IWH TECHNICAL REPORTS

RLPC:
Record Linkage Pre-Cleaning
Technical Documentation of Routines

Wilfried Ehrenfeld
Abstract

The primary objective of record linkage is the merger of different data sets on the basis of an unique identifier. The cases at hand are mostly company data sets from databanks with company characteristics (e.g. BvD Amadeus/Dafne), patent data sets (e.g. Patstat or DPMA) and funding data sets (e.g. BMBF funding catalog). These data sets shall be merged on the basis of the company names. Due to the fact that company names have varying notations in different databases - for example the corporate structure – a harmonization and standardization is necessary.

The routines described here implement the record linkage pre-cleaning (RLPC). They are used to create record linkage compatible names (RLName) from given (actor) names (Name). This includes converting special characters to ASCII characters, identifying corporate structures, isolating and separating bracketed expressions. The result is an expression which allows for a comparison with other names. Following this pre-cleaning, record linkage systems can be used to merge several data sets that have been pretreated in the same way.
Contents

1. Starting basis and outline of the procedure 3

2. Basic routines 4
   2.1. PreCleaning.do ................................. 4
   2.2. Programs_Load.do ............................ 6
   2.3. Programs_Basic_Routines.do .................. 6
       RLreplace ...................................... 6
       replace_at_end ................................ 6
       replace_at_beginning .......................... 6
       replace_in_middle .............................. 7
       RLtrim ......................................... 7
   2.4. Programs_Module_Management.do ................ 7
       start Outer_timer ............................... 7
       stop Outer_timer ............................... 7
       begin_step .................................... 7
       end_step ...................................... 8
       begin_substep ................................ 8
       end_substep .................................. 8
       update_clean_hist ............................. 8
   2.5. Programs_ASCII_Routines.do .................. 8
       remove_ascii_special_chars .................... 9
       remove_special_chars ........................... 9
       remove_ascii_std_chars ....................... 9
       char_condensing_one ........................... 9
       char_condensing_two .......................... 10
       char_condensing_RL2 ........................... 10
       condense_spacing_RL2 ......................... 10
   2.6. Programs_Legform_Routines.do ................ 10
       legform_detect ................................. 10
       set_legform ................................... 11

3. Routines of the RLPC steps - step 1 11
   3.1. prog_1_1_uppercase.do ....................... 12
   3.2. prog_1_2_replace_smgl_codes.do ............... 12
   3.3. prog_1_3_replace_coded_chars.do .............. 13
   3.4. prog_1_4_replace_bracket_symbols.do .......... 13
   3.5. prog_1_5_get_bracket_content.do .............. 13
   3.6. prog_1_6_replace_ascended_chars.do .......... 13
   3.7. prog_1_7_replace_and.do ..................... 14
4. Routines of the RLPC steps - step 2
   4.1. prog_2_1_condensing_part_1.do ........................................... 15
   4.2. prog_2_2_identify_legalforms.do ........................................... 15
   4.3. prog_2_2_set_legform.do ..................................................... 15
   4.4. prog_2_3_clear_company_word.do ........................................... 16
   4.5. prog_2_4_clean_name.do ...................................................... 17
   4.6. prog_2_5_replace_spelling_variation.do ................................... 17
   4.7. prog_2_6_remove_brackets.do ................................................ 17

5. Routines of the RLPC steps - step 3 ........................................... 18
   5.1. prog_3_1_condensing_part_2.do ............................................. 18

A. Appendix .................................................................................. 20
   A.1. Code Statistics ...................................................................... 20

List of Figures

1. Flowchart: Structure of the procedure in steps. ........................... 4
2. Flowchart: Loading basic routines. .............................................. 5
3. Flowchart: Functions of step 1 routines. .................................... 12
4. Flowchart: Functions of step 2 routines. .................................... 14
5. Flowchart: functions of step 3 routines. ...................................... 18
1. Starting basis and outline of the procedure

The primary objective of record linkage is the merger of different data sets on the basis of an unique identifier. The cases at hand are mostly company data sets from databanks with company characteristics (e.g. BvD Amadeus/Dafne), universities and non-university research facilities (e.g. Research Explorer - see Ehrenfeld 2015b), as well as patent data sets (e.g. Patstat/Regpat or DPMA), publication data (e.g. Web of Knowledge) and funding data sets (e.g. BMBF funding catalog (“Förderkatalog”)). These data sets shall be linked on the basis of company names. Due to the fact that company names have varying notations in different databases - for example the corporate structure - a harmonization and standardization is necessary.

