* die *resx*-Dateien automatisch umwandelt. Die erste Methode ist aber wesentlich komfortabler und weniger fehleranfällig.

Bevor der Doozer im XML-Markup zu verwenden ist, müssen Sie ihn beim SharpDevelop-Kern registrieren. Dies geschieht im Projekt *Startup*. Seine Verwendung im XML-Markup zeigt Listing 4.

Die Anwendung fragt die Toolbox-Einträge mit `AddInTree.BuildItems<ToolBoxItem>` ab.

Lokalisierung und Einstellungen

Im Beispielprojekt *Startup* im Ordner `\Src\Core\AddInSample\` befinden sich die Dateien `StringResources.resources` und `ImageResources.resources`. Sie enthalten die sogenannten neutralen Ressourcen, also die Ressourcen für die Hauptsprache. Lokalisierte Ressourcen lädt der SharpDevelop-Kern automatisch aus dem Unterordner `\data\resources\`.

Um auf die Ressourcen zuzugreifen, stehen die Methoden `GetString()` und `GetBitmapResource()` der Klasse `ResourceService` zur Verfügung.

Für das Speichern und Laden von Einstellungen gibt es ebenfalls ein Hilfsmittel in der `ICSharpCode.Core`-Bibliothek: `PropertyService`. Mithilfe der generischen Methoden `Get()` und `Set()` lassen sich die Einstellungen auslesen oder verändern; ein Aufruf von `Save()` speichert sie.

Die Beispielanwendung demonstriert die hier vorgestellten Merkmale des SharpDevelop-Kerns. Es handelt sich um ein sehr einfaches Zeichenprogramm, das sich um Menüeinträge, Symbole in der Symbolleiste und neue Werkzeuge erweitern lässt. Leider fällt die Dokumentation sehr spärlich aus. Am besten behelfen Sie sich mit einem Blick in den Quellcode der Bibliothek, die glücklicherweise recht klar aufgebaut ist, oder in den Quellcode von SharpDevelop. Dort finden sich auch weitere Beispielapplikationen.

Fazit

Die Bibliothek `ICSharpCode.Core` lohnt sich vor allem dort, wo Add-ins möglichst nahtlos mit der Hauptanwendung verschmelzen sollen. Durch das Pfadkonzept sind Funktionen, die ein Add-in zur Anwendung hinzufügt, nicht von eingebauten Funktionen zu unterscheiden. Der Gesamteindruck wird lediglich durch die praktisch nicht vorhandene Dokumentation getrübt. [jp]

[1] Apache log4net,
<http://logging.apache.org/log4net/>

[2] Lutz Roeder's Programming .NET,
www.lutzroeder.com/dotnet/

Listing 3

Ein eigener Doozer.

```

/// <summary>
/// Ein Doozer ist eine Erweiterung für den SharpDevelop-Kern. Es handelt sich
/// dabei um einen neuen "Baustein" für die addin-Dateien.
/// </summary>
public class ToolBoxItemDoozer : IDoozer
{
    public object BuildItem(object caller, Codon codon, System.Collections.ArrayList subItems)
    {
        // Instanz des ToolBoxItems erstellen
        ToolBoxItem item;
        if (codon.Properties.Contains("class"))
        {
            item = (ToolBoxItem)codon.AddIn.CreateObject(codon.Properties["class"]);
            if (item == null)
                item = new ToolBoxItem();
        }
        else
            item = new ToolBoxItem();

        if (subItems != null && subItems.Count > 0)
        {
            foreach (ToolBoxItem sub in subItems)
            {
                item.SubItems.Add(sub);
            }
        }
        if (codon.Properties.Contains("id"))
            item.Label = StringParser.Parse(codon.Properties["id"]);
        if (codon.Properties.Contains("label"))
            item.Label = StringParser.Parse(codon.Properties["label"]);
        if (codon.Properties.Contains("description"))
            item.Description = StringParser.Parse(codon.Properties["description"]);
        if (codon.Properties.Contains("tooltip"))
            item.ToolTip = StringParser.Parse(codon.Properties["tooltip"]);
        if (codon.Properties.Contains("icon"))
            item.Icon = (Bitmap)ResourceService.GetImageResource(codon.Properties["icon"]);

        return item;
    }

    public bool HandleConditions
    {
        get { return false; }
    }
}

```

Listing 4

Den eigenen Doozer im XML-Markup verwenden.

```

<Path Name="/Workbench/ToolBox">
  <ToolBoxItem
    id="brush" label="{res:AddInSample.Workbench.ToolBox.Brush.Title}"
    tooltip="{res:AddInSample.Workbench.ToolBox.Brush.ToolTip}"
    icon="AddInSample.Workbench.ToolBox.Brush"
    class="AddInSample.Base.Tools.BrushTool"/>
</Path>

```