OPTIMISATION ET FIABILISATION D'UNITEX

ou du labo à l'iPhone...

FRANÇOIS LIGER

Co-fondateur et CEO

GILLES VOLLANT

Co-fondateur et développeur

ANASTASIA YANNACOPOULOU

Co-fondatrice et linguiste en chef

francois@ergonotics.com • gilles@ergonotics.com • anastasia@ergonotics.com

ERGONOTICS

CONVEX

le convertisseur malin pour iOS

ASSISTANTV

la télécommande intelligente pour Freebox

Le convertisseur ma

UNITEX

+

GROSSE

BASE INSTALLÉE

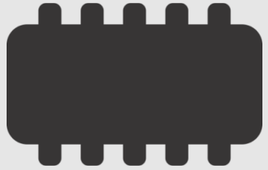
AU MONDE

argements

re (14

DÉFIS

Fiabilité



Mémoire

Performance

Multi-plateforme

MULTIPLATEFORME

amélioration du support multiplateforme

- ★Mobile : iOS, Androïd, Windows RT
- ★Serveur : Linux, Windows, Mac
- ★Support multiple pour les compilateurs : GCC, LLVM, Visual Studio

FIABILITÉ

- Le code était globalement “propre” mais
  - Quelques fuites mémoire
  - Quelques problèmes de multi-threading
- Dur à tester
  - Pas de mécanisme de reproduction systématique

LOGGER

- ★ Enregistre chaque opération and permet de la rejouer
- ★ Aide à la prévention des régressions
  - ★ Chaque modification peut être testée avec tout le jeu de tests pour valider son innocuité
  - ★ Si une modification ou un use case révèle un nouveau bug non détecté par le jeu de test, il suffit de l’ajouter au jeu de tests

