unitex-library Documentation

Version 3.1.0

The Unitex/GramLab devel team

04 January 2017

Table des matières

Note : Cette documentation est en cours de rédaction. Elle peut contenir des erreurs et des informations peuvent manquer. Si vous avez des commentaires ou questions, merci de nous contacter sur la liste de diffusion unitex-devel ou de remplir un rapport de bogue en utilisant notre système de suivi des incidents.

Unitex/GramLab est un système open source, multi-plateforme et multilingue, de traitement de corpus textuels à l'aide de ressources linguistiques. Ce document présente les différents éléments à connaître pour pouvoir utiliser efficacement le moteur Unitex lors du développement d'une application ayant des besoins liés au traitement du langage naturel.

Sommaire :

Note : Cette documentation est en cours de rédaction. Elle peut contenir des erreurs et des informations peuvent manquer. Si vous avez des commentaires ou questions, merci de nous contacter sur la liste de diffusion unitex-devel ou de remplir un rapport de bogue en utilisant notre système de suivi des incidents.

CHAPITRE 1

Invoquer la bibliothèque Unitex

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

Note : L'interface JNI est une bibliothèque dynamique standard du système, qui exporte en plus des fonctions C des fonctions JNI pour Java. *Qui peut le plus peut le moins* : on peut appeler les fonctions C à partir de la bibliothèque 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 bibliothèque. Les outils les plus testés, adaptés et optimisés à une utilisation en bibliothèque sont ceux dédiés à l'exécution de graphes, en comparaison à ceux destinés à leur création :

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

1.1 C

Les deux fonctions suivantes permettent d'appeler un outil de la bibliothèque Unitex, avec une syntaxe proche du traditionnel `main()` du C.

```
#include "UnitexTool.h"
int UnitexTool_public_run(int argc,char* const argv[],int* p_number_done,struct pos_tools_in_arg* pti
int UnitexTool_public_run_one_tool(const char*toolname,int argc,char* const argv[]);
```

A partir de la révision 4012 d'Unitex 3.1 beta, il est possible d'utiliser UnitexTool_public_run_string en utilisant une simple chaine de caractère avec les commandes :

```
int UnitexTool_public_run_string(const char* cmd_line);
int UnitexTool_public_run_string_ret_infos(const char* cmd_line, int* p_number_done, struct pos_tools
```

Par exemple :

1.1.1 CheckDic

```
    const char *CheckDic_Argv[] = {"CheckDic","c:\\foo\\bar.dic","DELAF"};
    int ret = UnitexTool_public_run_one_tool("CheckDic",3,CheckDic_Argv);

    // ou

    int ret = UnitexTool_public_run_string("UnitexTool CheckDic");
```

1.1.2 Tokenize

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

    // ou

    int retTok = UnitexTool_public_run_string("UnitexTool Tokenize -a \"*english/Alphabet.txt\"");
```

1.2 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);
```

Par example :

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

Note : Cette documentation est en cours de rédaction. Elle peut contenir des erreurs et des informations peuvent manquer. Si vous avez des commentaires ou questions, merci de nous contacter sur la liste de diffusion unitex-devel ou de remplir un rapport de bogue en utilisant notre système de suivi des incidents.

CHAPITRE 2

Accès aux fichiers et système de fichiers 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 (VFS (Virtual File System)), qui sont écrits en mémoire.

Les fichiers virtuels se reconnaissent à un préfixe :

Prefix	Implementation	License
`$:`	système de fichiers virtuel intégré à partir Unitex 3.0.	LGPLv2
`*:`	système de fichiers virtuel optimisé Ergonotics pour Unitex 2.1 ou 3.0	Propriétaire

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.

2.1 C

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

Indice : Le système de fichier virtuel n'ayant pas de notion de répertoire, les caractères `:` (deux-points), `/` (barre oblique) et `\` (barre oblique inversée) sont traités comme les autres, ce qui permet, in fine, d'avoir un comportement compatible.

2.1.1 `UnitexAbstractPathExists`

`UnitexAbstractPathExists` permet de savoir si un préfixe correspond à un système de fichiers virtuel.

