unitex-library Documentation

Release 3.1.0

The Unitex/GramLab devel team

January 04, 2017

Contents

Note: This documentation is still a work in progress. If you have any issues or questions, please ask on the unitex-devel mailing list or file a bug in our issue tracker.

Unitex/GramLab is an open source, cross-platform, multilingual, lexicon- and grammar-based corpus processing suite. Unitex tools are designed to perform several Natural Language Processing (NLP (Natural Language Processing)) tasks on a textual corpus relying on linguistic resources such as electronic dictionaries and grammars represented as finite state transducteurs (FSTs (Finite State Transducers)), recursive transition networks (RTNs (Recursive Transition Networks)) and lexicon-grammars.

This document is a guide to using the Unitex C/C++ library and and the Java Native Interface (JNI (Java Native Interface)), including methods descriptions and example snippets.

Contents:

Note: This documentation is still a work in progress. If you have any issues or questions, please ask on the unitex-devel mailing list or file a bug in our issue tracker.

CHAPTER 1

Linking your application

Unitex could be compiled into a shared library which exports all the engine high-level functions, and then that shared library linked when compiling an application. It is also possible to compile Unitex into a JNI library, and using it from Java.

Note: A JNI is a shared-object library (on Windows, a DLL) that contains entry points reachable in Java via the JVM. JNI lets you call Java methods from C functions and implement Java classes in C.

The exported functions can be called like any other Unitex command-line tool.

Every command-line tool has an exported function. The most frequently used tools, which are adapted and optimized for use as library, are those dedicated to the execution of graphs, compared to those for their creation:

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

1.1 C

The following two functions call a Unitex tool from the Unitex library, the used syntax is similar to call the `main()` function from 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[]);
```

Starting Unitex 3.1.4012-beta, it is possible to call a function from a character buffer using `UnitexTool_public_run_string()` as follows:

```
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
```

For instance:

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);

    // or

    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);

    // or

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

1.2 Java

1.2.1 execUnitexTool

```
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);
```

For example:

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

Note: This documentation is still a work in progress. If you have any issues or questions, please ask on the unitex-devel mailing list or file a bug in our issue tracker.

CHAPTER 2

File I/O and Unitex Virtual File System access

Unitex tools have a high I/O workload. To improve the I/O performance it is possible to use the Unitex Virtual File System (VFS (Virtual File System)) access layer which, at an application-level, behaves like a RAM drive and locate files on it.

Virtual files are identified by a prefix string:

Prefix	Implementation	License
`$:`	Open source Virtual File System included starting Unitex 3.0	LGPLv2
`*:`	Ergonotics Virtual File System for Unitex 2.1 and higher	Proprietary

The Unitex shared library export some common utility functions for file handling, these function are compatible with both the virtual file system and the standard file system.

2.1 C

Note: For those functions using file I/O with binary buffers, it is the caller of the function the one in charge of implementing the proper Unicode handling (UTF8 ou UTF16).

Hint: The Unitex VFS system is a flat file system, with no notion of directories/folders. Some characters like `:` (colon), `/` (slash) and `\` (backslash) are treat as the same as any other character, thus allowing a consistent behavior with a hierarchical file system.

2.1.1 `UnitexAbstractPathExists`

`UnitexAbstractPathExists` Checks whether a prefix corresponds to a virtual file system.

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

For example, to check the best VFS layer available you could implement the following function:

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

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

```
    return "$:";
  }

  return NULL;
}
```

2.1.2 GetUnitexFileReadBuffer

`GetUnitexFileReadBuffer` Returns the contents of the specified file as a read-only pointer. This pointer will remain valid until call `CloseUnitexFileReadBuffer`.

> **Warning:** Do not modify or remove the pointed file while it is being used, i.e. between the call of `GetUnitexFileReadBuffer` and `CloseUnitexFileReadBuffer`

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

2.1.3 WriteUnitexFile

`WriteUnitexFile` Creates a file using 1 or 2 binary buffers. If you need only 1 buffer, you need to set `buffer_suffix` to `NULL` and `size_suffix` to `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` Appends text to an existing file, or to a new file if the specified file does not exist.

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

2.1.5 RemoveUnitexFile

`RemoveUnitexFile` Deletes the specified file.

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

2.1.6 RenameUnitexFile

`RenameUnitexFile` Causes the filename referred to by *oldName* to be changed to *newName*.

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

2.1.7 CopyUnitexFile

`CopyUnitexFile` Copies an existing file to a new file. This function could be used to copy a file (in both senses) between the virtual and the standard file system.

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

2.1.8 CreateUnitexFolder

`CreateUnitexFolder` Creates a new directory under the standard file system. If the underlying file system is virtual, the function does nothing.

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

2.1.9 RemoveUnitexFolder

`RemoveUnitexFolder` Removes folder or directory structures. If the underlying file system is virtual, the function remove all files containing the given prefix in their name.

```
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: This documentation is still a work in progress. If you have any issues or questions, please ask on the unitex-devel mailing list or file a bug in our issue tracker.

CHAPTER 3

Persistent data access layer

Unitex language-related resources such as graphes `.fst2`, dictionaries `.bin` and `.inf` files, may become too large or complex, thus slow to load. To minimize the load time, the Unitex shared library provides some useful functions to preload these resources explicitly and reference them when calling other functions.

For each type of resource (dictionary, graph or alphabet), there exists a preload function. These functions return, in the buffer pointed to by `persistent_filename_buffer`, the name of the persistent resource. The number of bytes to return in the buffer (i.e., the buffer size) need to be equal to `buffer_size`.

Warning: After to preload a dictionary file, do not modify or remove it. Under the propietary VFS implementation, a file memory-mapping strategy might being used. As result, your program may produces unpredictable results or stop to working.

Note: The conventions used to create named persistent resources may change and depend upon the type of the persistent library installed. You should therefore be consistent in using the same file prefix (`$:` or `*:`) across all function calls.

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: This documentation is still a work in progress. If you have any issues or questions, please ask on the unitex-devel mailing list or file a bug in our issue tracker.

CHAPTER 4

Disabling console output

If you are looking to optimize the processing times and, under a multithreaded environment, to avoid to get mixed up messages printed to the console, it is advised to disable all the Unitex console outputs.

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);
```

Additionally, the C implementation allows you to redirect all messages to a callback function.

```
#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);
```

To disable all the Unitex console outputs you could call:

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

> **Note:** This documentation is still a work in progress. If you have any issues or questions, please ask on the unitex-devel mailing list or file a bug in our issue tracker.

CHAPTER 5

Copyright

January 04, 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: This documentation is still a work in progress. If you have any issues or questions, please ask on the unitex-devel mailing list or file a bug in our issue tracker.

CHAPTER 6

Credits

6.1 Authors

The main author of this document, originally in French, is Gilles Vollant from Ergonotics. He can be reached on <gilles_at_ergonotics.com>. If you want to know more about this collaboration, check out here: http://www.ergonotics.com/unitex-contribution/

6.2 Contributors

In chronological order of the first contribution:

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

6.3 Translators

The following people have contributed to the translation of this document:

English

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

CHAPTER 7

Indices and tables

- genindex
- search

CHAPTER 8

Contributing

We welcome everyone to contribute to improve these guidelines. Below are some of the things that you can do to contribute:

- Learn a bit about the RST (reStructuredText) markup language and Sphinx for creating documentation. A Restructured Text (reST) and Sphinx CheatSheet is available here
- Fork us and request a pull to the develop branch.
- Alternately, if you don't want to learn RST and Sphinx, submit a bug report or a feature request to our GitHub.

C

D

J

L

P

V