SectionPC lib SDK v0.6 10/03/2005

Programmée par Mathias Kende

Table des matières

Introduction

Généralité

La bibliothèque SectionPC est une bibliothèque fournissant un ensemble plus ou moins cohérent de fonctions pour être utilisé en C++.

Cette librairie peut être redistribué gratuitement et utilisée dans n'importe quel programme gratuit ou commercial (j'ai des doute qu'un programme commercial veuille l'utiliser, mais bon...). Dans tout les cas, si vous utilisez cette librairie dans un programme conséquent, ou que l'un de vos programmes repose largement sur cette librairie, je serais heureux de le savoir, et un mail de votre part serait le bienvenu (voir le paragraphe Me contacter).

Vous ne pouvez pas par contre la désassembler et/ou la décompiler. Tout le code compris dans cette librairie est protégé par les lois françaises et internationales en vigueurs sur la propriété intellectuelle.

La librairie utilise le fichier d'en tête"windows.h" et l'inclut donc automatiquement. Par conséquent, si vous voulez éliminez certaines parties de "windows.h", définissez les constantes nécessaires avant d'inclure le fichier "SectionPC.h".

Windows® est une marque déposée de Microsoft Corporation, toutes les marques citées dans ce documents sont des marques de leurs propriétaires respectifs.

But

Cette librairie a pour but d'implémenter des fonctions simples d'une manière simple. Plutôt que d'utiliser les lourds algorithme fournit par la bibliothèque standard, vous pouvez utilisez ceux de cette bibliothèque bien plus simple à utiliser.

De plus, certaines fonctions sont inédites et relativement utile, comme par exemple des fonctions reprenant les services fournis par la bibliothèque conio.h qui n'est plus supporté par la plupart des compilateurs. Dans certain cas, vous pouvez utiliser cette bibliothèque à la place de stdlib.h, iostream et conio.h se qui vous permet d'économisez beaucoup de place pour votre exécutable.

Limitation

La principale limitation de cette librairie est le fait qu'elle ne puisse être utilisée qu'en C++ et non pas en C car les espaces de noms qu'elle utilise sont un ajout du C++.

La version actuelle est compilée avec et pour Visual Studio. Elle est testé sur la version 2002, mais devrait fonctionner avec toute les versions précédentes et suivantes.

La version Dev C++ de la bibliothèque est parfaitement fonctionnelle excepté quelques fonctions qu'il est impossible de compiler avec ce logiciel. Ces fonctions sont donc désactivées dans la version Dev C++ (SectionPC.a).

Je n'ai pas la moindre idée de si cette librairie peut fonctionner ou non avec Borland C++ ou Borland C++ Builder (tout rapport sur ce point est le bien venu, voir la rubrique Me contacter)

Installation

L'installation des fichiers .h .lib et .a est décrit dans le paragraphe Utilisation.

Il y a quelques conditions spéciales d'installations selon les logiciels que vous utilisez :

Utilisation de la fonction GetDir sous Windows 95,98 et ME :

Si vous compter utilisez la fonction GetDir de l'espace de nom win32 vous devez possédez la dll shfolder.dll. Pour installer cette dll vous pouvez exécuter le fichier ShFolder.Exe distribué avec la librairie, ou bien en télécharger la dernière version sur le site

de Microsoft. Si l'une de vos applications utilise la fonction GetDir et est susceptible d'être utilisé sur une machine exécutant un système d'exploitation Windows 9.x (c'est-à-dire 95, 98 ou ME) vos utilisateurs doivent posséder la dll shfolder.dll. Vous pouvez donc redistribuer librement avec votre application l'exécutable ShFolder.Exe sous la condition qu'il ne soit pas modifié et que vous distribuiez avec les 3 fichiers textes dont le nom comment par ShFolder.

