Plugin-Entwicklung
für
USBSynC 0.3.0

Inhaltsverzeichnis

1. Einleitung

Da sich USBSynC noch in der Entwicklungsphase befindet, kann es vorkommen, dass sich die Schnittstelle ändert. Sollte dies der Fall sein wird dieses Dokument zeitnah aktualisiert werden.

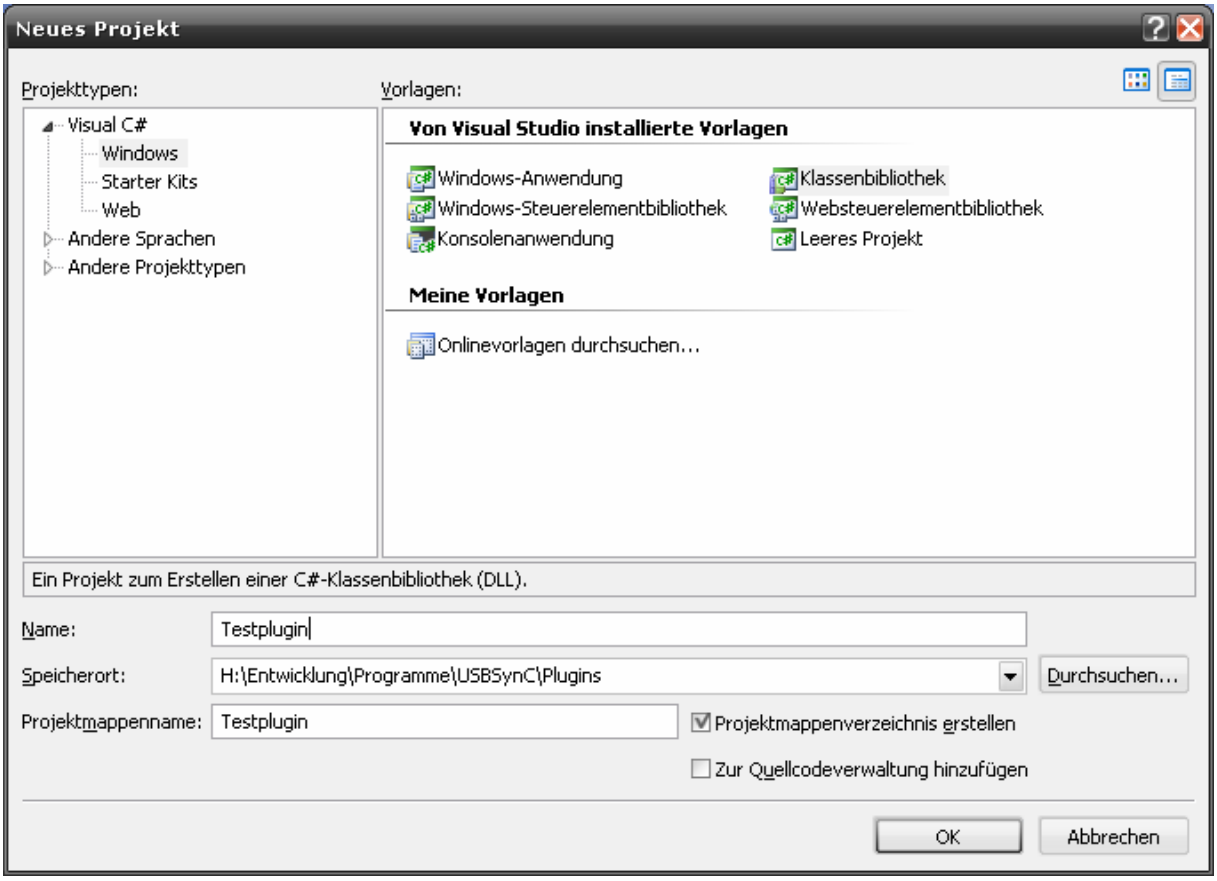
2. Was benötige ich?

Für die Entwicklung ist mit USBSynC bereits alles mitgeliefert, was Sie für die Entwicklung eigener Plugins benötigen. Da USBSynC in .NET 2.0 geschrieben ist, werden Sie eine Entwicklungsumgebung benötigen, die DLL-Dateien in .NET 2.0 kompiliert. Diese Dokumentation zeigt die Entwicklung unter VisualStudio 2005.