By using the procedures described here an adjusted, record linkage compatible name \texttt{RLName} is created from a given variable for the (actor) name \texttt{Name}. Therefore, the procedure is called record linkage pre-cleaning (short: RLPC). The following is a very brief description of the necessary steps. At the beginning of the procedure some basic variables and routines are defined. The are needed throughout the procedure. The procedure itself can be divided into three stages.

The first stage is a character clean up. For this, the actor name is completely converted to uppercase. It also includes the replacement of German umlauts, accented characters or characters with coding annotations with their ASCII equivalents. Double spaces in the name and spaces at the beginning or end of the name are removed. Subsequently, bracket symbols and notations for “and” are unified and bracketed expressions are extracted.

In a second step all non-ASCII characters are deleted from the name. Then the company structures of business enterprises are identified. This is done through an identification table, which currently holds more than 600 notations for different company structures. The original notations of the corporate structure are subsequently deleted from the company name. Thereafter, notations of frequently used terms with variable spellings are harmonized. Finally, all spaces are removed from the expression.

The routines in this report are based on the methods for the harmonization of patent data described by Magerman, Van Looy, and Song (2006). However, they have been extensively modified and expanded for the use with the data sets at hand, and translated into a modular system of Stata routines. After the pre-cleaning pretreated data sets can be merged. These procedures are well suited for the allocation of slightly varying notations for the same actor name.

From a technical point of view the problem of slightly differing notations is very different from the problem of varying denotations for the same institution. Well-known examples for this are Technical Universities (and their frequently used abbreviation “TU”) or the “classic” Rheinisch-Westfälische Technische Hochschule Aachen (short: RWTH Aachen). In these cases purely deterministic allocation of original data records or “fuzzy” (probabilistic)
methods alone can hardly ensure a reliable allocation. Instead, these cases can be standard-
ized by means of automatized replacement rules or through an additional table for different
notations of the same institution.

Record linkage systems that have been used in the past include the “Merge-Toolbox”
(Schnell, Bachteler, and Reiher 2005 or Schnell, Bachteler, and Bender 2004) and the
commercial software “Fuzzy Dupes”. An implementation in the course of the project
“RegDemo” can be found in Titze et al. (2015) and Ehrenfeld (2015a).

Figure 1 depicts the structural sequence of pre-cleaning. The following describes the
individual stages of the procedure more in-depth.

2. Basic routines

In this stage basic variables are defined and routines are loaded. This allows for the routines
to be provided as programs for the procedure. Figure 2 depicts the sequence of these basic
routines.

2.1. PreCleaning.do

This is the superordinate control file. It is used to activate all other routines. Here the paths
to routines and data sets are defined, the superordinate time measurement is controlled and
additional variables necessary for the RLPC are defined. The procedure uses the following
global paths and file names:

Figure 1: Flowchart: Structure of the procedure in steps.
$workdir$ is the working directory. This is where the general logfile $logname$ is saved.

$progdir$ is the path to the programs, which are saved as do-files.

$datadir$ is the path to the data sets $load_dataset$ and $save_dataset$.

$load_dataset$ is the file name of the data set to be edited.

$save_dataset$ is the file name of the treated data set.

$logname$ is the file name of the logfile.

The newly created variables are:

• $RLName$ is generated from the variable $Name$ and will later on contain the name that has been adjusted, translated into ASCII characters and, if applicable, condensed. This variable represents the main outcome of this procedure.
• temp_name is the \texttt{RLName} from the last stage. It is used later on for a comparison with the \texttt{RLName} from the current stage. This is helpful to identify changes (necessary for \texttt{clean_hist}).

• clean_hist states the steps actually applied to the individual entries of \texttt{RLName}.

• legal_form is a company’s legal form, which is identified and separated in step 2.2 (see sections 4.2 and 4.3).

• brackets contains bracketed expressions potentially separated during step 1.5 (see section 3.5).

2.2. Programs\textunderscore Load.do

Programs\textunderscore Load.do loads all of the following do-files and provides the programs contained therein.

