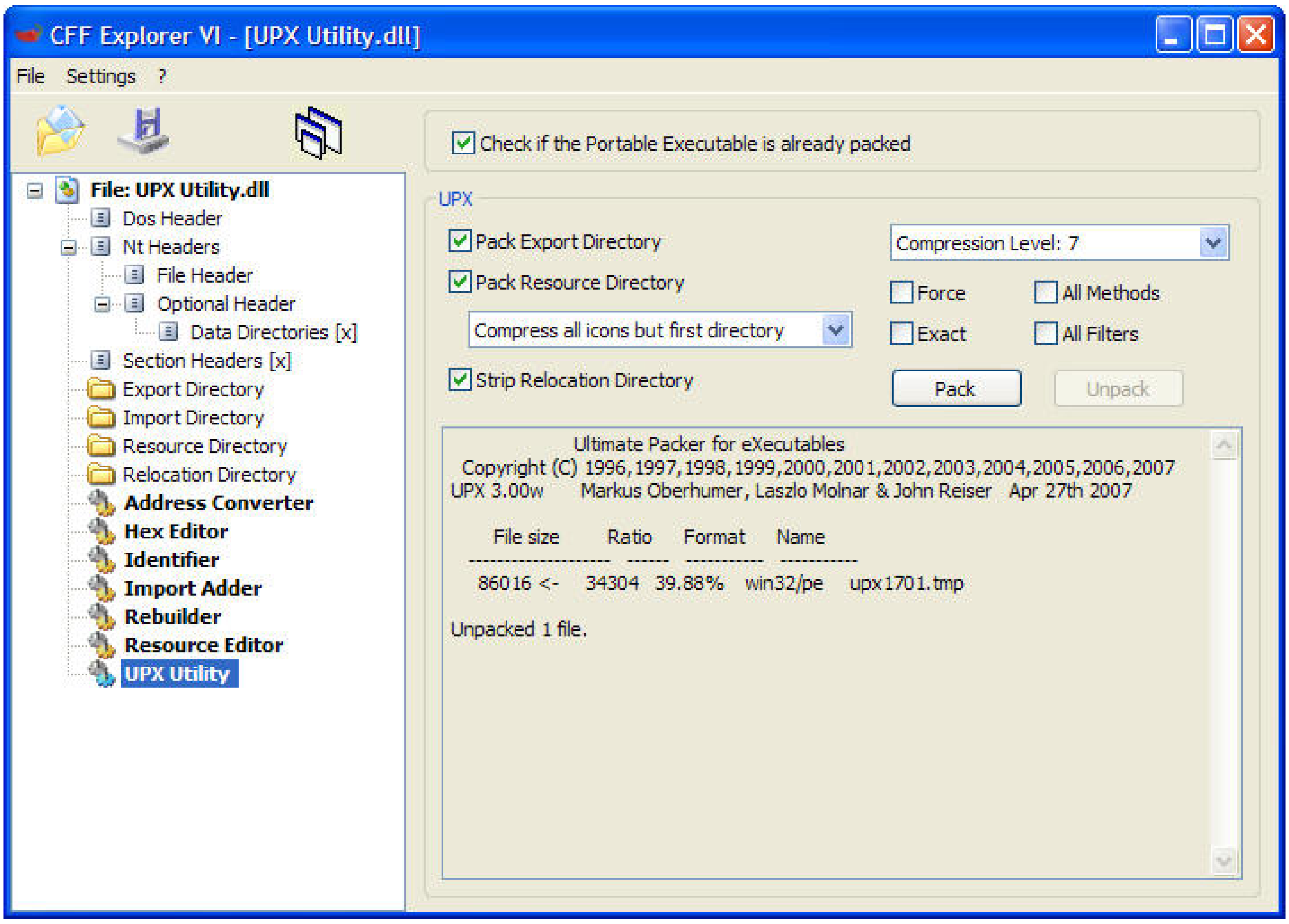
How to write an extension for the CFF Explorer

With the CFF Explorer VI (deployed with the Explorer Suite II) the possibility to write extensions has been introduced. Extension literally extend the functionalities of the CFF Explorer, integrating external software parts in its GUI. I wrote a little upx extension which I'm going to explain in this article.

- UPX Utility source code & compiled files (161 KB)