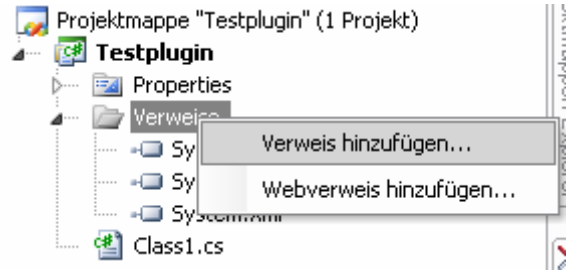
3. Die Entwicklung

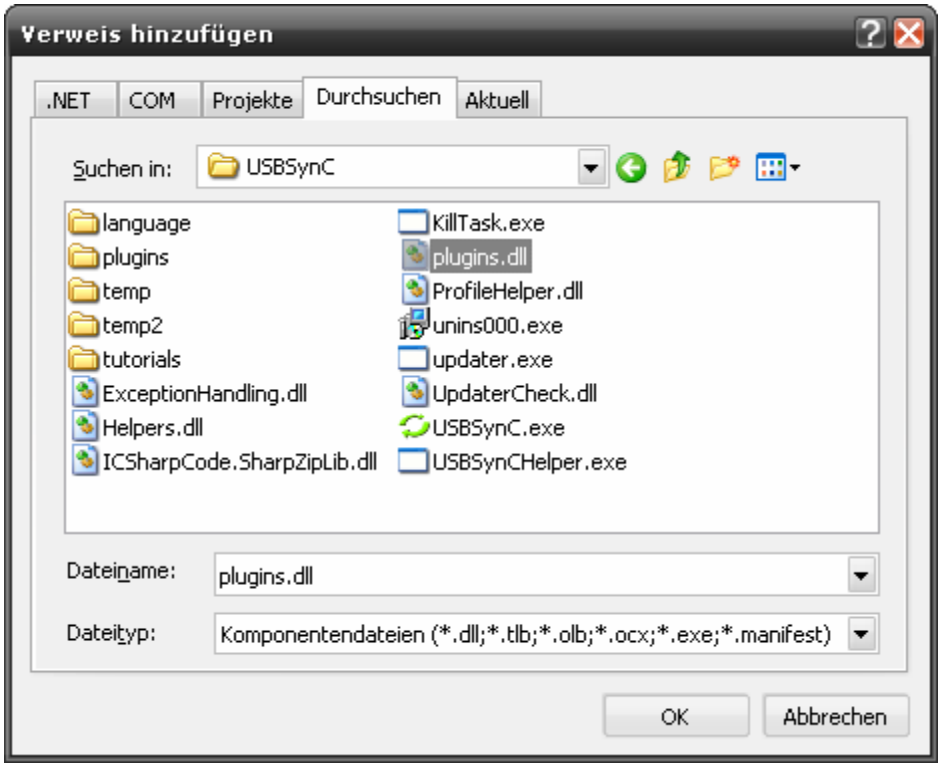
3.1 Neues Projekt

Starten Sie VisualStudio und legen Sie ein neues Projekt (hier in C#) vom Typ **Klassenbibliothek** an und geben Sie dem Projekt einen Namen:

