Utilisation de la librairie Unitex

Intégration des technologies de traitement de langage naturel d'Unitex au sein d'une application

Gilles Vollant

Ergonotics

gilles@ergonotics.com

Utilisation de la librairie Unitex 3.0

Ce document présente les différents éléments à connaitre pour pouvoir utiliser efficacement le moteur Unitex à partir d'une application ayant des besoins de traitement du langage naturel.

1) Lien avec la librairie Unitex

Unitex peut être compilé en tant que librairie dynamique standard, exportant des fonctions C, appelable à partir de votre logiciel, ou en tant que librairie JNI, appelable à partir d'un programme Java.

Note : la JNI est, techniquement, une librairie dynamique standard du système, qui exporte en plus des fonctions C des fonctions JNI pour le Java. Qui peut le plus peut le moins : on peut appeler les fonctions C à partir de la librairie JNI.

Ensuite, chaque outil Unitex est appelé avec des paramètres similaires à ceux utilisés avec les outils en ligne de commande.

Chaque outil Unitex est susceptible d'être appelé à partir de la librairie. Les outils les plus testés et optimisés adaptés à une utilisation en bibliothèque sont ceux dédié à l'exécution de graphes, plus que ceux dédiés à leur création :

- CheckDic
- Compress
- Concord
- Dico
- Fst2Txt
- Locate
- Normalize
- SortTxt
- Tokenize

a) Déclaration pour appel d'une librairie Unitex en C

```
#include "UnitexTool.h"

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[]);
```

Ces deux fonctions permettent d'appeler un outil de la librairie Unitex, avec une syntaxe proche du traditionnel main du C.

```
const char *CheckDic_Argv[] = {"CheckDic","c:\\Users\\GillesVollant\\Ergonotics\\Dela\\ufo-
contact.dic","DELAF"};
int ret = UnitexTool_public_run_one_tool("CheckDic",3,CheckDic_Argv);

const char* Tokenize_Argv[]={"UnitexTool","Tokenize","-a","*english/Alphabet.txt",UfoSntFileVFN};
int retTok = UnitexTool_public_run(5,Tokenize_Argv,NULL,NULL);
```

b) Déclaration pour appel d'une librairie Unitex en Java

```
import fr.umlv.unitex.jni.UnitexJni;

/**
 * Function to run UnitexTool with string or string array, like java exec in
 * java runtime
 * you can combine several tool using { }
 * (see UnitexTool in Unitex manual for more information)
 *
 * String [] strArrayCmds={"UnitexTool","{","Normalize","corpus.txt",
        "-r", "Norm.txt","}","{","Tokenize","corpus.txt", "-r", "Alphabet.txt","}"};
 *
 * UnitexLibAndJni.execUnitexTool(strArrayCmds);
     *
 *
 * @return value : the return value of the tools (0 for success)
 */
public native static int execUnitexTool(String[] cmdarray);


/**
 * Function to run UnitexTool with string or string array, like java exec in
 * java runtime
 * you can combine several tool using { }
 * (see UnitexTool in Unitex manual for more information)
 *
 * UnitexLibAndJni.execUnitexTool("UnitexTool Normalize \"corpus.txt\" -r \"Norm.txt\"");
 *
 * UnitexLibAndJni.execUnitexTool("UnitexTool Tokenize \"corpus.txt\" -a \"Alphabet.txt\"");
 *
 * UnitexLibAndJni.execUnitexTool("UnitexTool { Normalize \"corpus.txt\" -r \"Norm.txt\" }" +
 *                                      " { Tokenize \"corpus.txt\" -a \"Alphabet.txt\" }");
 *
 *
 * @return value : the return value of the tools (0 for success)
 */
public native static int execUnitexTool(String cmdline);
```

Exemple:

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

2) Accès fichier et système de fichier virtuel

Les outils Unitex utilisent beaucoup d'accès fichier pour s'échanger des informations. Afin d'économiser les accès disque, il est possible d'utiliser des fichiers virtuels, qui sont écrits en mémoire.

Les fichiers virtuels se reconnaissent à un préfixe (« * » pour le système de fichier virtuel optimisé Ergonotics pour Unitex 2.1 ou 3.0, « $: » pour le système de fichier virtuel minimal intégré dans Unitex 3.0.