Installation avec Dev C++ :

En raison de limitations dans l'implémentation du langage C++ faites par mingw, le compilateur de Dev C++, certaine fonctionnalités sont malheureusement désactivé sous Dev C++ (c'est-à-dire qu'elle n'ont pas été compilé dans SectionPC.a et que leur utilisation est supprimé grâce à des constantes de pré-compilation dans l'en tête SectionPC.h. Néanmoins il existe un problème dans l'un des fichiers d'en tête fournit avec Dev C++ qui empêche de compiler correctement un programme qui inclurait la librairie tel quel. Si vous utilisez la version 4.9.9.0 de Dev C++, vous pouvez remplacer le fichier d'en tête shlobj.h fournit avec le compilateur (qui se trouve dans le dossier include de votre répertoire d'installation) par celui fournit avec la librairie. Si vous posséder une autre version ou que vous ne souhaitez par remplacer le fichier, vous pouvez simplement éditer le fichier shlobj.h fournit avec Dev C++ avec n'importe quel éditeur de texte, puis de rechercher la ligne :

```
#define CSIDL_DESKTOP 0
```

Et de rajouter un peu en dessous les trois lignes manquante :

```
#define CSIDL_MYDOCUMENTS 12
#define CSIDL_MYMUSIC 13
#define CSIDL_MYVIDEO 14
```

La dernière méthode consiste à désactiver l'espace de nom win32. Comme cela est expliqué dans le paragraphe Constantes de pré-compilations.

Utilisation de la fonction ScriptEval :

parfois faut installer msscript.exe, déterminer quand…

Utilisation

Tout le code de la bibliothèque est fournit à l'intérieur d'espace de nom (namespace). Cette méthode a l'immense avantage d'être beaucoup plus simple d'utilisation qu'une méthode orienté objet, et permet toutefois de rester compatible avec n'importe quelle autre bibliothèque existante.

La meilleur méthode pour utiliser cette librairie est de copier le fichier "SectionPC.h" dans le dossier "include" de votre compilateur et de copier le fichier "SectionPC.lib" (pour Visual Studio) ou "SectionPC.a" (pour Dev C++) dans le dossier "lib" de votre compilateur.

Ensuite pour utiliser la librairie il vous suffit sous Visual C++ d'inclure l'en tête de la librairie avec la commande :

```
#include <SectionPC.h>
```

Le fichier contient les commandes nécessaires pour lier la librairie à votre programme.

Si vous souhaitez utiliser les fonctions d'algèbre linéaire, vous devez en plus copier dans le dossier include de votre compilateur, le dossier SectionPC qui contient les fichiers d'en-tête nécessaire. De plus, contrairement au autres partie de la bibliothèque, si vous désirer utiliser cette partie, vous devez définir une constante de pré-compilation (normalement, vous les définissez pour exclure une partie de la bibliothèque) :

```
#define SECTIONPC_USELINEAR
```

Sous Dev C++, en plus d'inclure le fichier d'en tête, vous devez lier manuellement votre programme à la librairie. Pour cela, aller dans les propriétés du projet (alt + P) puis dans l'onglet paramètres (en tout cas dans la version 4.9.9.0) et là, sous la boite de texte "éditeur de liens" cliquez sur le boutons "Ajouter un fichier" puis sélectionnez le fichier SectionPC.a qui doit se trouver dans le dossier lib de votre compilateur.

Sous visual C++ version 2003 (et si vous avez des erreurs de linkage faite cette modification aussi avec les versions précédentes) vous devez dans les propriété de votre projet dans la rubrique C/C++ puis sur la page génération de code, vous devez sélectionner l'option Multithread pour l'option bibliothèque de runtime lorsque vous êtes en configuration release et l'option Debogage multithread lorsque vous êtes en Debug.

Vous pouvez ensuite directement appeler les fonctions de la librairie, par exemple la fonction permettant d'effacer le presse papier peut être appelé ainsi :

```
SectionPC::Clipboard::ClearClipboard();
```

Comme vous le remarquez, la syntaxe est un peu longue. Cela sers au cas ou une autre librairie que vous utiliseriez déclarerait aussi la fonction ClearClipboard() afin qu'il n'y ait pas de conflit entre elle. Mais si vous n'utilisez pas de telle fonction, la longueur de l'appel est plutôt dérangeant. Qu'a cela ne tienne, il y a une solution. Il faut utiliser le mot clef using. Ainsi si vous utilisez le code suivant au début d'un fichier de votre programme :

```
using namespace SectionPC;
```

Alors il suffira dans la suite de votre programme que vous utilisiez la fonction :

```
Clipboard::ClearClipboard();
```

Et même vous pouvez utilisez :

```
using namespace SectionPC::Clipboard;
```

Pour pouvoir alors utiliser la fonction :

```
ClearClipboard();
```

Vous aurez donc compris que cette librairie utilise des espaces de nom (ou namespace) pour stocker ses fonctions. L'espace de nom principal est "SectionPC". Toute les fonctions et données de la librairie sans aucune exception se trouvent à l'intérieur de cet espace de nom.