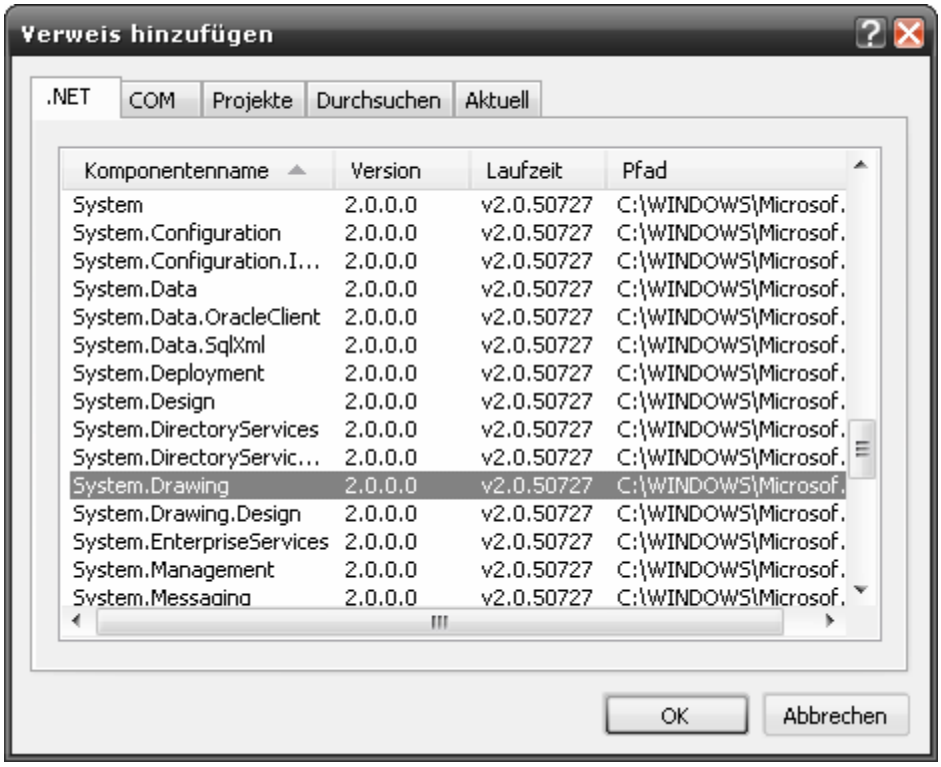


Klicken Sie auf OK und binden Sie im nächsten Schritt unter Verweise in der Projektmappe die **plugins.dll** aus dem USBSynC-Hauptverzeichnis ein:

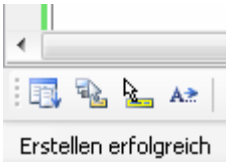



Verweis hinzufügen
.NET
COM
Projekte
Durchsuchen
Aktuell
Suchen in:
USBSynC
language
plugins
temp
temp2
tutorials
ExceptionHandling.dll
Helpers.dll
ICSharpCode.SharpZipLib.dll
KillTask.exe
plugins.dll
ProfileHelper.dll
unins000.exe
updater.exe
UpdaterCheck.dll
USBSynC.exe
USBSynCHelper.exe
Dateiname:
plugins.dll
Dateityp:
Komponentendateien (*.dll;*.tlb;*.olb;*.ocx;*.exe;*.manifest)
OK
Abbrechen

Fügen Sie als nächstes noch den Verweis **System.Drawing** hinzu:



Ersetzen Sie nun den kompletten Inhalt der **Class1.cs** mit dem Inhalt aus der mitgelieferten Datei **Starthilfe.txt**. Das Projekt sollte sich anschließend kompilieren lassen.



3.2 Informationen über das Plugin

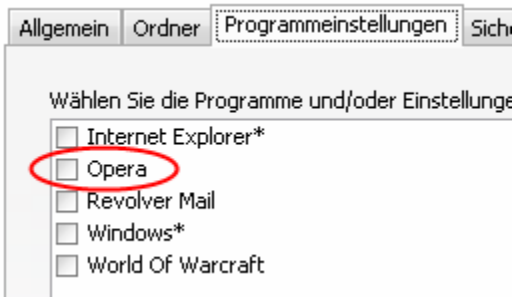
Beginnen Sie nun ein paar Informationen über Ihr Plugin zusammenzutragen. Ganz oben in der Class1.cs stehen ein paar Felder die ausgefüllt werden müssen:

```
string ID = "";
string name = "";
string author = "";
string version = "";
string contact = "";
string url = "";
string[ ] description = { "Zeile1", "Zeile2" };
bool setup = false;
bool singlesetup = true;
```

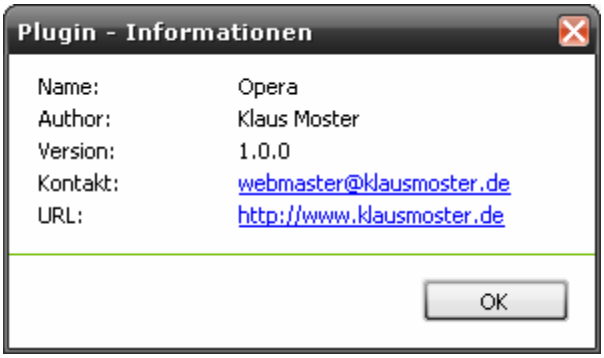
ID: Jedes Plugin erhält eine eindeutige ID. Das kann irgendeine Zeichenkette sein, die möglichst eindeutig ist. Zum Beispiel: Sollten Sie ein Plugin für den Mozilla Firefox verwenden, könnten Sie hier: „mozillafirefox“ eintragen. Sollte Ihnen keine eindeutige ID einfallen, so wenden Sie sich mit einer E-Mail an mich.