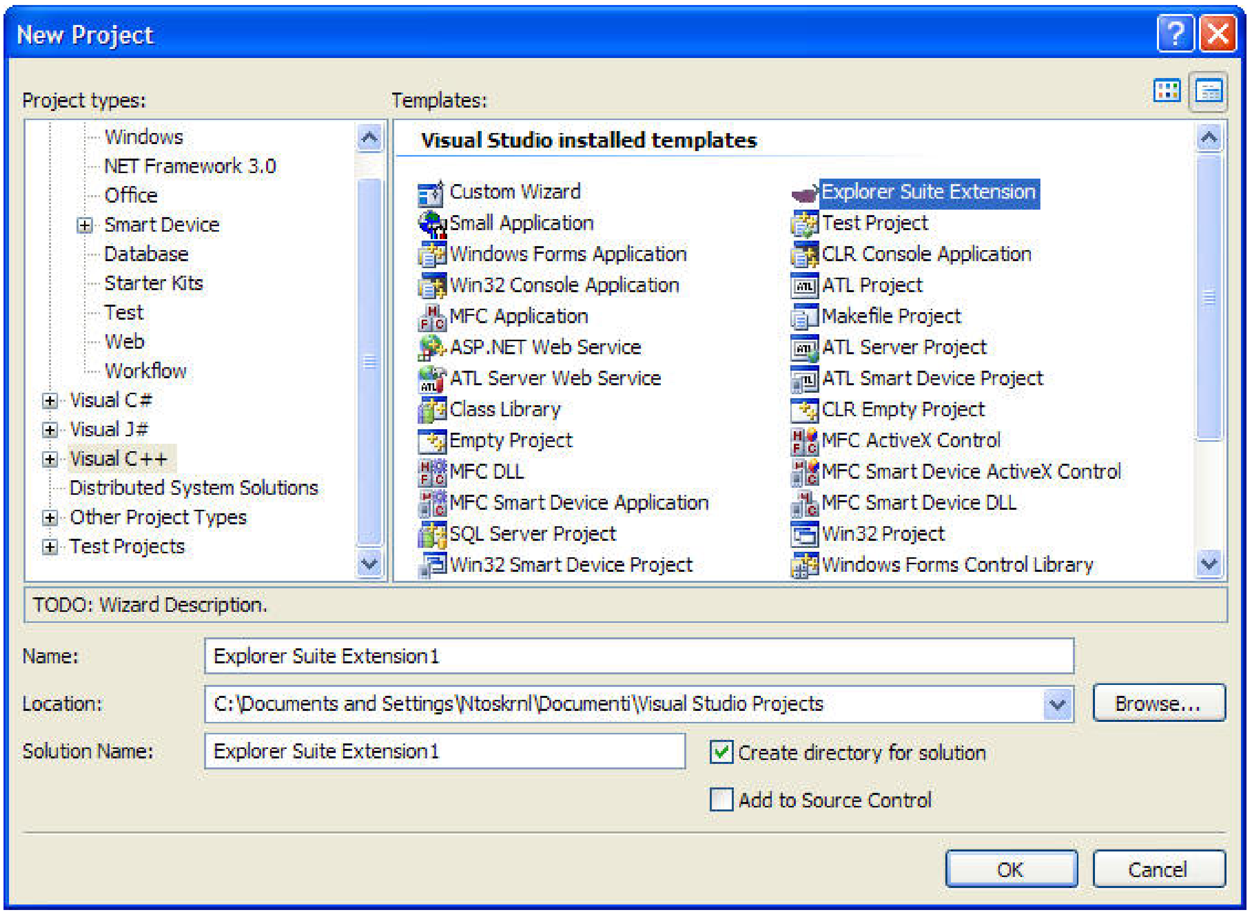


```
                       Ultimate Packer for eXecutables
   Copyright (C) 1996,1997,1998,1999,2000,2001,2002,2003,2004,2005,2006,2007
UPX 3.00w      Markus Oberhumer, Laszlo Molnar & John Reiser   Apr 27th 2007

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     86016 <-     34304   39.88%    win32/pe     upx1701.tmp

Unpacked 1 file.
```

The most professional way to implement extensions would have been through ActiveX. but instead I chose another, less powerful way. In fact, using COM objects is highly professional, but has two big drawbacks: it would take much of my time, and not everybody is capable (or willing) of writing COM objects. I decided to keep it simply, but nothing stands in the way of changing things in the future if necessary. To easily start writing extensions for the CFF Explorer, you should install the *ExSuiteExtWiz* msi file in the SDK directory. It's a wizard built for Visual Studio 2005. After the setup is complete, you can open your Visual Studio, click on *New Project* and then on *Explorer Suite Extension* in the Visual C++ section. As I said, I kept it simple, so we're going to work with pure Win32. The Visual Studio project dialog should be something like this:


New Project

Project types:

- Windows
- NET Framework 3.0
- Office
- Smart Device
- Database
- Starter Kits
- Test
- Web
- Workflow
- Visual C#
- Visual J#
- Visual C++
- Distributed System Solutions
- Other Project Types
- Test Projects

Templates:

Visual Studio installed templates

Custom Wizard	Explorer Suite Extension
Small Application	Test Project
Windows Forms Application	CLR Console Application
Win32 Console Application	ATL Project
MFC Application	Makefile Project
ASP.NET Web Service	ATL Server Project
ATL Server Web Service	ATL Smart Device Project
Class Library	CLR Empty Project
Empty Project	MFC ActiveX Control
MFC DLL	MFC Smart Device ActiveX Control
MFC Smart Device Application	MFC Smart Device DLL
SQL Server Project	Win32 Project
Win32 Smart Device Project	Windows Forms Control Library

TODO: Wizard Description.

Name: Explorer Suite Extension1

Location: C:\Documents and Settings\Ntoskrnl\Documenti\Visual Studio Projects

Solution Name: Explorer Suite Extension1

[✓] Create directory for solution

[ ] Add to Source Control

The wizard generates an extension which shows a dialog with a button. **To activate the extension in the CFF Explorer just put it in the "Extensions\CFF Explorer" directory or in any sub-directory of this path. Of course, the x64 version of the CFF Explorer needs an x64 version of your extension, and the Itanium CFF Explorer an Itanium one.** So, if you want your extension to run on various platforms, you'll have to compile it for them. Fortunately, this is very easy if you follow some basic rules to make your c++ code 64bit compatible. By the way, in this article I'm going to talk about the C++ code generated by the wizard, but nothing can stop you from writing your extension in assembly or whatever language you prefer. But why bother? The C++ code is already there and can be compiled for multiple platforms. So, I don't see the point, but it's really your choice.

The wizard generates the following files:

- ExtensionName.cpp: contains the core implementation of the extension.
- Extension.h: contains general definitions regarding the extensions framework.
- CFFExplorerSDK.h: basically, contains the Extensions API of the CFF Explorer.
- ExtensionName.rc: contains the default dialog.
- resource.h: standard resource definitions.
- stdafx.cpp/h: standard Visual C++ Afx header.

The files we're going to analyze are *ExtensionName.cpp* and *CFFExplorerSDK.h*, which, as I said, contains the CFF Explorer API. The first six functions - apart from the DllMain - we encounter in the cpp file are the extension's core functions:

- ExtensionLoad: called when the extension is being loaded for the first time. It contains the code to retrieve the CFF Explorer API.
- ExtensionUnload: called when the extension is being unloaded. Usually, occurs when the CFF Explorer is being closed. This function is empty for default.

