Projet RUIG-GIAN

Réseau universitaire international de Genève (RUIG)
Geneva International Academic Network (GIAN)

Analyse linguistique et extraction de collocations

Rapport final

Partenaires :

Organisation mondiale du commerce (OMC)

Laboratoire d'analyse et de technologie du langage (LATL)

Coordination :

Eric Wehrli

LATL - Département de linguistique
Université de Genève
1211 Genève 4

Eric.Wehrli@lettres.unige.ch

Table des matières

1 Sous-système COLLOCATION.....5

1.1 Index de la documentation 7

1.2 Manuel de l'utilisateur 11

1.3 Manuel de l'utilisateur - Images 19

Figure1.odc..... 21

Figure2.odc..... 22

Figure 3.odc..... 23

Figure4.odc..... 24

Figure5.odc..... 25

Figure6.odc..... 26

Figure7.odc..... 27

Figure 8.odc..... 28

Figure9.odc..... 29

Figure10.odc..... 30

1.4 Manuel du développeur 31

1.5 Manuel du développeur – Modules..... 40

CollocationAlign 41

CollocationBrowse 42

CollocationCmds..... 44

CollocationDB 46

CollocationDBAux..... 49

CollocationExe..... 50

CollocationFilter 51

CollocationGlobal..... 52

CollocationOptions..... 62

CollocationRead 63
CollocationSelect 64
CollocationValidate 66
CollocationWeb 69
2 Sous-système UTILS 71
2.1 Index de la documentation 73
2.2 Manuel de l'utilisateur 77
2.3 Manuel du développeur 83
2.4 Manuel du développeur –Modules 89
UtilsCmds 91
UtilsFileTree 93
UtilsFunctions 95
UtilsOptions 99
UtilsProcessFiles 102
UtilsScan 104

1. Sous-système

COLLOCATION

Emplacement : **sources/project/Collocation**

1.1 Index de la documentation

Emplacement: **Collocation/Docu/Sys-Map.odc**

Projet RUIG-GIAN, 2002-2003

Collocation Extraction and Visualization

Violeta Seretan

LATL, Violeta.Seretan@latl.unige.ch

User Manual
Developper Manual

Modules: CollocationAlign, CollocationBrowse, CollocationCmds, CollocationDB, CollocationDBAux, CollocationExe, CollocationFilter, CollocationGlobal, CollocationOptions, CollocationRead, CollocationSelect, CollocationValidate, CollocationWeb

Sources: CollocationAlign, CollocationBrowse, CollocationCmds, CollocationDB, CollocationDBAux, CollocationExe, CollocationFilter, CollocationGlobal, CollocationOptions, CollocationRead, CollocationSelect, CollocationValidate, CollocationWeb

Interface: Open interface

1.2. Manuel de l’utilisateur

Emplacement: **Collocation/Docu/User-Man/**

V.Seretan, Violeta.Seretan@lettres.unige.ch

Collocation Extraction and Visualization

- a brief visual guide -

November, 2004

User Manual

Contents

I. Overview
II. Collocation Extraction Interface
III. Collocation Visualization Interfaces
IV. Further Information

See also: Developper Manual, Subystem Map

I. Overview

This document explains how to use the tool for collocation extraction and visualization. It contains visual indications about the collocation extraction interface (menu option: Collocation -> Extraction) and the collocation visualization interfaces (menu options: Collocation -> Concordance and Collocation -> Alignment).

Back

II. Collocation Extraction Interface

This interface allows the user to extract collocations from a collection of files. The collocation extraction first performs the syntactic parsing of files, then applies a stochastic test for ranking the extracted word expressions according to the likelihood to constitute a collocation.

In order to process a collection of files, several steps need to be performed:
- select the input folder, i.e. the folder containing the collection of documents;
- optionally, filter the files to be processed (the filtering can be done automatically and/or manually);
- select the output folder, i.e. the folder where different result files (including the extracted collocations) will be stored.
  Note: for storing the collocation extracted in a database rather than in a text file, a datasource called "cooc" and pointing to the file "cooc.mdb" provided with this application must be defined by a system administrator.
- select the processing parameters (the options for parsing and for storing the extracted collocations).

The paragraphs below provide explanations for :

1. how to choose a folder with the files to be processed;

2. how to apply an automatic filter on the files contained by this folder;

3. how to further apply a manual selection of the files to be processed;

4. how to choose the folder in which the results will be stored;

5. how to set the processing parameters.

Back

1. *How to choose a folder with the files to be processed*

The input folder can be chosen either by:
- typing in the edit field the complete path of the folder;
- browsing the file structure on a disk drive.

Figure 1. *Input Folder*

Back

2. How to apply an automatic filter on the files contained by this folder

The content of the input folder (the files) can be restricted using several criteria:

- the file contains a **specific string in its name**
  To use this filter, type a string in the text field "include only files named".

- the file has a **specific type**
  To select only files of a desired type, fill in the field "include only files of types" with the desired file extension. Then click on the button "Add" close to this field. You can repeat the procedure to specify other types.
  ***Note**: The supported file formats are:*
  ASCII and Unicode files (e.g. txt, htm, html)
  ODC files (BlackBox specific file format)
  rtf files (rich text format from word processors like MS Word)
  UTF-8 (extension utf8)

- the file is contained in the **root** of the input folder or in **one of its subfolders**.
  To select all the files in the input folder, check the option "include files from subfolders".
  To select only the files in the root of the input folder, uncheck this options.

- the file is (not) contained in a **subfolder with a specific name**
  To exclude the files contained in specific folders, check the option 'exclude files from folders named' and add the names of the folders to be excluded in the list below it. To add a folder name for exclusion, type it in the list header and click on the button 'Add folder name'. All the folder names added to this list will be used for files filtering.
  ***Note:** Default value for this list: the folders named "Log" (output by the collocation extraction).*

- the file has been **last modified at a specific date**
  To apply a filter on the date of files last modification (includes creations), check the box 'include only files created/modified'. Then specify if the system should verify that the file was modified several days previously to the current date, or in a given days interval. To do that, choose either the radio button "in the last ... days" and enter the desired number of days, or the radio button "between ... and ...", and specify two dates.
  To specify the start limit, enter a date in the format DD-MM-YY in the field next to "between".
  To specify the end limit, enter it in the field next to "and".
  The default values that are proposed are between *yesterday* and *today*.

Figure 2. *Automatic Files Filter*

Back

3. *How to further apply a manual selection of the files to be processed*

The content of the input folder is filtered according to the automatic filter and is shown in a list, from which the user can manually choose which items (files or subfolders) to be processed.
To include or exclude items, use the buttons "Check All", "Uncheck All", "Invert" or use the mouse in combination with the keys Ctrl or Shift.

Figure 3. *Manual Files Selection*

Back

4. *How to choose the folder in which the results will be stored*

As in 1, either type the name of the folder directly or browse the disk structure.
If the folder name does not exist on the disk, it will be created when the output is produced.
***Note**: The result files will be written in the output folder, and they will be gathered in a folder having the same name as the folder containing the input files.*

Figure 4. *Output Folder*

Back

5. *How to set the processing parameters*

There are two kinds of parameters to set: for parsing and for collocation extraction.
Parsing parameters that must be set: **language** only (English and French are supported for now).
The others may keep the default values (*max alt* 10, *shallow parsing* checked, *filtering* checked, *output structure* unchecked).
Extraction parameters should specify where to store the collocations extracted.
Note: By storing collocations in a text file, some functionalities will not be available (collocation score computation, corpus frequency count, filtering, ordering).
Both for parsing and extraction, additional files are produced that concern the global statistics for a processing session (globalStat.txt) and Log files. If the parameter *output structure i*s selected, then files containing syntactic analyses are produced.
The parameters *save results in one file* and *results file per file* decide if the results are cumulated for all the files or are saved separately, file per file. In the last case, a mirror copy of the input folder's structure is created in the output folder.
The parameter *text files format* decide which type the results files will have: *txt* or *odc* (BlackBox internal format).

Figure 5. *Processing Parameters*

Back

III. Collocation Visualization Interfaces

These interfaces are represented by the Concordance tool and the Alignment tool.
They both displays the collocations previously extracted and their contexts (the sentences) in the originating files (the document in which they occur). The alignment tool displays, in addition, the hypothesized target context in the parallel documents, when available. That is, when translations exist for the originating documents, the system tries to identify the translation of the source sentence and displays it next to the source sentence.

The figures below explain how to use the visualization tools:

- How to display the whole list of collocations

The list in the left side displays the distinct collocations. To see all the occurrences of a collocation in the corpus, use the *first, previous, next* and *last* buttons at the bottom of the list.

Figure 6. *Collocation Visualization Interface - Navigation Buttons*

- How to display a subset of the extracted collocations

Use the button "Filter" to open the "Filter Collocations" interface. This allows to filter the collocations by using different criteria that can be combined:

- collocations containing a given word (key1 or key2);
- collocations of given types (multiple types can be selected from the list of types);
- collocations that have at least the given number of occurrences in the corpus;
- collocations that have the score bigger than a threshold;
- collocations that were (manually) validated.

The number of collocations to display can be limited by entering a maximum number of collocations to display. The order of score and corpus frequency is taken into account. The maximum number refers to distinct collocations. All the occurrences of each collocation in corpus are also shown.

Figure 7. *Collocation Filtering Interface*

- How to order the list of collocations

Use the button "Sort alphabetically" to alphabetically order the (filtered) list.
Use the button "Filter" and choose *Order by score* or *frequency* to apply the respective filter.

Figure 8. *Collocation Ordering*

- How to create a bilingual collocation database

The user can add collocation entries in a bilingual database of validated collocations. This database is stored in the table "BilingualCooc" in the database "cooc.mdb" provided with the application.
Use the button "Validate" to add the currently visualized collocation to a temporary list of collocations for validation. Edit the details of the entry in the temporary list. As long as for an entry the translation was entered and this entry is not already present in the bilingual database, it will be added in the database.
Use the buttons "Validate All" or "Validate Selected" to store the desired entries in the database.
The temporary list of collocations for validation can be consulted at any moment from the menu Collocation -> Validation List.

Figure 9. *Collocation Validation*

- How to consult the extracted collocations later

The interface "Display Options" helps to specify where the visualization tools will look for the data to be displayed. Choose for "Data Source" either *database* or *text file.* In the last case, choose a file named (or prefixed) "cooc" from an output folder used at extraction.
The language of the data to be displayed must be specified (when using the "cooc.mdb" database, the collocation data are stored in different tables for the 3 different languages).
For alignment, the paths of the existing parallel corpora need to be specified. Be default, the same path of the input folder is considered.

Figure 10. *Display Options*

Back

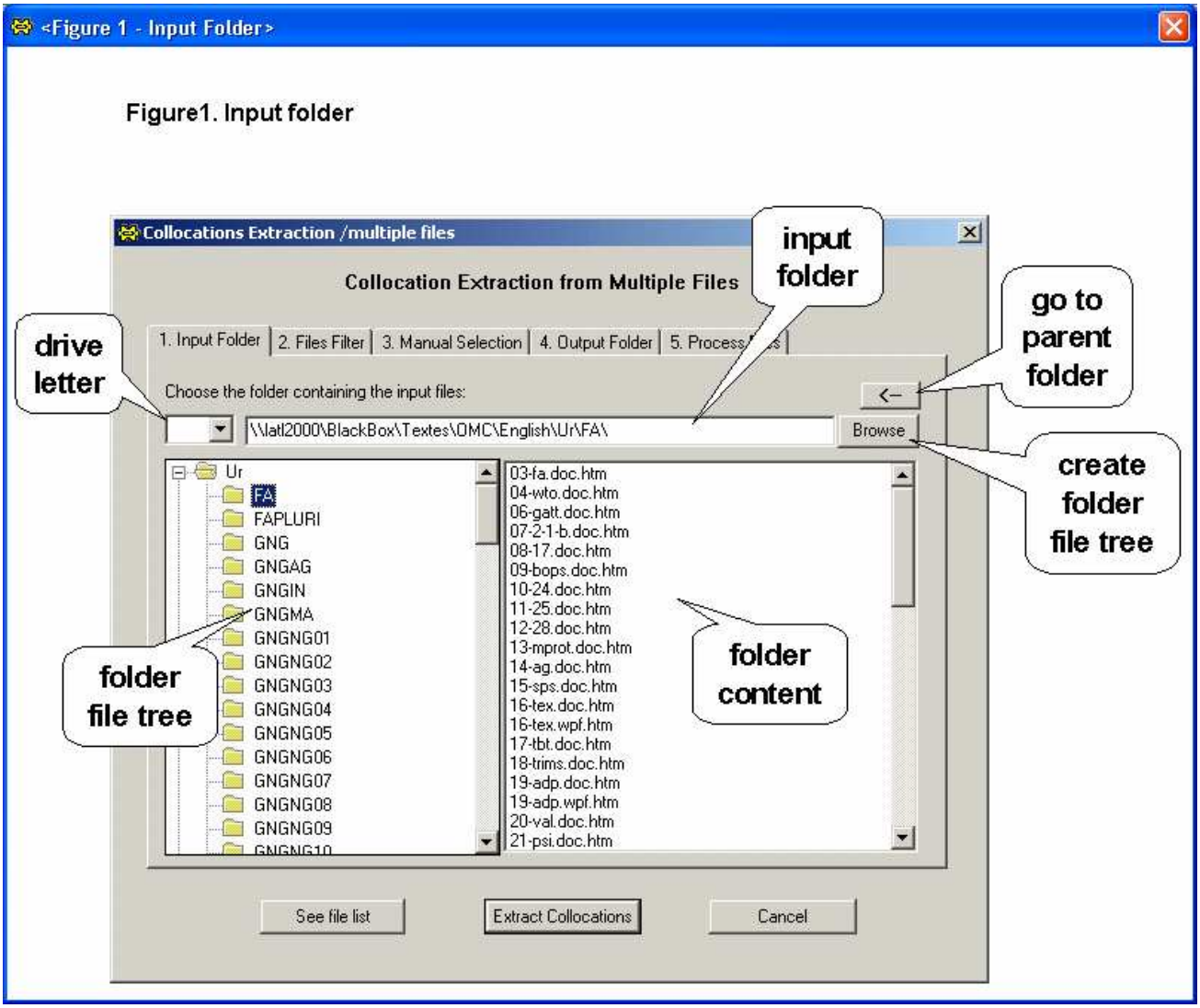
IV. Further Information

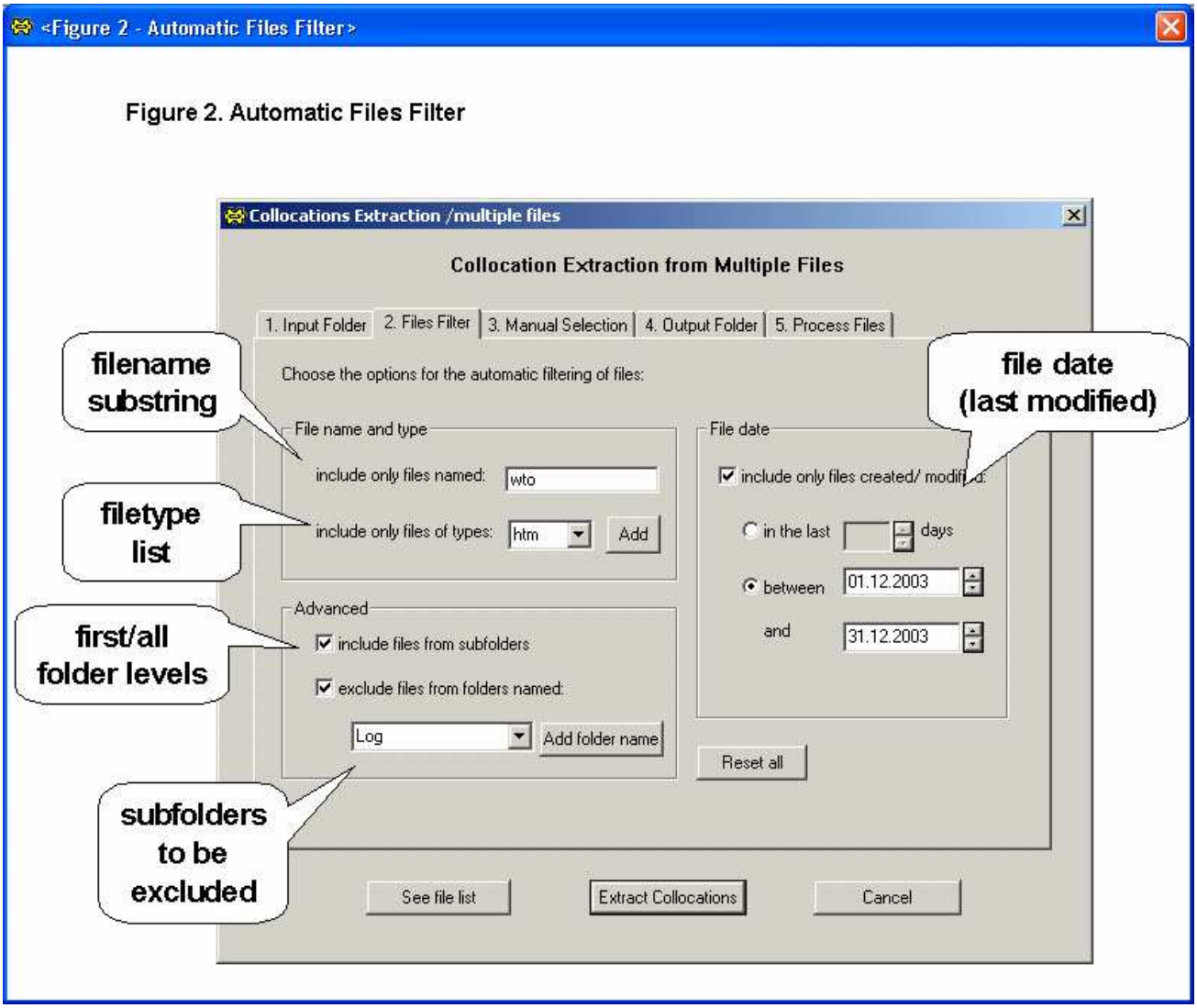
Contact **Violeta.Seretan@lettres.unige.ch** for further questions or comments.

