# Plugin Entwicklung für Mozilla Firefox mit Delphi

Von Padmalcom (2009)
www.jofre.de

Basiert auf der Plugin-Demo von Mike Scott (1996)

## 1 Was ist ein Plugin?

Erstmal möchte ich klarstellen, dass wir in diesem Tutorial ein Plugin schreiben, kein Addon. Der Unterschied ist, dass ein Plugin hauptsächlich unbekannte Dateien verarbeitet, zum Beispiel Div-x Filme oder Java Applets. Addons hingegen fügen einer Anwendung weitere Features hinzu so wie das automatische Blocken von Werbung oder das Verwalten von Passwörtern.
Genug der Theorie, los gehts!

## 2 Was benötige ich?

Es wird eine einfache Delphi-Entwicklungsumgebung benötigt (Ich habe **Turbo Delphi** benutzt da es kostenlos ist) und weiterhin einige Units aus einer Plugin-Demo von Mike Scott. Diese kann hier heruntergeladen werden:

http://ftp.newbielabs.com/Delphi%20Mozilla%20Plugin/SamplePlugin.zip

Aus diesem Archiv müssen folgende Dateien extrahiert und in unserem Projektordner abgelegt werden:

- NPForm.pas
- NPSubCls.pas
- NPPlugin.pas

## 3 Jetzt geht es los!

In Delphi erstellen wir eine neues DLL-Projekt über **Datei → Neu → Weitere...** und im sich nun öffnenden Fenster wählt man **Delphi Projekte → DLL-Experte.** So erhalten wir ein Grundgerüst für eine Dynamic Link Library.

Wir erweitern unsere Uses-Klausel wir in Abbildung 1 gezeigt.

```
uses
  SysUtils,
  Classes,
  NPPlugin in 'NPPlugin.pas',
  NPForm in 'NPForm.pas',
  NPSubCls in 'NPSubCls.pas';
```

**Abbildung 1: Uses-Klausel**

Das sind die drei Files die wir aus dem Archiv extrahiert haben. Sie bilden das Grundgerüst für unser Plugin.

Weiterhin müssen wir noch einige Funktionen exportieren, sodass der Firefox weiß mit welchen Methoden er die Schnittstellen finden und die Initialisierung und das Schließen des Plugins vornehmen kann.

```
exports
  NP_GetEntryPoints index 1,
  NP_Initialize index 2,
  NP_Shutdown index 3;
```

**Abbildung 2: Funktionsübergabe**

Nun erstellen wir die Form die am Ende unser Plugin darstellt. Ein Klick auf **Datei → Neu → VCL-Formular Anwendung** reicht jedoch nicht aus. Die neu generierte Unit muss noch für unsere Zwecke angepasst werden.

```
uses
  Windows, Messages, SysUtils, Variants
  Dialogs,NPForm, StdCtrls;


type
  TForm1 = class(TPluginForm)
    Button1: TButton;
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

initialization
RegisterPluginForm('', TForm1) ;

end.
```

**Abbildung 3: Änderungen an der Formdefinition**

1. Die Unit **NPForm** muss zu den Uses der Form **hinzugefügt** werden.
2. Die Klasse auf die die Form aufsetzt wird von **TForm zu TPluginForm geändert**, diese Klasse implementiert das Handling der Form als Plugin. Wer genauere Informationen wünscht kann gerne ein Blick in die Unit werfen ☺
3. Der **Initialization**-Bereich muss hinzugefügt werden. Dieser registriert unsere Form durch das Plugin-Framwork, sodass dieses automatisch vom Firefox erstellt wird. Der String am Anfang dient dazu beim Aufruf nur ein bestimmtes Plugin zu starten, da es ja prinzipiell möglich ist mehrere Plugins in einer DLL zu haben. Da wir jedoch nur ein Plugin erstellen, kann ein leerer String übergeben werden. Anmerkung: Verschiedene Plugins können im MIME über einen senkrechten Strich definiert werden, z.B. so „Application/x-test-pluginA|Application/x-test-pluginB“.