- ExtensionName: called to retrieve the extension's name. If this functions is missing, the file name is used as name for the extension.
- ExtensionDescription: called to retrieve the extension's description. Optional.
- ExtensionNeeded: called to evaluate if the extension is needed for the current file. If this function is missing, it is assumed that the extension is always needed/usable.
- ExtensionExecute: contains the execution code. In our case, it returns the data to create the dialog.

Now, I'll show you every of this functions taken from my UPX Utility, since a real world implementation surely is more useful than the skeleton code generated by the wizard.

The first and very important function is the *ExtensionLoad*. Here's the code:

```
extern "C" __declspec(dllexport) BOOL __cdecl ExtensionLoad(EXTINITDATA
*pExtInitData)
{
   //
   // Retrieves API Interface
   //

   pExtInitData->RetrieveExtensionApi(nCFFApiMask, &CFFApi);

   return TRUE;
}
```

The argument is the data necessary to initialize the extension by retrieving the CFF Explorer API. The *RetrieveExtensionApi* function takes two arguments: the API Mask and the API Structure. Basically, the API Mask is only an array of dwords ending with a null dword. The mask tells the CFF Explorer the functions requested by the extension. The addresses of those functions are put in the API Structure, which contains function pointers ordered accordingly to the API Mask. The declarations of these functions are contained in the CFFExplorerSDK.h. To get a better idea of what I mean, here's the API Mask and API Structure:

```
UINT nCFFApiMask[] =
{
   m_eaGetObjectAddress,
   m_eaGetObjectSize,
   m_eaObjectChanged,
   m_eaReplaceObject,
   m_eaFreeObject,

   m_eaIsPE64,
   m_eaIsRvaValid,
   m_eaRvaToOffset,
   m_eaVaToRva,
   m_eaVaToRva64,
   m_eaVaToOffset,
   m_eaVaToOffset64,
   m_eaOffsetToRva,
   m_eaSectionFromRva,
   m_eaEntryPoint,
   m_eaGetDataDirectory,
   m_eaAddSectionHeader,
   m_eaAddSection,
   m_eaDeleteSectionHeader,
   m_eaDeleteSection,
   m_eaAddDataToLastSection,
   m_eaRebuildImageSize,
   m_eaRebuildPEHeader,
   m_eaRealignPE,
   m_eaRemoveRelocSection,
   m_eaBindImports,
   m_eaRemoveStrongNameSignature,
   m_eaSetImageBase,
   m_eaAfterDumpHeaderFix,

   NULL
};

typedef struct _CFFAPI
{
   d_eaGetObjectAddress eaGetObjectAddress;
   d_eaGetObjectSize eaGetObjectSize;
   d_eaObjectChanged eaObjectChanged;
   d_eaReplaceObject eaReplaceObject;
   d_eaFreeObject eaFreeObject;

   d_eaIsPE64 eaIsPE64;
   d_eaIsRvaValid eaIsRvaValid;
   d_eaRvaToOffset eaRvaToOffset;
   d_eaVaToRva eaVaToRva;
   d_eaVaToRva64 eaVaToRva64;
   d_eaVaToOffset eaVaToOffset;
   d_eaVaToOffset64 eaVaToOffset64;
   d_eaOffsetToRva eaOffsetToRva;
   d_eaSectionFromRva eaSectionFromRva;
   d_eaEntryPoint eaEntryPoint;
   d_eaGetDataDirectory eaGetDataDirectory;
   d_eaAddSectionHeader eaAddSectionHeader;
   d_eaAddSection eaAddSection;
   d_eaDeleteSectionHeader eaDeleteSectionHeader;
   d_eaDeleteSection eaDeleteSection;
   d_eaAddDataToLastSection eaAddDataToLastSection;
   d_eaRebuildImageSize eaRebuildImageSize;
   d_eaRebuildPEHeader eaRebuildPEHeader;
   d_eaRealignPE eaRealignPE;
   d_eaRemoveRelocSection eaRemoveRelocSection;
   d_eaBindImports eaBindImports;
   d_eaRemoveStrongNameSignature eaRemoveStrongNameSignature;
   d_eaSetImageBase eaSetImageBase;
   d_eaAfterDumpHeaderFix eaAfterDumpHeaderFix;

} CFFAPI, *PCFFAPI;

CFFAPI CFFApi;
```

So, after the extension retrieved the API, it can access one of its functions like this: CFF.eaGetObjectAddress(...). This method I implemented seems uselessly complicated, but allows an easy introduction of newer API functions without backwards compatibility issues. I'll discuss later how to use the API functions.

The unload function, as I said, is empty:

```
extern "C" __declspec(dllexport) VOID __cdecl ExtensionUnload()
{
}
```

The function to retrieve the extension's name:

```
extern "C" __declspec(dllexport) WCHAR * __cdecl ExtensionName()
{
   return L"UPX Utility";
}
```

In this case, it's twice as useless, since the file name is exactly the same.

Then comes the function to retrieve the extension's description:

```
extern "C" __declspec(dllexport) WCHAR * __cdecl ExtensionDescription()
{
   return L"A simple GUI for the UPX program.";
}
```

And now, an important function, even if optional. In fact, the *ExtensionNeeded* function should always be implemented, since it makes no sense to load your extension for, let's say, an x64 Portable Executable if it's only capable of processing x86 ones. In this case, UPX can only handle x86 files, which cannot be .NET assemblies. So, the function filters all files which do not respect this criteria:

```
extern "C" __declspec(dllexport) BOOL __cdecl ExtensionNeeded(VOID *pObject,
UINT uSize)
{
   __try
   {
      if (uSize < sizeof (IMAGE_DOS_HEADER) + sizeof (IMAGE_NT_HEADERS))
         return FALSE;
      //
      // Check if it's a 32bit x86 PE file. If it's not, return FALSE
      //

      IMAGE_DOS_HEADER *pDosHeader = (IMAGE_DOS_HEADER *) pObject;

      if (pDosHeader->e_magic != IMAGE_DOS_SIGNATURE) return FALSE;

      IMAGE_NT_HEADERS *pNtHeaders = (IMAGE_NT_HEADERS *) (pDosHeader->e_lfanew
+
         (ULONG_PTR) pDosHeader);

      if (pNtHeaders->Signature != IMAGE_NT_SIGNATURE) return FALSE;

      if (pNtHeaders->FileHeader.Machine != IMAGE_FILE_MACHINE_I386 ||
         pNtHeaders->OptionalHeader.Magic != IMAGE_NT_OPTIONAL_HDR32_MAGIC)
         return FALSE;

      //
      // Check if it's a .NET executable. If so, UPX can't be used
      //

      if (pNtHeaders->OptionalHeader.NumberOfRvaAndSizes <
(IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR + 2))
         return TRUE;

      IMAGE_DATA_DIRECTORY DataDir;

      CFFApi.eaGetDataDirectory(pObject, uSize,
IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR, &DataDir);

      if (DataDir.VirtualAddress != 0 || DataDir.Size != 0) return FALSE;
   }

   __except (EXCEPTION_EXECUTE_HANDLER)
   {
      return FALSE;
   }

   return TRUE;
}
```