Mais la librairie est elle-même divisé en un certain nombre d'espace de nom distinct regroupant par thème un ensemble de fonctions.

Chacun des chapitres suivants porte le nom de l'un des espaces de nom de la librairie, et les fonctions données le sont sans leur espace de nom. Vous devrez donc le rajouter ou le déclarer à l'aide du mots clef "using" pour utilisez les fonctions de la librairie.

Constantes de pré-compilations

Blablabla, désactiver ceci cela…

De toute manière, je dois améliorer la manière dont c'est géré dans la lib.

Problèmes connus

Blablabla…

Peut-être que la détection automatique du compilo dans SectionPC.h n'est pas terrible alors il faut modifier ça.

Et puis aussi le problème du débug, que je vais virer tout de suite (ça y est, c'est fait).

Me contacter

Pour tout rapports de bogues, suggestions d'améliorations, questions sur le fonctionnement de la bibliothèque, vous pouvez m'écrire à : sectionpc@gmail.com.

Ascii

Introduction

Cet espace de nom fourni 2 fonctions de manipulation de chaînes de caractères pour convertir des chaînes de caractère en chaîne contenant les codes Ascii correspondants et vice-versa.

```
bool SectionPC::Ascii::AsciiToHex(char*,char*,int*);
bool SectionPC::Ascii::HexToAscii(char*,char*,int*);
```

AsciiToHex

Cette fonction convertie une chaîne de caractères en une autre composée des codes hexadécimaux des caractères de la chaîne passée en arguments.

Utilisation :

```
bool AsciiToHex ( char* InStr,            //Chaîne à convertir.
                  char* OutBuffer,        //Tampon de sortie.
                  int* BufferSize);       //Taille du tampon de sortie.
```

Paramètres :

InStr [in] : C'est un pointeur vers une chaînes de caractères terminée par un dont vous voulez récupérer les codes Ascii.

OutBuffer [out] : Pointeur vers une zone de mémoire suffisamment grande pour obtenir les codes hexadécimaux de la chaînes pointé par InStr.

BufferSize [in/out] : en entier qui doit contenir la taille total de BufferSize. Si la fonction réussie BufferSize contient après son appelle la longueur de la chaîne écrite dans OutBuffer sans compter le zéro final.

Valeur renvoyée :

Si la fonction réussie la valeur renvoyée est true, sinon la fonction renvoie false. La principale cause d'échec de la fonction peut être un tampon trop petit. Celui-ci doit faire au moins deux fois la taille de InStr plus trois octets. Si le tampon est trop petit, la fonction écrit dans OutBuffer (BufferSize-1) caractères puis complète avec un zéro.

Surcharge :

Il existe une version surchargé de la fonction dont le dernier arguments n'est pas un pointeur, mais juste un entier, cette version ne renvoie donc pas la quantité d'octets écrits. Le prototype de cette fonction est :

```
bool AsciiToHex (char* InStr, char* OutBuffer, int BufferSize);
```

HexToAscii

Cette fonction convertie une chaîne de caractères en une autre composée des codes hexadécimaux des caractères de la chaîne passée en arguments.

Utilisation :

```
bool HexToAscii ( char* InStr,            //Chaîne à convertir.
                  char* OutBuffer,        //Tampon de sortie.
                  int* BufferSize);       //Taille du tampon de sortie.
```

Paramètres :

InStr [in] : C'est un pointeur vers une chaînes de caractères terminée par un zéro qui contient uniquement des chiffres et des lettres entre A et F (en majuscule ou minuscule) et dont vous voulez obtenir l'équivalent Ascii.

OutBuffer [out] : Pointeur vers une zone de mémoire suffisamment grande pour obtenir la chaîne InStr convertie en Ascii.

BufferSize [in/out] : en entier qui doit contenir la taille total de BufferSize. Si la fonction réussie BufferSize contient après son appelle la longueur de la chaîne écrite dans OutBuffer sans compter le zéro final.