USBSynC lädt jede ID nur einmal. Sollten sich im plugin-Ordner zwei Plugins mit identischer ID befinden, so wird nur eins von beiden geladen.

name: Name des Plugins. Dieser Name wird später in der Übersicht angezeigt:



author: Der Name des Autors, also Ihr Name. Dieser Name wird mit anderen Informationen im Info-Dialog angezeigt:

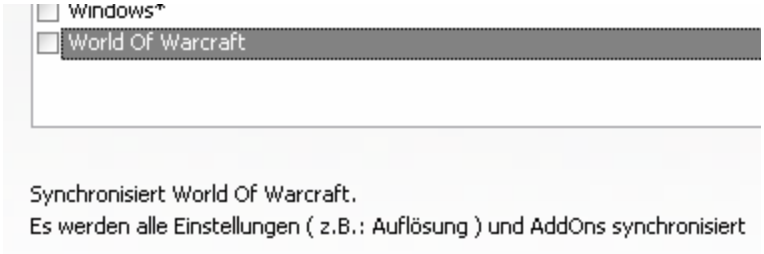


version: Die Versionsnummer des Plugins. Bitte beginnen Sie bei 1.0.0 und erhöhen Sie die Versionsnummer bei jeder öffentlichen Änderung am Plugin. Bei neuen Features ist die zweite Zahl zu erhöhen, bei reinen Bugfixes die dritte Stelle.

contact: Eine E-Mail – Adresse unter der Sie bei Fragen vom Anwender erreichbar sind. Im Zuge der Anwenderfreundlichkeit entwickeln Sie bitte nur dann Plugins, wenn Sie auch bereit sind Fragen zu beantworten und geben Sie hier eine gültige E-Mail – Adresse ein.

url: Ihre Internetadresse, falls vorhanden.

description: Eine kurze Beschreibung zum Plugin. Dies ist ein Stringarray mit maximal 5 Feldern. Pro Feld = eine Zeile. Beschreibung wird unter der Liste angezeigt:



setup: Jedes Plugin kann eine erweiterte Konfiguration benötigen. Das kann den Umfang der Synchronisation beinhalten oder bestimmte Ordnerangaben. Ein Beispiel ist der Einstellungsdialog beim Internet-Explorer Plugin. Ist dieses Feld auf **true** gesetzt, wird USBSynC den Knopf **Konfigurieren** entsperren und zusätzlich ein kleines Sternchen am Namen anzeigen. Auch erwartet USBSynC die Konfiguration des Plugins bevor das Profil abgeschlossen werden

kann. Wie eine Konfiguration realisiert werden kann, wird im Kapitel x.y.z beschrieben.

singlesetup: Es ist manchmal erforderlich, dass ein Plugin auf beiden PCs separat konfiguriert wird, z.B. wenn eine Ordnerangabe benötigt wird, die sich bei beiden PCs unterscheiden kann. In diesem Falle dieses Feld auf **false** setzen. Der Konfigurieren-Dialog wird dann bei beiden PCs angezeigt.

In anderen Fällen reicht die Konfiguration auf dem ersten PC. Der Synchronisationsumfang z.B. beim Internet Explorer muss nur einmal angegeben werden. Denn sollen die Feeds synchronisiert werden, so sollen diese auf beiden PCs synchronisiert werden, sonst würde es kein Sinn machen. In solchen Fällen reicht die Konfiguration auf dem ersten PC und das Feld wird auf **true** gesetzt.

3.3 Synchronisationsablauf

An dieser Stelle ein paar Worte zum Synchronisationsablauf. Wird auf einem PC eine Synchronisation ausgelöst, zunächst „PC -> Stick“, sollen alle Daten vom PC geholt werden und auf den USB-Stick gesichert werden.

Was tut USBSynC nun und was muss das Plugin machen?