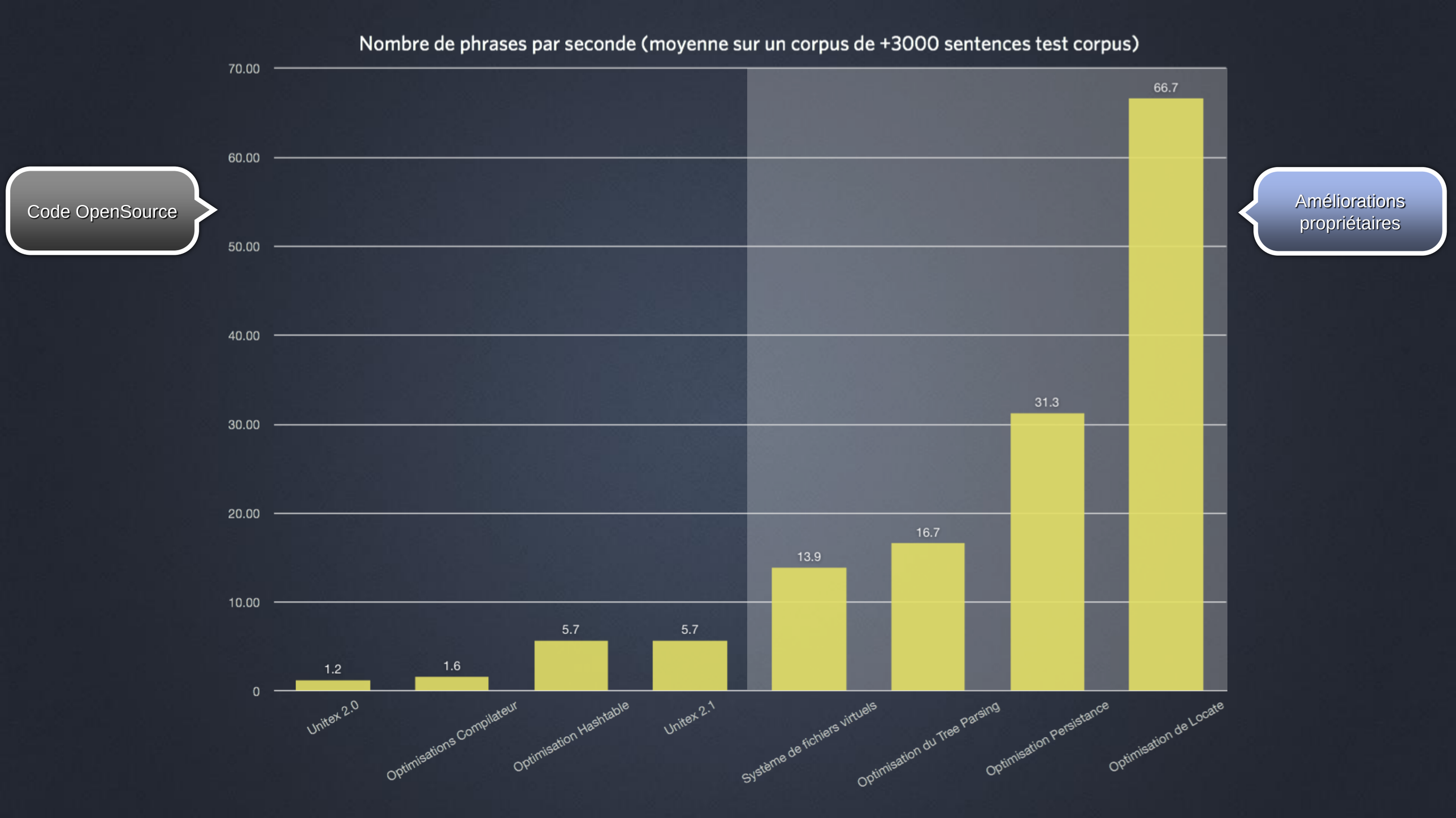
PERFORMANCE

Point de départ

Phrase traitée en 0,8s
en moyenne

Notre impératif

Traitement quasi-temps réel


Nombre de phrases par seconde (moyenne sur un corpus de +3000 sentences test corpus)
70.00
60.00
50.00
40.00
30.00
20.00
10.00
0
1.2
1.6
5.7
5.7
13.9
16.7
31.3
66.7
Unitex 2.0
Optimisations Compilateur
Optimisation Hashtable
Unitex 2.1
Système de fichiers virtuels
Optimisation du Tree Parsing
Optimisation Persistance
Optimisation de Locate
Code OpenSource
Améliorations propriétaires

PASSAGE EN LIBRAIRIE

- ★ À l’origine, collection d’exécutables indépendants
- ★ Amélioration de la vitesse (démarrage de chaque exécutable)
- ★ Modèle d’application monolithique pour les mobiles
- ★ Nécessité de supprimer les fuites mémoire et d’une meilleure gestion des erreurs

API C

- int UnitexTool_public_run(int argc,char* const argv[],int* p_number_done,struct pos_tools_in_arg* ptia);

- int UnitexTool_public_run_one_tool(const char*toolname,int argc,char* const argv[]);

JNI

- public native static int execUnitexTool(String[] cmdarray);
- public native static int execUnitexTool(String cmdline);

Exemple

- UnitexJni.execUnitexTool(new String[] {"UnitexToolLogger","Normalize",PFX+txt,"-r"+dirRes+"Norm.txt"});

VIRTUALISATION

- Sans virtualiser les fichiers, Unitex écrit le résultat de chaque commande sur le disque dur.
- Problèmes de performances importants, et même d'usure du matériel
- Création d'un espace de fichier virtuel et stockage des fichiers uniquement en mémoire.
- Les fichiers virtuels se reconnaissent à leur préfixe, qui permet à Unitex de savoir si il s'adresse au système de fichier virtuel ou au « vrai » système de fichier
- Deux implémentations : la notre et celle d'Unitex 3