The arguments of this function are a pointer to the file and its size. All files in the CFF Explorer are objects, since the software wasn't designed to open only Portable Executables, even if to this point it's the only supported format.

Finally, the *ExtensionExecute* function:

```
EXTEVENTSDATA eed;

extern "C" __declspec(dllexport) VOID *  __cdecl ExtensionExecute(LPARAM lParam)
{
   eed.cbSize = sizeof (EXTEVENTSDATA);

   eed.hInstance = hInstance;
   eed.DlgID = DLG_EXTENSION;
   eed.DlgProc = DlgProc;

   return (VOID *) &eed;
}
```

As mentioned before, this function returns the necessary data to create the extension's GUI. The *DlgProc* is the callback function which receives all the Win32 messages of the created dialog.

The API is divided in two class of functions: the general ones and the format-bound ones. The first class is format-independent. So, they can be used with every format supported by the CFF Explorer. On the other hand, the format-bound class can only be used accordingly to the current object format.

The only functions which, in my opinion, need a bit of explanation are the ones of the first class, since they are less intuitive than the others and more bound to the implementation of extensions in the CFF Explorer. First, let's see the declarations:

```
//
// Syntax: VOID *eaGetObjectAddress(HWND hWnd);
//
// Description: retrieves the current object base address
//

typedef VOID * (__cdecl *d_eaGetObjectAddress)(HWND hWnd);

//
// Syntax: UINT eaGetObjectSize(HWND hWnd);
//
// Description: retrieves the current object size
//

typedef UINT (__cdecl *d_eaGetObjectSize)(HWND hWnd);

//
// Syntax: VOID eaObjectChanged(HWND hWnd);
//
// Description: tells the CFF Explorer that the current object has been modified
//
```

```
typedef VOID (__cdecl *d_eaObjectChanged)(HWND hWnd);

//
// Syntax: BOOL eaReplaceObject(HWND hWnd, VOID *pNewObject, UINT uNewSize);
//
// Description: replaces the current object
//

typedef BOOL (__cdecl *d_eaReplaceObject)(HWND hWnd, VOID *pNewObject, UINT
uNewSize);

//
// Syntax: VOID eaFreeObject(VOID *pObject);
//
// Description: frees memory that has been allocated by others CFF APIs
//

typedef VOID (__cdecl *d_eaFreeObject)(VOID *pObject);
```

As you can see, almost all of these function take an HWND type as first parameter. That's because the CFF Explorer is capable of opening multiple files (and windows) in the same instance, and can only retrieve the current file when it knows which dialog is requesting it. Of course, the internal implementation is completely different, but extensions have to identify themselves through their window handle. Talking in coding terms:

```
LRESULT CALLBACK DlgProc(HWND hDlg, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
   switch (uMsg)
   {

   case WM_INITDIALOG:
      {
         break;
      }

   case WM_COMMAND:
      {
         switch (LOWORD(wParam))
         {

         case IDC_OK:
            {
               VOID *pObject = CFFApi.eaGetObjectAddress(hDlg);
               UINT ObjSize = CFFApi.eaGetObjectSize(hDlg);

               break;
            }
         }
         break;
      }
   }

   return FALSE;
}
```

This means that *eaGetObjectAddress* gets the current object memory address and *eaGetObjectSize* gets its size. I think this should be pretty clear now.

I'm not going to paste the format-bound functions. They're very easy. And keep in mind that these functions may increase significantly.