Beim Drücken auf „PC -> Stick“ legt USBSynC für jedes Plugin einen temporären Ordner an. Es wird dann die Funktion **GetFiles()** des Plugins gestartet und der Pfad dieses Ordners wird im Parameter **strPath** mitgeliefert. Das Plugin kann nun tun was es will. Alle Dateien die gesichert werden soll können in diesen Ordner kopiert werden. Auch Unterordner können angelegt werden.

Danach packt USBSynC diesen Ordner zusammen. An Kompression brauchen Sie also nicht zu denken, das erledigt USBSynC.

Auf dem zweiten PC, nach Klick auf „Stick -> PC“ werden diese Dateien in einem temporären Ordner zur Verfügung gestellt, in derselben Ordnerstruktur. Die Funktion S**etFiles()** wird aufgerufen und dieser Pfad wird ebenfalls als Parameter **strPath** mitgeliefert. Das Plugin kann diese Dateien nun weiter verarbeiten.

3.4 Init()

Beim Laden der Plugins wird diese Funktion aufgerufen. Sie dient zur Initialisierung des Plugins. USBSynC erwartet als Rückgabewert ein vollständiges XML-Dokument, das mindestens diesen Aufbau hat:

<settings>

</settings>

Dazwischen können – wie auch immer – sämtliche Einstellungen gespeichert werden. Sollte ihr Plugin keine Einstellungen benötigen, belassen Sie das XML-Dokument in der oben genannten Form. In der Datei **Starthilfe.txt** ist diese Form schon angelegt.

3.5 Setup()

Klickt der Anwender auf die Schaltfläche **Konfigurieren**, so wird genau diese Funktion aufgerufen. Der Variable **pluginsettings** wird das XML-Dokument mitgeliefert, das USBSynC durch **Init()** erhalten hat. Das geänderte Dokument erwartet USBSynC als Rückgabewert zurück und liefert es bei jedem Aufruf wieder mit.

In der Setup-Funktion können Sie beliebige Einstellungsdialoge einblenden. Wie so etwas funktionieren könnte, zeigt Kapitel x.y.z.

3.6 isConfigured

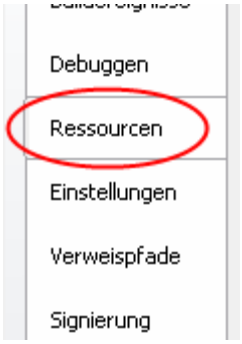
Klickt der Anwender auf **Fertig** und möchte so das Profil abschließend, fragt USBSynC bei jedem Plugin nach ob es korrekt konfiguiert ist. Dieses „Nachfragen“ geschieht durch den Aufruf dieser Methode. Beispiel: Im Internet Explorer-Plugin erwarte ich die Auswahl von mindestens einer Option. Denn es würde kein Sinn machen, wenn der Anwender den Internet Explorer synchronisieren möchte, dann aber keine einzige Option markiert. In der Funktion **isConfigured()** überprüfe ich, ob mindestens eine Auswahl getroffen ist und liefere dementsprechend **true** oder **false**.

Sollte Ihre Plugin auf eine Konfiguration verzichten, so tragen Sie hier bitte immer „return true“ ein.

3.7 getLogo()

Zu jedem Plugin wird ein kleines Logo angezeigt. Durch diese Funktion wird das Logo abgefragt. Wie erreicht man das Einbinden eines Logos?

Klicken Sie auf **Projekt** und **Eigenschaften** und links auf **Ressourcen**:



Erzeugen Sie eine Ressourcendatei, in dem Sie den Hinweis „Dieses Projekt enthält keine Standardressourcendatei. Klicken Sie hier, um eine zu erstellen.“ Anklicken.

Wählen Sie unter **Ressource hinzufügen** den Eintrag **Vorhandene Datei hinzufügen…** und wählen Sie eine vorhandene Bilddatei aus. Die Bildgröße ist grundsätzlich egal, da USBSynC das Logo skaliert.

Geben Sie das Icon in der Funktion mit diesem Aufruf als Image-Objekt zurück:

```
public Image getLogo( )
{                                                    1                    2
    return ( ( System.Drawing.Image )( Testplugin.Properties.Resources.icon ) );
}
```

1. **Testplugin** ist der Name des Projektes, passen Sie diesen Namen an den Namen Ihres Projektes an.
2. **icon** ist der Name meines Bildes in den Ressourcen. Passen Sie diesen Namen an den Namen Ihres Bildes an.