API C

- void GetUnitexFileReadBuffer(const char*name,UNITEXFILEMAPPED** amf, const void**buffer,size_t *size_file);
- void CloseUnitexFileReadBuffer(UNITEXFILEMAPPED *,const void*buffer,size_t size_file);
- int WriteUnitexFile(const char*name,const void*buffer_prefix,size_t size_prefix,const void*buffer_suffix,size_t size_suffix);
- int AppendUnitexFile(const char*name,const void*buffer_data,size_t size_data);
- int RemoveUnitexFile(const char*name);
- int RenameUnitexFile(const char*oldName,const char*newName);
- int CopyUnitexFile(const char*srcName,const char*dstName);
- int CreateUnitexFolder(const char*name);
- int RemoveUnitexFolder(const char*name);
- int UnitexAbstractPathExists(const char* path);

API Java

- public native static boolean writeUnitexFile(String fileName, char[] fileContent);
- public native static boolean writeUnitexFile(String fileName, byte[] fileContent);
- public native static boolean writeUnitexFile(String fileName, String fileContent);
- public native static boolean writeUnitexFileUtf(String fileName, String fileContent);
- public native static boolean writeUnitexFileUtf(String fileName, String fileContent, boolean isBom);
- public native static boolean appendUnitexFile(String fileName, byte[] fileContent);
- public native static char[] getUnitexFileDataChar(String fileName);
- public native static byte[] getUnitexFileData(String fileName);
- public native static String getUnitexFileString(String fileName);
- public native static boolean removeUnitexFile(String fileName);
- public native static boolean createUnitexFolder(String folderName);
- public native static boolean removeUnitexFolder(String folderName);
- public native static boolean renameUnitexFile(String fileNameSrc, String fileNameDst);
- public native static boolean copyUnitexFile(String fileNameSrc, String fileNameDst);

PERSISTANCE

- ★ Les dictionnaires et graphes sont gros (surtout les dictionnaires), et complexe à charger (surtout les graphes)
- ★ Les recharger à chaque exécution de Dico et Locate est couteux
- ★ La persistence permet de les charger une fois pour toute à l'initialisation de l'application
- ★ Deux implémentations : la notre et celle d'Unitex 3

API C

- `int persistence_public_load_dictionary(const char*filename,char* persistent_filename_buffer,size_t buffer_size);`
- `void persistence_public_unload_dictionary(const char*filename);`
- `int persistence_public_load_fst2(const char*filename,char* persistent_filename_buffer,size_t buffer_size);`
- `void persistence_public_unload_fst2(const char*filename);`
- `int persistence_public_load_alphabet(const char*filename,char* persistent_filename_buffer,size_t buffer_size);`
- `void persistence_public_unload_alphabet(const char*filename);`

API Java

- public native static String loadPersistentDictionary(String filename);
- public native static void freePersistentDictionary(String filename);
- public native static String loadPersistentFst2(String filename);
- public native static void freePersistentFst2(String filename);
- public native static String loadPersistentAlphabet(String filename);
- public native static void freePersistentAlphabet(String filename);

ALLOCATEURS PERSONNALISÉS

- ★ UNITEX passe beaucoup de temps à gérer les allocations et désallocations mémoire
- ★ Permet l’optimisation en fonction des cas d’usage souhaités
- ★ “Personnalisation” des allocateurs par outils
- ★ Complémentarité avec la persistance, en particulier pour les graphes
- ★ Implémentation propriétaire Ergonotics

SUPPRESSION DES SORTIES

- Permet de supprimer les sorties d'Unitex sur la console
- API C

```
int SetStdWriteCB(enum stdwrite_kind swk, int trashOutput, t_fnc_stdOutWrite fnc_stdOutWrite, void*
privatePtr);

int GetStdWriteCB(enum stdwrite_kind swk, int* p_trashOutput, t_fnc_stdOutWrite* p_fnc_stdOutWrite,
void** p_privatePtr);
```

- API Java

```
public native static boolean setStdOutTrashMode(boolean flushMode);

public native static boolean setStdErrTrashMode(boolean flushMode);
```

MULTI-THREADING

- ★Modifications dans Unitex
- ★suppression de toute les variables globales
- ★Adaptation des espaces de virtualisation et de persistence

NOTRE CONTRIBUTION

- Nous avons ajouté à Unitex ce qui nous manquait pour pouvoir l'utiliser dans un cadre industriel
  - Performance, Fiabilité, Logger
  - API de virtualisation permettant d'intégrer des gestionnaires de virtualisation, persistance et allocateur mémoire personnalisé
- La grande majorité de nos améliorations a été reversée gratuitement dans le domaine public
- http://igm.univ-mlv.fr/~unitex/ufo.pdf

AU DELÀ...