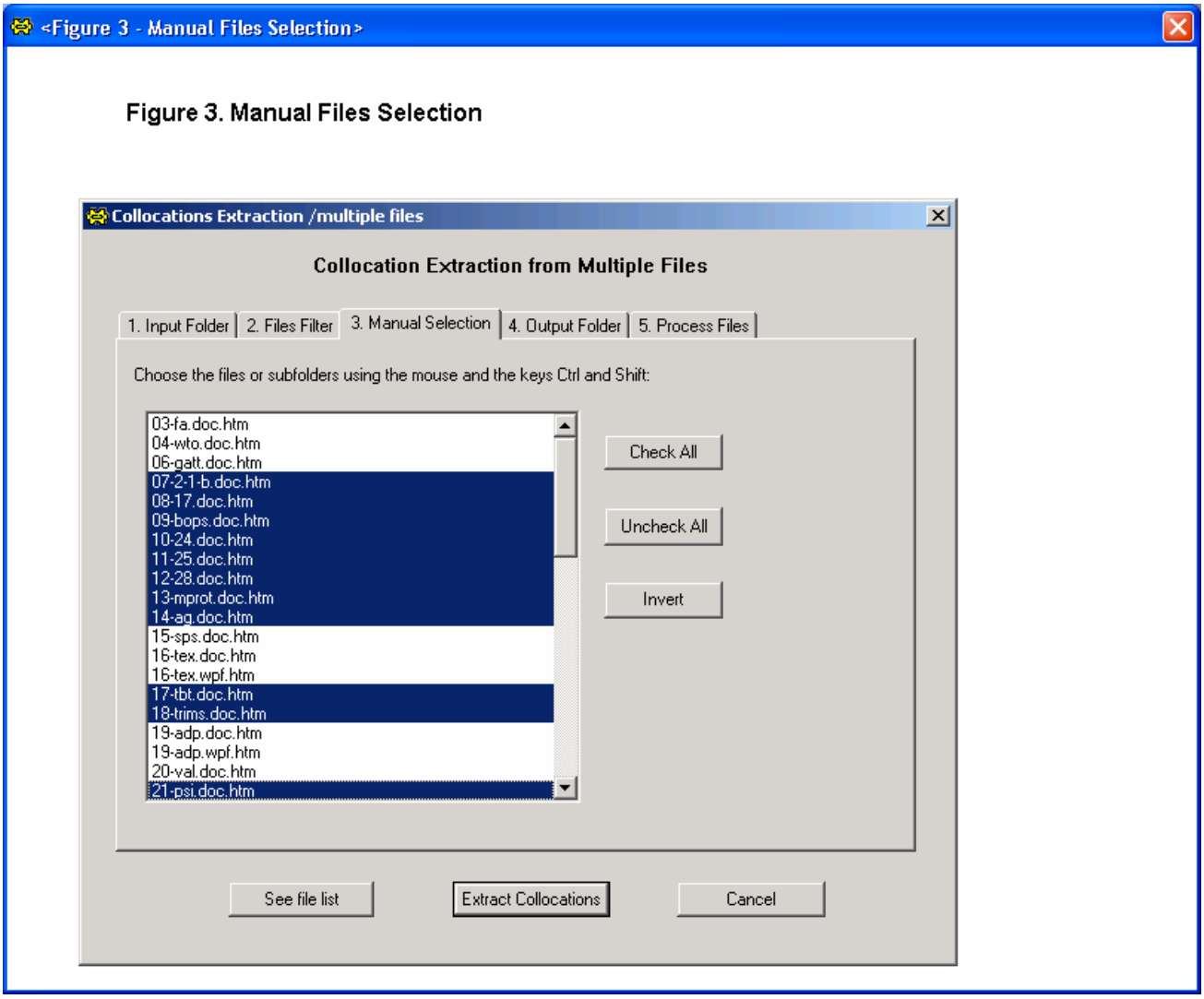
Back

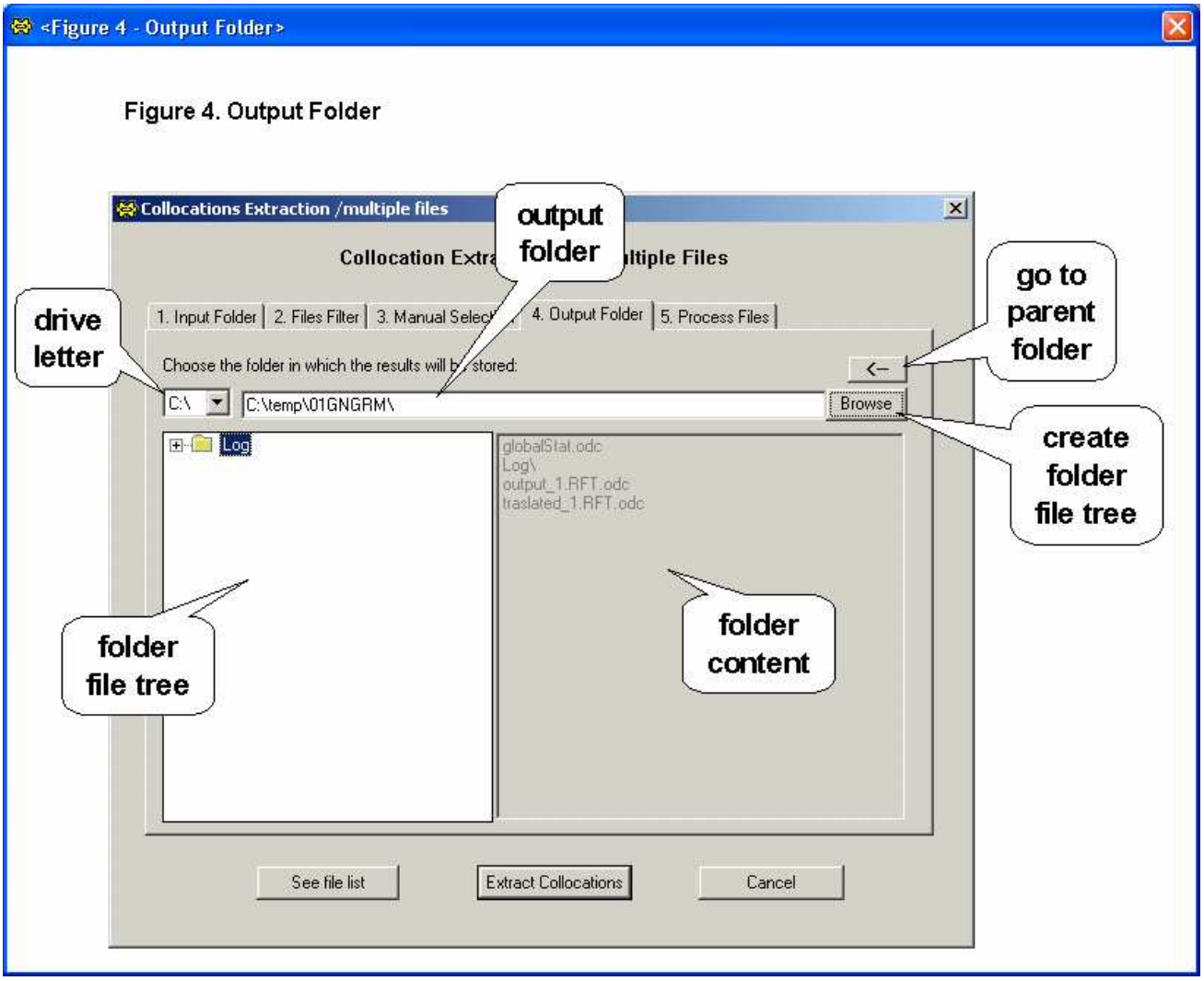
1.3 Manuel de l’utilisateur - Images

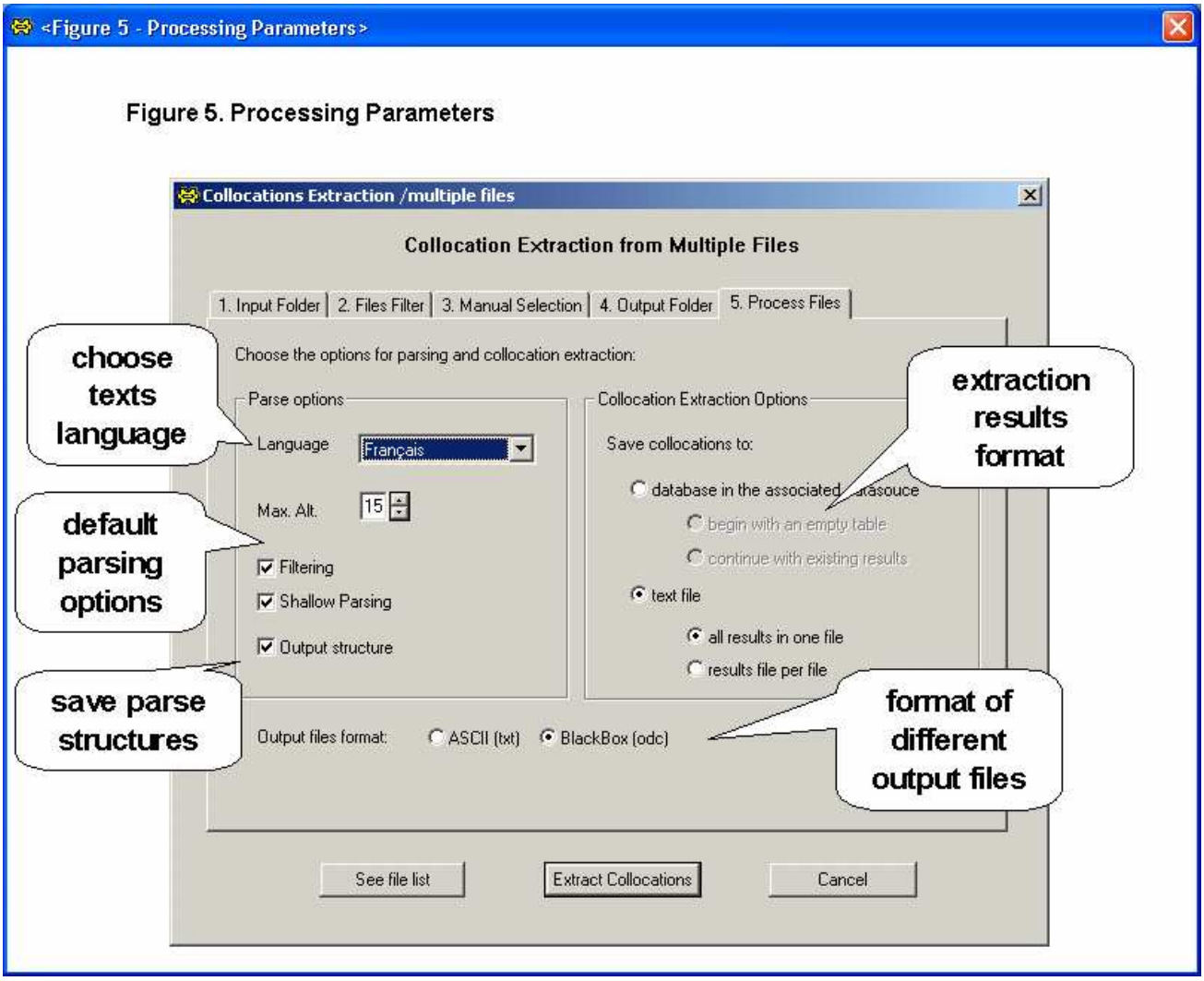
Emplacement: **Collocation/Docu/Figures/**


<Figure 1 - Input Folder>
Figure1. Input folder
Collocations Extraction /multiple files
Collocation Extraction from Multiple Files
input folder
go to parent folder
drive letter
1. Input Folder
2. Files Filter
3. Manual Selection
4. Output Folder
5. Process Files
Choose the folder containing the input files:
\\latl2000\BlackBox\Textes\OMC\English\Ur\FA\
Browse
create folder file tree
Ur
FA
FAPLURI
GNG
GNGAG
GNGIN
GNGMA
GNGNG01
GNGNG02
GNGNG03
GNGNG04
GNGNG05
GNGNG06
GNGNG07
GNGNG08
GNGNG09
GNGNG10
folder file tree
03-fa.doc.htm
04-wto.doc.htm
06-gatt.doc.htm
07-2-1-b.doc.htm
08-17.doc.htm
09-bops.doc.htm
10-24.doc.htm
11-25.doc.htm
12-28.doc.htm
13-mprot.doc.htm
14-ag.doc.htm
15-sps.doc.htm
16-tex.doc.htm
16-tex.wpf.htm
17-tbt.doc.htm
18-trims.doc.htm
19-adp.doc.htm
19-adp.wpf.htm
20-val.doc.htm
21-psi.doc.htm
folder content
See file list
Extract Collocations
Cancel


<Figure 2 - Automatic Files Filter>
Figure 2. Automatic Files Filter
Collocations Extraction /multiple files
Collocation Extraction from Multiple Files
1. Input Folder
2. Files Filter
3. Manual Selection
4. Output Folder
5. Process Files
Choose the options for the automatic filtering of files:
filename substring
filetype list
first/all folder levels
subfolders to be excluded
file date (last modified)
File name and type
include only files named:
wto
include only files of types:
htm
Add
Advanced
include files from subfolders
exclude files from folders named:
Log
Add folder name
File date
include only files created/ modified:
in the last
days
between
01.12.2003
and
31.12.2003
Reset all
See file list
Extract Collocations
Cancel


<Figure 3 - Manual Files Selection>
Figure 3. Manual Files Selection
Collocations Extraction /multiple files
Collocation Extraction from Multiple Files
1. Input Folder
2. Files Filter
3. Manual Selection
4. Output Folder
5. Process Files
Choose the files or subfolders using the mouse and the keys Ctrl and Shift:
03-fa.doc.htm
04-wto.doc.htm
06-gatt.doc.htm
07-2-1-b.doc.htm
08-17.doc.htm
09-bops.doc.htm
10-24.doc.htm
11-25.doc.htm
12-28.doc.htm
13-mprot.doc.htm
14-ag.doc.htm
15-sps.doc.htm
16-tex.doc.htm
16-tex.wpf.htm
17-tbt.doc.htm
18-trims.doc.htm
19-adp.doc.htm
19-adp.wpf.htm
20-val.doc.htm
21-psi.doc.htm
Check All
Uncheck All
Invert
See file list
Extract Collocations
Cancel


<Figure 4 - Output Folder>
Figure 4. Output Folder
Collocations Extraction /multiple files
output folder
Collocation Extra
ltiple Files
drive letter
go to parent folder
1. Input Folder
2. Files Filter
3. Manual Selec
4. Output Folder
5. Process Files
Choose the folder in which the results will b stored:
C:\
C:\temp\01GNGRM\
Browse
Log
globalStat.odc
Log\
output_1.RFT.odc
traslated_1.RFT.odc
create folder file tree
folder file tree
folder content
See file list
Extract Collocations
Cancel


<Figure 5 - Processing Parameters>
Figure 5. Processing Parameters
Collocations Extraction /multiple files
Collocation Extraction from Multiple Files
1. Input Folder
2. Files Filter
3. Manual Selection
4. Output Folder
5. Process Files
Choose the options for parsing and collocation extraction:
Parse options
Language
Français
Max. Alt.
15
Filtering
Shallow Parsing
Output structure
Collocation Extraction Options
Save collocations to:
database in the associated
begin with an empty table
continue with existing results
text file
all results in one file
results file per file
Output files format:
ASCII (txt)
BlackBox (odc)
See file list
Extract Collocations
Cancel
choose texts language
default parsing options
save parse structures
extraction results format
format of different output files

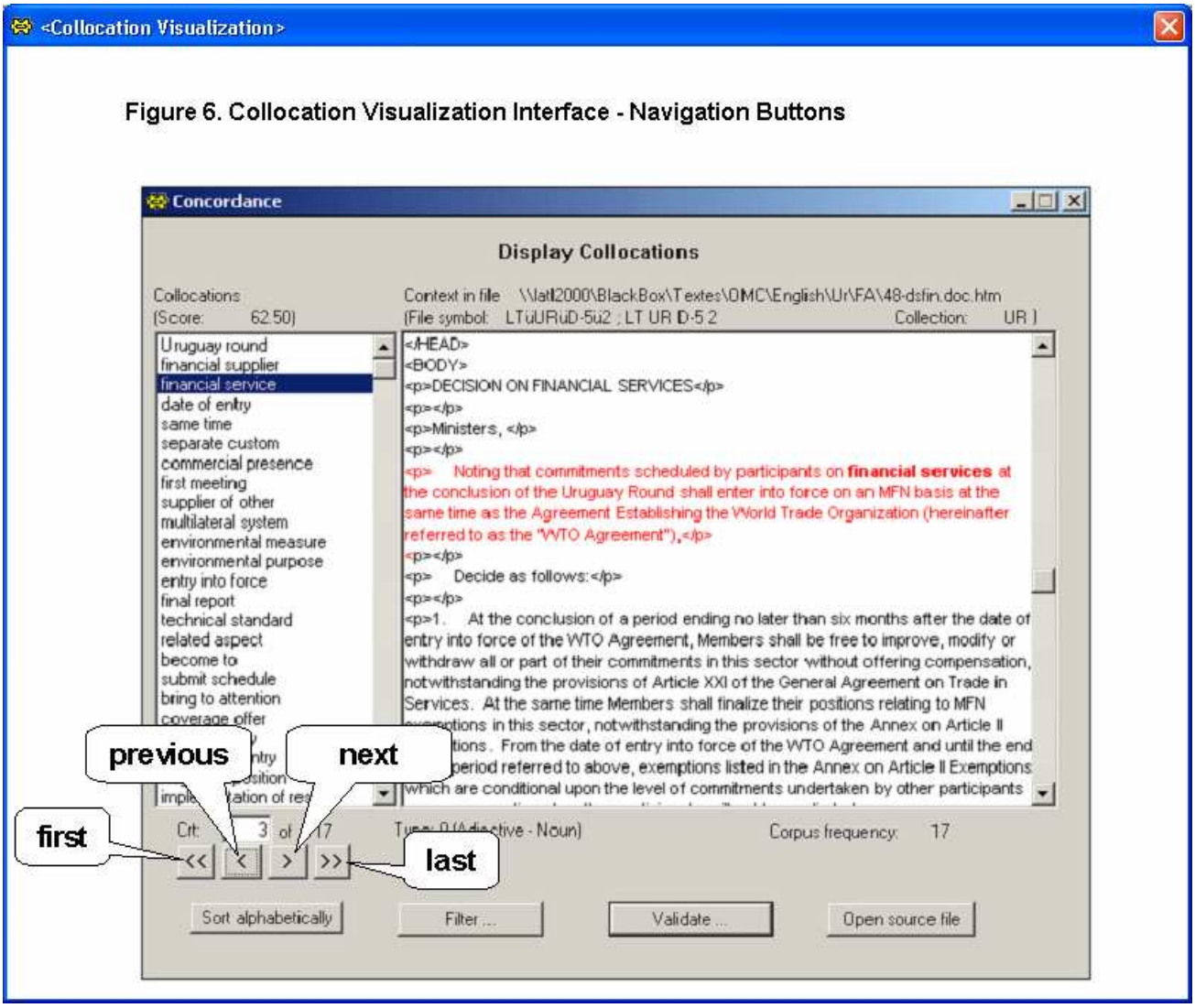


Figure 6. Collocation Visualization Interface - Navigation Buttons

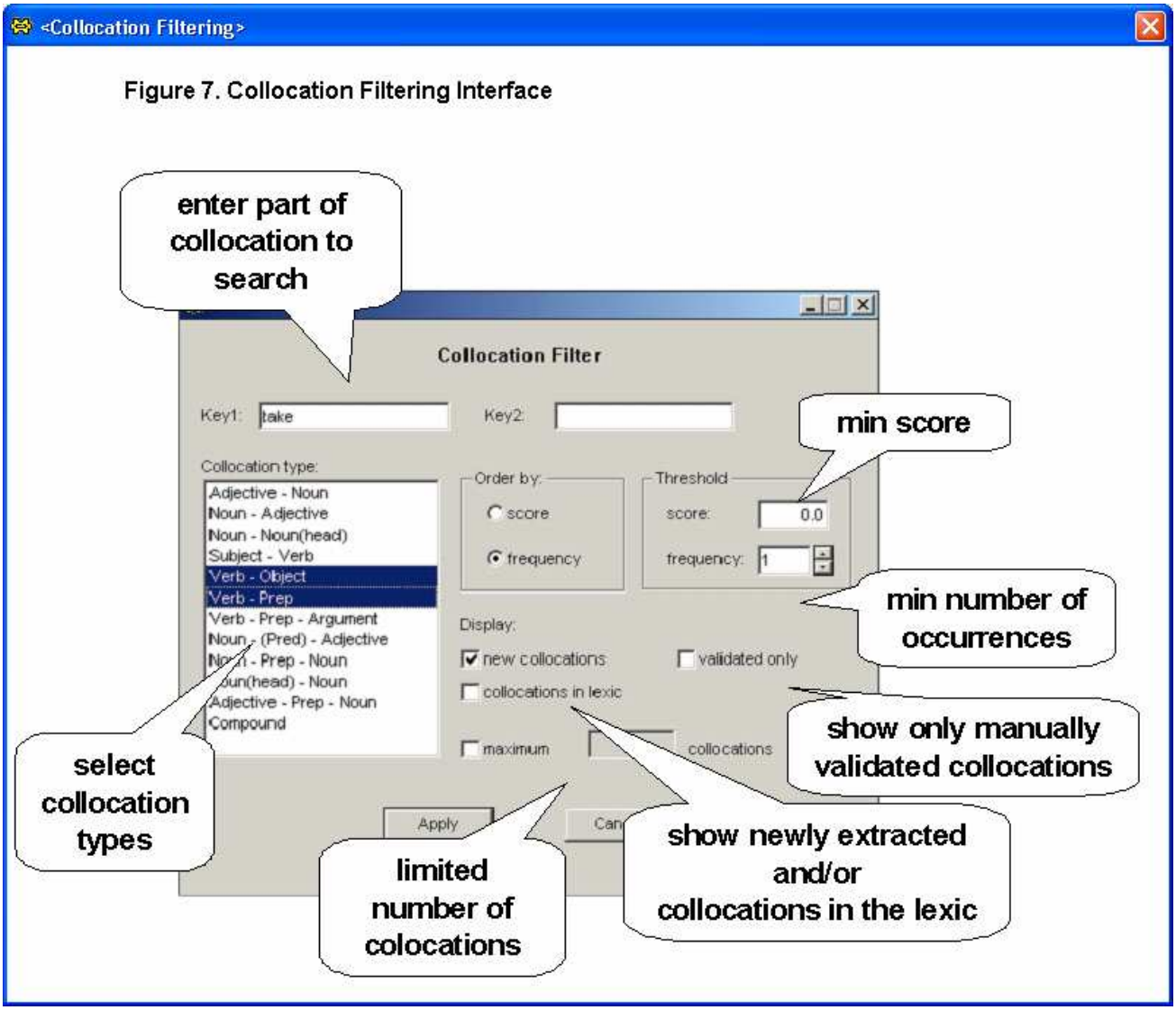
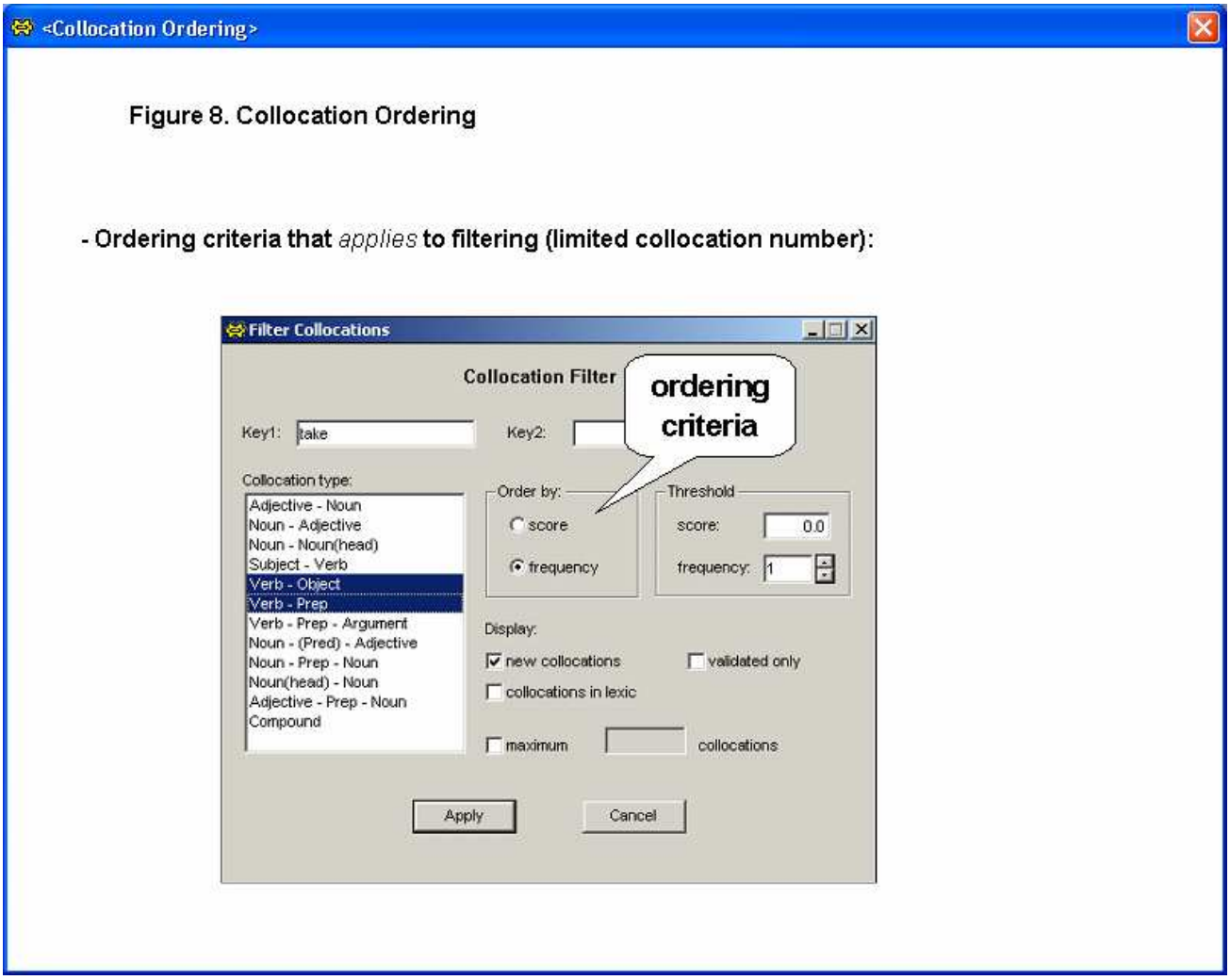