To conclude this article, here's the code of the main file of the UPX Utility extension. It should be noted that I used external code to redirect the console's output and that I modified parts of it to make it work with TCHARs. You can retrieve the original code on codeproject.

```
#include "stdafx.h"
#include "Extension.h"
#include "CFFExplorerSDK.h"
#include "resource.h"
#include "Redirect.h"

#define WM_EXITTHREAD      (WM_USER + 300)

HINSTANCE hInstance;

BOOL IsAlreadyPacked(VOID *pPE);
VOID EnableControls(HWND hDlg, BOOL bPack, BOOL bUnpack);

LRESULT CALLBACK DlgProc(HWND hDlg, UINT uMsg, WPARAM wParam, LPARAM lParam);

BOOL APIENTRY DllMain(HMODULE hModule, DWORD dwReason, LPVOID lpReserved)
{
   switch (dwReason)
   {
   case DLL_PROCESS_ATTACH:
      {
         hInstance = (HINSTANCE) hModule;
      }

   case DLL_THREAD_ATTACH:
   case DLL_THREAD_DETACH:
   case DLL_PROCESS_DETACH:
      break;
   }

    return TRUE;
}

UINT nCFFApiMask[] =
{
   m_eaGetObjectAddress,
   m_eaGetObjectSize,
   m_eaObjectChanged,
   m_eaReplaceObject,
   m_eaFreeObject,

   m_eaIsPE64,
   m_eaIsRvaValid,
   m_eaRvaToOffset,
   m_eaVaToRva,
   m_eaVaToRva64,
   m_eaVaToOffset,
   m_eaVaToOffset64,
   m_eaOffsetToRva,
   m_eaSectionFromRva,
   m_eaEntryPoint,
   m_eaGetDataDirectory,
   m_eaAddSectionHeader,
   m_eaAddSection,
   m_eaDeleteSectionHeader,
   m_eaDeleteSection,
   m_eaAddDataToLastSection,
   m_eaRebuildImageSize,
   m_eaRebuildPEHeader,
   m_eaRealignPE,
   m_eaRemoveRelocSection,
   m_eaBindImports,
   m_eaRemoveStrongNameSignature,
   m_eaSetImageBase,
   m_eaAfterDumpHeaderFix,

   NULL
};

typedef struct _CFFAPI
{
   d_eaGetObjectAddress eaGetObjectAddress;
   d_eaGetObjectSize eaGetObjectSize;
   d_eaObjectChanged eaObjectChanged;
   d_eaReplaceObject eaReplaceObject;
   d_eaFreeObject eaFreeObject;

   d_eaIsPE64 eaIsPE64;
   d_eaIsRvaValid eaIsRvaValid;
   d_eaRvaToOffset eaRvaToOffset;
   d_eaVaToRva eaVaToRva;
   d_eaVaToRva64 eaVaToRva64;
   d_eaVaToOffset eaVaToOffset;
   d_eaVaToOffset64 eaVaToOffset64;
   d_eaOffsetToRva eaOffsetToRva;
   d_eaSectionFromRva eaSectionFromRva;
   d_eaEntryPoint eaEntryPoint;
   d_eaGetDataDirectory eaGetDataDirectory;
   d_eaAddSectionHeader eaAddSectionHeader;
   d_eaAddSection eaAddSection;
   d_eaDeleteSectionHeader eaDeleteSectionHeader;
   d_eaDeleteSection eaDeleteSection;
   d_eaAddDataToLastSection eaAddDataToLastSection;
   d_eaRebuildImageSize eaRebuildImageSize;
   d_eaRebuildPEHeader eaRebuildPEHeader;
   d_eaRealignPE eaRealignPE;
   d_eaRemoveRelocSection eaRemoveRelocSection;
   d_eaBindImports eaBindImports;
   d_eaRemoveStrongNameSignature eaRemoveStrongNameSignature;
   d_eaSetImageBase eaSetImageBase;
   d_eaAfterDumpHeaderFix eaAfterDumpHeaderFix;

} CFFAPI, *PCFFAPI;

CFFAPI CFFApi;

extern "C" __declspec(dllexport) BOOL __cdecl ExtensionLoad(EXTINITDATA
*pExtInitData)
{
   //
   // Retrieves API Interface
   //

   pExtInitData->RetrieveExtensionApi(nCFFApiMask, &CFFApi);

   return TRUE;
}

extern "C" __declspec(dllexport) VOID __cdecl ExtensionUnload()
{
}
```

```
//
// If this function is missing, the file name is used as name for the extension
//

extern "C" __declspec(dllexport) WCHAR * __cdecl ExtensionName()
{
   return L"UPX Utility";
}

//
// This function is not necessary
//

extern "C" __declspec(dllexport) WCHAR * __cdecl ExtensionDescription()
{
   return L"A simple GUI for the UPX program.";
}

//
// If this function is missing, it is assumed that the extension
// is always needed/usable
//

extern "C" __declspec(dllexport) BOOL __cdecl ExtensionNeeded(VOID *pObject,
UINT uSize)
{
   __try
   {
      if (uSize < sizeof (IMAGE_DOS_HEADER) + sizeof (IMAGE_NT_HEADERS))
         return FALSE;
      //
      // Check if it's a 32bit x86 PE file. If it's not, return FALSE
      //

      IMAGE_DOS_HEADER *pDosHeader = (IMAGE_DOS_HEADER *) pObject;

      if (pDosHeader->e_magic != IMAGE_DOS_SIGNATURE) return FALSE;

      IMAGE_NT_HEADERS *pNtHeaders = (IMAGE_NT_HEADERS *) (pDosHeader->e_lfanew
+
         (ULONG_PTR) pDosHeader);

      if (pNtHeaders->Signature != IMAGE_NT_SIGNATURE) return FALSE;

      if (pNtHeaders->FileHeader.Machine != IMAGE_FILE_MACHINE_I386 ||
         pNtHeaders->OptionalHeader.Magic != IMAGE_NT_OPTIONAL_HDR32_MAGIC)
         return FALSE;

      //
      // Check if it's a .NET executable. If so, UPX can't be used
      //

      if (pNtHeaders->OptionalHeader.NumberOfRvaAndSizes <
(IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR + 2))
         return TRUE;

      IMAGE_DATA_DIRECTORY DataDir;

      CFFApi.eaGetDataDirectory(pObject, uSize,
IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR, &DataDir);