- ★ Un gestionnaire de persistance de ressource et système de fichier virtuel optimisé, compatible multi-thread (notre implémentation)
- ★ Un gestionnaire d’allocation mémoire spécifique et optimisé par opération
- ★ Un format de fichier graphe binaire optimisé en vitesse de chargement et en taille, offrant la possibilité de crypter les graphes sur disque
- ★ Un format binaire INP remplaçant le format INF pour les dictionnaires, immédiat à charger et ne demandant pas d’allocation mémoire

Exemple C

```c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "UnitexTool.h"
#include "UnitexLibIO.h"
#include "PersistenceInterface.h"
#include "SyncTool.h"
#include "AbstractFilePlugCallback.h"

#ifdef HAS_UNITEX_NAMESPACE
using namespace unitex;
#endif

#ifdef _NOT_UNDER_WINDOWS
   #define UNITEX_PATH_SEPARATOR_CHAR '/'
   #define UNITEX_PATH_SEPARATOR_STRING "/"
#else
   #define UNITEX_PATH_SEPARATOR_CHAR '\\'
   #define UNITEX_PATH_SEPARATOR_STRING "\\"
#endif

const char* getVirtualFilePfx()
{
	if (UnitexAbstractPathExists("*") != 0)
		return "*";

	if (UnitexAbstractPathExists("$:") != 0)
		return "$:";

	return "";
}
```

Exemple C

```c
char* combineUnitexFileComponent(const char* resource_folder,const char* filename)
{
	if (resource_folder == NULL)
		resource_folder = "";
	if (filename == NULL)
		filename = "";

	size_t len_resource = strlen(resource_folder);
	size_t len_filename = strlen(filename);

	char* buffer = (char*)malloc(len_resource+len_filename+4);

	int must_add_separator = 0;
	if (len_resource > 0)
	{
		char last_folder_char = * (resource_folder + len_resource - 1);
		must_add_separator = ((last_folder_char == '\\') || (last_folder_char == '/')) ? 0 : 1;
	}

	memcpy(buffer,resource_folder,len_resource);
	if (must_add_separator != 0)
		*(buffer + len_resource) = UNITEX_PATH_SEPARATOR_CHAR;
	memcpy(buffer + len_resource + must_add_separator,filename,len_filename);
	*(buffer + len_resource + must_add_separator + len_filename) = '\0';
	return buffer;
}
```