Valeur renvoyée :

Si la fonction réussie la valeur renvoyée est true, sinon la fonction renvoie false. La principale cause d'échec de la fonction peut être un tampon trop petit. Celui-ci doit faire au moins la moitié de la taille de InStr plus 1 octets. Si le tampon est trop petit, la fonction écrit dans OutBuffer (BufferSize-1) caractères puis complète avec un zéro.

Surcharge :

Il existe une version surchargée de la fonction dont le dernier argument n'est pas un pointeur, mais juste un entier, cette version ne renvoie donc pas la quantité d'octets écrits. Le prototype de cette fonction est :

```
bool HexToAscii (char* InStr, char* OutBuffer, int BufferSize);
```

Hash

Introduction

Cet espace de nom fourni un ensemble de fonction de manipulation de chaînes de caractères et particulièrement, deux fonction de Hash permettant de calculer des valeurs caractéristique pour une chaîne.

LCase
UCase

Ces deux fonctions servent à convertir en majuscules (UCase) et en minuscules (LCase) une chaîne de caractère passé en argument. Toute deux écrivent par-dessus la chaîne qui leur est fournie. Si vous voulez que la fonction crée une nouvelle chaîne, reportez-vous dans les exemples à la fonction LCase_Copy.

Utilisation :

```
char* LCase (char* Str);                    //la chaîne à convertir.
char* LCase (char* Str);                    //la chaîne à convertir.
```

Paramètre :

Str [in/out] : Pointeur vers une chaîne de caractères terminée par un zéro à convertir en majuscules ou minuscules.

Valeur renvoyée :

La fonction renvoie Str, c'est-à-dire que le pointeur renvoyé est le même que celui qui est passé en argument, ceci afin de pouvoir passer la fonction comme argument pour une autre fonction.

Hash

Cette fonction permet de calculer une valeur sur 4 octets caractéristique d'une chaîne de caractères. Cette fonction n'est pas du tout sécurisé et il est tout a fait aisé de trouver deux chaînes différente renvoyant pourtant le même hash. Mais l'utilité d'une telle fonction est de permettre d'utiliser un switch avec une chaîne de caractère. Par exemple pour une interface en ligne de commande, lorsque l'utilisateur tape la commande qu'il veut effectuer, plutôt que de comparer sa commande avec des commandes préenregistré à l'aide de la fonction strcomp qui doit s'utiliser à l'intérieur d'un if, vous pouvez utiliser un switch en comparant le hash de la chaîne avec les hash pré calculé grâce au logiciel HASH_CLI des commandes auxquelles vous vous attendez.

Il faut savoir que le hash s'arrête à la fin de la chaîne, ou au bout de 16 caractères ou lorsqu'il y a un signe de ponctuation. De plus le hash n'est pas sensible au différence entre majuscules et minuscule. La fonction n'est donc pas adaptée à un fichier tout entier, mais seulement à de courtes commandes. Pour information sur la fonction voir le code source de la fonction Hash ainsi que l'exemple d'utilisation. Si vous avez besoin d'une fonction sécurisé et pouvant hacher tout un fichier reporté vous à la fonction md5

Utilisation :

```
long Hash(char* CMD);                       //La chaîne qu'il faut "hacher".
```

Paramètre :

CMD [in] : c'est un pointeur vers la chaîne qui sera "hachée". Il faut faire attention au fait que la fonction modifie cette chaîne car elle est, avant d'être hachée, passée à la fonction

LCase (en réalité une version particulière de LCase qui se limite au 16 premiers caractères) pour que tout soit en minuscule.

Valeur renvoyée :

La fonction renvoie un nombre sur 4 octets (long ou int) qui est le hash de la chaîne qui lui a été passé en argument.

TextXP

Introduction

Cet espace de nom fourni quelques fonctions qui se trouvait auparavant dans la bibliothèque conio.h mais qui n'existe plus maintenant et permettant de formater un petit peu la sortie sur console. Tout cet espace de nom fonctionne autour d'une classe advio (pour advanced input/output). Je cous conseille de déclarer dès le début de vos programme un objet de cette classe tel que :

```
advio console;
```

Que vous pourrez ensuite utilisez à votre convenance.