Figure 7. Collocation Filtering Interface


<Collocation Ordering>
Figure 8. Collocation Ordering
- Ordering criteria that applies to filtering (limited collocation number):
Filter Collocations
Collocation Filter
ordering criteria
Key1: take
Key2:
Collocation type:
Adjective - Noun
Noun - Adjective
Noun - Noun(head)
Subject - Verb
Verb - Object
Verb - Prep
Verb - Prep - Argument
Noun - (Pred) - Adjective
Noun - Prep - Noun
Noun(head) - Noun
Adjective - Prep - Noun
Compound
Order by:
score
frequency
Threshold
score: 0.0
frequency: 1
Display:
new collocations
validated only
collocations in lexic
maximum
collocations
Apply
Cancel

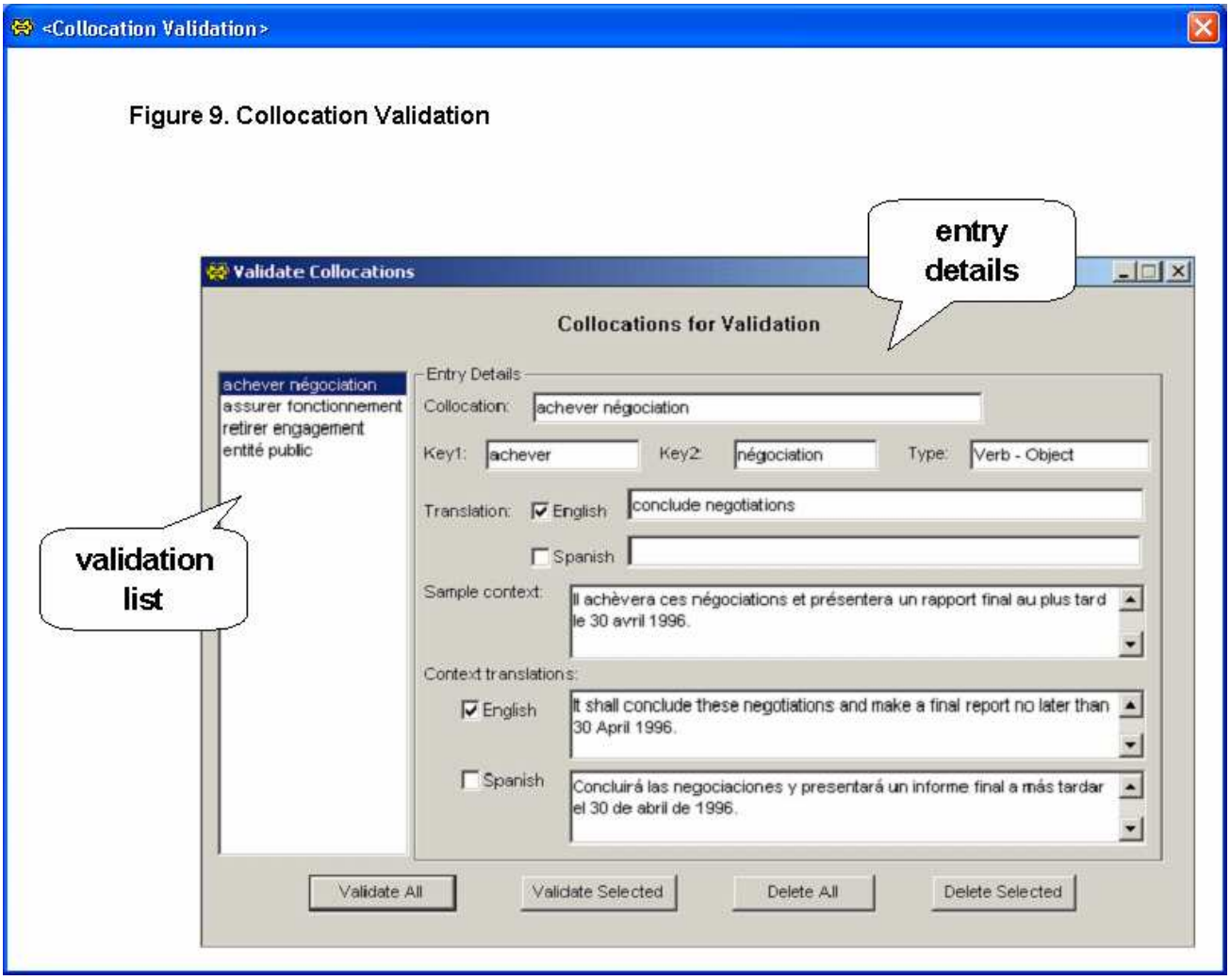


Figure 9. Collocation Validation

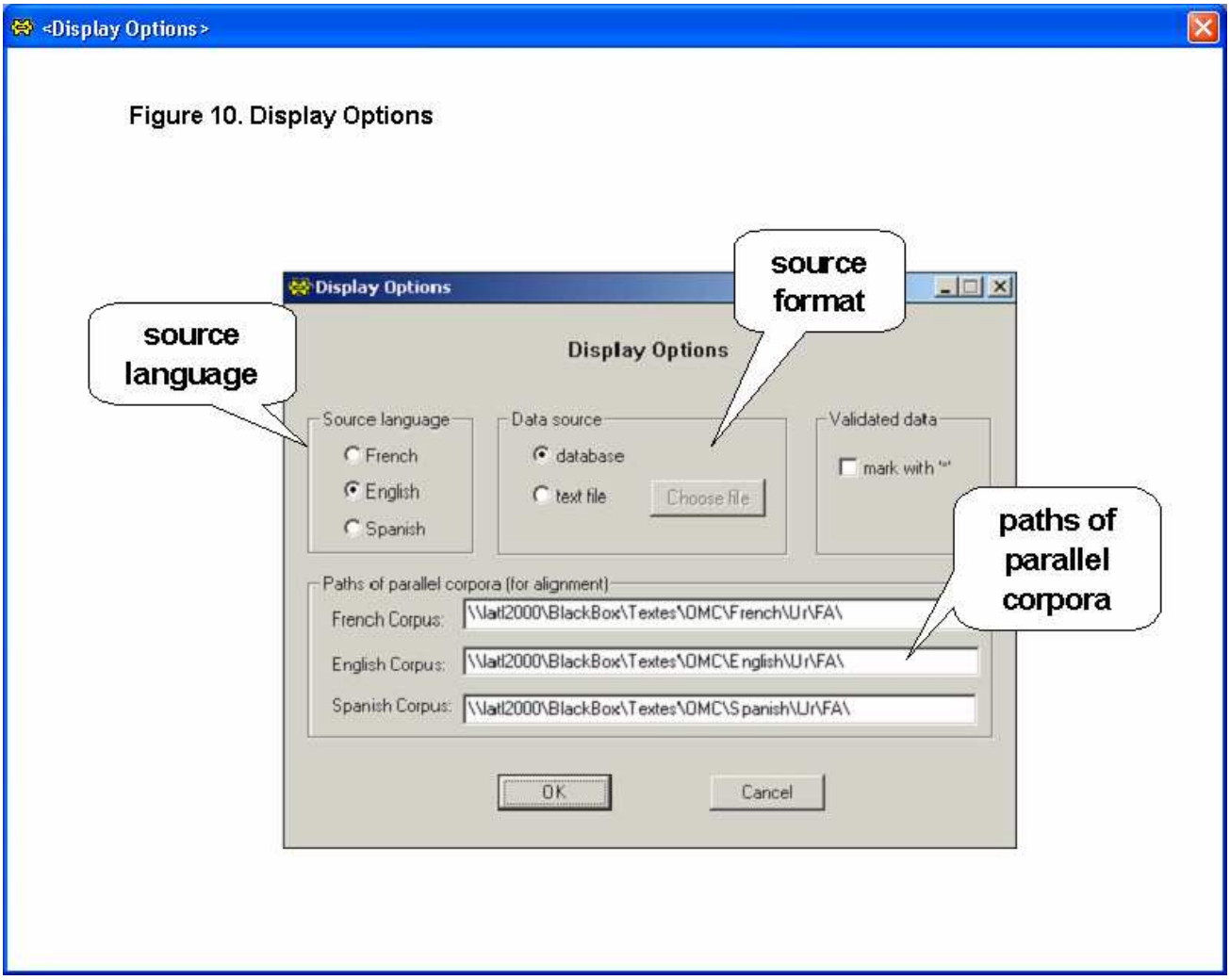


Figure 10. Display Options

1.4 Manuel du développeur

Emplacement: **Collocation/Docu/Dev-Man.odc**

V.Seretan, Violeta.Seretan@lettres.unige.ch

Collocation Extraction and Visualization

Developper Manual

November, 2004

Contents

Overview
Architecture
Relations with other subsystems
Compilation
Open the interface
Terminology

See also: User Manual, Subystem Map

Overview

This document describes the Collocation subsystem, a collection of modules that deal with the collocation extraction and visualization in parallel corpora. The extraction is based on Fips parser. The Collocation subsystem imports modules of Fips, and not the vice-versa. It needs the subsystem Utils, that allows the execution of Fips on a corpus of files (rather than on a single file) and that also implements useful general procedures.

Collocation subsystem mainly does the following:

- execute the Fips parser on a selection of files, extract collocation and store the collocation data either in a file or in a database;

- read collocation data;

- apply different filters on collocation data;

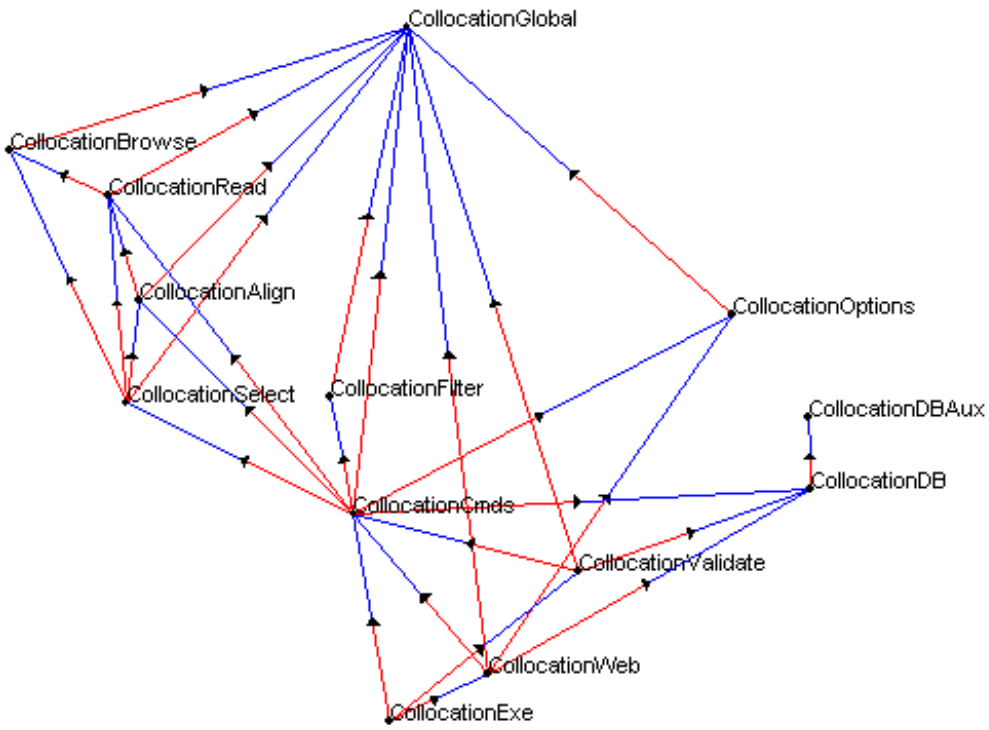
- visualize collocation with a concordance and an alignment tool implemented in the same subsystem;

- allow a user to validate a series of interesting collocations and to store them in the database.

Back

Architecture

The relationship between the subsystem's modules is shown below (the directed arrow means the module at the arrow source imports, i.e., uses, depends on, the module at the target).



The **root modules** (the main modules, not imported) are CollocationCmds, CollocationValidate and CollocationWeb.

CollocationExe is used just to gather the root modules into a single compilation unit; this facilitates the creation of exe applications.

CollocationWeb implements the extraction of collocates from the Web using the Google APIs. This module uses other modules for collocation extraction (from Google's results) and visualization.

CollocationCmds is the most important module of the subsystem. It contains the main GUI commands and controls the visualization.

CollocationValidate supports the creation of a database of manually validated collocations during the visualization with the concordance and alignment tools.

CollocationSelect implements the actions to execute when the users selects a collocation displayed by the visualization tools: show its features, context, source file, etc.

CollocationAlign implements the sentence alignment between source and target documents in the parallel corpus (in the 3 languages currently taken into account: Fr, En, Sp).

CollocationBrowse implements the functions of browsing through all the instances of the currently selected collocation in the visualization tools (sets the index that uniquely identifies the collocation instance and enable/disable browsing buttons First, Prev, Next, Last).

CollocationDB is used to support the interface of Collocation subsystem to the associated database and to perform the SQL queries for the collocation score computation. CollocationDBAux is an auxiliary module used in SQL queries during the collocation score computation.

CollocationOptions saves and retrieves the options of collocation subsystem, stored as txt files.

CollocationFilter applies filters, on several criteria, on the collocations extracted in order to allow only a part of them to be visualized.

CollocationRead read (possibly filtered) collocations either from the associated database of from an ASCII file, and builds the collocation list on the interface form of visualization tools.

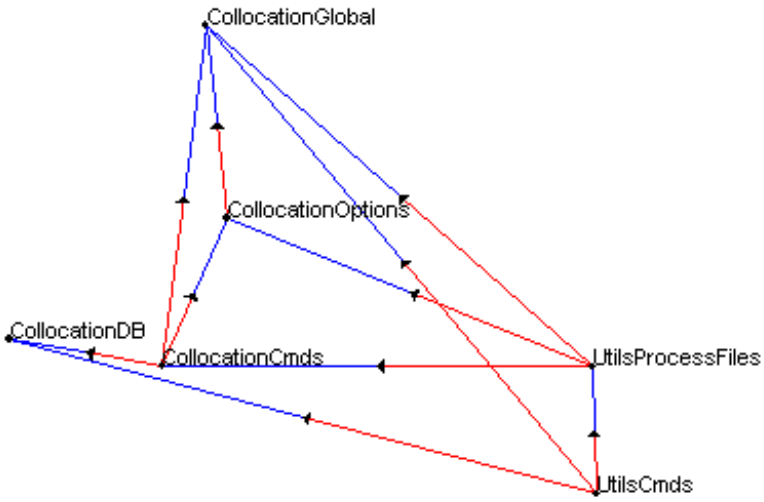
CollocationGlobal contains the main global variables of teh system, through which the modules communicate, as well as guard procedures for these variables.

Back

Relationship with other subsystems

1. Collocation and Utils

The relationship between the modules of Collocation and Utils subsystem is shown below.



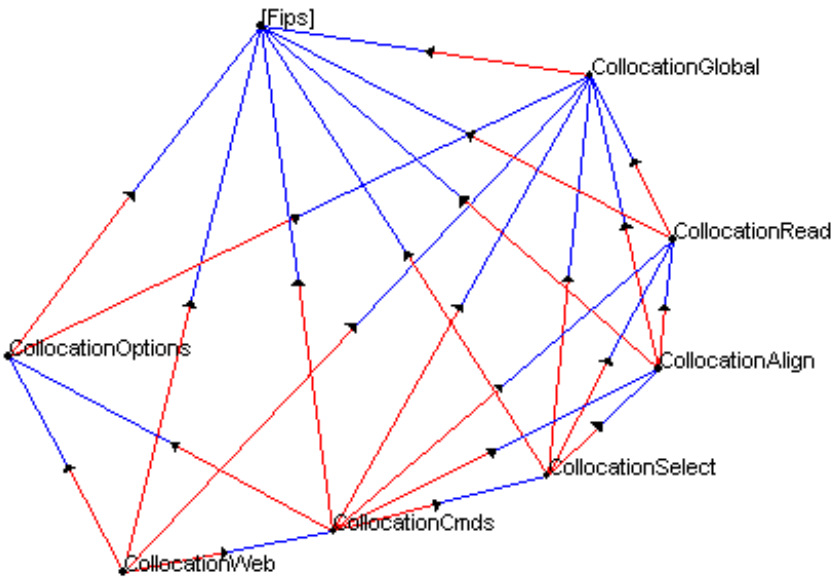
UtilsProcessFiles imports modules of Collocation subsystem since it performs the collocation extraction on files selected with Utils Multiple File Selection tool.

UtilsCmds does the same for setting up the database used in collocation extraction.

There is no import from *Collocation* subsystem to *Utils* subsystem, except for the module UtilsFunctions (with general-use procedures).

2. Collocation and Fips

The relationship between the modules of Collocation and Fips subsystem is shown below.



CollocationWeb imports Fips in order to run the Fips Term Extractor tool.

CollocationCmds imports Fips in order to get the Fips parser options (to know where the Fips Term Extractor tool stored its results).

CollocationOptions imports the module FipsXMLTools to store the options files in XML format.