Il existe des fonctions d'accès simplifiées aux fichiers, qui sont compatibles à la fois avec le système de fichiers virtuel et les fichiers du système de fichiers standard.

a) Déclaration pour appel d'une librairie Unitex en C

```
#include "UnitexLibIO.h"

int UnitexAbstractPathExists(const char* path);
```

UnitexAbstractPathExists permet de savoir si un préfixe correspond à un système de fichiers virtuel. Ainsi, l'exemple suivant retournera le système de fichier virtuel le plus optimisé disponible sous Unitex 3.0 :

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

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

      return NULL;
}
```

```
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);
```

GetUnitexFileReadBuffer permet d'obtenir un pointeur en lecture seule sur le contenu d'un fichier. Ce pointeur sera valide jusqu'à l'appel de la fonction CloseUnitexFileReadBuffer correspondant, sachant que ce fichier ne doit en aucun cas être modifié (et à plus forte raison supprimé) entre l'appel de GetUnitexFileReadBuffer et de CloseUnitexFileReadBuffer.

WriteUnitexFile permet de créer un fichier à partir d'un ou deux buffers binaires mémoires (si un seul buffer est utile, il suffira de positionner buffer_suffix à NULL et size_suffix à zéro). AppendUnitexFile permet d'ajouter du contenu à la fin d'un fichier.

Toutes ces fonctions manipulant les fichiers avec des buffers binaires, c'est à l'application appelante de gérer la problématique d'encodage Unicode (UTF8 ou UTF16).

RemoveUnitexFile, RenameUnitexFile et CopyUnitexFile permettent respectivement de supprimer, renommer, ou copier un fichier. Notons que CopyUnitexFile permet de copier un fichier entre le système de fichiers virtuel et le système de fichiers standard (dans les 2 sens).

CreateUnitexFolder n'agit que pour le système de fichier standard et permet de créer un répertoire.

RemoveUnitexFolder permet de supprimer un répertoire (dans le système de fichiers standard) ou de supprimer tous les fichiers avec un préfixe donné (dans le système de fichier virtuel).

Le système de fichier virtuel n'ayant pas de notion de répertoire (« : », « / » et « \ » sont traités comme les autres caractères), cela perme in-fine d'avoir un comportement compatible.

b) Déclaration pour appel d'une librairie Unitex en Java

```
/**
 * function to known how many abstract file system are installed
 *
 * @return the number of Abstract file system installed in Unitex
 */
public native static int numberAbstractFileSpaceInstalled();

/*
 * file function below are compatible with both disk file system and
 * abstract file system
 */
/**
 * writeUnitexFile* function create file to be used by Unitex.
 */
/**
 * create a file from a raw binary char array
 */
public native static boolean writeUnitexFile(String fileName,
      char[] fileContent);

/**
 * create a file from a raw binary byte array
 */
public native static boolean writeUnitexFile(String fileName,
      byte[] fileContent);

/**
 * create a file from a string using UTF16LE encoding with BOM (native
 * Unitex format)
 */
public native static boolean writeUnitexFile(String fileName,
      String fileContent);

/**
 * create a file from a string using UTF8 encoding without BOM
 */
public native static boolean writeUnitexFileUtf(String fileName,
      String fileContent);

/**
 * create a file from a string using UTF8 encoding with or without BOM
 */
public native static boolean writeUnitexFileUtf(String fileName,
      String fileContent, boolean isBom);

/**
 * append to a file a raw binary byte array
 */
public native static boolean appendUnitexFile(String fileName,
      byte[] fileContent);

/**
 * read a file to a raw binary char array representation
 */
public native static char[] getUnitexFileDataChar(String fileName);

/**
 * read a file to a raw binary byte array representation
 */
public native static byte[] getUnitexFileData(String fileName);

/**
 * read and decode a file to a string.
 */
public native static String getUnitexFileString(String fileName);
```

```
/**
 * remove a file
 */
public native static boolean removeUnitexFile(String fileName);

/**
 * create a folder, if needed
 */
public native static boolean createUnitexFolder(String folderName);

/**
 * remove a folder and the folder content
 */
public native static boolean removeUnitexFolder(String folderName);

/**
 * rename a file
 */
public native static boolean renameUnitexFile(String fileNameSrc,
      String fileNameDst);

/**
 * copy a file
 */
public native static boolean copyUnitexFile(String fileNameSrc,
      String fileNameDst);

/**
 * tests whether a path is already present in Unitex's abstact file space
 */
public native static boolean unitexAbstractPathExists(String path);

/**
 * retrieve array of file in abstract space
 */
public native static String[] getFileList(String path);
```