Comme vous ne pouvez pas pour l'instant rediriger la sortie de cet objet, il est inutile d'en créer plusieurs instances qui seraient toutes identiques. L'objet console est de toute manière globale à tout un projet.

Cette classe advio définit des fonctions de mise en forme (pour l'instant assez limité) et de manipulation de la console qui permettent de dépasser la simple utilisation d'entrée sortie basique fournit par stdio.h et iostream. Néanmoins pour l'entrée sortie elle-même je vous recommande d'utiliser iostream dont les fonctions sont sur et extrêmement optimisé. Si pourtant vous souhaiter économiser la centaine de kilo-octet que coûte le linkage à iostream vous pouvez utiliser les fonctions d'entrée sortie fournit directement par la classe advio. Celles-ci sont moins puissantes que celle de iostream mais tout a fait fonctionnelles.

Pour résumer, toutes les fonctions de cette espace de nom doivent être utilisé au travers de l'objet console et non pas directement ce qui provoquerait une erreur.

Sortie de données

Vous pouvez écrire des données dans la console grâce à la classe advio et particulièrement grâce à l'objet console déjà défini de la même manière qu'avec l'objet cout. La plupart des types de bases sont surchargé ce qui rend pratique son utilisation.

Utilisation :

```
console << var;         //écrit dans la console le contenu de var
console << 45.23;       //écrit le chiffre en question
console << crlf;        //écrit un retour à la ligne
console << var1 << " " << var2 << crlf;  //On peut enchainer les valeurs
```

Paramètre :

L'opérateur << qui est surchargé peut prendre en paramètre les type suivant : char, int, long, unsigned int, unsigned long, float, double et char*. De plus 3 caractère spéciaux sont définit : crlf qui permet de faire un retour à la ligne suivit d'un retour chariot, tab qui fait une tabulation horizontale et bip qui émets un bip sonore.

Valeur renvoyée :

L'opérateur << renvoie un pointeur vers l'objet auquel il s'applique de tel sorte que les expressions puissent être enchaînées comme avec cout. En réalité vous pouvez même mélanger les entrées et les sorties sur la même ligne puisqu'elles s'effectuent avec le même objet.

Entrée de données

Vous pouvez demander à l'utilisateur d'entrer des données dans la console en utilisant la classe advio et particulièrement l'objet console déjà définit sur le même mode qu'avec l'objet cin. Cette fonctionnalité de la classe n'est pas extrêmement robuste. Bien que suffisante pour

un programme lambda, je vous conseille de tester exhaustivement votre programme si vous utiliser cette classe pour lire des données complexes.

Utilisation :

```
console >> var;               //lis des données depuis la console
console >> var1 >> var2;      //lis plusieurs données différentes.
```

Paramètre :

L'opérateur >> peut prendre comme paramètre des variables de type : char, int, long, double ou char* (c'est-à-dire une chaîne de caractère).

Valeur renvoyée :

L'opérateur >> renvoie un pointeur vers l'objet auquel il s'applique de tel sorte que les expressions puissent être enchaînées comme avec cout. En réalité vous pouvez même mélanger les entrées et les sorties sur la même ligne puisqu'elles s'effectuent avec le même objet.

Entrée/Sortie

Comme toute les manipulations se font sur un seul objets, il est tout a fait possible d'enchaîner les entrées et les sorties dans une seule expression. Néanmoins, je vous recommande de ne pas utiliser en entrée et en sortie une même variable dans une même expression car certains compilateurs en optimisant les expression vont effectuer les actions dans un ordre différend de celui prévu par le programmeur ce qui fait qu'une variable ne sera pas initialisé correctement lors de son utilisation. Essayez par exemple avec le code donné ci-dessous et réessayer ensuite en séparant avant l'affichage des données.

Utilisation :

```
char nom[20];
int age;
console << "Comment t'appelle tu ? ">> nom << "Quel age as-tu ? " >> age <<
"Tu as " << age << " ans et tu t'appelle " << nom <<'.' << crlf;
```

gotoxy

Cette fonction est l'équivalent de l'ancien gotoxy de conio.h ou du LOCATE en BASIC. Elle permet de choisir la position du curseur dans la console en nombre de caractère par rapport à l'origine. Le premier caractère en haut à gauche est le caractère (0,0).