CollocationSelect uses constants defined in FipsConst to know whether the current collocation is a compound or an idiom.

CollocationAlign uses procedures for tagged text processing from FipsXMLTools.

CollocationRead calls a procedure from FipsCmds to clean the source (and target) file it must display; the same procedure was called at extraction time by Fips Term Extractor, therefore it has to be called again at display time.

CollocationGlobal uses constants defined in FipsConst in order to set up the list of possible syntactic types for collocations.

There is no import from Fips subsystem to Collocation subsystem.

Back

Compilation

To compile:

⇨

❗ DevCompiler.CompileThis
CollocationGlobal CollocationOptions CollocationFilter CollocationBrowse CollocationRead CollocationAlign CollocationSelect CollocationDBAux CollocationDB CollocationCmds CollocationWeb CollocationValidate CollocationExe ◢

⇦

Back

Open the interface

Use the command below to open the interface of the tool:

❗"UtilsCmds.OnTermExtractor"

Back

Terminology

The following terms (explained below) are often used in the documentation of modules of Collocation subsystem.

alignment- method of finding the equivalent of a piece of text in the parallel corpora.

alignment tool (for collocations) - a visualization tool that displays both source and target collocation contexts for the items of the collocation list.

associated database - the database associated with the subsystem. Stores the extracted and validated collocations. Allows the score computation and collocation filtering. Must be configured as a datasource in the operating system. The datasource name set by the program by default is "cooc".

cooccurrence - collocation candidate (word pair extracted from text, having a collocation score assigned).

collocation context - the phrase in which the collocation (instance) occurs, i.e., where it has been extracted from.

collocation (collocation instance) - an occurrence of a collocation type in corpus. A collocation may occur several times, i.e., have multiple instances in the corpus.

collocations (collocation data) - collocation information (candidates extracted and ranked by a score).

collocation list - list of collocations in visualization (multiple instances included).

collocation type - a collocation seen as a lexeme pair, as opposed to a collocation instance.

collocation type list - list of collocations types in visualization (no instances).

concordance tool (for collocations) - a visualization tool that displays the source collocation context for the items of the collocation list.

currently selected collocation - the instance currently chosen in the visualization tool, for the currently selected collocation type.

currently selected collocation type - the collocation type selected in the collocation list of the visualization tool.

lexicalized collocation - a collocation that belongs to the parser's lexicon.

parallel corpora - corpora with versions of documents in several languages (here, Fr, En and Sp).

source context - the collocation context.

source corpus - the documents from which the collocation extraction has been done.

source file - the file from which the collocation (instance) has been extracted.

source filename (of a collocation) - the filename of the source file.

source language - the language of collocation data.

source window - the window showing the content of the source file.

subsystem - the program (the group of modules that all deal with the task described).

target context - the equivalent of the source context in the target file (obtained through alignment).

target corpus - the version of the source corpus in the target language.

target file - the version of the source file in the target language.

target language - the language used in the alignment tool to display the target context.

target window - the window showing the content of the target file.

validate (a collocation) - add information and store the collocation entry into a validation table.

validation table - a table with collocations manually validated by the user. Table names: MonolingualCooc and BilingualCooc.

visualization tools - the concordance and alignment tools.

Back

1.5 Manuel du développeur - Modules

Emplacement: **Collocation/Docu/**

CollocationAlign

Last updated Nov 05, 2004 (0006), Violeta.Seretan@latl.unige.ch

DEFINITION CollocationAlign;

IMPORT CollocationGlobal, TextModels;

PROCEDURE AlignCooc (cooc: CollocationGlobal.CoocRec; OUT posBeg, posEnd: INTEGER);
PROCEDURE GetSentenceLimits (file: TextModels.Model; crt: INTEGER; OUT sentBeg, sentEnd: INTEGER);

END CollocationAlign.

This modules implements the procedures that determine the translation equivalent for a sentence in the source file that represents the context of a collocation, i.e., they do the collocation contexts alignment.

PROCEDURE **AlignCooc** (cooc: CollocationGlobal.CoocRec; OUT posBeg, posEnd: INTEGER);
Aligns the source sentence that contains the collocation with the corresponding sentence in the target file.
First the paragraphs are aligned then the sentence inside the paragraphs.
The paragraph-level alignment method determine the initial target candidate ("pivot") using the relative documents lengths, then look in its neighborhood for a better candidate. Two criteria are used, one content based (relies on the matching of paragraph numbering or identification), the other length-based (relies on the matching of relative paragraph sizes in a context).
The sentence-level alignment method does for now a simple sentence-to-sentence correspondence.

PROCEDURE **GetSentenceLimits** (file: TextModels.Model; crt: INTEGER; OUT sentBeg, sentEnd: INTEGER);
Given a file and a position in that file, return the limits of the sentence around that position.

CollocationBrowse

Last updated Nov 05, 2004 (0015), Violeta.Seretan@latl.unige.ch

```
DEFINITION CollocationBrowse;

   IMPORT Dialog, CollocationGlobal;

   VAR
      firstCooc: INTEGER;
      lastCooc: INTEGER;
      nextCooc: INTEGER;
      prevCooc: INTEGER;

   PROCEDURE ComputeIndexes;
   PROCEDURE CountInstances (crt: INTEGER): INTEGER;
   PROCEDURE CrtCoocGuard (VAR par: Dialog.Par);
   PROCEDURE EqualCoocs (cooc1, cooc2: CollocationGlobal.CoocRec): BOOLEAN;
   PROCEDURE FindIth (index: INTEGER): INTEGER;
   PROCEDURE NbInstancesGuard (VAR par: Dialog.Par);
   PROCEDURE OnCoocFirstGuard (VAR par: Dialog.Par);
   PROCEDURE OnCoocLastGuard (VAR par: Dialog.Par);
   PROCEDURE OnCoocNextGuard (VAR par: Dialog.Par);
   PROCEDURE OnCoocPrevGuard (VAR par: Dialog.Par);

END CollocationBrowse.
```

This modules implements the procedures of navigation through the instances of the currently selected collocation. It allows the sequential and direct access to all instances of a collocation. It also implements guards for the access buttons (First, Prev, Next, Last) and field (number of instance) on the associated GUIs (Concordance.odc and Alignment.odc).

VAR **firstCooc**: INTEGER
The index (in the entire array of collocations) of the first instance of currently selected collocation.

VAR **lastCooc**: INTEGER
The index (in the entire array of collocations) of the last instance of currently selected collocation.

VAR **nextCooc**: INTEGER
The index (in the entire array of collocations) of the next instance of currently selected collocation.

VAR **prevCooc**: INTEGER
The index (in the entire array of collocations) of the previous instance of currently selected collocation.

PROCEDURE **ComputeIndexes**
Determine the indexes of the first, last, previous and next instance of the currently selected collocation.

PROCEDURE **CountInstances** (crt: INTEGER): INTEGER
Return the number of instances of the currently selected collocation.

PROCEDURE **CrtCoocGuard** (VAR par: Dialog.Par)
Guard for the field of direct access to the instance with a given number. Shows the current instance number.

PROCEDURE **EqualCoocs** (cooc1, cooc2: CollocationGlobal.CoocRec): BOOLEAN
Predicate that is TRUE iff the two collocations have the same keys indexes, same type and same preposition.

PROCEDURE **FindIth** (index: INTEGER): INTEGER
Find index-th instance of currently selected collocation. Assign it to the current collocation.

PROCEDURE **NbInstancesGuard** (VAR par: Dialog.Par)

Guard for the caption field in the interface, displays the total number of instances of the currently selected collocation.

PROCEDURE **OnCoocFirstGuard** (VAR par: Dialog.Par)
Guard for the 'First' browse button. Disable it under certain conditions (no other collocation instance before; empty collocation list; only one instance).

PROCEDURE **OnCoocLastGuard** (VAR par: Dialog.Par)
Guard for the 'Last' browse button. Disable it under certain conditions.

PROCEDURE **OnCoocNextGuard** (VAR par: Dialog.Par)
Guard for the 'Next' browse button. Disable it under certain conditions.

PROCEDURE **OnCoocPrevGuard** (VAR par: Dialog.Par)
Guard for the 'Prev' browse button. Disable it under certain conditions.

CollocationCmds

Last updated Nov 09, 2004 (0032), Violeta.Seretan@latl.unige.ch

```
DEFINITION CollocationCmds;

   IMPORT Dialog;

   VAR
      scoreOK: BOOLEAN;

   PROCEDURE ComputeScore;
   PROCEDURE FilterGuard (VAR par: Dialog.Par);
   PROCEDURE FormatGuard (VAR par: Dialog.Par);
   PROCEDURE OKApplyGuard (VAR par: Dialog.Par);
   PROCEDURE OnCoocAlignment;
   PROCEDURE OnCoocConcordance;
   PROCEDURE OnDisplayOK;
   PROCEDURE OnFilter;
   PROCEDURE OnFilterOK;
   PROCEDURE OnSortAlph;
   PROCEDURE OpenSourceFile;
   PROCEDURE OpenTargetFile;
   PROCEDURE SortGuard (VAR par: Dialog.Par);
   PROCEDURE TargetLanguageNotifier (op, from, to: INTEGER);

END CollocationCmds.
```

This module contains the main commands of the Collocation subsystem that implements the collocation visualization (concordance and alignment tools) and validation.

VAR **scoreOK**: BOOLEAN
TRUE iff the collocation score computed ok.

PROCEDURE **ComputeScore**
Action for menu item Collocation - Compute Score. Compute the collocation score on the collocations extracted in the current language of Fips.

PROCEDURE **FilterGuard** (VAR par: Dialog.Par)
Guard for button 'Filter' on the visualization forms Concordance.odc and Alignment.odc. Disable the button if collocation data is read from an ASCII file.

PROCEDURE **FormatGuard** (VAR par: Dialog.Par)

PROCEDURE **OKApplyGuard** (VAR par: Dialog.Par)
Guard for button 'OK/Apply' on the GUI forms 'Display.odc', 'Filter.odc'. Set its label to 'Apply' if at least one visualization tool is open (Concordance, Alignment). Set to 'OK' otherwise.

PROCEDURE **OnCoocAlignment**
Action for menu item Collocation - Alignment. Start the alignment tool: read collocation data, build the collocation dialog list, open the 'Alignment.odc' dialog.

PROCEDURE **OnCoocConcordance**
Action for menu item Collocation - Concordance. Start the concordance tool: read collocation data, build the collocation dialog list, open 'Concordance.odc' dialog.

PROCEDURE **OnDisplayOK**
Action for button 'OK' on GUI form 'Display.odc'. Save the display parameters and restart the visualization tools that were already open.

PROCEDURE **OnFilter**
Action for menu item Collocation - Filter and for button 'Filter' on the visualization tools. Read display parameters and open 'Filter.odc' dialog.

PROCEDURE **OnFilterOK**
Action for button 'OK/Apply' on the GUI form 'Filter.odc'. Save the filter parameters and restart the visualization tools that were already open.

PROCEDURE **OnSortAlph**
Action for button 'Sort alphabetically' on the visualization forms. Restart the visualization tools with the parameter 'CG.param.sort_alphabetically' set to TRUE. The current collocation table, already filtered, will be ordered alphabetically.

PROCEDURE **OpenSourceFile**
Action for button 'Open source file ... ' on the visualization forms. Open the source file of the currently selected collocation and highlight the source context.

PROCEDURE **OpenTargetFile**
Action for button 'Open target file ... ' on the visualization forms. Align the context of currently selected collocation. Open the target file and display the target context.

PROCEDURE **SortGuard** (VAR par: Dialog.Par)
Guard for the button 'Sort Alphabetically' on the visualization forms. Disable it if the collocation data is read from an ASCII file, not from a table.

PROCEDURE **TargetLanguageNotifier** (op, from, to: INTEGER)
Notifier for the radio buttons on the visualization forms that allow to choose the target language between several alternatives. Set the target language, align the context of currently selected collocation. Open the target document if needed and highlight the target source context.

CollocationDB

Last updated Nov 10, 2004 (0033), Violeta.Seretan@latl.unige.ch

```
DEFINITION CollocationDB;

   IMPORT SqlDB, Dialog, Files;

   TYPE
      Entry = RECORD
         cooc: Dialog.String;
         index1, index2, typeInt: INTEGER;
         typeStr, translation: Dialog.String;
         context1, context2: ARRAY 1000 OF CHAR;
         sourceLanguage, targetLanguage, sourceFile, targetFile: Files.Name;
         posLex1, posLex2, contextPos1, contextPos2: INTEGER
      END;

   VAR
      compte: RECORD
         n: INTEGER
      END;
      stored1: INTEGER;
      stored2: INTEGER;

   PROCEDURE Clear (VAR db: SqlDB.Database);
   PROCEDURE ClearAllTemp (VAR db: SqlDB.Database);
   PROCEDURE ComputeLogRatioScore (VAR db: SqlDB.Database; languageIndex: INTEGER);
   PROCEDURE Connect (IN name: ARRAY OF CHAR; VAR db: SqlDB.Database; OUT res: INTEGER);
   PROCEDURE CoocContextInBilingualDatabase (db: SqlDB.Database; index1, index2: INTEGER; IN
sourceLanguage, targetLanguage, sourceFile, targetFile: ARRAY OF CHAR; pos1, pos2: INTEGER):
BOOLEAN;
   PROCEDURE FillFcontingence (VAR db: SqlDB.Database; IN inputCoocTable: ARRAY OF CHAR);
   PROCEDURE FillFcooccooc (VAR db: SqlDB.Database);
   PROCEDURE FillFcoocwithkeys (VAR db: SqlDB.Database; languageIndex: INTEGER);
   PROCEDURE FillFscore (VAR db: SqlDB.Database);
   PROCEDURE FillFtemp (VAR db: SqlDB.Database; IN inputCoocTable: ARRAY OF CHAR);
   PROCEDURE StoreBilingualEntry (e: Entry; db: SqlDB.Database; silent: BOOLEAN);
   PROCEDURE StoreMonolingualEntry (e: Entry; db: SqlDB.Database; silent: BOOLEAN);

END CollocationDB.
```

This module contains procedures that support the interface of Collocation subsystem to the associated database and perform SQL queries for the collocation score computation.

TYPE **Entry** = RECORD

```
      cooc: Dialog.String;
      index1, index2, typeInt: INTEGER;
      typeStr, translation: Dialog.String;
      context1, context2: ARRAY 1000 OF CHAR;
      sourceLanguage, targetLanguage, sourceFile, targetFile: Files.Name;
      posLex1, posLex2, contextPos1, contextPos2: INTEGER
   END;
```

Type that represents a collocation (monolingual or bilingual) to validate.

cooc: Dialog.String;
Key of the entry (a string of characters).

index1, **index2**: INTEGER;
Indexes of the two lexemes of the entry (as defined in the parser's lexicon).

typeInt: INTEGER;
A numeric code representing the syntactic type of the entry (see CollocationGlobal.coocTypes).

typeStr: Dialog.String;
The syntactic type of the entry (as a string of characters).

translation: Dialog.String;
The translation proposed for the entry.

context1, **context2**: ARRAY 1000 OF CHAR;
Sample usage of the collocation in source and target languages, extracted from corpus.

sourceLanguage, targetLanguage: Files.Name;
Names of source and target language of the (bilingual) entry.

sourceFile, targetFile: Files.Name;
Names of source and target files from which the contexts have been extracted.

posLex1, posLex2: INTEGER
File position of the two lexemes of the entry in the source file.

VAR **compte**: RECORD
    n: INTEGER
  END;

Variable that receives the number of rows in a table after a call table.Read.

VAR **stored1, stored2**: INTEGER;
Number of monolingual and bilingual entries stored at the validation of collocations.

PROCEDURE **Clear** (VAR db: SqlDB.Database);
Clear the following tables from the given database: f_cooc, e_cooc, s_cooc (the cooccurrences extracted).

PROCEDURE **ClearAllTemp** (VAR db: SqlDB.Database);
Clear all the tables used for collocation score computation.

PROCEDURE **ComputeLogRatioScore** (VAR db: SqlDB.Database; languageIndex: INTEGER);
Perform the computation of log likelihood collocation score for the cooccurrences extracted for a given language. The computation uses a series of SQL queries implemented by different procedures of this module.

PROCEDURE **Connect** (IN name: ARRAY OF CHAR; VAR db: SqlDB.Database; OUT res: INTEGER);
Establish a connection to a database identified as a system datasource with the name 'name'.
Return a variable db that points to the database, if the connection was possible; otherwise, db is NIL.
Set res to the result of the operation: 0 if success, an error number if failure.

PROCEDURE **CoocContextInBilingualDatabase** (db: SqlDB.Database; index1, index2: INTEGER; IN sourceLanguage, targetLanguage, sourceFile, targetFile: ARRAY OF CHAR; pos1, pos2: INTEGER): BOOLEAN;
Returns TRUE iff in the table **BilingualCooc** of the associated database another entry is found that has the same lexeme indexes, source and target language, source and target file, and position in these files, as the given entry.

PROCEDURE **FillFcontingence** (VAR db: SqlDB.Database; IN inputCoocTable: ARRAY OF CHAR);
Used for collocation score computation. Add in the table **contingence** of database db the values for the contingency table for each co-occurrence: joint and marginal lexemes frequencies.

PROCEDURE **FillFcooccooc** (VAR db: SqlDB.Database);
Used for collocation score computation. Add in the table **coocooc** of database db information about the number of occurrence of each lexeme in **temp** table. These numbers are the marginal frequencies of lexemes.

PROCEDURE **FillFcoocwithkeys** (VAR db: SqlDB.Database; languageIndex: INTEGER);
Used for collocation score computation. Add in the table **coocwithkeys** of database db information about the frequency of each co-occurrence, by counting rows in the SQL query result **coocwithprep** for the given language. These numbers are the joint frequencies of lexemes.

PROCEDURE **FillFscore** (VAR db: SqlDB.Database);
Used for collocation score computation. Add in the table **coocscore** of database db thelog likelihood values for each contingency table values (rows in table contingence).

PROCEDURE **FillFtemp** (VAR db: SqlDB.Database; IN inputCoocTable: ARRAY OF CHAR);
Used for collocation score computation. Add in the table **temp** of database db the list of lexemes from the table given. Consider only the lexemes that are not part of a lexicalized term (collocation, compound or idiom).

PROCEDURE **StoreBilingualEntry** (e: Entry; db: SqlDB.Database; silent: BOOLEAN);
Given the collocation entry e, store it in the database db, in its table BilingualCooc. If silent is TRUE, do not show the result of the operation (collocation stored or not).

PROCEDURE **StoreMonolingualEntry** (e: Entry; db: SqlDB.Database; silent: BOOLEAN);
Given the collocation entry e, store it in the database db, in its table MonolingualCooc. If silent is TRUE, do not show the result of the operation (collocation stored or not).

CollocationDBAux

Last updated Nov 10, 2004 (0034), Violeta.Seretan@latl.unige.ch

```
DEFINITION CollocationDBAux;

   VAR
      nbocc: INTEGER;
      val: ARRAY 25 OF CHAR;

END CollocationDBAux.
```

Auxiliary module for CollocationDB module. Useful for some SQL operations during the collocation score computation.

```
VAR
   nbocc: INTEGER;
   val: ARRAY 25 OF CHAR;
```

Used to add variable values to database tables using INSERT clause.

CollocationExe

Last updated Nov 05, 2004 (0007), Violeta.Seretan@latl.unige.ch

DEFINITION CollocationExe;

END CollocationExe.

This is an empty module that imports the root modules of Utils and Collocation subsystems, in order to facilitate the creation of the final application. Root modules: UtilsEdit, UtilsCmds, CollocationCmds, CollocationValidate, CollocationWeb.

CollocationFilter

Last updated Nov 05, 2004 (0050), Violeta.Seretan@latl.unige.ch

```
DEFINITION CollocationFilter;

   IMPORT Files;

   VAR
      record: LexicalizedRecord;

   PROCEDURE ApplyFilterOptions (OUT coocFiltered: BOOLEAN);

END CollocationFilter.
```

Module that deals with the filtering of extracted collocations according to the filter options defined in the GUI form 'Filter.odc'. It mainly adds the filtered collocations to an output table in the associated database 'CollocationGlobal.cocDB'. It finally indicated what is the name of the output table by setting the variable 'CollocationGlobal.coocTable' to this table's name.

VAR **record**: LexicalizedRecord
Exported variable used to read a record from a table.

PROCEDURE **ApplyFilterOptions** (OUT coocFiltered: BOOLEAN)
Apply the filter according to 'CG.coocFilterOptions' and set the name of output table.