Exemple C

```
void* processUnitexWork(const char* alphabetName,const char* normFullPath,
                                                const char* dictionaryFullName,const char* graphFullFileName,
                            const char* corpusPath,
                                                const void* buffer_prefix,size_t size_prefix,const void*buffer_suffix,size_t
size_suffix,
                            const char* inputCorpusEncoding, const char* outputCorpusEncoding)
{


        char* originalCorpusTextFileNameTxt = combineUnitexFileComponent(corpusPath,"corpus.txt") ;
        char* originalCorpusTextFileNameSnt = combineUnitexFileComponent(corpusPath,"corpus.snt") ;
        char* originalCorpusTextFileNameSntDir = combineUnitexFileComponent(corpusPath,"corpus_snt") ;
        char* originalCorpusTextFileNameConcordInd = combineUnitexFileComponent(originalCorpusTextFileNameSntDir,"concord.ind") ;

        SetStdWriteCB(stdwrite_kind_out, 1, NULL,NULL);
        SetStdWriteCB(stdwrite_kind_err, 1, NULL,NULL);


        CreateUnitexFolder(originalCorpusTextFileNameSntDir);

        int result_write = WriteUnitexFile(originalCorpusTextFileNameTxt,buffer_prefix,size_prefix,buffer_suffix,size_suffix);
          WriteUnitexFile("y:\\avir\\iii.txt",buffer_prefix,size_prefix,buffer_suffix,size_suffix);

        const char *Normalize_Argv[] = {"UnitexTool","Normalize",originalCorpusTextFileNameTxt,"-r",normFullPath,"-
k",inputCorpusEncoding};
        int nb_Normalize_Argv = 7;
        if (inputCorpusEncoding == NULL)
                nb_Normalize_Argv = 5;
        else
        if ((*inputCorpusEncoding) == '\0')
                nb_Normalize_Argv = 5;
    int result_Normalize = UnitexTool_public_run(nb_Normalize_Argv,(char**)Normalize_Argv,NULL,NULL);


        const char *Tokenize_Argv[] = {"UnitexTool","Tokenize",originalCorpusTextFileNameSnt,"-a",alphabetName};
        int nb_Tokenize_Argv = 5;
        int result_Tokenize = UnitexTool_public_run(nb_Tokenize_Argv,(char**)Tokenize_Argv,NULL,NULL);

        const char *Dico_Argv[] = {"UnitexTool","Dico","-t",originalCorpusTextFileNameSnt,"-a",alphabetName,dictionaryFullName};
        int nb_Dico_Argv = 7;
        int result_Dico = UnitexTool_public_run(nb_Dico_Argv,(char**)Dico_Argv,NULL,NULL);

        const char *Locate_Argv[] = {"UnitexTool","Locate","-t",originalCorpusTextFileNameSnt,"-a",alphabetName,
                "-L","-R","--all","-b","-Y",graphFullFileName,
                "-q",outputCorpusEncoding
        };
```

Exemple C

```
	int nb_Locate_Argv = 14;
	if (outputCorpusEncoding == NULL)
		nb_Locate_Argv = 12;
	else
	if ((*outputCorpusEncoding) == '\0')
		nb_Locate_Argv = 12;
	int result_Locate = UnitexTool_public_run(nb_Locate_Argv,(char**)Locate_Argv,NULL,NULL);


	int result_Concord = 0;


	if ((result_write != 0) || (result_Normalize != 0) || (result_Tokenize != 0) || (result_Dico != 0) || (result_Locate != 0) ||
(result_Concord != 0))
		return NULL;


	UNITEXFILEMAPPED *amf=NULL;
	const void*pBufSnt = NULL;
	size_t sizeSnt = 0;
	GetUnitexFileReadBuffer(originalCorpusTextFileNameConcordInd,&amf,&pBufSnt,&sizeSnt);
	void * return_buffer = (void*)malloc(sizeSnt+8);
	if (return_buffer != NULL)
	{
		memcpy(return_buffer,pBufSnt,sizeSnt);
		for (size_t walk=0;walk<8;walk++)
		{
			*(((char*)return_buffer) + sizeSnt + walk) = '\0';
		}
	}
	CloseUnitexFileReadBuffer(amf, pBufSnt, sizeSnt);



	free(originalCorpusTextFileNameTxt);
	free(originalCorpusTextFileNameSnt);
	free(originalCorpusTextFileNameSntDir);
	free(originalCorpusTextFileNameConcordInd);
	return return_buffer;
}
```