      if (DataDir.VirtualAddress != 0 || DataDir.Size != 0) return FALSE;
```

```
    }

    __except (EXCEPTION_EXECUTE_HANDLER)
    {
        return FALSE;
    }

    return TRUE;
}

//
// Here is where the extension is executed
//

EXTEVENTSDATA eed;

extern "C" __declspec(dllexport) VOID *  __cdecl ExtensionExecute(LPARAM lParam)
{
    eed.cbSize = sizeof (EXTEVENTSDATA);

    eed.hInstance = hInstance;
    eed.DlgID = DLG_EXTENSION;
    eed.DlgProc = DlgProc;

    return (VOID *) &eed;
}

//
// Here starts the utility code
//

class CConsoleRedir : public CRedirect
{
public:
    HWND hDlg;
    TCHAR TempName[MAX_PATH];
protected:
    void OnChildWrite(UINT OutputID, TCHAR *lpszOutput);
protected:
    virtual void OnChildStarted(TCHAR *lpszCmdLine);
    virtual void OnChildStdOutWrite(TCHAR *lpszBuffer);
    virtual void OnChildStdErrWrite(TCHAR *lpszBuffer);
    virtual void OnChildTerminate();
    virtual void OnChildTerminated();
};

void CConsoleRedir::OnChildStarted(TCHAR *lpszCmdLine)
{
    // useless in this case
}

// Send stdout or stderr text to the display window.

void CConsoleRedir::OnChildWrite(UINT OutputID, TCHAR *lpszOutput)
{
    SendDlgItemMessage(hDlg, ED_RESULT, EM_REPLACESEL, FALSE, (LPARAM)
lpszOutput);
}

// Send stdout text to the display window.

void CConsoleRedir::OnChildStdOutWrite(TCHAR *lpszBuffer)
{
   OnChildWrite(0, lpszBuffer);
}

// Send stderr text to the display window.

void CConsoleRedir::OnChildStdErrWrite(TCHAR *lpszBuffer)
{
   OnChildWrite(1, lpszBuffer);
}

// Child process is terminating

void CConsoleRedir::OnChildTerminate()
{
   PostMessage(hDlg, WM_EXITTHREAD, 0, 0);
}

// Child process is terminated correctly.

void CConsoleRedir::OnChildTerminated()
{
   //
   // The temp file has been processed and can now be read
   //

   HANDLE hTempFile = CreateFile(TempName, GENERIC_READ, FILE_SHARE_READ, NULL,
      OPEN_EXISTING, 0, 0);

   if (hTempFile == INVALID_HANDLE_VALUE) return;

   DWORD FileSize = GetFileSize(hTempFile, NULL);

   BYTE *BaseAddress = (BYTE *) VirtualAlloc(NULL, FileSize,
      MEM_COMMIT, PAGE_READWRITE);

   DWORD BRW;

   ReadFile(hTempFile, BaseAddress, FileSize, &BRW, NULL);

   CloseHandle(hTempFile);

   DeleteFile(TempName);

   //
   // Replace the CFF Object with the processed file
   //

   CFFApi.eaReplaceObject(hDlg, BaseAddress, FileSize);

   VirtualFree(BaseAddress, 0, MEM_RELEASE);

   //
   // Change GUI if necessary
   //

   if (IsDlgButtonChecked(hDlg, CB_CHECKPACK) == BST_CHECKED)
   {
      if (IsAlreadyPacked(CFFApi.eaGetObjectAddress(hDlg)) == TRUE)
         EnableControls(hDlg, FALSE, TRUE);
      else
         EnableControls(hDlg, TRUE, FALSE);
   }
   else
   {
      EnableControls(hDlg, TRUE, TRUE);
   }
}

VOID EnableControls(HWND hDlg, BOOL bPack, BOOL bUnpack)
{
   EnableWindow(GetDlgItem(hDlg, CB_PACKEXPORTS), bPack);
   EnableWindow(GetDlgItem(hDlg, CB_PACKRES), bPack);
   EnableWindow(GetDlgItem(hDlg, CB_ICONS), bPack);
   EnableWindow(GetDlgItem(hDlg, CB_STRIPRELOCS), bPack);
   EnableWindow(GetDlgItem(hDlg, CB_PACKLEVEL), bPack);
   EnableWindow(GetDlgItem(hDlg, CB_FORCE), bPack);
   EnableWindow(GetDlgItem(hDlg, CB_ALLMETHODS), bPack);
   EnableWindow(GetDlgItem(hDlg, CB_EXACT), bPack);
   EnableWindow(GetDlgItem(hDlg, CB_ALLFILTERS), bPack);
   EnableWindow(GetDlgItem(hDlg, IDC_PACK), bPack);

   EnableWindow(GetDlgItem(hDlg, IDC_UNPACK), bUnpack);
}

BOOL IsAlreadyPacked(VOID *pPE)
{
   __try
   {
      IMAGE_DOS_HEADER *pDosHeader = (IMAGE_DOS_HEADER *) pPE;

      if (pDosHeader->e_magic != IMAGE_DOS_SIGNATURE) return FALSE;

      IMAGE_NT_HEADERS *pNtHeaders = (IMAGE_NT_HEADERS *) (pDosHeader->e_lfanew
+
         (ULONG_PTR) pDosHeader);

      if (pNtHeaders->Signature != IMAGE_NT_SIGNATURE) return FALSE;

      if (pNtHeaders->FileHeader.Machine != IMAGE_FILE_MACHINE_I386 ||
         pNtHeaders->OptionalHeader.Magic != IMAGE_NT_OPTIONAL_HDR32_MAGIC)
         return FALSE;

      IMAGE_SECTION_HEADER *pSectionHeader = IMAGE_FIRST_SECTION(pNtHeaders);

      if (memcmp(pSectionHeader->Name, "UPX", 3) != 0) return FALSE;
   }

   __except (EXCEPTION_EXECUTE_HANDLER)
   {
      return FALSE;
   }

   return TRUE;
}

static CConsoleRedir cr;

BOOL RunUpx(HWND hDlg, TCHAR *CmdLine, TCHAR *TempName)
{
   SetDlgItemText(hDlg, ED_RESULT, _T(""));
   SendDlgItemMessage(hDlg, ED_RESULT, EM_SETSEL, 0, 0);

   EnableControls(hDlg, FALSE, FALSE);

   VOID *pObject = CFFApi.eaGetObjectAddress(hDlg);
   UINT ObjSize = CFFApi.eaGetObjectSize(hDlg);
   //
   // Write temp file
   //

   HANDLE hTempFile = CreateFile(TempName, GENERIC_READ | GENERIC_WRITE,
FILE_SHARE_READ, NULL,
      CREATE_ALWAYS, 0, 0);

   if (hTempFile == INVALID_HANDLE_VALUE) return FALSE;

   DWORD BRW;