Utilisation :

```
bool gotoxy(short int x,      //Abscisse de la position ou placer le curseur
            short int y);     //Ordonnée
console.gotoxy(10,5);         //Utilisation de l'objet console.
```

Paramètres :

x [in] : coordonnée horizontale de la position à laquelle le curseur doit être envoyé.
y [in] : coordonnée verticale de la position à laquelle le curseur doit être envoyé.

Valeur renvoyée :

Si la fonction réussit elle renvoie true, sinon elle renvoie false. La cause la plus commune d'erreur est l'indication d'une coordonné à l'extérieur de la console. Sachez pourtant que la console est bien plus large que la fenêtre qui est visible par l'utilisateur dans la plupart des cas.

TextColor

Comme son nom l'indique cette fonction permet de choisir la couleur du texte ainsi que celle de l'arrière plan.

Lorsque vous appeler la fonction en donnant en paramètre la couleur que vous voulez, vous effacer automatiquement les sélections précédentes. Par défaut le texte est blanc et l'arrière plan est noir. Mais si vous spécifiez une couleur différente pour l'arrière plan, vous effacez ce qui est enregistré pour le texte qui se retrouve donc noir. Pour pallier à cela vous devez donc indiquer systématiquement la couleur du texte à chaque fois que vous modifiez la couleur de l'arrière plan.

Toute les couleurs que vous pouvez utiliser se trouve dans l'enum ADVIO elle-même dans SectionPC::TextXP.

Utilisation :

```
bool SetColor(unsigned short c);                //La fonction elle même
console.SetColor(ADVIO::TEXTE_JAUNE|ADVIO::FOND_BLEU); //Son utilisation
```

Paramètre :

c [in] : ce paramètre représente la couleur que vous voulez utilisez pour le texte ou pour le fond. Si vous voulez modifier la couleur du texte et du fond simultanément, vous devez le faire en une seule instruction. Pour utiliser les couleurs vous devez utiliser la syntaxe tel que montrez dans l'exemple ci-dessus. Vous devez séparer les différente constante (au maximum 2, une pour le texte et une pour le fond) par le caractère | (le ou binaire).

La liste exhaustive des constantes utilisables se trouve dans l'annexe 2.

Valeur renvoyée :

Si la fonction réussit elle renvoie true, sinon elle renvoie false.

SetTitle

Cette fonction permet de modifier le titre de la console.

Utilisation :

```
bool SetTitle(char* Title);
console.SetTitle("HelloWorld");
```

Paramètre :

Title [in] : pointeur vers une chaîne terminée par un zéro. C'est cette chaîne qui est affiché dans la barre de titre de la fonction.

Valeur renvoyée :

Si la fonction réussit elle renvoie true, sinon elle renvoie false.

cls

Cette fonction permet d'effacer la console.

Utilisation :

```
bool cls();
console.cls();
```

Paramètre :

La fonction ne prend pas de paramètre.

Valeur renvoyée :

Si la fonction réussit elle renvoie true, sinon elle renvoie false (en fait, elle ne peut pas rater et renvoie toujours true).

beep

Cette fonction emet un beep sur le haut parleur système.

Utilisation :

```
bool beep();
console.beep();
console << bip;                //Cette syntaxe est absolument équivalente.
```

Paramètre :

La fonction ne prend pas de paramètre.

Valeur renvoyée :

Si la fonction réussit elle renvoie true, sinon elle renvoie false (en fait, elle ne peut pas rater et renvoie toujours true).

pause

Cette fonction met la console et le programme en pause, c'est-à-dire bloque l'exécution jusqu'à ce qu'une touche soit pressée par l'utilisateur.

Utilisation :

```
bool pause(bool verbose=false);
console.pause(true);
```

Paramètre :

verbose [in] : c'est un booléen. S'il vaut false (valeur par défaut) alors la fonction se mettra simplement en pause et attendra que l'utilisateur appuie sur une touche. S'il est mis à true alors la fonction affichera dans la console le message standard : "Appuyez sur une touche pour continuer..."

Valeur renvoyée :

Si la fonction réussit elle renvoie true, sinon elle renvoie false.

SetPos

Cette fonction permet de déplacer la console et de modifier ça taille.