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

Ainsi, l'exemple suivant retournera le système de fichier virtuel le plus optimisé disponible sous Unitex 3.0 :

```
const char* getVirtualFilePrefix() {
  if (UnitexAbstractPathExists("*:") != 0) {
    return "*:";
  }

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

```
  }

  return NULL;
}
```

2.1.2 GetUnitexFileReadBuffer

`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,

Avertissement : Le fichier ne doit en aucun cas être modifié (et à plus forte raison supprimé) entre l'appel de `GetUnitexFileReadBuffer` et de `CloseUnitexFileReadBuffer`.

```
void GetUnitexFileReadBuffer(const char*  name, UNITEXFILEMAPPED** amf,
                             const void** buffer, size_t* size_file);
```

2.1.3 WriteUnitexFile

`WriteUnitexFile` permet de créer un fichier à partir d'un ou deux buffers binaires (si un seul buffer est utile, il suffira de positionner `buffer_suffix` à `NULL` et `size_suffix` à `0`).

```
int WriteUnitexFile(const char* name, const void* buffer_prefix, size_t size_prefix,
                                      const void* buffer_suffix,size_t size_suffix);
```

2.1.4 AppendUnitexFile

`AppendUnitexFile` permet d'ajouter du contenu à la fin d'un fichier.

```
int AppendUnitexFile(const char* name,const void* buffer_data,size_t size_data);
```

2.1.5 RemoveUnitexFile

`RemoveUnitexFile` permet de supprimer un fichier.

```
int RemoveUnitexFile(const char* name);
```

2.1.6 RenameUnitexFile

`RenameUnitexFile` permet de renommer un fichier.

```
int RenameUnitexFile(const char* oldName,const char* newName);
```

2.1.7 CopyUnitexFile

`CopyUnitexFile` permet de copier un fichier. Notons que cette function est capable de copier un fichier entre le système de fichiers virtuel et le système de fichiers standard (dans les 2 sens).

```
int CopyUnitexFile(const char* srcName,const char* dstName);
```

2.1.8 CreateUnitexFolder

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

```
int CreateUnitexFolder(const char* name);
```

2.1.9 RemoveUnitexFolder

`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).

```
int RemoveUnitexFolder(const char* name);
```

2.2 Java

2.2.1 numberAbstractFileSpaceInstalled

```
/**
 * 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();
```

2.2.2 writeUnitexFile

```
/**
 * 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);
```

2.2.3 appendUnitexFile

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

2.2.4 getUnitexFileDataChar

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

2.2.5 getUnitexFileData

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

2.2.6 getUnitexFileString

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

2.2.7 removeUnitexFile

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

2.2.8 createUnitexFolder

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

2.2.9 removeUnitexFolder

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

2.2.10 renameUnitexFile

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

2.2.11 copyUnitexFile

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

2.2.12 unitexAbstractPathExists

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

Example :

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

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

  return null;
}
```

2.2.13 getFileList

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

Note : Cette documentation est en cours de rédaction. Elle peut contenir des erreurs et des informations peuvent manquer. Si vous avez des commentaires ou questions, merci de nous contacter sur la liste de diffusion unitex-devel ou de remplir un rapport de bogue en utilisant notre système de suivi des incidents.

CHAPITRE 3

Persistance des ressources linguistiques

Les ressources linguistiques (graphes `.fst2`, dictionnaires `.bin` et fichiers `.inf`, et dans une bien moindre mesure les fichiers `Alphabet.txt`) 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 persistante (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).

Note : Les fichiers dictionnaires utilisés doivent rester présents et non modifiés, car la technique de mappage de fichiers en mémoire peut être utilisée.

Avertissement : 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.

3.1 C

3.1.1 persistence_public_load_dictionary

```
int persistence_public_load_dictionary(const char* filename,
                                       char* persistent_filename_buffer,
                                       size_t buffer_size);
```

3.1.2 persistence_public_is_persisted_dictionary_filename

```
int persistence_public_is_persisted_dictionary_filename(const char*filename);
```

3.1.3 persistence_public_unload_dictionary

```
void persistence_public_unload_dictionary(const char* filename);
```

3.1.4 persistence_public_load_fst2

```
int persistence_public_load_fst2(const char* filename,
                                 char* persistent_filename_buffer,
                                 size_t buffer_size);
```

3.1.5 persistence_public_is_persisted_fst2_filename

```
int persistence_public_is_persisted_fst2_filename(const char*filename);
```

3.1.6 persistence_public_unload_fst2

```
void persistence_public_unload_fst2(const char* filename);
```