   WriteFile(hTempFile, pObject, ObjSize, &BRW, NULL);

   CloseHandle(hTempFile);


   //
   // Start UPX
   //

   cr.hDlg = hDlg;
   _tcscpy_s(cr.TempName, MAX_PATH, TempName);

   if (cr.StartChildProcess(CmdLine, FALSE) == FALSE)
   {
      DeleteFile(TempName);
      return FALSE;
   }

   //
   // Wait for upx to process the file
   //

   return TRUE;
}

VOID ScreenToClient(HWND hWnd, RECT *rc)
{
   ScreenToClient(hWnd, (LPPOINT) rc);
   ScreenToClient(hWnd, ((LPPOINT) rc) + 1);
   if (((DWORD)GetWindowLong(hWnd, GWL_EXSTYLE)) & WS_EX_LAYOUTRTL)
   {
      LONG temp = rc->left;
      rc->left = rc->right;
      rc->right = temp;
   }
}

LRESULT CALLBACK DlgProc(HWND hDlg, UINT uMsg, WPARAM wParam, LPARAM lParam)
{

#define CMDLINESIZE      2048

   static TCHAR Buffer1[MAX_PATH];
   static TCHAR Buffer2[MAX_PATH];
   static TCHAR Buffer3[MAX_PATH];
   static TCHAR TempName[MAX_PATH];
   static TCHAR CmdLine[CMDLINESIZE];

   switch (uMsg)
   {
```

```
   case WM_INITDIALOG:
      {
         CheckDlgButton(hDlg, CB_CHECKPACK, BST_CHECKED);

         CheckDlgButton(hDlg, CB_PACKEXPORTS, BST_CHECKED);

         CheckDlgButton(hDlg, CB_PACKRES, BST_CHECKED);

         SendDlgItemMessage(hDlg, CB_ICONS, CB_ADDSTRING, 0, (LPARAM) _T("Don't
compress icons"));
         SendDlgItemMessage(hDlg, CB_ICONS, CB_ADDSTRING, 0, (LPARAM)
_T("Compress all icons but first"));
         SendDlgItemMessage(hDlg, CB_ICONS, CB_ADDSTRING, 0, (LPARAM)
_T("Compress all icons but first directory"));
         SendDlgItemMessage(hDlg, CB_ICONS, CB_ADDSTRING, 0, (LPARAM)
_T("Compress all icons"));
         SendDlgItemMessage(hDlg, CB_ICONS, CB_SETCURSEL, 2, 0);

         CheckDlgButton(hDlg, CB_STRIPRELOCS, BST_CHECKED);

         SendDlgItemMessage(hDlg, CB_PACKLEVEL, CB_ADDSTRING, 0, (LPARAM)
_T("Compression Level: 1"));
         SendDlgItemMessage(hDlg, CB_PACKLEVEL, CB_ADDSTRING, 0, (LPARAM)
_T("Compression Level: 2"));
         SendDlgItemMessage(hDlg, CB_PACKLEVEL, CB_ADDSTRING, 0, (LPARAM)
_T("Compression Level: 3"));
         SendDlgItemMessage(hDlg, CB_PACKLEVEL, CB_ADDSTRING, 0, (LPARAM)
_T("Compression Level: 4"));
         SendDlgItemMessage(hDlg, CB_PACKLEVEL, CB_ADDSTRING, 0, (LPARAM)
_T("Compression Level: 5"));
         SendDlgItemMessage(hDlg, CB_PACKLEVEL, CB_ADDSTRING, 0, (LPARAM)
_T("Compression Level: 6"));
         SendDlgItemMessage(hDlg, CB_PACKLEVEL, CB_ADDSTRING, 0, (LPARAM)
_T("Compression Level: 7"));
         SendDlgItemMessage(hDlg, CB_PACKLEVEL, CB_ADDSTRING, 0, (LPARAM)
_T("Compression Level: 8"));
         SendDlgItemMessage(hDlg, CB_PACKLEVEL, CB_ADDSTRING, 0, (LPARAM)
_T("Compression Level: 9"));
         SendDlgItemMessage(hDlg, CB_PACKLEVEL, CB_ADDSTRING, 0, (LPARAM)
_T("Compression Level: Best"));
         SendDlgItemMessage(hDlg, CB_PACKLEVEL, CB_ADDSTRING, 0, (LPARAM)
_T("Compression Level: Brute"));
         SendDlgItemMessage(hDlg, CB_PACKLEVEL, CB_ADDSTRING, 0, (LPARAM)
_T("Compression Level: Ultra-Brute"));
         SendDlgItemMessage(hDlg, CB_PACKLEVEL, CB_SETCURSEL, 6, 0);


         if (IsAlreadyPacked(CFFApi.eaGetObjectAddress(hDlg)) == TRUE)
            EnableControls(hDlg, FALSE, TRUE);
         else
            EnableControls(hDlg, TRUE, FALSE);

         break;
      }

   case WM_SIZE:
      {
         RECT rcPackButton, rcCheckButton, rcCompLevelBox;

         GetWindowRect(GetDlgItem(hDlg, IDC_PACK), &rcPackButton);
         GetWindowRect(GetDlgItem(hDlg, CB_CHECKPACK), &rcCheckButton);
         GetWindowRect(GetDlgItem(hDlg, CB_PACKLEVEL), &rcCompLevelBox);
         ScreenToClient(hDlg, &rcPackButton);
         ScreenToClient(hDlg, &rcCheckButton);
         ScreenToClient(hDlg, &rcCompLevelBox);

         RECT rc, rcp;

         GetClientRect(hDlg, &rc);

         int WLimit = rc.right < rcCompLevelBox.right + 23 ?
rcCompLevelBox.right + 3 :  rc.right - 20;

         MoveWindow(GetDlgItem(hDlg, GB_CHECKPACK), 10, rcCheckButton.top - 17,
WLimit,
            (rcCheckButton.bottom - rcCheckButton.top) + 25, TRUE);

         MoveWindow(GetDlgItem(hDlg, GB_UPX), 10, rcCheckButton.top + 32,
WLimit, rc.bottom -
            ((rcCheckButton.top + 32) + 8), TRUE);

         MoveWindow(GetDlgItem(hDlg, ED_RESULT), 20, rcPackButton.bottom + 10,
WLimit - 20, rc.bottom -
            ((rcPackButton.bottom + 10) + 18), TRUE);

         break;
      }

   case WM_COMMAND:
      {
         switch (LOWORD(wParam))
         {

         case CB_CHECKPACK:
            {
               if (IsDlgButtonChecked(hDlg, CB_CHECKPACK) == BST_CHECKED)
               {
                  if (IsAlreadyPacked(CFFApi.eaGetObjectAddress(hDlg)) == TRUE)
                     EnableControls(hDlg, FALSE, TRUE);
                  else