CollocationGlobal

Last updated Feb 02, 2005 (0115) , Violeta.Seretan@latl.unige.ch

```
DEFINITION CollocationGlobal;

   IMPORT Dialog, Files, SqlDB, LatlIO, TextMappers, TextModels, Views, TextViews, Windows;

   CONST
      ASCIIFile = 1;
      DBInfo0 = "Cooccurrence present in database";
      DBInfo1 = "Cooccurrence context present in database";
      DBInfo2 = "Cooccurrence contexts present in database";
      Database = 0;
      nbTypes = 12;
      nb_lang = 3;

   TYPE
      CoocIndexRec = RECORD
         index1, index2: INTEGER;
         key1, prep, key2: Dialog.String;
         validated: BOOLEAN;
         type: INTEGER
      END;

      CoocParam = RECORD
         terms: Dialog.List;
         crtFileName: Dialog.String;
         fileSymbol: ARRAY 256 OF CHAR;
         fileCollection: ARRAY 256 OF CHAR;
         sel_lang, sort: INTEGER;
         sort_alphabetically: BOOLEAN;
         instanceNb: INTEGER;
         DBInfo: ARRAY 256 OF CHAR;
         markValid: BOOLEAN
      END;

      CoocRec = RECORD
         index1, index2, type, posCharLex1, posCharLex2, posCharSentence, occ: INTEGER;
         score: REAL;
         key1, prep, key2, inFile: Dialog.String;
         lexicalized, validated, contextValidated: BOOLEAN
      END;

      FilterOptions = RECORD
         type: Dialog.Selection;
         freq: INTEGER;
         key1, key2: Dialog.String;
         score: REAL;
         max: INTEGER;
         limitedNb, validOnly, includeLexicalized, includeExtracted: BOOLEAN
      END;

   VAR
      coocArray: POINTER TO ARRAY OF CoocRec;
      coocDB: SqlDB.Database;
      coocDBName: Files.Name;
      coocFile: LatlIO.Text;
      coocFilterOptions: FilterOptions;
```

```
    coocIndexArray: POINTER TO ARRAY OF CoocIndexRec;
    coocSource: INTEGER;
    coocTable: Files.Name;
    coocTypes: ARRAY 12 OF RECORD
          str: Dialog.String;
          id: INTEGER
       END;
    corpora_paths: ARRAY 3 OF Dialog.String;
    crtCooc: INTEGER;
    dataPath: ARRAY 256 OF CHAR;
    f_src: TextMappers.Formatter;
    f_trg: TextMappers.Formatter;
    flagAlign: BOOLEAN;
    flagSelectEach: BOOLEAN;
    languages: ARRAY 3 OF ARRAY 30 OF CHAR;
    nbCooc: INTEGER;
    nbCoocUnique: INTEGER;
    nbSameCooc: INTEGER;
    ordered: BOOLEAN;
    param: CoocParam;
    path: ARRAY 256 OF CHAR;
    readOK: BOOLEAN;
    realCoocIndex: POINTER TO ARRAY OF INTEGER;
    shortFileName: Files.Name;
    sourceContextBeg: INTEGER;
    sourceContextEnd: INTEGER;
    sourceFileName: Files.Name;
    source_lang: INTEGER;
    srcName: Files.Name;
    srcViewOnForm: TextViews.View;
    t_src: TextModels.Model;
    t_trg: TextModels.Model;
    targetContextBeg: INTEGER;
    targetContextEnd: INTEGER;
    targetFileName: Files.Name;
    target_lang: INTEGER;
    trgName: Files.Name;
    trgViewOnForm: TextViews.View;
    v_src: Views.View;
    v_trg: Views.View;
    w_src: Windows.Window;
    w_trg: Windows.Window;

PROCEDURE ClearViewsOnForm;
PROCEDURE ClearVisualizationParameters;
PROCEDURE CollectionGuard (VAR par: Dialog.Par);
PROCEDURE CoocTypeGuard (VAR par: Dialog.Par);
PROCEDURE CorporaPathsGuard (VAR par: Dialog.Par);
PROCEDURE CorpusFreqGuard (VAR par: Dialog.Par);
PROCEDURE DBInfoGuard (VAR par: Dialog.Par);
PROCEDURE FileContentGuard (VAR par: Dialog.Par);
PROCEDURE FilenameGuard (VAR par: Dialog.Par);
PROCEDURE FilterMaxGuard (VAR par: Dialog.Par);
PROCEDURE GetSourceDir (OUT dirName: Files.Name);
PROCEDURE HideGuard (VAR par: Dialog.Par);
PROCEDURE IdOfType (id: INTEGER; OUT res: INTEGER): INTEGER;
PROCEDURE InitTypeList (OUT list: Dialog.Selection);
PROCEDURE IsValidType (id: INTEGER): BOOLEAN;
PROCEDURE LangTarget1Guard (VAR par: Dialog.Par);
PROCEDURE LangTarget2Guard (VAR par: Dialog.Par);
```

```
    PROCEDURE LanguageGuard (VAR par: Dialog.Par);
    PROCEDURE LexicalizedGuard (VAR par: Dialog.Par);
    PROCEDURE ScoreGuard (VAR par: Dialog.Par);
    PROCEDURE SourceFileGuard (VAR par: Dialog.Par);
    PROCEDURE SymbolGuard (VAR par: Dialog.Par);
    PROCEDURE TargetFileGuard (VAR par: Dialog.Par);

END CollocationGlobal.
```

This module contains the main global variables used by the modules of Collocation subsystem (which basically does the extraction of collocations from texts and the visualization of the results in parallel documents).
The variables of this module are used mostly in the visualization part. The main variables are: the list of collocations to be displayed (coocArray), and the options for visualization (filtering options, interface variables).
Most of the procedures are GUI guards for the global variables.
Associated GUIs: *Concordance.odc, Alignment.odc, Display.odc, Filter.odc.*

CONST **ASCIIFile**
The collocations are to be read from a text file.

CONST **DBInfo0** = "Cooccurrence present in database";
GUI message indicating that the collocation has been validated by the user (it is stored in the validation table, in the associated database).

CONST **DBInfo1** = "Cooccurrence context present in database";
GUI message indicating that the collocation and its source context have been validated by the user (it is stored in the validation table, in the associated database).

CONST **DBInfo2** = "Cooccurrence contexts present in database";
GUI message indicating that the collocation and its source and target contexts have been validated by the user (it is stored in the validation table, in the associated database).

CONST **Database** = 0;
The collocations are to be read from the associated database.

CONST **nbTypes** = 12;
Number of syntactic types for collocations.

CONST **nb_lang** = 3;
Number of languages in the visualization of parallel documents. This variable is used to define the parallel corpora paths and the target languages (according to a source language). For now, there are 3 languages for which parallel documents are available (French, English and Spanish).

```
TYPE CoocIndexRec = RECORD
      index1, index2: INTEGER;
      key1, prep, key2: Dialog.String;
      validated: BOOLEAN;
      type: INTEGER
END;
```

Type for representing a collocation type (as opposed to collocation instances). The collocation instances of a type will be read each time the user selects a type in the interface dialog list.

index1: INTEGER;
The index (in parser's lexicon) of the first key of collocation type.

index2: INTEGER;
The index (in parser's lexicon) of the second key of collocation type.

key1: Dialog.String;

The string for the first key of collocation type.

prep: Dialog.String;
The string for the preposition possibly occurring in the collocation type.

key2: Dialog.String;
The string for the second key of collocation type.

validated: BOOLEAN;
TRUE iff the collocation type belongs to the list of collocations validated by the user.

type: INTEGER
The syntactic type of collocation type. Possible values are found in the variable coocTypes.

```
TYPE
   CoocParam = RECORD
      terms: Dialog.List;
      crtFileName: Dialog.String;
      sourceContext: ARRAY 10010 OF CHAR;
      targetContext: ARRAY 10010 OF CHAR;
      fileSymbol: ARRAY 256 OF CHAR;
      fileCollection: ARRAY 256 OF CHAR;
      sel_lang, sort: INTEGER;
      sort_alphabetically: BOOLEAN;
      instanceNb: INTEGER;
      DBInfo: ARRAY 256 OF CHAR;
      markValid: BOOLEAN
   END;
```

The type used for representing collocation data on the visualization interfaces.

terms: Dialog.List;
The dialog list containing the string for the collocations visualized (the two keys and possibly a middle preposition).

crtFileName: Dialog.String;
The source filename of a collocation, relative to the path of the corpus to which it belongs.

fileSymbol: ARRAY 256 OF CHAR;
For the OMC corpora, an identifier of the source file. It is the value of the META attribute "SYMBOL" in the header of file in HTML format. For other types of files, this value is empty.

fileCollection: ARRAY 20 OF CHAR;
For the OMC corpora, an identifier of the collection to which the source file belongs. It is the value of the META attribute "COLLECTION" in the header of file in HTML format.

sel_lang: INTEGER;
The target language selected for visualization. One language at a time is used for visualization in the alignment tool.

sort: INTEGER;
Sort criterion for the collocation list: 0 - score; 1 - frequency. Used also for collocation filtering.

sort_alphabetically: BOOLEAN;
TRUE iff the collocation list is sorted alphabetically. Not used for collocation filtering.

instanceNb: INTEGER;
The instance number of the collocation currently showed in the visualization tool. A collocation is uniquely listed in the collocation list but it may occur several times, i.e., have multiple instances in the corpus. See also: nbSameCooc.

DBInfo: ARRAY 256 OF CHAR;
String indicating whether the currently selected collocation is in the validation table. Possible values: constants DBInfo0, DBInfo1, DBInfo2.

markValid: BOOLEAN
If TRUE, a star * will be inserted before the validated collocations in the list.

CoocRec = RECORD
  index1, index2, type, posCharLex1, posCharLex2, posCharSentence, occ: INTEGER;
  score: REAL;
  key1, prep, key2, inFile: Dialog.String
END;
The type representing a collocation.

index1, index2: INTEGER;
The indexes (in the parser's lexicon) of the two collocation keys.

type: INTEGER;
The syntactic type of a collocation. Possible values are found in the variable coocTypes.

posCharLex1, posCharLex2: INTEGER;
The (file) position of the two collocation keys in the source file.

posCharSentence: INTEGER;
The file position of the beginning of the sentence in which occurs the collocation.

occ: INTEGER;
The number of occurrences of the collocation in the source file.

score: REAL;
The collocation score calculated by UtilsOdbcScore. The bigger the score, the more likely that the two keys form a true collocation.

key1, prep, key2: Dialog.String;
The collocation string: the two keys and the optional middle preposition.

inFile: Dialog.String;
The name of the source file of the collocation.

lexicalized: BOOLEAN;
TRUE iff the collocation belongs to the parser's lexicon.

validated: BOOLEAN;
TRUE iff a collocation instance of the type index1, index2 exists in the validation table in the associated database.

contextValidated: BOOLEAN;
TRUE iff the same collocation instance (same index1, index2, posCharLex1, posCharLex2) exists in the validation table in the associated database.

TYPE **FilterOptions** = RECORD
    type: Dialog.Selection;
    freq: INTEGER;
    key1, key2: Dialog.String;
    score: REAL;
    max: INTEGER;
    limitedNb, validOnly, includeLexicalized, includeExtracted: BOOLEAN
  END;

The type for representing the filtering options (used in visualization).

type: Dialog.Selection;
Syntactic types included for visualization.

freq: INTEGER;
Frequency threshold. Only collocation types having at list freq occurrences in the corpus will be included.

key1, key2: Dialog.String;
Keys of collocation. Only collocation types having the given keys will be included. Allows the search for given collocations.

score: REAL;
Score threshold. Only collocation types having at least this score will be included.

max: INTEGER;
Number limitation. Maximum number of collocation types to be included. N.B.: it applies to collocation types, not instances. The number of instances may be much greater.

limitedNb: BOOLEAN
TRUE iff there is a number limitation.

validOnly: BOOLEAN
If TRUE, only validated collocations will be included.

includeLexicalized: BOOLEAN
If TRUE, also the lexicalized collocations will be included. Otherwise they are not included.

includeExtracted: BOOLEAN
If TRUE, the collocations extracted will be included (default case). Otherwise they are not included.

VAR **coocArray**: POINTER TO ARRAY OF CoocRec;
The array of collocation instances that will be displayed (collocation list).

VAR **coocFile**: LatlIO.Text;
The source file for the selected collocation in the list (the file where the collocation occur).

VAR **coocDB**: SqlDB.Database;
The database associated with the subsystem. Stores the extracted and validated collocations. Allows the score computation and collocation filtering.

VAR **coocDBName**: Files.Name;
Name of the associated database.

VAR **coocFile**: LatlIO.Text;
Input file (stores the extracted collocations).

VAR **coocFilterOptions**: FilterOptions;
Filtering options (shown in the GUI *Filter.odc*).

VAR **coocIndexArray**: POINTER TO ARRAY OF CoocIndexRec;
The array of collocations types that will be displayed. Does not contain collocation instances.

VAR **coocSource**: INTEGER;
Where the collocations are read from. Possible values: constants ASCIIFile, Database.

VAR **coocTable**: Files.Name;
Name of the input table, from which the collocation will be read. It changes depending on display and filtering options.

VAR **coocTypes**: ARRAY 12 OF RECORD

```
    str: Dialog.String;
    id: INTEGER
END;
```
Collocations syntactic types possible: string and ID number.
Example values: "Adjective - Noun", 0 = FipsConsts.adjectiveNoun.

VAR **corpora_paths**: ARRAY 3 OF Dialog.String;
The paths for the parallel corpora (used for displaying target collocation context).

VAR **crtCooc**: LONGINT;
The index of the currently selected collocation in the collocations list coocArray.

VAR **dataPath**: ARRAY 256 OF CHAR;
The path to data directory (where option files are stored). It is relative to the path of the subsystem.

VAR **f_src**: TextMappers.Formatter;
The formatter (writer) connected to the source file of the currently selected collocation. Allows to read text from that file.

VAR **f_trg**: TextMappers.Formatter;
The formatter (writer) connected to the target file of the currently selected collocation. Allows to read text from that file.

VAR **flagAlign**: BOOLEAN;
TRUE if the visualization tool is the alignment tool; FALSE if it is the concordance tool.

VAR **flagSelectEach**: BOOLEAN;
If TRUE, work with a collocation type list instead of collocation instances. Retrieve all the instances on-the-fly (for efficiency purpose).

VAR **languages**: ARRAY 3 OF ARRAY 30 OF CHAR;
The language names for the languages used (for parallel corpora): French, English and Spanish.

VAR **nbCooc**: LONGINT;
Total number of collocations in the collocation list.

VAR **nbCoocUnique**: INTEGER;
Total number of collocation types in the collocation list (instances not counted).

VAR **nbSameCooc**: LONGINT;
The number of instances of the currently selected collocation type.

VAR **ordered**: BOOLEAN;
TRUE if the collocation data are grouped alphabetically.

VAR **param**: CoocParam;
Collocation information used by the visualization interface.

VAR **path**: ARRAY 256 OF CHAR;
The path to the subsystem resources (by default, the working directory: the empty string).

VAR **readOK**: BOOLEAN;
TRUE iff the reading of the collocation data was ok.

VAR **realCoocIndex**: POINTER TO ARRAY OF INTEGER;
Hash table for collocation list. Contains indexes to the beginning of collocation instances groups, for each collocation type in cooclist.

VAR **shortFileName**: Files.Name;
A (possibly shortened) version of the source filename to be displayed in the interface when the filename is too

long for the corresponding field.

VAR **sourceContextBeg**: INTEGER;
File position of the beginning of the source context for the currently selected collocation.

VAR **sourceContextEnd**: INTEGER;
File position of the end of the source context for the currently selected collocation.

VAR **sourceFileName**: Files.Name;
The filename of the source file for the currently selected collocation.

VAR **source_lang**: INTEGER;
The source language of the collocation data. Possible values: the indexes of the array languages (0 for French, 1 for English, 2 for Spanish).

VAR **srcName**: Files.Name;
The source filename for the collocation last visualized.

VAR **srcViewOnForm**: TextViews.View;
The text view on the GUI showing the source file content, with the source context colored and automatically scrolled to.

VAR **t_src**: TextModels.Model;
The text model (file content) of the source file.

VAR **t_trg**: TextModels.Model;
The text model (file content) of the target file.

VAR **targetContextBeg**: INTEGER;
File position of the beginning of the target context for the currently selected collocation.

VAR **targetContextEnd**: INTEGER;
File position of the end of the target context for the currently selected collocation.

VAR **targetFileName**: Files.Name;
The filename of the target file for the currently selected collocation.

VAR **target_lang**: INTEGER;
The target language: the language used in the alignment tool to display the target context. Possible values: values of the array languages.

VAR **trgName**: Files.Name;
The target filename for the collocation last visualized.

VAR **trgViewOnForm**: TextViews.View;
The text view on the GUI showing the target file content, with the target context colored and automatically scrolled to.

VAR **v_src**: Views.View;
The view (showed by the window) of the source file model t_src. The source context is colored and automatically scrolled to.

VAR **v_trg**: Views.View;
The view (showed by the window) of the target file model t_trg. The target context is colored and automatically scrolled to.

VAR **w_src**: Windows.Window;
The window displaying the source file view v_src.

VAR **w_trg**: Windows.Window;

The window displaying the target file view v_trg.

PROCEDURE **ClearViewsOnForm**;
Empty the source and target file views on the GUI.

PROCEDURE **ClearVisualizationParameters**;
Init all parameters of current collocation used on the GUI.

PROCEDURE **CollectionGuard** (VAR par: Dialog.Par);
Display the collection identifier on the visualization GUI (*Concordance.odc* and *Alignment.odc*)

PROCEDURE **CoocTypeGuard** (VAR par: Dialog.Par);
Display the syntactic type of current collocation on the GUI.

PROCEDURE **CorporaPathsGuard** (VAR par: Dialog.Par);
Make the option for corpora pathes read-only if the visualization tool is not the alignment tool. Interface: '*Display.odc*'.

PROCEDURE **CorpusFreqGuard** (VAR par: Dialog.Par);
Display the frequency filter option on GUI *(Filter.odc).*

PROCEDURE **DBInfoGuard** (VAR par: Dialog.Par);
Display the validation information for the current collocation on the visualization GUI.

PROCEDURE **FileContentGuard** (VAR par: Dialog.Par);
Make the text field that displays context read-only. Problem: background becomes grey. Not used.

PROCEDURE **FilenameGuard** (VAR par: Dialog.Par);
Displays the short filename of the source file for the current collocation on the visualization GUI.

PROCEDURE **FilterMaxGuard** (VAR par: Dialog.Par);
Disable the control showing the maximum number of collocation in filter when there is not a number limitation. GUI: *Filter.odc*.

PROCEDURE **GetSourceDir** (OUT dirName: Files.Name);
Set the source directory name for collocation data. In this version, the data contain the full source filenames including path, therefore the source directory is set to the empty string.

PROCEDURE **HideGuard** (VAR par: Dialog.Par);
Set the label of the control to the empty string (make invisible).

PROCEDURE **IdOfType** (id: INTEGER; OUT res: INTEGER): INTEGER;
For the collocation syntactic type ID id, retrieve the index in the types list.

PROCEDURE **InitTypeList** (OUT list: Dialog.Selection);
Create the dialog list showing collocation syntactic types from the list of types typesList.

PROCEDURE **IsValidType** (id: INTEGER): BOOLEAN;
Return TRUE iff id is an existing syntactic type ID.

PROCEDURE **LangTarget1Guard** (VAR par: Dialog.Par);
Set the label of the first radio button on visualization GUI corresponding to the first target language alternative. Disable it if the source language is undefined.

PROCEDURE **LangTarget2Guard** (VAR par: Dialog.Par);
Set the label of the second radio button on visualization GUI corresponding to the second target language alternative. Disable it if the source language is undefined.

PROCEDURE **LanguageGuard** (VAR par: Dialog.Par);
Set read-only the source language radio buttons on the display options (*Display.odc*). Not used.

PROCEDURE **LexicalizedGuard** (VAR par: Dialog.Par);
Display a string in the visualization GUI indicating whether the current collocation is lexicalized ('Collocation in lexic') or is newly extracted ('New collocation').

PROCEDURE **ScoreGuard** (VAR par: Dialog.Par);
Displays the score of the current collocation on the visualization GUI. Only keep the first 3 decimals.

PROCEDURE **SymbolGuard** (VAR par: Dialog.Par);
Display the source file symbol identifier in the visualization GUI.

PROCEDURE **SortGuard** (VAR par: Dialog.Par);
Guard for the button on the interface that sorts the collocation list. Sets its label either to "Sort alphabetically" or to "Sort by score", depending on the value of variable **sort**.

PROCEDURE **SourceFileGuard** (VAR par: Dialog.Par);
Disable the button 'Open source file' on the visualization GUI when no collocation is currently selected.

PROCEDURE **TargetFileGuard** (VAR par: Dialog.Par);
Disable the button 'Open source file' on the visualization GUI when no collocation is currently selected.

