PortaNum , documentation Plugin

version 1.0.0

J. Colineau, 12 décembre 2007

1 Introduction

Ce document spécifie le format des dll utilisables comme "plug-in" pour le programme PortaNum.

2 Principe d'utilisation par PortaNum

L'installation d'un plugin consiste à ajouter une dll (ainsi que d'autres fichiers, éventuellement) au sous-répertoire "\plugins\".

Au démarrage, PortaNum lit le contenu de ce répertoire, charge les dll qu'il trouve, vérifie si elles sont destinées à l'application, et, si oui, ajoute les fonctions de traitement de ces dll aux fonctions de base du programme. Elles apparaissent alors dans le menu Traitements, à la suite des fonctions de base.

Chaque fonction nouvelle à adjoindre à PortaNum fera l'objet d'une dll indépendante.

Lorsqu'un traitement est sélectionné, l'utilisateur peut accéder à la page de propriétés de cette fonction, si elle existe, à l'aide de la fonction F8. Il peut également régler l'amplitude de l'effet de cette fonction, si ce réglage est prévu par le concepteur, en appuyant sur la touche R (pour "réglage") puis sur les flèches UP (^) et DOWN (V), pour modifier ce réglage . La nouvelle valeur du réglage est affichiée de façon temporaire. L'utilisateur peut retrouver le réglage par défaut (valeur 50) à l'aide de la flèche < .

3 Spécifications de la dll

Une dll utilisable avec PortaNum doit comporter au moins les fonctions suivantes:

3.1 PnPluginAppName

Cette fonction doit retourner la chaîne de caractères "PortaNum" (maj. ou minuscules indifféremment),
afin de vérifier que la dll est bien destinée à l'application PortaNum

```
PNPLUGIN_API const char* PnPluginAppName( void )
{
    //version;
    const char * appName;
    appName = "PortaNum";
    return appName;
}
```

3.2 PnPluginInfo

Cette fonction retourne dans une structure comportant plusieurs champs de strings, un certain nombre d'informations qui seront intégrées dans le programme PortaNum.

```
typedef struct
{     const char * version;
      const char * param;
      const char * funcName;
      const char * hint
}  plInfo;

PNPLUGIN_API int PnPluginInfo(plInfo * pInfo )
{
      if (pInfo)
      {
            *pInfo = Info;
            return 0;
      }
      else return 1;
}
```

- version est le numéro de version de la dll (sous la forme d'un string "0.3.0"
- param est le nom du paramètre de réglage (ex: "seuil"), ou de la grandeur de sortie ex: "focus". param peut être une chaîne vide.
- funcName est le nom de la fonction, tel qu'il apparaît dans le menu "Traitements"
- hint est un texte court d'explication du rôle de la fonction

3.3 PnPluginFunction

Cette fonction effectue le traitement demandé. Elle utilise les définitions de la librairie Intel Ipp.
Il n'est pas nécessaire d'utiliser cette librairie. Il faut alors inclure les déclarations de types suivantes:

```
typedef unsigned char   Ipp8u;

typedef struct {
    int width;
    int height;
} IppiSize;

typedef enum {
ippStsErr             =  -2,   /* (erreur)  erreur non spécifiée */
ippStsNoErr           =   0,   /* (OK)  pas d'erreur*/
} IppStatus;
```

Le paramètre mode (valeur par défaut 0) permet de prévoir des variantes de la fonction. Il est utile en particulier pour les phases de test, mais peut être exploité en usage normal, si PortaNum gère l'accès à ce paramètre.
Si la valeur de mode est –1, on utilise le mode défini en interne par la page de configuration, si elle existe. Sinon, c'est le mode 0 qui est appelé.
Le paramètre coef (valeur par défaut 50, plage de 0 à 100) permet à l'utilisateur de gérer de manière interactive l'amplitude de l'effet réalisé. Selon les cas, il sera actif ou non.

```
PNPLUGIN_API IppStatus PnPluginFunction( Ipp8u* pSrc, int srcStep, Ipp8u*
pDst, int dstStep, IppiSize roiSize, int mode, int coef )
{
    IppStatus status = 0;  // 0 = no error
      Ipp8u * pS = pSrc;
      Ipp8u * pD = pDst;
      int w, iy, ix;

      // function implementation ...

      // votre code …

    return status;
}
```

Le code de retour, de type IppStatus, utilise la spécification ipp (voir codes d'erreur dans ippdefs.h)
D'une manière générale, un code négatif signale une erreur, un code nul signale que l'opération s'est déroulée corrrectement, un code positif signale un avertissement.
Par défaut, on utilisera seulement les deux codes suivants :

```
ippStsErr             =  -2,   /* (erreur)  erreur non spécifiée */
ippStsNoErr           =   0,   /* (OK)  pas d'erreur*/
```

on pourra utiliser une valeur entière positive comme code de retour (par exemple pour des plugins réalisant des calculs sur des images, comme le plugin "Focus".