Das Format Ihres Bildes (png, jpg, bmp, …) ist egal, da durch diesen Aufruf das Bild in ein Objekt vom Typ Image umgewandelt wird.

3.8 GetFiles()

Zusammen mit **SetFiles()** die wichtigste Funktion. Sie wird aufgerufen wenn eine Synchronisation durch Klick auf **PC –> Stick** gestartet wird.

Parameter:

SettingsThisPC:	Das Einstellungs-XML-Dokument von dem PC auf dem **PC -> Stick** ausgelöst wurde.
SettingsOtherPC:	Das Einstellungs-XML-Dokument vom jeweils anderen PC.
strPath:	Der Ordner, in den sämtliche Einstellungen gespeichert werden. Dieser Ordner steht auf dem anderen PC zur Verfügung.
strEmpty:	Keine Bedeutung, wird eventuell wegfallen.
swDebug:	Zur Vereinfachung der Entwicklung, wird der StreamWriter swDebug übergeben. In diesen StreamWriter können Zeichenketten zu Debugzwecken geschrieben werden. Aufruf: swDebug.WriteLine(„Eintrag“); swDebug.flush(); Bitte regelmäßig den Schreibpuffer mit flush() leeren, damit im Falle einer Ausnahme keine Daten verloren gehen.

Die Debugdatei trägt den Namen debug_toStick.txt oder debug_toPC.xml und liegt im USBSynC-Hauptordner.

3.9 SetFiles()

SetFiles() ist fast identisch zu **GetFiles()** aufgebaut.

Parameter:

SettingsThisPC:	Das Einstellungs-XML-Dokument von dem PC auf dem **PC -> Stick** ausgelöst wurde.
SettingsOtherPC:	Das Einstellungs-XML-Dokument vom jeweils anderen PC.
strPath:	Der Ordner, in den sämtliche Einstellungen gespeichert wurden. Dieser Ordner entpricht vom Inhalt her, dem Ordner **strPath** der durch **GetFiles()** gefüllt wurde.
strStorePath:	Jedem Plugin wird ein Dauerhafter Ordner im Pfad C:_Programme_USBSynC_plugins_Store zugewiesen. In diesen Pfad können Dateien geschrieben werden, die Dauerhaft gespeichert bleiben. Anwendungsbeispiel: Wird ein Wallpaper synchronisiert, so kann nicht sicher gestellt werden, dass das Bild auf dem anderen PC an derselben Stelle liegt. Somit kann das Bild in den Storepfad gelegt werden und von dort als Wallpaper gesetzt werden.
swDebug:	Zur Vereinfachung der Entwicklung, wird der StreamWriter swDebug übergeben. In diesen StreamWriter können Zeichenketten zu Debugzwecken geschrieben werden. Aufruf: swDebug.WriteLine(„Eintrag“); swDebug.flush();

Bitte regelmäßig den Schreibpuffer mit flush() leeren, damit im Falle einer Ausnahme keine Daten verloren gehen.