2.3. Programs\textunderscore Basic\textunderscore Routines.do

The file Programs\textunderscore Basic\textunderscore Routines.do contains several basic programs for the modification of the string \texttt{RLName}. Mainly these are commands for replacing character strings.

\textbf{RLreplace}

\texttt{RLreplace} replaces the character string \texttt{<search string>} with the character string \texttt{<replacement string>} in \texttt{RLName} - regardless of the position of \texttt{<search string>} in \texttt{RLName} (normal replacement).

\textit{Usage:} \texttt{RLreplace "<search string>" "<replacement string>"}

\textbf{replace\_at\_end}

This command replaces the character string \texttt{<search string>} with the character string \texttt{<replacement string>} in \texttt{RLName} if \texttt{<search string>} is located at the \textit{end} of \texttt{RLName}.

\textit{Usage:} \texttt{replace\_at\_end "<search string>" "<replacement string>"}

\textbf{replace\_at\_beginning}

This command replaces the character string \texttt{<search string>} with the character string \texttt{<replacement string>} in \texttt{RLName} if \texttt{<search string>} is located at the \textit{beginning} of \texttt{RLName}.

\textit{Usage:} \texttt{replace\_at\_beginning "<search string>" "<replacement string>"}
replace_in_middle

This command replaces the character string `<search string>` with the character string `<replacement string>` in `RLName` if `<search string>` is located in the middle of `RLName`. (see RLreplace).

Usage: `replace_in_middle "<search string>" "<replacement string>"`

RLtrim

RLtrim deletes all spaces at the beginning and the end of `RLName`, as well as all double spaces within `RLName`.

Usage: RLtrim

2.4. Programs_Module_Management.do

start_outer_timer

Starts time measurement through the outer/superordinate timer (timer 1). Opens the timelog for writing in `$workdir/$timelog`.

$timelog = "$logname" + "_TimeStats.log"

Usage: `start_outer_timer`

stop_outer_timer

Stops the outer timer (timer 1) and closes the timelog $timelog.

Usage: `stop_outer_timer`

begin_step

Can be found at the beginning of a new `<step>` of the procedure. Starts the inner timer (timer 2) and replaces the temporary name of the last step (`temp_name`) with the current `RLName`.

Usage: `begin_step "<step>"`
end_step

Can be found at the end of a <step> of the procedure. Deletes superfluous spaces in \texttt{RLName} (using \texttt{RLtrim}), stops the inner timer (\texttt{timer 2}), writes the inner time measurement into the \texttt{timelog} and updates the entries on the actually applied steps (\texttt{update_clean_hist}).

\textit{Usage:} \texttt{end_step "<step>"}

begin_substep

Can be found at the beginning of each new <sub step> of the procedure. Starts the timer on level 3 (\texttt{timer 3}).

\textit{Usage:} \texttt{begin_substep "<sub step>"}

end_substep

Can be found at the end of a <sub step> of the procedure. Stops the timer on level 3 (\texttt{timer 3}) and writes the time measurement into the \texttt{Timelog}.

\textit{Usage:} \texttt{end_substep "<sub step>"}

update_clean_hist

Writes the inner time measurement (\texttt{timer 2}) into the \texttt{timelog} and updates the entries on the actually applied steps in \texttt{clean_hist}. This lists only the numbers of the steps which actually brought about an alteration in \texttt{RLName}. The identification is done by comparing \texttt{temp_name} and \texttt{RLName}.

\textit{Usage:} \texttt{update_clean_hist}

2.5. Programs\_ASCII\_Routines.do

Each of these routines eliminates a different set of ASCII characters in \texttt{RLName}. They are used for the string condensing in step 2.1 or 3.1 of the RLPC routines (see section 4.1 or section 5.1).
**remove_ascii_special_chars**
This command replaces some “special” ASCII characters in **RLName** with spaces. They are the following:

<table>
<thead>
<tr>
<th>Number</th>
<th>33-39</th>
<th>42</th>
<th>44-47</th>
<th>58-59</th>
<th>61</th>
<th>63-64</th>
<th>92</th>
<th>94-96</th>
<th>124</th>
<th>126</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>! &quot; # $ % &amp; ´ * , . / : ; = ? @ \ ^ _</td>
<td>~</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Usage:** remove_ascii_special_chars

**remove_special_chars**
This command replaces the following special characters in **RLName** with spaces:

\[ \frac{2 \cdot \pi \mu \$}{\phantom{1000}} \]

**Usage:** remove_special_chars

**remove_ascii_std_chars**
This command replaces all “normal” ASCII characters in **RLName** with spaces. It is only used for testing purposes to form the difference of all characters in **RLName** to the “special” characters.