3.4 PnPluginDialog

Cette fonction appelle une page de dialogue. Cette page, facultative, peut être simplement utilisée pour afficher une fenêtre "à propos", ou une aide simplifiée. Elle peut être le point d'entrée d'une interface plus complexe, permettant par exemple le choix et la sauvegarde de différents paramètres de configuration.

```
PNPLUGIN_API IppStatus PnPluginDialog(HINSTANCE hInstance, HWND hwnd)
{

}
```

Cette page de dialogue sert souvent à choisir une ou des options de traitement, que l'on voudra sauvegarder pour les retrouver au prochain lancement du programme. Il est possible alors (mais non obligatoire) d'utiliser pour celà le fichier d'initialisation du programme Portanum: portanum.ini. Voici un exemple de lecture et de sauvegarde dans le fichier .ini d'un paramètre de mode, au lancement de la dll.

```
static int selectedMode = 0;
static char szPath[255];

BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                              )
{
      char strMode[80];
      char *c;
    switch (ul_reason_for_call)
      {
            case DLL_PROCESS_ATTACH:
            case DLL_THREAD_ATTACH:
                  c=szPath+GetModuleFileName(0,szPath,255);
                  while (*c != '\\') c--;
                        *c =0;
                  strcat(szPath,"\\PortaNum.ini");

      GetPrivateProfileString("pluginFaussesCouleurs","LUT","0",strMode,80,
szPath);
                  selectedMode =atoi(strMode);
                  readLUT(selectedMode);
            case DLL_THREAD_DETACH:
            case DLL_PROCESS_DETACH:
                  itoa(selectedMode,strMode,10);

      WritePrivateProfileString("pluginFaussesCouleurs","LUT",strMode,szPat
h);
                  break;
    }
    return TRUE;
}
```

4 Exemple d'utilisation dans un programme

4.1 Déclarations

```
// formats d'appel
typedef struct plInfo
{     const char * version;
      const char * param;
      const char * funcName;
      const char * hint;
} Info;

#define PNPLUGIN_API __declspec(dllimport)
typedef IppStatus (PNPLUGIN_API *plFuncPtr)(Ipp8u* ,int ,Ipp8u* ,int
,IppiSize ,int ,int );
typedef const char* (PNPLUGIN_API  *plAppName)( void );
typedef int (PNPLUGIN_API *plInfoPtr)(plInfo *);
typedef int (PNPLUGIN_API *plDialogPtr)(HINSTANCE, HWND);

plAppName PluginAppName;
plInfoPtr GetPluginInfo = NULL;
```

```
HINSTANCE hPlugin = NULL;
plFuncPtr PlFunc = NULL;
plDialogPtr PlDialog = NULL;

TList *PluginList;

typedef struct Plugin
{
  const char * filename;
  HINSTANCE hPlugin;
  const char * FuncName;
  plFuncPtr Func;
  plInfo info;
} TPlugin;
typedef TPlugin* PPlugin;

PPlugin APlugin;
```

4.2 Chargement des plugins

Extrait de l'application pnPluginTest

```
   // -------------- chargement plugin -----------------

     PluginList = new TList;
     AnsiString PluginPath = GetCurrentDir() + "\\plugins\\";
     AnsiString FullName;

    TSearchRec sr;
    int iAttributes = faAnyFile;
    if ( FindFirst( PluginPath + "*.dll", iAttributes, sr ) == 0 )
    {
        do
        {
            if ( ( sr.Attr & iAttributes ) == sr.Attr )
            {
                APlugin = new TPlugin;
                APlugin->filename = sr.Name.c_str();
                FullName = PluginPath+ APlugin->filename;
                APlugin->hPlugin = LoadLibrary(FullName.c_str());
                if (APlugin->hPlugin)
                {
                  PluginAppName  = (plAppName) GetProcAddress(APlugin-
>hPlugin, "PnPluginAppName");
                  if (PluginAppName)
                  {
                        if (UpperCase(PluginAppName()) == "PORTANUM")
                        {       GetPluginInfo = (plInfoPtr)
GetProcAddress(APlugin->hPlugin, "PnPluginInfo");
                                GetPluginInfo(&APlugin->info);
                               APlugin->Func =(plFuncPtr)
GetProcAddress(APlugin->hPlugin, "PnPluginFunction");
                               RadioGroup1->Items->Add(APlugin-
>info.funcName);
                                PluginList->Add(APlugin);
                        }
                  }
                }
            }
        }
        while ( FindNext( sr ) == 0 );
        FindClose( sr );
    }

    // -----------------------------------------------------
```