4. Kompilieren und Testen

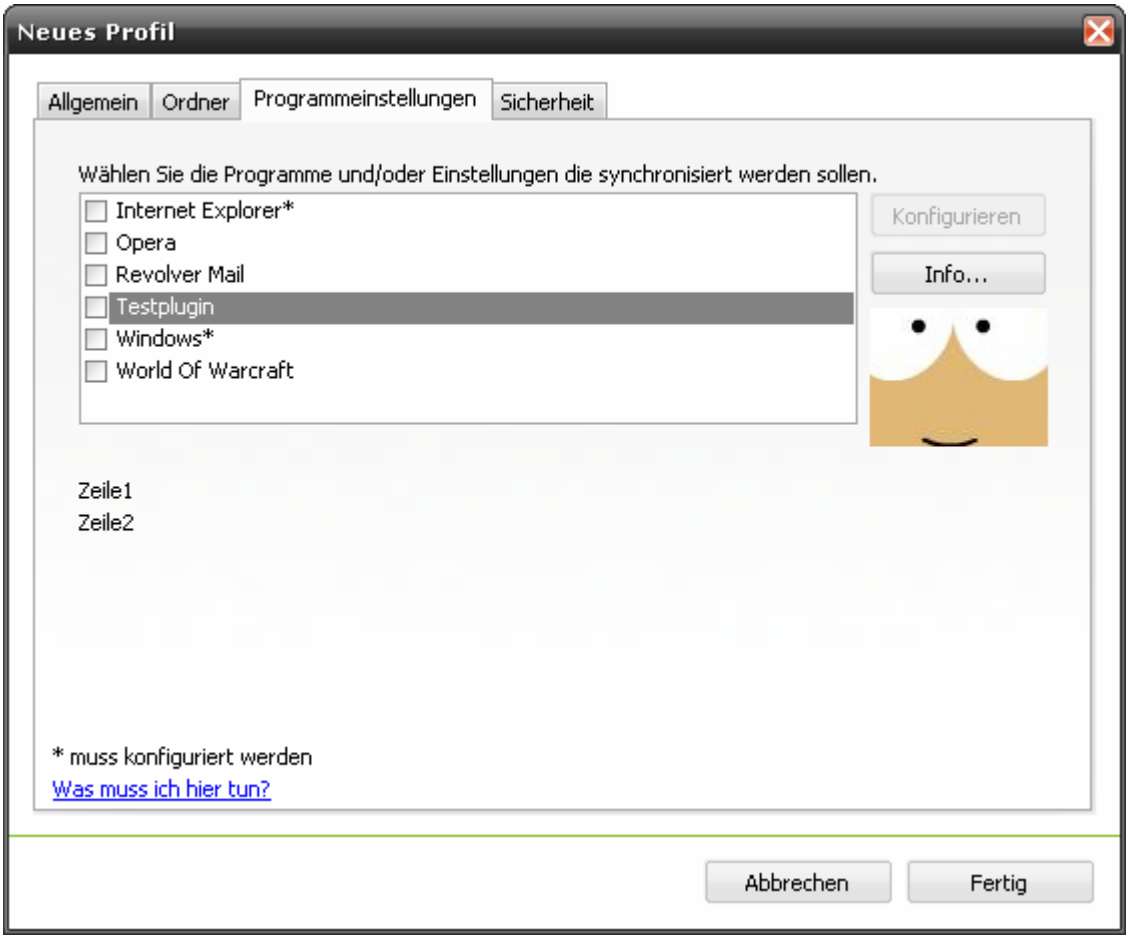
4.1 Kompilieren

Bitte kompilieren Sie das Projekt immer als R**elease**, damit keine Debug-Informationen mehr enthalten sind.

4.2 Testen

Die erzeugte DLL-Datei einfach in den Unterordner **plugins** im USBSynC-Ordner legen und USBSynC starten:

Das Plugin sollte nun in der Liste der Programmeinstellungen erscheinen:



5. Plugin API

5.1 Einleitung

Für die Plugin-Entwicklung werden Funktionen bereit gestellt um die Entwicklung zu vereinfachen.

Um auf diese Funktionen Zugriff zu erhalten, muss ein Objekt der Klasse **PluginHelper** erzeugt werden. Zum Beispiel so:

```
namespace plugins
{
    public class thisplugin : plugins.pluginsBaseInterface
    {
        // Hilfsklasse
        PluginHelper pluginhelper = new PluginHelper( );
```

Im Beispielcode der Datei **Starthilfe.txt** ist dies bereits geschehen und es kann global an jeder Stelle auf **pluginhelper** zugegriffen werden. p**luginhelper** stellt nun eine Reihe von nützlichen Funktionen die im Folgenden beschrieben sind.

5.2 CopyFiles(String strSource, String strDestination)

Kopiert die Datei, angegeben in **strSource**, in den Ordner **strDestination**. Der Ordner muss erzeugt sein.

5.3 CopyFolders(String strSource, String strDestination)

Kopiert den Inhalt inkl. aller Unterordner von **strSource** in den Ordner **strDestination**.

strDestination muss nicht erzeugt sein.

5.4 DeleteFolder(String Path)

Löscht den in **Path** angegeben Ordner inkl. aller Unterordner.

5.5 ExportToXML(String Key, String XMLFile)

Exportiert einen Registryschlüssel in die angegebene XML-Datei. Dabei werden sämtliche Unterschlüssel berücksichtigt.

Aufruf: ExportToXML(„HKCUControlPanelCursors“, „C:key.xml”)

Derzeit warden nur HKCU und HKLM unterstützt.

5.6 ImportFromXML(String XMLFile)

Import eine durch die Funktion E**xportToXML** exportierte Datei.