Nun kann die Form nach Belieben mit Komponenten gefüllt und mit Funktionen versehen werden.

## 4 Die Resourcedatei

Woher weiß der Firefox wer das Plugin erstellt hat und wie es aufgerufen werden kann? Das alles sagt ihm eine *.rc Datei die wie folgt aufgebaut ist:

```
/* Test Res */

1 VERSIONINFO
FILEVERSION 1, 0, 0, 1
PRODUCTVERSION 1, 0, 0, 1
FILEFLAGSMASK VS_FFI_FILEFLAGSMASK
FILEOS VOS__WINDOWS32
FILETYPE VFT_DLL
{
 BLOCK "StringFileInfo"
 {
  BLOCK "040904e4"
  {
   VALUE "CompanyName", "Jofre\000"
   VALUE "FileDescription", "Test Firefox Plugin\000"
   VALUE "FileExtents", "jfp\000"
   VALUE "FileOpenName", "Jofre's plugin file (*.jfp)\000"
   VALUE "FileVersion", "1, 0, 0, 1\000"
   VALUE "InternalName", "Testplugin\000"
   VALUE "LegalCopyright", "Copyright by Jofre \251 2009\000"
   VALUE "MIMEType", "application/x-jofre-plugin\000"
   VALUE "OriginalFilename", "JofrePlugin.dll\000"
   VALUE "ProductName", "Jofre's FF Plugin\000"
   VALUE "ProductVersion", "1, 0, 0, 1\000"
  }
 }

 BLOCK "VarFileInfo"
 {
  VALUE "Translation", 1033, 1252
 }
}
```

**Abbildung 4: Die Res-Datei**

Ich werde nicht auf jeden einzelnen Wert eingehen, da diese weitestgehen selbsterklärend sind und der Rest gegoogelt werden kann ☺
**Wichtig ist jedoch der MIMEType**. Darüber wird definiert wie das Plugin aufgerufen wird. Nennen wir sie wie hier im Beispiel „application/x-jofre-plugin“, dann wird das Plugin später im HTML-Code wir in Abbildung 5 gezeigt aufgerufen:

```
<EMBED TYPE="application/x-jofre-plugin" ALIGN=CENTER WIDTH=500 HEIGHT=500>
```

**Abbildung 5: Einbinden in den HTML-Code**

Haben wir die *.rc erstellt, nennen wir sie wie „Plugin.rc“ und löschen die im Verzeichnis liegende „<Projektname>.res“. Dann suchen wir in **Delphi’s Bin Verzeichnis die Datei brcc32.exe** (Bei liegt diese „C:\Program Files\Borland\BDS\4.0\Bin“), kopieren sie in unser Projektverzeichnis und ziehen die Plugin.rc Datei darauf. Die Exe kompiliert die Datei dann zu einer Plugin.res Datei. Die brcc32.exe und die Plugin.rc kann nun gelöscht werden.

Diese Resourcendatei binden wir jetzt in unser Haputscript ein, direkt **über** den Uses:

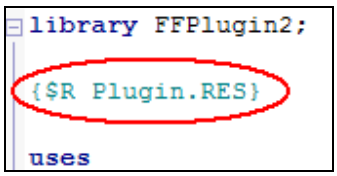


**Abbildung 6: *.res einbinden**

Über **Projekt → Alle Projekte erstellen** compiliert Delphi nun unser Projekt zu einer *.dll. Wir sind fertig!

## 5 Deployment

Um das Plugin zu installieren kopieren wir einfach die DLL in das Plugin-Verzeichnis des Firefox.

**Achtung**: Das Plugin muss mit den beiden Buchstaben „**NP**“ beginnen, sonst wird es vom Firefox nicht erkannt.

Nun kann wie in Abbildung 5 beschrieben das Plugin in den HTML-Code eingebunden werden. Sofern es im richtigen Ordner platziert wurde wird es angezeigt.

Falls alles läuft, herzlichen Glückwunsch! Falls was nicht funktioniert, schaut euch das Beispielprojekt von Mike an!

Viel Spaß!