4.3 Fonction de traitement

Extrait de l'application pnPluginTest
Ici, le choix du traitement se fait en choisissant un élément d'un "radiogroup"

```
   switch ( RadioGroup1->ItemIndex )
    {
    case 0:
        status = PnScNormal ( pSrc, srcDstStep, pDst, srcDstStep, roiSize,
mode, coef );
        break;
    case 1:
        status = PnScSombre ( pSrc, srcDstStep, pDst, srcDstStep, roiSize,
mode, coef );
        break;
    case 2:
        status = PnScContrasteLumiere ( pSrc, srcDstStep, pDst, srcDstStep,
roiSize, mode, coef );
        break;
    case 3:
        status = PnScTexteImprime ( pSrc, srcDstStep, pDst, srcDstStep,
roiSize, mode, coef );
        break;
    case 4:
        status = PnScTexteManuel ( pSrc, srcDstStep, pDst, srcDstStep,
roiSize, mode, coef );
        break;
    default:      //on n'a pas sélectionné une fonction de base
       APlugin = (TPlugin *) PluginList->Items[ RadioGroup1->ItemIndex -
5];
        lblCoefficient->Caption = APlugin->info.param;
        if (APlugin->Func) {
                 PlFunc = APlugin->Func ;
                status = PlFunc ( pSrc, srcDstStep, pDst, srcDstStep,
roiSize, mode, coef );
        }
        break;
     }
```

4.4 Appel fonction Dialog

```
PlDialog = (plDialogPtr) GetProcAddress(APlugin->hPlugin,
"PnPluginDialog");
// la fonction est optionnelle: tester l'existence
if (PlDialog)
      PlDialog(APlugin->hPlugin, Form1->Handle);
```

4.5 Déchargement des plugins

Ne pas oublier, à la fermeture du programme, de libérer les dll.

Exemple de code de libération des plugins:

```
(Plugin *) APlugin ;
 for (int i =0; i < PluginList->Count ;i++ )
 {
     APlugin = (Plugin *) PluginList->Items[i];
     FreeLibrary(APlugin->hPlugin);
     delete PluginList->Items[i];
 }
delete PluginList;
```

5 Evolutions

Cette version est une première phase. Un certain nombre d'évolutions sont susceptibles de se produire:

- des fonctions complémentaires, optionnelles, pourront être ajoutées.
- on pourra développer des versions plus évoluées de plugins: par exemple pour permettre de définir des scénarios utilisateur. Il suffira de définir des extensions, qui pourront éventuellement être normalisées.

On cherchera cependant à préserver une compatibilité des versions ultérieures avec celle-ci.

6 Exemple de plugins

Voici quelques exemples de plugins qui pourront être réalisés:

- détection et suivi du visage de l'orateur
- aide au réglage de mise au point pour les caméras à réglage de focus manuel (webcam)
- superposition de contours colorés à l'image naturelle (vision "augmentée")
- effets spéciaux (ex: "postérisation")
- colormap
- plugin de personnalisation des traitements de l'image (regroupement de l'ensemble des fonctions de PortaNum ancienne version, avec une fonction d'édition d'une chaîne de traitements personnalisée)

7 Plugins actuellement disponibles (liste non exhaustive)

7.1 pluginExemple2.dll v0.3.0

- inversion d'une partie de l'image
- page de configuration
- mode –1: sélection par page de config
- mode 0 : séparation verticale
- mode 1 : séparation horizontale

7.2 pluginFace.dll v0.3.0

- détection de visages (utilise la librairie OpenCV)
- page de configuration
- mode –1: sélection par page de config
- mode 0 : détection
- mode 1 : zoom sur le premier visage détecté
- possibilité de filtrage de la position

7.3 pluginLUT.dll v0.3.0

- transformation de la luminance de l'image à l'aide d'une LUT couleur
- page de configuration
- 5 LUT disponibles (spectrum, fire, jet, autumn, copper)

7.4 PluginContours.dll v0.3.0

- utilise la librairie OpenCV
- superposition des contours à l'image d'origine
- page de configuration
- réglage du seuil de détection des contours
- choix de la couleur des contours
- possibilité de filtrage passe-bas de l'image d'origine
- possibilité d'affichage des contours seuls

7.5 PluginFocus.dll v0.3.0

- aide au réglage de la mise au point d'une caméra ou webcam
- visualisation d'un indicateur de mise au point

7.6 PluginNagao.dll v0.3.0

- implémentation d'un filtre de Nagao

7.7 PluginPoster.dll v0.3.0

- effet de postérisation par réduction du nombre de couleurs dans l'image
- réglage de l'effet: joue sur le nombre de couleurs retenues
- page de configuration
- possibilité de filtrage passe-bas

D'autres plugins expérimentaux ont été réalisés.

8 Test de dll et Plugins: pnDllTest.exe

Ce programme a été réalisé pour tester la fonctionnalité et les performances de la dll de traitement Portanum et des plugins. Il permet d'intégrer un nouveau plugin, de vérifier qu'il est reconnu par l'application, que l'on peut appeler sa page de configuration, et utiliser son paramètre de réglage. On peut appeler une série d'images de test pour vérifier l'effet du traitement. Il permet aussi de quantifier sa complexité en termes de cycles processeur par pixel d'image, ainsi qu'en temps d'exécution.