CollocationOptions

Last updated Nov 05, 2004 (0054), Violeta.Seretan@latl.unige.ch

```
DEFINITION CollocationOptions;

   IMPORT CollocationGlobal;

   TYPE
      FilterParam = RECORD
         filterOpt: CollocationGlobal.FilterOptions;
         sort: INTEGER
      END;

   PROCEDURE ReadDisplayParam;
   PROCEDURE ReadFilterParam;
   PROCEDURE SaveDisplayParam;
   PROCEDURE SaveFilterParam;

END CollocationOptions.
```

Module for saving and retrieving options of collocation subsystem. The path to options files is defined in CollocationGlobal.dataPath.

TYPE
   FilterParam = RECORD
      filterOpt: CollocationGlobal.FilterOptions;
      sort: INTEGER
   END;

Type for collocation filtering and sorting options.

filterOpt: CollocationGlobal.FilterOptions;
Collocation filtering options.

sort: INTEGER
Collocation sorting options. Possible values: those of filed sort of type CollocationGlobal.CoocParam.

PROCEDURE **ReadDisplayParam**;
Read display parameters 'CollocationGlobal.param' from option file 'display.txt' file in XML format.

PROCEDURE **ReadFilterParam**;
Read filter parameters 'CollocationGlobal.coocFilterOptions' and 'CollocationGlobal.param.sort' from 'filter.txt' file in XML format.

PROCEDURE **SaveDisplayParam**;
Save display parameters 'CollocationGlobal.param' to 'display.txt' file in XML format

PROCEDURE **SaveFilterParam**;
Save filter parameters 'CollocationGlobal.coocFilterOptions' and 'CollocationGlobal.param.sort' to file 'filter.txt' in XML format.

CollocationRead

Last updated Nov 05, 2004 (0018), Violeta.Seretan@latl.unige.ch

```
DEFINITION CollocationRead;

   IMPORT Dialog, Files, TextModels, Views, TextMappers, SqlDB, CollocationGlobal;

   VAR
      dbIndexRec: DBCoocUniqueRec;
      dbRec: DBCoocRec;
      dbValidRec: DBValidRec;

   PROCEDURE BuildCollocationList (OUT string: Dialog.String);
   PROCEDURE ChooseFileGuard (VAR par: Dialog.Par);
   PROCEDURE GetTargetDocument (IN fileNameIn: Files.Name; OUT fileNameOut: Files.Name): BOOLEAN;
   PROCEDURE OnChooseFile;
   PROCEDURE OpenSourceFile (path, name: Files.Name; OUT t: TextModels.Model; OUT f:
TextMappers.Formatter; OUT v: Views.View): BOOLEAN;
   PROCEDURE ReadSelectedList (db: SqlDB.Database; coocTable, crtIndex1, crtIndex2: ARRAY OF CHAR;
OUT coocArray: POINTER TO ARRAY OF CollocationGlobal.CoocRec);

END CollocationRead.
```

This module implements procedures used to read the collocations data from an ASCII file or database table, and to build the collocation dialog list.

VAR **dbIndexRec**: DBCoocUniqueRec
Exported variable, used to receive the record read from the table with unique collocations.

VAR **dbRec**: DBCoocRec
Exported variable, used to receive the record read from the table with collocations (non unique, but containing all instances).

VAR **dbValidRec**: DBValidRec;
Exported variable, used to receive the record read from the table with validated collocations.

PROCEDURE **BuildCollocationList** (OUT string: Dialog.String)

PROCEDURE **ChooseFileGuard** (VAR par: Dialog.Par)
Guard for the button "Choose file..." in the GUI form 'DisplayOptions.odc'. Disable it if the collocation data will not be read from an ASCII file.

PROCEDURE **GetTargetDocument** (IN fileNameIn: Files.Name; OUT fileNameOut: Files.Name): BOOLEAN
Given the filename of the source document, output the name of target document (for the current target language.
Return TRUE if the source filename is compatible with the path defined for the source corpus.

PROCEDURE **OnChooseFile**
Open the text file containing collocation data. Notifier for the button "Choose file..." in the GUI form 'DisplayOptions.odc'.

PROCEDURE **OpenSourceFile** (path, name: Files.Name; OUT t: TextModels.Model; OUT f: TextMappers.Formatter; OUT v: Views.View): BOOLEAN
Open the file specified by the given path and name. Return the text content, an associated writer and a view.

PROCEDURE **ReadSelectedList** (db: SqlDB.Database; coocTable, crtIndex1, crtIndex2: ARRAY OF CHAR; OUT coocArray: POINTER TO ARRAY OF CollocationGlobal.CoocRec)
Given a collocation (crtIndex1, crtIndex2), read all its instances into the collocation array 'coocArray'. Called on selection change in the collocation dialog list.

CollocationSelect

Last updated Nov 05, 2004 (0004), Violeta Seretan

```
DEFINITION CollocationSelect;

   IMPORT Files, CollocationGlobal, LatlIO, TextMappers, TextModels;

   PROCEDURE CrtCoocNotifier (op, from, to: INTEGER);
   PROCEDURE DisplaySourceFile (relativeFileName: Files.Name; cooc: CollocationGlobal.CoocRec; open:
BOOLEAN);
   PROCEDURE DisplayTargetFile (relativeFileName: Files.Name; posBeg, posEnd: INTEGER; open:
BOOLEAN);
   PROCEDURE GetFileID (sourceFile: LatlIO.Text; IN nameID: ARRAY OF CHAR; OUT value: ARRAY OF
CHAR);
   PROCEDURE HighlightCrtCollocation (f_src: TextMappers.Formatter; t_src: TextModels.Model; cooc:
CollocationGlobal.CoocRec);
   PROCEDURE Init;
   PROCEDURE OnCoocBrowse (n: INTEGER);
   PROCEDURE OnCoocSel (op, from, to: INTEGER);

END CollocationSelect.
```

This module implements the actions which are executed when a collocation instance has been selected for visualization. These actions are triggered by the elements of associated GUIs (Concordance.odc, Alignment.odc). Examples: selection of a collocation in the list (when its first instance is actually selected); selection of another instance of it with the buttons of browse (*First, Prev, Next, Last*); selection by entering the instance number. The action to perform consists mainly in displaying the source context and updating the information on the current collocation (file name, file symbol, file collection).

PROCEDURE **CrtCoocNotifier** (op, from, to: INTEGER)
Notifier for the field indicating the current number of instance for the selected collocation. Access collocation instance by the number given and perform the main action on selection changed.

PROCEDURE **DisplaySourceFile** (relativeFileName: Files.Name; cooc: CollocationGlobal.CoocRec; open: BOOLEAN);
Display the collocation context: locate the phrase containing the current collocation instance 'cooc', and color it while highlighting the collocation keys (bold font face). The content of the source file is automatically scrolled to this phrase. If open is TRUE, open the source file in a window.

PROCEDURE **DisplayTargetFile** (relativeFileName: Files.Name; posBeg, posEnd: INTEGER; open: BOOLEAN)
Find the target document in the selected target language (by using the paths defined for the parallel corpora). Color the span between 'posBeg' and 'posEnd' (the target context) and scroll the file content to it. IF open is TRUE, open the document in a target window.

PROCEDURE **GetFileID** (sourceFile: LatlIO.Text; IN nameID: ARRAY OF CHAR; OUT value: ARRAY OF CHAR)
Given the name 'nameID' of a META tag attribute in the file 'sourceFile', return the attribute's value. If the file is not in HTML format, the returned value is empty. The attributes are used in the header of WTO files in order to identify files.

PROCEDURE **HighlightCrtCollocation** (f_src: TextMappers.Formatter; t_src: TextModels.Model; cooc: CollocationGlobal.CoocRec)
Highlight the collocation keys in the source document (bold font face).

PROCEDURE **Init**
Initialize program's variables related to the last selection.

PROCEDURE **OnCoocBrowse** (n: INTEGER)
Action to perform on click on collocations navigation buttons (First, Previous, Next, Last) in the visualization interfaces (Concordance.odc, Alignment.odc). Update the current instance number and perform the main action on selection changed.

PROCEDURE **OnCoocSel** (op, from, to: INTEGER)
Action to perform on selection change in the collocations list (main action on selection changed).

CollocationValidate

Last updated Nov 05, 2004 (0055), Violeta.Seretan@latl.unige.ch

```
DEFINITION CollocationValidate;

   IMPORT Dialog, Files;

   TYPE
      Term = RECORD
         string, type, translation1, targetLanguage1, translation2, targetLanguage2: Dialog.String;
         sampleContext, targetContext1, targetContext2: ARRAY 1000 OF CHAR;
         sourceFile, targetFile1, targetFile2: Files.Name;
         key1, key2: Dialog.String;
         addTranslation1, addTranslation2: BOOLEAN
      END;

   VAR
      entry: Term;
      param: RECORD
         validatedList: Dialog.Selection
      END;

   PROCEDURE AddTranslation1Guard (VAR par: Dialog.Par);
   PROCEDURE AddTranslation2Guard (VAR par: Dialog.Par);
   PROCEDURE DeleteAll;
   PROCEDURE DeleteSelected;
   PROCEDURE OnEntryChanged (op, from, to: INTEGER);
   PROCEDURE OnValidate;
   PROCEDURE OnValidateAll;
   PROCEDURE OnValidateSelected;
   PROCEDURE OnValidatedChange (op, from, to: INTEGER);
   PROCEDURE ThereIsSelectionGuard (VAR par: Dialog.Par);
   PROCEDURE ValidateAllGuard (VAR par: Dialog.Par);
   PROCEDURE ValidateGuard (VAR par: Dialog.Par);

END CollocationValidate.
```

This module contains procedures used for building a database of collocation terminology by manually validating collocations from the concordance/alignment tools. Its main function is to allow the interaction between the visualization tools and the database:
- entry collocation details, store collocation entries, check if an entry is already stored;
- perform operations with the terms in the validation list.
Associated GUI form: '*Validate.odc*'.

```
TYPE
   Term = RECORD
      string, type, translation1, targetLanguage1, translation2, targetLanguage2: Dialog.String;
      sampleContext, targetContext1, targetContext2: ARRAY 1000 OF CHAR;
      sourceFile, targetFile1, targetFile2: Files.Name;
      key1, key2: Dialog.String;
      addTranslation1, addTranslation2: BOOLEAN
   END;
```

Holds information about the terms to validate. Part of this information is added automatically and the rest is to be added by the user. The term will be converted into two entries corresponding to the two target languages.

string: Dialog.String;
Term key, as a string.

type: Dialog.String;
Term syntactic type, as a string.

translation1: Dialog.String;
The translation of term in the first target language.

targetLanguage1: Dialog.String;
The name of the first target language. Possible values: that of CollocationGlobal.languages.

translation2: Dialog.String;
The translation of term in the second target language.

targetLanguage2: Dialog.String;
The name of the second target language. Possible values: that of CollocationGlobal.languages.

sampleContext: ARRAY 1000 OF CHAR;
Sample context of term use in the source language. Default value: the term source context (phrase).

targetContext1: ARRAY 1000 OF CHAR;
Sample context of term use in the first target language. Default value: target context in the first target file.

targetContext2: ARRAY 1000 OF CHAR;
Sample context of term use in the second target language. Default value: target context in the second target file.

sourceFile: Files.Name;
The full name of the source file of term (the file from which it has been extracted).

targetFile1: Files.Name;
The full name of the target file of term, for the first target language.

targetFile2: Files.Name;
The full name of the target file of term, for the second target language.

key1:Dialog.String;
The key of the first lexeme of term.

key2: Dialog.String;
The key of the second lexeme of term.

addTranslation1: BOOLEAN
If TRUE, the translation and sample context for the first target language will be stored in the validation list.

addTranslation2: BOOLEAN
If TRUE, the translation and sample context for the second target language will be stored in the validation list.

VAR **entry**: Term;
Interactor interface-program (validation entry on the associated GUI form). Holds information about the term to validate that is currently selected in the validation list.

VAR **param**: RECORD
    validatedList: Dialog.Selection
END;
A dialog list containing terms to validate.

PROCEDURE **AddTranslation1Guard** (VAR par: Dialog.Par);
Guard for the radio button of the first target language. Check the button if a translation is added for the corresponding language in the validation entry on the associated GUI form.

PROCEDURE **AddTranslation2Guard** (VAR par: Dialog.Par);

Guard for the radio button of the second target language. Check the button if a translation is added for the corresponding language in the validation entry on the associated GUI form.

PROCEDURE **DeleteAll**;
Action for the button 'Delete All' on the associated GUI form. Empty the validation dialog list on the associated GUI form.

PROCEDURE **DeleteSelected**;
Action for button 'Delete Selected' on the associated GUI form. Delete the selected items from the validation dialog list. Select the first remaining item, if any.

PROCEDURE **OnEntryChanged** (op, from, to: INTEGER);
Notifier for the validation entry on the associated GUI form. If any of the entry fields changed, update the current entry in the validation list.

PROCEDURE **OnValidate**;
Action for click on button 'Validate...' on the visualization forms. Open the validation form 'Validate.odc'. Add a new entry (corresponding to the term to validate) to the validation entry list, and to the validation dialog list. Select it in the dialog list.

PROCEDURE **OnValidateAll**;
Action for button 'Validate All' on the associated GUI form. Iterate the validation entry list, create two terms per entry (corresponding to the two target languages) and store them into the database. Report the number of entries stored.

PROCEDURE **OnValidateSelected**;
Action for the button 'Validate Selected' on the associated GUI form. Store the selected terms into the database. Report the number of entries stored.

PROCEDURE **OnValidatedChange** (op, from, to: INTEGER);
Notifier for the validation dialog list on the associated GUI form. When a new item is included into the selection, assign the entry that correspond to this item to the validation entry on the associated GUI form.

PROCEDURE **ThereIsSelectionGuard** (VAR par: Dialog.Par);
Guard for buttons 'Validate Selected' and 'Delete Selected' on the associated GUI form. Disable the button if there is no item selected on the validation dialog list.

PROCEDURE **ValidateAllGuard** (VAR par: Dialog.Par);
Guard for buttons 'Validate All' and 'Delete All' on the associated GUI form. Disable the buttons if the validation dialog list is empty.

PROCEDURE **ValidateGuard** (VAR par: Dialog.Par);
Guard for button 'Validate...' on the visualization forms. Disable button if the collocation dialog list on the visualization tools is empty.

CollocationWeb

Last updated Nov 16, 2004 (0059), Violeta.Seretan@latl.unige.ch

```
DEFINITION CollocationWeb;

   IMPORT Dialog;

   VAR
      ClientInfo: RECORD
         clientKey: Dialog.String
      END;
      GUI: RECORD
         keys: Dialog.String;
         nbRes: INTEGER
      END;

   PROCEDURE DateTest;
   PROCEDURE Do;
   PROCEDURE DoFromDownloaded;
   PROCEDURE DoFromDownloadedNoClean;
   PROCEDURE DoFromFile;
   PROCEDURE LoadFile;

END CollocationWeb.
```

This module implements procedures that allow to extract collocation from the web by parsing the google query results. A query is executed with Google API by calling an exernal bat comand file. The results received are stored in a file, that is later cleaned (improper snippets, that are empty or do not containg the search word, are skipped), then parsed. Finally the score computation takes place, and the visualization that uses the concordance tool of Collocation subsystem. Before each processing (word or list of words) the previous results are removed from the associated database.

VAR **ClientInfo**: RECORD
```
      clientKey: Dialog.String
   END;
```
Google API client information. Contains a Google API client key that can be obtained by the user from Google website. The access to automatic querying is limited to 1000 queries/client/day.

VAR **GUI**: RECORD
```
      keys: Dialog.String;
      nbRes: INTEGER
   END;
```
Interactor program-interface.

keys: Dialog.String;
The search key(s). One or more words.

nbRes: INTEGER
How many snippets to retrieve from Google as query results. Can be between 10 and 1000. Results are got in blocks of 10. Not more than 1000 results can be retrieved from a query (Google limitation).

PROCEDURE **Do**;
Validate interface user input, get the results from Google, clean and parse results, open concordance tool.

PROCEDURE **DoFromDownloaded**;
Access results previously retrieved. Clean and parse them, then open concordance tool. Results are found by specifying the search word and the results number.

PROCEDURE **DoFromDownloadedNoClean**;
Same as DoFromDownloaded, without cleaning the results.

PROCEDURE **DoFromFile**;
Perform the search procedure for a number of words stored in a file (each word on a separate line).

PROCEDURE **LoadFile**;
Set the file that specifies the list of words to process.

2. Sous-système

UTILS

Emplacement : **sources/project/Utils**

2.1 Index de la documentation

Emplacement: **Utils/Docu/Sys-Map.odc**

Part of Projet RUIG-GIAN, 2002-2003

Utils -

Selection and processing of multiple files

Violeta Seretan

LATL, Violeta.Seretan@latl.unige.ch

User Manual
Developper Manual

Modules: UtilsCmds, UtilsProcessFiles, UtilsScan, UtilsFileTree, UtilsOptions, UtilsFunctions

Sources: UtilsCmds, UtilsProcessFiles, UtilsScan, UtilsFileTree, UtilsOptions, UtilsFunctions

Interface: Open interface

2.2 Manuel de l’utilisateur

Emplacement: **Utils/Docu/User-Man.odc**

V.Seretan, Violeta.Seretan@lettres.unige.ch

Utils - Selection and processing of multiple files

December, 2004

User Manual

Contents

Overview

Interface

1. how to choose a folder with the files to be processed;
2. how to apply an automatic filter on the files contained by this folder;
3. how to further apply a manual selection of the files to be processed;
4. how to choose the folder in which the results will be stored;
5. how to set the processing parameters.

See also: Developper Manual, Subystem Map

Overview

This document explains how to use the interface for multiple files selection and processing. This interface allows the user to select a collection of files situated in a source folder, according to several filters (mainly, the file location, name, type, and date). Then, the interface allows to process the selected files with a processor (some

Back

Interface

The rest of the document provide explanations for :

1. *How to choose a folder with the files to be processed*

The input folder can be chosen either by:
- typing in the edit field the complete path of the folder;
- browsing the file structure on a disk drive.

Figure 1. *Input Folder*

Back

2. How to apply an automatic filter on the files contained by this folder

The content of the input folder (the files) can be restricted using several criteria:

- the file contains a **specific string in its name**
  To use this filter, type a string in the text field "include only files named".

- the file has a **specific type**
  To select only files of a desired type, fill in the field "include only files of types" with the desired file extension. Then click on the button "Add" close to this field. You can repeat the procedure to specify other types.
  ***Note**: The supported file formats are:*
  ASCII and Unicode files (e.g. txt, htm, html)
  ODC files (BlackBox specific file format)
  rtf files (rich text format from word processors like MS Word)
  UTF-8 (extension utf8)

- the file is contained in the **root** of the input folder or in **one of its subfolders**.
  To select all the files in the input folder, check the option "include files from subfolders".
  To select only the files in the root of the input folder, uncheck this options.

- the file is (not) contained in a **subfolder with a specific name**
  To exclude the files contained in specific folders, check the option 'exclude files from folders named' and add the names of the folders to be excluded in the list below it. To add a folder name for exclusion, type it in the list header and click on the button 'Add folder name'. All the folder names added to this list will be used for files filtering.
  ***Note:** Default value for this list: the folders named "Log" (output by the collocation extraction).*

- the file has been **last modified at a specific date**
  To apply a filter on the date of files last modification (includes creations), check the box 'include only files created/modified'. Then specify if the system should verify that the file was modified several days previously to the current date, or in a given days interval. To do that, choose either the radio button "in the last ... days" and enter the desired number of days, or the radio button "between ... and ...", and specify two dates.
  To specify the start limit, enter a date in the format DD-MM-YY in the field next to "between".
  To specify the end limit, enter it in the field next to "and".
  The default values that are proposed are between *yesterday* and *today*.

Figure 2. *Automatic Files Filter*

Back

3. *How to further apply a manual selection of the files to be processed*

The content of the input folder is filtered according to the automatic filter and is shown in a list, from which the user can manually choose which items (files or subfolders) to be processed.
To include or exclude items, use the buttons "Check All", "Uncheck All", "Invert" or use the mouse in combination with the keys Ctrl or Shift.

Figure 3. *Manual Files Selection*

Back

4. *How to choose the folder in which the results will be stored*

As in 1, either type the name of the folder directly or browse the disk structure.
If the folder name does not exist on the disk, it will be created when the output is produced.

***Note**: The result files will be written in the output folder, and they will be gathered in a folder having the same name as the folder containing the input files.*

Figure 4. *Output Folder*

Back

5. *How to set the processing parameters*

The parameters currently implemented concern the following implemented processors: parsing and collocation extraction.

Parsing parameters:
- *language* (English and French are supported for now) - must be set;
- others - may keep the default values (*max alt* 10, *shallow parsing* checked, *filtering* checked, *output structure* unchecked).