3.1.7 persistence_public_load_alphabet

```
int persistence_public_load_alphabet(const char* filename,
                                     char* persistent_filename_buffer,
                                     size_t buffer_size);
```

3.1.8 persistence_public_is_persisted_alphabet_filename

```
int persistence_public_is_persisted_alphabet_filename(const char*filename);
```

3.1.9 persistence_public_unload_alphabet

```
void persistence_public_unload_alphabet(const char* filename);
```

3.2 Java

3.2.1 loadPersistentDictionary

```
public native static String loadPersistentDictionary(String filename);
```

3.2.2 freePersistentDictionary

```
public native static void freePersistentDictionary(String filename);
```

3.2.3 loadPersistentFst2

```
public native static String loadPersistentFst2(String filename);
```

3.2.4 freePersistentFst2

```
public native static void freePersistentFst2(String filename);
```

3.2.5 loadPersistentAlphabet

```
public native static String loadPersistentAlphabet(String filename);
```

3.2.6 freePersistentAlphabet

```
public native static void freePersistentAlphabet(String filename);
```

Note : Cette documentation est en cours de rédaction. Elle peut contenir des erreurs et des informations peuvent manquer. Si vous avez des commentaires ou questions, merci de nous contacter sur la liste de diffusion unitex-devel ou de remplir un rapport de bogue en utilisant notre système de suivi des incidents.

CHAPITRE 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.

4.1 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* privat
/* 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 preced
   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);
```

4.2 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);
```

Note : Cette documentation est en cours de rédaction. Elle peut contenir des erreurs et des informations peuvent manquer. Si vous avez des commentaires ou questions, merci de nous contacter sur la liste de diffusion unitex-devel ou de remplir un rapport de bogue en utilisant notre système de suivi des incidents.

CHAPITRE 5

Copyright

04 January 2017

5.1 Licenses

5.1.1 FDL : GNU Free Documentation License

This document is licensed under the terms of the GNU Free Documentation License version 1.3.

download http://www.gnu.org/licenses/fdl-1.3.txt

5.1.2 LGPL : GNU Lesser General Public License

The examples in this document are licensed under the terms of the GNU Lesser General Public License, version 2.1 (LGPLv2).

download http://www.gnu.org/licenses/lgpl-2.1.txt

5.1.3 LGPL-LR : Lesser General Public License For Linguistic Resources

Linguistic resources, as electronic dictionaries and local grammars, are licensed under the terms of the Lesser General Public License For Linguistic Resources (LGPL-LR)

download http://bit.do/LGPL-LR

Note : Cette documentation est en cours de rédaction. Elle peut contenir des erreurs et des informations peuvent manquer. Si vous avez des commentaires ou questions, merci de nous contacter sur la liste de diffusion unitex-devel ou de remplir un rapport de bogue en utilisant notre système de suivi des incidents.

CHAPITRE 6

Crédits

6.1 Auteurs

L'auteur de ce document, en version française, est Gilles Vollant <gilles_at_ergonotics.com> de la société Ergonotics. Si vous voulez en savoir davantage sur la collaboration d'Ergonotics au projet Unitex/GramLab, visitez le site : http://www.ergonotics.com/unitex-contribution/

6.2 Collaborateurs

Par ordre chronologique :

- Cristian Martinez <cristian.martinez_at_univ-paris-est.fr>
- Eric Laporte <eric.laporte_at_univ-paris-est.fr>

6.3 Traducteurs

Les personnes suivantes ont contribué à la traduction de ce document :

English

- Cristian Martinez <cristian.martinez_at_univ-paris-est.fr>

CHAPITRE 7

Indices et tableaux

— genindex
— search

CHAPITRE 8

Comment contribuer

Nous vous invitons à contribuer à l'amélioration de ce document. Voici quelques suggestions :

— Découvrez un peu plus sur le langage de balisage RST (reStructuredText) et le générateur de documentation Sphinx. Une antisèche de la syntaxe RST avec Sphinx est disponible ici.
— Dupliquez (Fork) le contenu du projet et ensuite formulez votre demande de contribution (Pull Request) à la branche de développement.
— Alternativement, si vous ne voulez pas apprendre la syntaxe RST ou Sphinx, soumettez un rapport de bogue ou une demande de fonctionnalité à notre GitHub.

B

C

J

L

P

S