                     EnableControls(hDlg, TRUE, FALSE);
               }
               else
               {
                  EnableControls(hDlg, TRUE, TRUE);
               }

               break;
            }

         case IDC_UNPACK:
            {
               GetModuleFileName((HMODULE) hInstance, Buffer1, MAX_PATH);
               _tsplitpath_s(Buffer1, Buffer2, MAX_PATH, Buffer3, MAX_PATH,
NULL, 0, NULL, 0);
               _tcscat_s(Buffer2, MAX_PATH, Buffer3);
               _tcscat_s(Buffer2, MAX_PATH, _T("upx.exe"));
               GetShortPathName(Buffer2, CmdLine, MAX_PATH); // upx.exe

               GetTempPath(MAX_PATH, Buffer1);
               GetTempFileName(Buffer1, _T("upx"), 0, Buffer2);
               GetShortPathName(Buffer2, TempName, MAX_PATH); // temp file name

               // add file to unpack
               _tcscat_s(CmdLine, CMDLINESIZE, _T(" -d "));
               _tcscat_s(CmdLine, CMDLINESIZE, TempName);

               RunUpx(hDlg, CmdLine, TempName);

               break;
            }

         case IDC_PACK:
            {
               GetModuleFileName((HMODULE) hInstance, Buffer1, MAX_PATH);
               _tsplitpath_s(Buffer1, Buffer2, MAX_PATH, Buffer3, MAX_PATH,
NULL, 0, NULL, 0);
               _tcscat_s(Buffer2, MAX_PATH, Buffer3);
               _tcscat_s(Buffer2, MAX_PATH, _T("upx.exe"));
               GetShortPathName(Buffer2, CmdLine, MAX_PATH); // upx.exe

               GetTempPath(MAX_PATH, Buffer1);
               GetTempFileName(Buffer1, _T("upx"), 0, Buffer2);
               GetShortPathName(Buffer2, TempName, MAX_PATH); // temp file name

               TCHAR CompLevel[][30] =
               {
                  _T("-1"),            // 0
                  _T("-2"),            // 1
                  _T("-3"),            // 2
                  _T("-4"),            // 3
                  _T("-5"),            // 4
                  _T("-6"),            // 5
                  _T("-7"),            // 6
                  _T("-8"),            // 7
                  _T("-9"),            // 8
                  _T("--best"),        // 9
                  _T("--brute"),       // 11
                  _T("--ultra-brute")  // 12
               };

               _tcscat_s(CmdLine, CMDLINESIZE, _T(" "));
               int nSelItem = (int) SendDlgItemMessage(hDlg, CB_PACKLEVEL,
CB_GETCURSEL, 0, 0);
               _tcscat_s(CmdLine, CMDLINESIZE, CompLevel[nSelItem]);

               // more general

               if (IsDlgButtonChecked(hDlg, CB_FORCE) == BST_CHECKED)
                  _tcscat_s(CmdLine, CMDLINESIZE, _T(" --force"));

               if (IsDlgButtonChecked(hDlg, CB_EXACT) == BST_CHECKED)
                  _tcscat_s(CmdLine, CMDLINESIZE, _T(" --exact"));

               if (IsDlgButtonChecked(hDlg, CB_ALLMETHODS) == BST_CHECKED)
                  _tcscat_s(CmdLine, CMDLINESIZE, _T(" --all-methods"));

               if (IsDlgButtonChecked(hDlg, CB_ALLFILTERS) == BST_CHECKED)
                  _tcscat_s(CmdLine, CMDLINESIZE, _T(" --all-filters"));

               // PE related

               if (IsDlgButtonChecked(hDlg, CB_PACKEXPORTS) == BST_CHECKED)
                  _tcscat_s(CmdLine, CMDLINESIZE, _T(" --compress-exports=1"));
               else
                  _tcscat_s(CmdLine, CMDLINESIZE, _T(" --compress-exports=0"));
               if (IsDlgButtonChecked(hDlg, CB_PACKRES) == BST_CHECKED)
               {
                  TCHAR CompIcon[][40] =
                  {
                     _T("--compress-icons=0"),
                     _T("--compress-icons=1"),
                     _T("--compress-icons=2"),
                     _T("--compress-icons=3"),
                  };

                  _tcscat_s(CmdLine, CMDLINESIZE, _T(" "));
                  nSelItem = (int) SendDlgItemMessage(hDlg, CB_ICONS,
CB_GETCURSEL, 0, 0);
                  _tcscat_s(CmdLine, CMDLINESIZE, CompIcon[nSelItem]);
               }
               else
               {
                  _tcscat_s(CmdLine, CMDLINESIZE, _T(" --compress-
resources=0"));
               }

               if (IsDlgButtonChecked(hDlg, CB_STRIPRELOCS) == BST_CHECKED)
                  _tcscat_s(CmdLine, CMDLINESIZE, _T(" --strip-relocs=1"));
               else
                  _tcscat_s(CmdLine, CMDLINESIZE, _T(" --strip-relocs=0"));

               // add file to pack

               _tcscat_s(CmdLine, CMDLINESIZE, _T(" "));
               _tcscat_s(CmdLine, CMDLINESIZE, TempName);

               RunUpx(hDlg, CmdLine, TempName);

               break;
            }
         }
         break;
      }

   case WM_EXITTHREAD:
      {
         cr.TerminateChildProcess();
         break;
      }
   }

   return FALSE;
}
```

That's all!

Daniel Pistelli