Extraction parameters:
- where to store the collocations extracted: text file or database.
By storing collocations in a text file, some functionalities will not be available (collocation score computation, corpus frequency count, filtering, ordering).

Output of processors
For both parsing and extraction, additional files are produced that concern the global statistics for a processing session (globalStat.txt) and Log files. If the parameter *output structure is* selected, then files containing syntactic analyses are produced.

The parameters *save results in one file* and *results file per file* decide if the results are cumulated for all the files or are saved separately, file per file. In the last case, a mirror copy of the input folder's structure is created in the output folder.

The parameter *text files format* decide which type the results files will have: *txt* or *odc* (BlackBox internal format).

Figure 5. *Processing Parameters*

Back

2.3 Manuel du développeur

Emplacement: **Utils/Docu/Dev-Man.odc**

V.Seretan, Violeta.Seretan@lettres.unige.ch

Utils Subsystem

Developper Manual

November, 2004

Contents

Overview
Architecture
Compilation
Open the interface
How to develop it
Terminology
Comments

See also: User Manual, Subystem Map

Overview

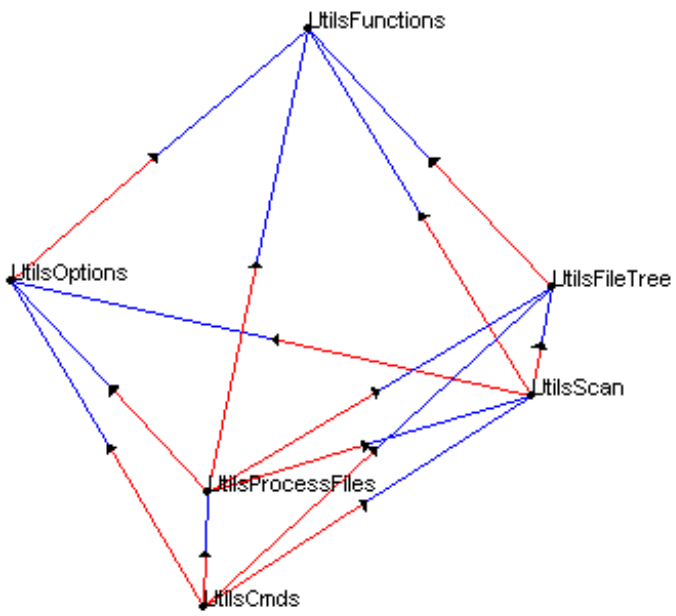
This document describes the Utils subsystem, a collection of modules that allow to select a list of files (using an interface) according to multiple filtering options, then to process these files (the processors connected now are the Fips Parser, Fips Term Extractor, and Its Translator).

Other processors (of a file given by its full name) can be easily added (see How to develop it).

Back

Architecture

The relationship between the Utils modules is depicted below (the directed arrow means the module at the arrow source imports, i.e., uses, depends on, the module at the target).



UtilsCmds contains the main interface procedures of the system (GUI guards, notifiers for the associated GUI 'Select.odc'; menu commands such as *OnParse*, *OnTermExtractor* etc.).

UtilsProcessFiles controls the execution of the process selected by UtilsCmds on the list of selected files. This list is set according to the parameters in UtilsScan.

UtilsScan implements the scanning of folder's structure and the application of different filters on files (automatic or manual). UtilsScan uses UtilsFileTree for operations with dialog trees displaying folder's structure.

UtilsOptions is used to store and retrieve parameters of UtilsScan.

UtilsFunctions contains procedures of general use, such as operations with files and strings of characters.

Back

Compilation

To compile:

⇨

❗ DevCompiler.CompileThis
UtilsFunctions UtilsOptions UtilsFileTree UtilsScan UtilsProcessFiles UtilsCmds ◢

⇦

Back

Open the interface

Use the command below to open the interface of multiple files selection:

❗"StdCmds.OpenToolDialog('Utils/Rsrc/Select.odc', 'Multiple files selection')"

Back

How to develop it

To retrieve the list of files selected:
Use exported variable UtilsProcessFiles.filelist of type UtilsProcessFiles.DocNames, described below:

```
DynString* = POINTER TO ARRAY OF CHAR.
DocNames* = POINTER TO RECORD
    path*: DynString;
    name*: DynString;
    next*: DocNames
END;
```

Then traverse the list and process each file using your own processor, like below:

```
PROCEDURE ParcoursListe;
    VAR liste: UtilsScan.DocNames;
BEGIN
    UtilsProcessFiles.SetFileList;
    liste := UtilsProcessFiles.filelist;
        (* retrieve the list of selected files *)
    WHILE liste # NIL DO
        YourOwnProcessor.ProcessFile(liste.path^$ + liste.name^$);
        liste := liste.next END;
    END;
END ParcoursListe;
```

Optionally, you can edit the associated GUI ('Select.odc') and add a command button linked to the previous procedure ('ParcoursListe').

To open the form, see Open the interface.

Back

Terminology

The following terms (explained below) are often used in the documentation of modules of Utils subsystem.

associated GUI - the interface form 'Select.odc';

dialog tree - the files tree;

(file) content list - the list that shows the content of source directory (the first level) for manual selection;

file list - files selection;

(files) selection - the list of files selected with the file selection interface;

file selection interface - the associated GUI interface;

files tree - a dialog tree displaying the file structure of a directory;

processing button - the button on the associated GUI that process selected files;

processing - the action performed on the files selection;

processor - the processing to perform on the files selection, for instance: parsing, collocation extraction, translation);

selected files - files in selection;

selection parameters - the parameters used to make the files selection;

source directory - the directory (local or remote, on the network) that contains the files for selection;

target directory - the directory in which the results of processing will be stored.

Back

Comments

- Strange display of the dialog file tree
The subsystem currently displays the content of the source folder in a on-the-fly fashion. It builds up the dialog file tree as the user clicks on a subfolder, not all in once at the beginning (this would have taken many minutes on big drives). As an inconvenient, the display is a bit "weird", because of some automatic scrolling which is performed in order to re-display the modified tree.
To change the way the folder's content is displayed to the classical mode (in which the whole tree is build up at the beginning), set the variable UtilsFileTree.expandOnTheFly to FALSE.

Back

2.4 Manuel du développeur - Modules

Emplacement: **Utils/Docu/**

UtilsCmds

Last updated Dec 10, 2004 (0290) , Violeta.Seretan@latl.unige.ch

```
DEFINITION UtilsCmds;

   IMPORT Dialog, StdTabViews;

   PROCEDURE FilesActionGuard (VAR par: Dialog.Par);
   PROCEDURE OnAdvancedOptions;
   PROCEDURE OnFileNamesDisplay;
   PROCEDURE OnFilesAction;
   PROCEDURE OnParse;
   PROCEDURE OnTermExtractor;
   PROCEDURE OnTranslate;
   PROCEDURE OutputDatabaseGuard (VAR par: Dialog.Par);
   PROCEDURE SelectionCountNotifier (op, from, to: INTEGER);
   PROCEDURE TabNotifier (tv: StdTabViews.View; from, to: INTEGER);
   PROCEDURE TitleGuard (VAR par: Dialog.Par);

END UtilsCmds.
```

This module contains the GUI procedures for Utils subsystem. Associated GUI: 'Select.odc'.

PROCEDURE **FilesActionGuard** (VAR par: Dialog.Par);
Guard for processing button. Set its label as prepared in pre-set. If the processing is running, set the label to "Stop".

PROCEDURE **OnAdvancedOptions**;
Action for click on button "Advanced...", section Parse Options (tab Process Files) on the associated GUI. Save parser parameters and open the parser options form.

PROCEDURE **OnFileNamesDisplay**;
Action for the button "See file list". Save all options and display the names of files in the selection in the Log window. If there are too many files, only display their number.

PROCEDURE **OnFilesAction**;
Action for processing button. Save all parameters. Apply the file selection according to the selection parameters. Set the processor according to the button's label. Process the file list, but if the processing is already running, then stop it.

PROCEDURE **OnParse**;
Action for a menu item. Pre-set the label of processing button to "Parse files".

PROCEDURE **OnTermExtractor**;
Action for a menu item. Pre-set the label of processing button to "Extract Collocations".
Set some parsing parameters for term extraction.

PROCEDURE **OnTranslate**;
Action for a menu item. Pre-set the label of processing button processor to "Translate".

PROCEDURE **OutputDatabaseGuard** (VAR par: Dialog.Par);
Guard for the radio buttons in the section "Save Results" (tab Process Files) on the associated form. Disable buttons if the processor is not collocation extraction.

PROCEDURE **SelectionCountNotifier** (op, from, to: INTEGER);
Display the selection information each time a new item from the content list is included/excluded in the selection.

PROCEDURE **TabNotifier** (tv: StdTabViews.View; from, to: INTEGER);
Action for tab changing in the associated GUI. If the tab 'Input Folder' is opened, then browse the source directory. If the tab 'Output Folder' is opened, browse the target directory. If the tab 'Files Filter' is quit, apply the filter on the content list.

PROCEDURE **TitleGuard** (VAR par: Dialog.Par);
Guard for caption field on the associated form. Set the label depending on the processor.

UtilsFileTree

Last updated Dec 10, 2004 (0039) , Violeta.Seretan@latl.unige.ch

```
DEFINITION UtilsFileTree;

   IMPORT Dialog;

   CONST
      excludeFiles = TRUE;
      includeFiles = FALSE;

   TYPE
      Mask = POINTER TO ARRAY OF BOOLEAN;

      Dlg = RECORD
         path: Dialog.String;
         tree: Dialog.Tree;
         mask: Mask
      END;

   VAR
      expandOnTheFly: BOOLEAN;

   PROCEDURE BuildTree (VAR dlg: Dlg; includeFiles: BOOLEAN);
   PROCEDURE CopyMask (VAR source, target: Mask);
   PROCEDURE DifferentMasks (VAR tree: Dialog.Tree; mask1, mask2: Mask; OUT node: Dialog.TreeNode):
BOOLEAN;
   PROCEDURE Empty (VAR dlg: Dlg);
   PROCEDURE ExpandLevels (VAR dlg: Dlg);
   PROCEDURE GetDifferentNode (VAR tree: Dialog.Tree; number: INTEGER; OUT foundNode:
Dialog.TreeNode);
   PROCEDURE GetExpMask (VAR tree: Dialog.Tree; OUT mask: Mask);
   PROCEDURE GetPath (VAR dlg: Dlg; VAR path: Dialog.String);
   PROCEDURE WriteMask (mask: Mask);

END UtilsFileTree.
```

This module implements operations on the dialog tree representing a directory file structure: create, delete, get the directory path, creation by expansion (on-the-fly, i.e., create its substructure on the next level as the user clicks on a subdirectory node).
The implementation is rooted in the module ObxFileTree.

CONST **excludeFiles** = TRUE;
Possible value for the parameter includeFiles in BuildTree procedure. If used, the files in the directory will not be added to the dialogue tree.

CONST **includeFiles** = FALSE;
Possible value for the parameter includeFiles in BuildTree procedure. If used, the files in the directory will be added to the dialogue tree.

TYPE **Mask** = POINTER TO ARRAY OF BOOLEAN;
An array with the expanded values for each node in the dialog tree that contains the files.

TYPE **Dlg** = RECORD
   path: Dialog.String;
   tree: Dialog.Tree;
   mask: Mask
END;
Type for a dialog tree displaying files.

path: Dialog.String;
The directory path.

tree: Dialog.Tree;
The tree dialog.

mask: Mask
The expansion mask of the dialog tree.

VAR **expandOnTheFly**: BOOLEAN;
Display parameter, which is TRUE if the files tree is to be expanded on user click; FALSE if completed at the beginning

PROCEDURE **BuildTree** (VAR dlg: Dlg; includeFiles: BOOLEAN);
Build the dialog tree representing the structure of the directory given by dlg.path.

PROCEDURE **CopyMask** (VAR source, target: Mask);
Copy the expansion mask from source to target.

PROCEDURE **DifferentMasks** (VAR tree: Dialog.Tree; mask1, mask2: Mask; OUT node: Dialog.TreeNode): BOOLEAN;
Compare two node lists and return TRUE iff they are different; in this case, node is the first node that differs.

PROCEDURE **Empty** (VAR dlg: Dlg);
Clears the dialog tree.

PROCEDURE **ExpandLevels** (VAR dlg: Dlg);
Browse the tree and add the next levels to all the nodes expanded by the user.

PROCEDURE **GetDifferentNode** (VAR tree: Dialog.Tree; number: INTEGER; OUT foundNode: Dialog.TreeNode);
Return the node in the dialog tree having the given number. The preorder is considered:
parent - first children - ... - last children.

PROCEDURE **GetExpMask** (VAR tree: Dialog.Tree; OUT mask: Mask);
Create a list representing the expansion mask of all the nodes in a dialog tree.

PROCEDURE **GetPath** (VAR dlg: Dlg; VAR path: Dialog.String);
Retrieve the path of the node selected in the file tree dialog.

PROCEDURE **WriteMask** (mask: Mask);
Print to Log window an expansion mask.

UtilsFunctions

Last updated Dec 10, 2004 (0279) , Violeta.Seretan@latl.unige.ch

```
DEFINITION UtilsFunctions;

   IMPORT Dialog, Files, TextModels, Views, Dates, Windows, TextViews;

   PROCEDURE ClearTags (inText: ARRAY OF CHAR; OUT outText: ARRAY OF CHAR);
   PROCEDURE CommentSel;
   PROCEDURE ControlChar (ch: CHAR): BOOLEAN;
   PROCEDURE CountSelected (VAR list: Dialog.Selection): INTEGER;
   PROCEDURE DeleteFile (filename: Files.Name);
   PROCEDURE DetectSourceLanguage (): INTEGER;
   PROCEDURE ExtractFilePathName (IN fileName: Files.Name; OUT path, name: Files.Name);
   PROCEDURE ExtractSpan (file: TextModels.Model; beg, end: INTEGER; VAR span: ARRAY OF CHAR;
maxlen: INTEGER);
   PROCEDURE FileCreatedAfter (filename: Files.Name; date: Dates.Date; time: Dates.Time): BOOLEAN;
   PROCEDURE FileExists (filename: Files.Name): BOOLEAN;
   PROCEDURE FindInPath (fileName, string: ARRAY OF CHAR): INTEGER;
   PROCEDURE FindWindow (title: ARRAY OF CHAR): BOOLEAN;
   PROCEDURE FipsOptionsSetUp;
   PROCEDURE GetCharOrd;
   PROCEDURE GetDriveLetter (IN dir: ARRAY OF CHAR): CHAR;
   PROCEDURE GetFileModel (IN fileName: ARRAY OF CHAR; OUT text: TextModels.Model);
   PROCEDURE GetFileSize (IN fileName: Files.Name): INTEGER;
   PROCEDURE GetSelection (OUT out: POINTER TO ARRAY OF CHAR);
   PROCEDURE GetStringFromModel (model: TextModels.Model; OUT string: POINTER TO ARRAY OF
CHAR);
   PROCEDURE HasExtension (IN filename: Files.Name): BOOLEAN;
   PROCEDURE HasNumbering (IN string: Dialog.String): BOOLEAN;
   PROCEDURE HasSubDirs (IN dir: ARRAY OF CHAR): BOOLEAN;
   PROCEDURE InsertLogString;
   PROCEDURE InsertProcHeader;
   PROCEDURE IsDash (ch: CHAR): BOOLEAN;
   PROCEDURE IsDigit (ch: CHAR): BOOLEAN;
   PROCEDURE IsLetter (ch: CHAR): BOOLEAN;
   PROCEDURE IsPunctuation (ch: CHAR): BOOLEAN;
   PROCEDURE IsSpace (ch: CHAR): BOOLEAN;
   PROCEDURE NewSentence (file: TextModels.Model; pos: INTEGER): BOOLEAN;
   PROCEDURE OpenView (v: Views.View; loc: Files.Locator; IN name: Files.Name; OUT w:
Windows.Window);
   PROCEDURE ParentDirectory (IN dir: ARRAY OF CHAR; OUT parentDir: ARRAY OF CHAR);
   PROCEDURE PathToFileSpec (IN fileName: ARRAY OF CHAR; OUT loc: Files.Locator; OUT name:
Files.Name);
   PROCEDURE ReadListFromFile (IN filename: Files.Name; OUT list: POINTER TO ARRAY OF INTEGER);
   PROCEDURE ReadString (t: TextModels.Model; startPos: INTEGER; OUT endPos: INTEGER; OUT str:
Dialog.String);
   PROCEDURE ReadWord (t: TextModels.Model; min_len, startPos: INTEGER; OUT endPos: INTEGER;
OUT word: Dialog.String);
   PROCEDURE Register (model: TextModels.Model; fileName: Files.Name; IN format: ARRAY OF CHAR);
   PROCEDURE ReplaceQuotes (VAR str: ARRAY OF CHAR);
   PROCEDURE RetrieveSrcView (OUT textView: TextViews.View);
   PROCEDURE RetrieveTrgView (OUT textView: TextViews.View);
   PROCEDURE ShortenFileName (IN filename: ARRAY OF CHAR; desiredLength: INTEGER; OUT
newFilename: ARRAY OF CHAR);
   PROCEDURE ShowWindowsNames;
   PROCEDURE Trim (VAR input: ARRAY OF CHAR; trimList: ARRAY OF CHAR);
   PROCEDURE UndoReplaceQuotes (VAR str: ARRAY OF CHAR);
   PROCEDURE ValidWordChar (ch: CHAR): BOOLEAN;
```

```
    PROCEDURE ValidatePath (path: Files.Name): BOOLEAN;
    PROCEDURE WorkingDirectory (IN dir: ARRAY OF CHAR; OUT wrkDir: ARRAY OF CHAR);
    PROCEDURE WriteCursorPosition;

END UtilsFunctions.
```

This module consist of procedures of general use, mainly operations with files and strings.

PROCEDURE **ClearTags** (inText: ARRAY OF CHAR; OUT outText: ARRAY OF CHAR);
Given the text inText, return its version in which the tags were stripped.

PROCEDURE **CommentSel**;
Add comment marks arround the text in the current selection; make the selection italic.

PROCEDURE **ControlChar** (ch: CHAR): BOOLEAN;
Return TRUE iff the character ch is a control character.

PROCEDURE **CountSelected** (VAR list: Dialog.Selection): INTEGER;
Return the number of the items which were selected in a dialog list.

PROCEDURE **DeleteFile** (filename: Files.Name);
Delete the specified file.

PROCEDURE **DetectSourceLanguage** (): INTEGER;
Return the language of Fips parser: an integer code defined in FipsLexTools.

PROCEDURE **ExtractFilePathName** (IN fileName: Files.Name; OUT path, name: Files.Name);
Extract path and file name from a complete filename.

PROCEDURE **ExtractSpan** (file: TextModels.Model; beg, end: INTEGER; VAR span: ARRAY OF CHAR; maxlen: INTEGER);
Extract from file the text span that is between the indicated limits. Allow only a given maximal length.

PROCEDURE **FileCreatedAfter** (filename: Files.Name; date: Dates.Date; time: Dates.Time): BOOLEAN;
Checks whether the file modification date is after the specified date and time.

PROCEDURE **FileExists** (filename: Files.Name): BOOLEAN;
Checks whether the specified file exists.

PROCEDURE **FindInPath** (fileName, string: ARRAY OF CHAR): INTEGER;
Return the position of occurrence of the desired string in the given filename.

PROCEDURE **FindWindow** (title: ARRAY OF CHAR): BOOLEAN;
Return TRUE iff the window with the given title (name) is currently opened in the system.

PROCEDURE **FipsOptionsSetUp**;
Default settings for Fips parser options. May be distribution - dependent.

PROCEDURE **GetCharOrd**;
Print to the Log window the numeric codes of characters in the current text selection.

PROCEDURE **GetDriveLetter** (IN dir: ARRAY OF CHAR): CHAR;
Given a directory path, return the drive letter if there is one in the path, and a blank space otherwise.

PROCEDURE **GetFileModel** (IN fileName: ARRAY OF CHAR; OUT text: TextModels.Model);
Return the model (the data) of a text file specified by its complete filename.

PROCEDURE **GetFileSize** (IN fileName: Files.Name): INTEGER;
Return the size (in characters) of the file specified.

PROCEDURE **GetSelection** (OUT out: POINTER TO ARRAY OF CHAR);
Get the text that is currently selected in the system.

PROCEDURE **GetStringFromModel** (model: TextModels.Model; OUT string: POINTER TO ARRAY OF CHAR);
Convert a model to a string. The inverse operation for TextModels.NewFromString.

PROCEDURE **HasExtension** (IN filename: Files.Name): BOOLEAN;
Return TRUE iff a full stop is found after the first position in string and it is followed by a letter.