Comme elle n'est pas parfaitement au point et qu'il serait nécessaire qu'elle soit complété par d'autre fonction (récupération de la taille et des coordonnées actuelles par exemple) je ne donne pas sa description pour l'instant. Il suffit néanmoins de savoir que ces paramètres sont des coordonnées en pixel depuis le bords supérieur droit de l'écran.

Annexe 1 : Exemples

Utilisation de l'espace de nom Ascii

Ce programme donne un petit exemple d'utilisation de la bibliothèque utilisant les fonctions de l'espace de nom Ascii.

```
#include <iostream>
#include <SectionPC.h>
using namespace std;
using namespace SectionPC::Ascii;

void main()
{
  char Texte_Ascii[120];
  char Texte_Hexa[255];
  cin >> Texte_Ascii;
  AsciiToHex(Texte_Ascii, Texte_Hexa,255);
  cout << Texte_Hexa << endl;
  HexToAscii(Texte_Hexa, Texte_Ascii,120);
  cout << Texte_Ascii;
  cin.get();cin.get();
}
```

Code source de HASH_CLI

Ce programme sert à obtenir les constantes que renvoie la fonction Hash, avant de compiler. Il s'agit du code source du programme HASH_CLI distribué avec la bibliothèque.

```
#include <SectionPC.h>
using namespace SectionPC::Hash;
using namespace SectionPC::Clipboard;
using namespace SectionPC::TextXP;
advio console;

void main()
{
  int cmd,size;
  char CMD[20];

  do
  {
    console << ">" >> CMD;
    cmd=Hash(CMD);
    size=Str(cmd,CMD,20);
    SetClipboardText(&CMD[CMD[0]==' '?1:0],size);
    cout << cmd << endl;
  }while (cmd!=CMD_QUIT);
}
```

Fonction LCase_Copy

Cette fonction permet de mettre une chaîne en majuscule tout en n'effaçant pas la chaîne passée en argument.

```
using namespace SectionPC::Hash;
long LCase_Copy(char* InStr, char* OutStr, long Size)
{
  long ret=StrCopy(InStr,OutStr,Size);    //Copie la chaine
  LCase(OutStr);                          //Mets la copie en majuscule
  return ret;
  //Retourne la taille de la chaîne OutStr sans le zéro final.
```

```
}
```

Code source de la fonction Hash

Ceci est le code source de la fonction Hash

```
long SectionPC::Hash::Hash(char* CMD)
{
  if (CMD==NULL) return 0;
  lcase(CMD);    //Ce n'est pas exactement la fonction LCase.
  unsigned long CalcHash=0;
  for (int J=0;J<4;++J)
  {
    for (int I=0;I<4;++I)
    {
      if ((*CMD==0)||(*CMD==',')||(*CMD=='=')||
          (*CMD==':')||(*CMD=='.')||(*CMD==' '))
        return CalcHash;
      CalcHash=CalcHash^(int)(*CMD<<(I*8));
      ++CMD;
    }
  }
  return CalcHash;
}
```

Exemple d'utilisation de la fonction Hash

blablabla.

Annexe 2 : Les constantes

Les couleurs définit par SectionPC::TextXP::ADVIO

Les couleurs que vous pouvez utilisez lors de l'appel à la fonction SetColor de la classe advio dans l'espace de nom SectionPC::TextXP sont divisé en deux groupe, celles concernant le texte et celles concernant le fond. Je vous donne ici la liste des constantes définissant la couleurs du texte, celle définissant la couleurs du fond sont les même mais il faut remplacer le mot TEXTE par le mot FOND (qui l'eut cru ?).

```
TEXTE_NOIR,
TEXTE_BLEU,
TEXTE_VERT,
TEXTE_ROUGE,
TEXTE_CYAN,
TEXTE_MAGENTA,
TEXTE_JAUNE,
TEXTE_GRIS,
TEXTE_BLANC,
TEXTE_BLEU_CLAIR,
TEXTE_VERT_CLAIR,
TEXTE_ROUGE_CLAIR,
TEXTE_CYAN_CLAIR,
TEXTE_MAGENTA_CLAIR,
TEXTE_JAUNE_CLAIR,
TEXTE_BLANC_CLAIR
```