Exemple C

```c
int  main(int argc, char** argv)
{

        if (argc < 2)
        {
                // usage();
                return 0;
        }

        const char* pfx_prefix = getVirtualFilePfx();
        //pfx_prefix="y:\\avir\\wrkuni\\";
    const char* ressourceDir=*(argv+1);

        int nbLoop = 1;
        if (argc > 2)
                nbLoop=atoi(*(argv+2));

        int persist = 1;
        if (argc > 3)
                persist=atoi(*(argv+3));

    char* graphResDir = combineUnitexFileComponent(ressourceDir, "graph");
    char* dictionnaryResDir = combineUnitexFileComponent(ressourceDir, "dictionnary");
    char* othersResDir = combineUnitexFileComponent(ressourceDir, "others");


    char* AlphabetName = combineUnitexFileComponent(othersResDir,"Alphabet.txt");
        char* normFullPath = combineUnitexFileComponent(othersResDir,"Norm.txt");
    char* dictionnaryName = combineUnitexFileComponent(dictionnaryResDir, "dela-en-public.bin");
        char* graphName = combineUnitexFileComponent(graphResDir, "AAA-hoursgilles.fst2");

        char AlphabetNamePersisted[0x200];
        char dictionnaryNamePersisted[0x200];
        char graphNamePersisted[0x200];
        char normInVfs[0x200];



        if (persist)
        {
                persistence_public_load_alphabet(AlphabetName,AlphabetNamePersisted,sizeof(AlphabetNamePersisted)-1);

        persistence_public_load_dictionary(dictionnaryName,dictionnaryNamePersisted,sizeof(dictionnaryNamePersisted)-1);
                persistence_public_load_fst2(graphName,graphNamePersisted,sizeof(graphNamePersisted)-1);
                sprintf(normInVfs,"%sresInVfs%cNorm.txt",pfx_prefix,UNITEX_PATH_SEPARATOR_CHAR);
                CopyUnitexFile(normFullPath,normInVfs);
        }
        else
        {
                strcpy(AlphabetNamePersisted,AlphabetName);
                strcpy(dictionnaryNamePersisted,dictionnaryName);
                strcpy(graphNamePersisted,graphName);
                strcpy(normInVfs,normFullPath);
        }
```

Exemple C

```
    char corpusFolder[0x80];
    char workDirectoryName[0x100];
    size_t arbitraty_unique_value = 0;
    arbitraty_unique_value = (size_t)(&arbitraty_unique_value);
    sprintf(corpusFolder,"%sCorpusFolder",pfx_prefix);
    CreateUnitexFolder(corpusFolder);

    /* now the work, it can be splitted in several thread */
    sprintf(workDirectoryName,"%s%cworkUnitexThread_%08lx%08lx",corpusFolder,UNITEX_PATH_SEPARATOR_CHAR,
            ((unsigned long)((arbitraty_unique_value>>16)>>16)),((unsigned long)(arbitraty_unique_value)));
    CreateUnitexFolder(workDirectoryName);
    hTimeElapsed beginAll = SyncBuidTimeMarkerObject();

    for (int i=0;i<nbLoop;i++)
    {
                    const char* inputString = "I want watch at 4:00 am see at 6:00 pm before leave at 15.47";
        char* stringResult = (char*)processUnitexWork(

    AlphabetNamePersisted,normInVfs,dictionnaryNamePersisted,graphNamePersisted,workDirectoryName,
                            NULL,0,inputString,strlen(inputString),"utf8-no-bom,bom","utf8-no-bom");

                    if (stringResult != NULL)
                    {
                            if ((i+1) == nbLoop)
                                    printf("result is %s\n",stringResult);
                            free(stringResult);
                    }
    }
    unsigned int allTimeMsec = SyncGetMSecElapsed(beginAll);
    RemoveUnitexFolder(workDirectoryName);
    /* work finish, cleanup */
    printf("all time %d msec (%d msec by run)\n",allTimeMsec,allTimeMsec/nbLoop);

    if (persist)
    {
            persistence_public_unload_dictionary(dictionnaryNamePersisted);
            persistence_public_unload_fst2(graphNamePersisted);
            RemoveUnitexFile(normInVfs);
    }

        free(AlphabetName);
    free(graphResDir);
    free(dictionnaryResDir);
    free(othersResDir);


    free(dictionnaryName);
        free(graphName);
        free(normFullPath);
        return 0;
}
```

MERCI

http://www.ergonotics.com/unitex-contribution/

unitex-contribution@ergonotics.com

francois@ergonotics.com • gilles@ergonotics.com • anastasia@ergonotics.com