PROCEDURE **HasNumbering** (IN string: Dialog.String): BOOLEAN;
Return TRUE if the string represents a paragraph numbering, e.g., after stripping of period and parentheses, it is on the form: 0-9, a-z, {i|v|x|l|c|d|m].

PROCEDURE **HasSubDirs** (IN dir: ARRAY OF CHAR): BOOLEAN;
Return TRUE iff the given directory contains at least a subdirectory.

PROCEDURE **InsertLogString**;
Insert "Log.Ln; Log.String();" at the cursor position.

PROCEDURE **InsertProcHeader**;
Create a PROCEDURE block, having the text from the current selection as name.

PROCEDURE **IsDash** (ch: CHAR): BOOLEAN;
Return TRUE iff the character ch is a (kind of) hyphen or dash.

PROCEDURE **IsDigit** (ch: CHAR): BOOLEAN;
Return TRUE iff the character ch is in the range 0..9.

PROCEDURE **IsLetter** (ch: CHAR): BOOLEAN;
Return TRUE iff the character ch is in the range a..z, A..Z or it is a accented letter.

PROCEDURE **IsPunctuation** (ch: CHAR): BOOLEAN;
Return TRUE iff the character ch is a dot, an exclamation or an question mark.

PROCEDURE **IsSpace** (ch: CHAR): BOOLEAN;
Return TRUE iff the character ch is a blank space, a tab or a non-breaking space.

PROCEDURE **NewSentence** (file: TextModels.Model; pos: INTEGER): BOOLEAN;
Checks whether pos is the beginning of a new sentence. Used because the lexical analysis of Fips returns too broad text spans.

PROCEDURE **OpenView** (v: Views.View; loc: Files.Locator; IN name: Files.Name; OUT w: Windows.Window);
Open a window with the specifications v, loc, name.

PROCEDURE **ParentDirectory** (IN dir: ARRAY OF CHAR; OUT parentDir: ARRAY OF CHAR);
Return the complete path of the parent directory.

PROCEDURE **PathToFileSpec** (IN fileName: ARRAY OF CHAR; OUT loc: Files.Locator; OUT name: Files.Name);
Return the file specifications (locator and name) for the file specified by its complete filename.

PROCEDURE **ReadListFromFile** (IN filename: Files.Name; OUT list: POINTER TO ARRAY OF INTEGER);
Read an array of integers from the file specified. The resulting array is dynamic.

PROCEDURE **ReadString** (t: TextModels.Model; startPos: INTEGER; OUT endPos: INTEGER; OUT str: Dialog.String);
Read a string from t beginning at position pos and return the position endPos of its end.

PROCEDURE **ReadWord** (t: TextModels.Model; min_len, startPos: INTEGER; OUT endPos: INTEGER; OUT word: Dialog.String);
Read a word from t beginning at position pos, of minimal length min_len and return the word w and the position endPos of its end. Skip the HTML-like tagged text.

PROCEDURE **Register** (model: TextModels.Model; fileName: Files.Name; IN format: ARRAY OF CHAR);
Save the text model to a file specified by "fileName" in the desired format ("txt" or "odc").

PROCEDURE **ReplaceQuotes** (VAR str: ARRAY OF CHAR);
Replace simple quotes ' with stars *. Necessary for storing strings to databases.

PROCEDURE **RetrieveSrcView** (OUT textView: TextViews.View);
Get the first text view on the form that has the focus.

PROCEDURE **RetrieveTrgView** (OUT textView: TextViews.View);
Get the second text view on the form that has the focus.

PROCEDURE **ShortenFileName** (IN filename: ARRAY OF CHAR; desiredLength: INTEGER; OUT newFilename: ARRAY OF CHAR);
Takes a filename and replace part of the path with '...' in order to shorten it at the desired length.

PROCEDURE **ShowWindowsNames**;
Print to the Log window the names of the windows currently opened in the system.

PROCEDURE **Trim** (VAR input: ARRAY OF CHAR; trimList: ARRAY OF CHAR);
Trim from input the left and right the symbols that are in list trimList.

PROCEDURE **UndoReplaceQuotes** (VAR str: ARRAY OF CHAR);
Replace stars * with simple quotes '. Undo operation for ReplaceQuotes.

PROCEDURE **ValidWordChar** (ch: CHAR): BOOLEAN;
Return TRUE iff the character ch a letter or a hyphen.

PROCEDURE **ValidatePath** (path: Files.Name): BOOLEAN;
If the directory specified by path doesn't exist, then create it.

PROCEDURE **WorkingDirectory** (IN dir: ARRAY OF CHAR; OUT wrkDir: ARRAY OF CHAR);
Return the short name of the parent directory.

PROCEDURE **WriteCursorPosition**;
Print to the Log window the file position of the cursor of the system.

UtilsOptions

Last updated Dec 10, 2004 (0289) , Violeta.Seretan@latl.unige.ch

```
DEFINITION UtilsOptions;

   IMPORT Dialog, Dates;

   TYPE
      ScanParam = RECORD
         source, target: Dialog.String;
         srcdrive, trgdrive: Dialog.List;
         dirContentList, filteredContent: Dialog.Selection;
         fname: ARRAY 256 OF CHAR;
         ftype: ARRAY 16 OF CHAR;
         fileTypeList: Dialog.Combo;
         include, existExcluded, stop: BOOLEAN;
         stopDir: Dialog.String;
         excludedFolders: Dialog.Combo;
         lastModif: TimeRecord;
         common_file: BOOLEAN;
         outputFormat: ARRAY 10 OF CHAR;
         outputFormatFlag: INTEGER;
         newDB: BOOLEAN;
         label: Dialog.String
      END;

      TimeRecord = RECORD
         thereIsFilter, byNbDays: BOOLEAN;
         nbDays: INTEGER;
         fromDay, toDay: Dates.Date
      END;

   VAR
      dataPath: Dialog.String;
      path: Dialog.String;

   PROCEDURE ReadScanParam (VAR param: ScanParam);
   PROCEDURE ResetAllScanParam (VAR param: ScanParam);
   PROCEDURE SaveScanParam (param: ScanParam);

END UtilsOptions.
```

Module for saving and retrieving options of Utils subsystem. The path to options files is defined in dataPath. Associated GUI: 'Select.odc'.

```
TYPE
   ScanParam = RECORD
      source, target: Dialog.String;
      srcdrive, trgdrive: Dialog.List;
      dirContentList, filteredContent: Dialog.Selection;
      fname: ARRAY 256 OF CHAR;
      ftype: ARRAY 16 OF CHAR;
      fileTypeList: Dialog.Combo;
      include, existExcluded, stop: BOOLEAN;
      stopDir: Dialog.String;
      excludedFolders: Dialog.Combo;
      lastModif: TimeRecord;
      common_file: BOOLEAN;
      outputFormat: ARRAY 10 OF CHAR;
```

```
    outputFormatFlag: INTEGER;
    newDB: BOOLEAN;
    label: Dialog.String
  END;
```

Scan parameters: parameters for the selection of files.

source, **target**: Dialog.String;
Source and target folders.

srcdrive, **trgdrive**: Dialog.List;
Source and target drive letters.

dirContentList, **filteredContent**: Dialog.Selection;
Lists that display the content (complete or filtered according to the filtering parameters) of the source folder.

fname: ARRAY 256 OF CHAR;
Filtering parameter: string that must be contained by the names of files in source folder.

ftype: ARRAY 16 OF CHAR;
Filtering parameter. Deprecated (replaced by 'fileTypeList').

fileTypeList: Dialog.Combo;
Filtering parameter: the type of a file in the source folder must match one of the types in this list.

include: BOOLEAN;
Filtering parameter: include subdirectories of the source folder.

existExcluded: BOOLEAN;
Filtering parameter: exclude files that are in the list of excluded folders.

stop: BOOLEAN;
Filtering parameter: there are stop directories (to be excluded). Deprecated (replaced by 'existExcluded').

stopDir: Dialog.String;
Filtering parameter: directory to be excluded. Deprecated (replaced by 'excludedFolders').

excludedFolders: Dialog.Combo;
Filtering parameter: list of names of folders to be excluded, e.g. "Log".

lastModif: TimeRecord;
Filtering parameter: include only file whose last modification date (or creation date) satisfy the time parameters.

common_file: BOOLEAN;
Output option: save results to a common file. Applies to processor "Term Extractor" only.

outputFormat: ARRAY 10 OF CHAR;
Output option: save results in ASCII or BlackBox file format (possible values: "txt" for ASCII, "odc" for BlackBox file format).

outputFormatFlag: INTEGER;
Output option: save results in ASCII or BlackBox file format (0 for ASCII, 1 for BlackBox file format). Used on the associated GUI.

newDB: BOOLEAN;
Output option: the coocurences database is cleared before storing new results. Applies to processor "Term Extractor" only.

label: Dialog.String
The label of the action button in the associated GUI.

TYPE **TimeRecord** = RECORD
  thereIsFilter, byNbDays: BOOLEAN;
  nbDays: INTEGER;
  fromDay, toDay: Dates.Date
END;

The type for time filtering parameters: parameters for files filtering based on the file modification date and time.

thereIsFilter: BOOLEAN;
TRUE iff there is a time filtering.

byNbDays: BOOLEAN;
TRUE iff the user chooses "last modified in the last n days".
FALSE iff the user chooses "last modified between date and date".

nbDays: INTEGER;
Number of days, for the condition "file last modified 'nbDays' days ago".

fromDay, **toDay**: Dates.Date
Limits of time interval, for the condition "file last modified between date... and date...".

VAR **dataPath**: Dialog.String;
Path to options files of subsystem.

VAR **path**: Dialog.String;
Path to subsystem.

PROCEDURE **ReadScanParam** (VAR param: ScanParam);
Read scan parameters from a file in XML format (scan.txt).

PROCEDURE **ResetAllScanParam** (VAR param: ScanParam);
Initialization of scan parameters and time filtering parameters.

PROCEDURE **SaveScanParam** (param: ScanParam);
Save scan parameters in a file in XML format (scan.txt).

UtilsProcessFiles

Last updated Dec 10, 2004 (0036) , Violeta.Seretan@latl.unige.ch

```
DEFINITION UtilsProcessFiles;

   IMPORT UtilsScan, FipsTools;

   CONST
      Parser = 1;
      Show = 0;
      TermExtractor = 2;
      Translator = 3;

   VAR
      crtFileNo-: INTEGER;
      filelist-: UtilsScan.DocNames;
      nbFiles-: INTEGER;
      options: FipsTools.Option;
      processor: INTEGER;

   PROCEDURE ProcessFileList (filelist: UtilsScan.DocNames; actionType: INTEGER);
   PROCEDURE SetFileList;
   PROCEDURE ShowSelectionInfo;
   PROCEDURE StopProcessing;

END UtilsProcessFiles.
```

This module (1) retrieves the list of files in the selection defined according to the parameters of module UtilsScan, and (2) performs a given processing on this file list. The processing of the list is implemented as a Service.Action, thus the system is not "frozen" but refreshes after each file being processed.
Associated GUI: 'Select.odc'.

CONST **Parser** = 1;
Defines a type of processing to perform on the files list: correspond to Fips parser.

CONST **Show** = 0;
Defines a type of processing to perform on the files list: correspond to simply listing the files names.

CONST **TermExtractor** = 2;
Defines a type of processing to perform on the files list: correspond to Fips Term Extractor.

CONST **Translator** = 3;
Defines a type of processing to perform on the files list: correspond to Its translator.

VAR **crtFileNo**-: INTEGER;
The number of the currently processed file, from 1 to nbFiles.

VAR **filelist**-: UtilsScan.DocNames;
List of files to process (contains files path and name).

VAR **nbFiles**-: INTEGER;
Total number of files to process (length of filelist).

VAR **options**: FipsTools.Option;
Fips parser option.

VAR **processor**: INTEGER;
What processing to perfom on the file list; possible values: constants Show, Parser, Term Extractor,

Translator.

PROCEDURE **ProcessFileList** (filelist: UtilsScan.DocNames; actionType: INTEGER);
Main procedure: start the processing for the given file list, with the action to perform given by 'actionType' (possible values: the constants Show, Parser, Term Extractor, Translator).

PROCEDURE **SetFileList**;
Scan the input directory according to the parameters in UtilsScan, and build up the file list to process.

PROCEDURE **ShowSelectionInfo**;
Show the file list information: number of files, total size of files.

PROCEDURE **StopProcessing**;
Notifier for the "Stop" button: finish processing (after the current file being processed) and show results.

UtilsScan

Last updated Dec 09, 2004 (0035) , Violeta.Seretan@latl.unige.ch

```
DEFINITION UtilsScan;

   IMPORT Dialog, UtilsFileTree, UtilsOptions;

   TYPE
      DocNames = POINTER TO RECORD
         path, name: DynString;
         next: DocNames
      END;

   VAR
      dlg: UtilsFileTree.Dlg;
      param: UtilsOptions.ScanParam;

   PROCEDURE AddFileTypeGuard (VAR par: Dialog.Par);
   PROCEDURE AddFolderNameGuard (VAR par: Dialog.Par);
   PROCEDURE BackGuard1 (VAR par: Dialog.Par);
   PROCEDURE BackGuard2 (VAR par: Dialog.Par);
   PROCEDURE BuildList (OUT list: Dialog.Selection; filtered: BOOLEAN; IN path: ARRAY OF CHAR);
   PROCEDURE CommonFileGuard (VAR par: Dialog.Par);
   PROCEDURE DateBetweenGuard (VAR par: Dialog.Par);
   PROCEDURE DateByGuard (VAR par: Dialog.Par);
   PROCEDURE DisableGuard (VAR par: Dialog.Par);
   PROCEDURE ExcludedFoldersGuard (VAR par: Dialog.Par);
   PROCEDURE FormatNotifier (op, from, to: INTEGER);
   PROCEDURE ListNotifier (n, op, from, to: INTEGER);
   PROCEDURE NbDaysGuard (VAR par: Dialog.Par);
   PROCEDURE NewDBGuard (VAR par: Dialog.Par);
   PROCEDURE OnAddFileType;
   PROCEDURE OnAddFolderName;
   PROCEDURE OnBack1;
   PROCEDURE OnBack2;
   PROCEDURE OnBrowse1;
   PROCEDURE OnBrowse3;
   PROCEDURE OnCheckAll;
   PROCEDURE OnInvertSelection;
   PROCEDURE OnUncheckAll;
   PROCEDURE ResetAll;
   PROCEDURE SaveToTextFile (op, from, to: INTEGER);
   PROCEDURE Scan (VAR src: ARRAY OF CHAR; OUT nbFiles, totalSize: INTEGER): DocNames;
   PROCEDURE SelectDirectory (VAR tree: Dialog.Tree; IN nodeName: ARRAY OF CHAR);
   PROCEDURE TreeNotifier (op, from, to: INTEGER);

END UtilsScan.
```

This module implements procedures for browsing the directory structure on the disk drives and selecting the files
belonging to the source directory. Several selection parameters can be defined:

1. filtering parameters, to apply an automatic filter on the files of the source folder, as follows:
   - include/exclude files from subdirectories of source directory, recursively, as opposed to including only the files in that directory;
   - exclude files from directories that have a given name;
   - include only files of given types;
   - include only files whose name contains a string
   - include only files having the last modification date in a given interval or a given number of days ago.

2. manual selection (on the result of the automatic selection).
Associated GUI form: 'Select.odc'.

TYPE **DocNames** = POINTER TO RECORD
  path, name: DynString;
  next: DocNames
END;
List of files in the selection.

path, **name**: DynString;
Path and (short) name of the file in the list.

next: DocNames
Pointer to the next file in the selection list.

VAR **dlg**: UtilsFileTree.Dlg;
File tree dialog that displays the structure of source folder defined in 'param' .

VAR **param**: UtilsOptions.ScanParam;
Selection parameters.

PROCEDURE **AddFileTypeGuard** (VAR par: Dialog.Par);
Guard for the button "Add" of the combo box for types. Disable the button when the item to add is empty.

PROCEDURE **AddFolderNameGuard** (VAR par: Dialog.Par);
Guard for the button "Add" of the combo box for excluded folders.
Disable the button when the item to add is empty, or there is no option to exclude folders.

PROCEDURE **BackGuard1** (VAR par: Dialog.Par);
Guard for the button "<--" (Back) next to the source folder: disable if source folder is empty.

PROCEDURE **BackGuard2** (VAR par: Dialog.Par);
Guard for the button "<--" (Back) next to the source folder: disable if target folder is empty.

PROCEDURE **BuildList** (OUT list: Dialog.Selection; filtered: BOOLEAN; IN path: ARRAY OF CHAR);
Build the list 'list' with the first-level content of directory given by 'path'.
If filtered is TRUE, apply the filter set by the selection parameters.

PROCEDURE **CommonFileGuard** (VAR par: Dialog.Par);
Guard for radio buttons 'all results in one file' and 'results file per file' on the associated GUI.
Disable it when the option 'Save results to' - 'text file' was not chosen.

PROCEDURE **DateBetweenGuard** (VAR par: Dialog.Par);
Guard for fields 'date1' and 'date2' in the option 'between date1 and date2' on the associated GUI.
Disable it when the option 'include only files created/ modified' or 'between date1 and date2' was not chosen.

PROCEDURE **DateByGuard** (VAR par: Dialog.Par);
Guard for radio buttons 'in the last n days' and 'between date... and date...' on the associated GUI.
Disable them when the option 'include only files created/ modified' was not chosen.

PROCEDURE **DisableGuard** (VAR par: Dialog.Par);
Guard for permanently disabling some items, such as the list showing the content of the target folder.

PROCEDURE **ExcludedFoldersGuard** (VAR par: Dialog.Par);
Guard for the combo box for excluded folders. Disable the box if there is no option to exclude folders.

PROCEDURE **FormatNotifier** (op, from, to: INTEGER);
Notifier for the radio buttons for the output format of result files. Set a string variable accordingly.

PROCEDURE **ListNotifier** (n, op, from, to: INTEGER);

Notifier for the lists displaying the drive letters. Set the source (or target) folder
and file tree dialog to the selected drive.

PROCEDURE **NbDaysGuard** (VAR par: Dialog.Par);
Guard for field 'n' in the option 'in the last n days' on the associated GUI.
Disable it when the option 'include only files created/ modified' or 'in the last n days' was not chosen.

PROCEDURE **NewDBGuard** (VAR par: Dialog.Par);
Guard for radio buttons 'begin with empty table' and 'continue with existing results'.
Disable these buttons when the option 'Sve results to' - 'database in the associated datasource' was not chosen.

PROCEDURE **OnAddFileType**;
Notifier for the button "Add" of the combo box for types. Add a new item to the combo box.

PROCEDURE **OnAddFolderName**;
Notifier for the button "Add" of the combo box for excluded folders. Add a new item to the combo box.

PROCEDURE **OnBack1**;
Notifier for the button "<--" (Back) next to the source folder: set the source folder to its parent
and re-build the source file tree.

PROCEDURE **OnBack2**;
Notifier for the button "<--" (Back) next to the target folder: set the target folder to its parent
and re-build the target file tree.

PROCEDURE **OnBrowse1**;
Notifier for the button "Browse" next to the source directory in the associated GUI.
Build the file tree dialog that allows to browse and set the source directory.

PROCEDURE **OnBrowse3**;
Notifier for the button "Browse" next to the target directory in the associated GUI.
Build the file tree dialog that allows to browse and set the target directory.

PROCEDURE **OnCheckAll**;
Notifier for button "Check All" on the associated GUI. Select all items in the list of manual selection.

PROCEDURE **OnInvertSelection**;
Notifier for button "Invert" on the associated GUI. Invert the selection in the list of manual selection.

PROCEDURE **OnUncheckAll**;
Notifier for button "Uncheck All" on the associated GUI. Uncheck all items in the list of manual selection.

PROCEDURE **ResetAll**;
Reset selection parameters.

PROCEDURE **SaveToTextFile** (op, from, to: INTEGER);
Notifier for the radio button 'Save results to' - 'text file'.
Show a warning explaining the limitations of choosing this option.

PROCEDURE **Scan** (VAR src: ARRAY OF CHAR; OUT nbFiles, totalSize: INTEGER): DocNames;
Main procedure. Scan the structure of folder 'src' and return the list of files selected according to the selection parameters; return the number of files in the selection and their total size (in bytes).

PROCEDURE **SelectDirectory** (VAR tree: Dialog.Tree; IN nodeName: ARRAY OF CHAR);
Select in the given dialog tree the node having the label 'nodeName'.

PROCEDURE **TreeNotifier** (op, from, to: INTEGER);
Notifier for the file tree dialogs on the associated GUI.
On user double click, set the source (or the target) folder and re-creates the file tree.

On user click, set the source (or the target) folder.