```
public String getVirtualFilePfx()
{
     if (UnitexJni.unitexAbstractPathExists("*"))
      return "*";

     if (UnitexJni.unitexAbstractPathExists("$:"))
      return "$:";

     return null;
}
```

3) Persistance des ressources linguistiques

Les ressources linguistiques (graphes .fst2, dictionnaires .bin./inf, et dans une bien moindre mesure les fichiers alphabet) prennent beaucoup de temps pour être chargées. Les dictionnaires sont gros, les graphes sont complexes. Pour économiser ce temps à chaque lancement d'un outil, il est possible de pré-charger ces ressources avant d'appeler les outils Unitex.

Pour cela, il existe pour chaque type de ressource (dictionnaire, graphe et alphabet) une fonction de pré-chargement. Cette fonction retourne le nom de la ressource persistente (dans le buffer persistent_filename_buffer de taille buffer_size qui devra être fourni par l'appelant en C, comme valeur de retour des fonctions Java).

A noter que les fichiers dictionnaires utilisés doivent rester présents et non modifiés, car la technique de mappage de fichiers en mémoire peut être utilisée.

Attention : la règle de constitution du nom de la ressource persistante pouvant évoluer et dépendre du type de librairie de persistance installée, c'est bien ce nom qui devra être fourni comme paramètre aux outils Unitex correspondant (Locate, Dico…), et éventuellement aux fonctions de déchargement correspondant.

a) Déclaration pour appel d'une librairie Unitex en 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);
```

b) Déclaration pour appel d'une librairie Unitex en 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);
```

4) Suppression des sorties console

Dans un but d'optimisation, de performance et de compatibilité multithread (pour éviter un mélange de sortie des outils s'exécutant simultanément), il est conseillé (hors debuggage) de supprimer les sorties console d'Unitex.

a) Déclaration pour appel d'une librairie Unitex en C

```
/* There is a set of callbacks for rerouting stdin, stdout and stderr IO */
/* t_fnc_stdOutWrite (for stdout and stderr) and t_fnc_stdIn (for stdin) define
   the callback.
   the callback must return the number of char processed (the actual size if operating normally)
*/

enum stdwrite_kind { stdwrite_kind_out=0, stdwrite_kind_err } ;

typedef size_t (ABSTRACT_CALLBACK_UNITEX *t_fnc_stdOutWrite)(const void*Buf, size_t size,void*
privatePtr);
/* SetStdWriteCB sets the callback for one of the two (stdout or stderr) output streams
   if trashOutput == 1, fnc_stdOutWrite must be NULL and the output will just be ignored
   if trashOutput == 0 and fnc_stdOutWrite == NULL and the output will be the standard output
   if trashOutput == 0 and fnc_stdOutWrite != NULL the callback will be used

   the callback is called with a zero size in only one case: when SetStdWriteCB is called, the
preceding
   callback is called a last time (with its associated privatePtr) with a zero size.
   This may allow you, for instance, to close a file or free some memory...

   privatePtr is a private value which is passed as the last parameters of a callback
   GetStdWriteCB sets the current value on *p_trashOutput, *p_fnc_stdOutWrite and **p_privatePtr

   returns 1 if successful and 0 if an error occurred on SetStdWriteCB or GetStdWriteCB
   */

UNITEX_FUNC int UNITEX_CALL SetStdWriteCB(enum stdwrite_kind swk, int trashOutput,
                                          t_fnc_stdOutWrite fnc_stdOutWrite,void* privatePtr);
UNITEX_FUNC int UNITEX_CALL GetStdWriteCB(enum stdwrite_kind swk, int* p_trashOutput,
                                          t_fnc_stdOutWrite* p_fnc_stdOutWrite,void**
p_privatePtr);
```

L'implémentation C permet de plus de rediriger les sorties vers une fonction callback.

```
#include "AbstractFilePlugCallback.h"

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

b) Déclaration pour appel d'une librairie Unitex en Java

```
/**
 * allow ignore (flushMode is TRUE) or emit normal message to stdout
 */
public native static boolean setStdOutTrashMode(boolean flushMode);

/**
 * allow ignore (flushMode is TRUE) or emit error message to stderr
 */
public native static boolean setStdErrTrashMode(boolean flushMode);
```

Pour supprimer les sorties console d'Unitex, on pourra donc appeler :

```
UnitexJni.setStdOutTrashMode (true);
UnitexJni.setStdErrTrashMode (true);
```

//
Les 3 url
Nouveauté ergonotics

- Optimim virt et persist, compatible
- Gestionnaire mémoire
- Inp et pack fst2
-