These characters are:

<table>
<thead>
<tr>
<th>Number</th>
<th>32</th>
<th>40-41</th>
<th>48-57</th>
<th>65-90</th>
<th>97-122</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>&lt;Space&gt;</td>
<td>( )</td>
<td>0-9</td>
<td>A-Z</td>
<td>a-z</td>
</tr>
</tbody>
</table>

**Usage:** remove_ascii_std_chars

**char_condensing_one**
This routine replaces all characters in **RLName** that are not A-Z, 0-9, (, ) or <space> with spaces. A suitable regular expression is used to do this.

**Usage:** char_condensing_one

Wilfried Ehrenfeld

9

30.11.2015
char_condensing_two

This routine deletes all characters in RLName that are not A-Z or 0-9. A suitable regular expression is used to do this.

In contrast to char_condensing_one, this routine deletes affected characters. Furthermore, spaces are also deleted here.

Usage: char_condensing_two

char_condensing_RL2

This routine matches char_condensing_two for (the experimental) RLName2.

Usage: char_condensing_RL2

condense_spacing_RL2

This routine condenses (deletes spaces) names in letter spacing in RLName2. The names containing letter spacing are identified by their structure following the form “at least three single letters with spaces between them”. A suitable (nontrivial) regular expression is used for this. Subsequently, the routine deletes all superfluous (double) spaces in RLName2.

Usage: condense_spacing_RL2

2.6. Programs_Legform_Routines.do

These routines are needed for the identification and separation of legal forms in step 2.2 of the RLPC routines (see sections 4.2 and 4.3).

legform_detect

This routine recognizes legal forms in RLName and replaces them with their short forms in RLName. The short form is made recognizable for subsequent steps by inserting « and » (e.g. «KG»). The routine takes particularly short long forms (shorter than 5 characters) at the beginning and the end of RLName into account. These have to be separated by an additional space in RLName. For recognition in the middle the long form has to be separated by spaces before and behind it. This measure is implemented in order to reduce incorrectly (positive) recognized corporate structures.

The list of legal forms (long and short) is:

  step_2_2_legal_form_replacement.tsv.
This file has to be in the program directory \$progdir and should be sorted by the length of the long form in descending (!) order. This helps recognizing special forms (e.g. GGMBH vs. GMBH).

The list currently comprises more than 620 entries. It contains two columns, which are separated by the <TAB>-character (file extension tsv). It has the following format:

<search text/long form><TAB><short form><CR LF>.

When searching for the legal form, the routine can be told to search at the beginning, the end or in the middle of RLName. Therefore, “replace_at_end”, “replace_at_beginning” or “replace_in_middle” can be used as parameters for <replacetype>.

Usage: legform_detect "<replacetype>"

set_legform

This command enters the legal form <set_string> into the (empty) variable legal_form if <search_string> is contained in RLName. This means that the first entry found is maintained. Typically, the <search_string> is the short form of the legal form marked by « and » (e.g. «KG») from step 2.2 or the routine legform_detect. The expression <set_string> is then typically the “normal” short form.

Example: set_legform "<GMBH>" "GMBH".

Usage: set_legform "<search_string>" "<set_string>"

3. Routines of the RLPC steps - step 1

This is where the actual procedure starts. The first step of the RLPC procedure replaces symbols in RLName or transfers them to standard ASCII characters. Figure 3 shows the sequence of this first step.

Each program of the RLPC procedure in step 1 through 3 is “framed” by begin_step and end_step. Oftentimes RLtrim is run in a sub-step as the last routine.
3.1. prog_1_1_uppercase.do

This routine converts all letters of RLName into uppercase. Due to the fact that Stata ignores some characters when using the upper-function, these characters are also replaced with their uppercase versions. This also relates to German umlauts. Furthermore, “ß” is converted into “SS”.

Usage: prog_1_1_uppercase

Routines used: RLreplace

3.2. prog_1_2_replace_smgl_codes.do

This program translates special characters of the Standard Generalized Markup Language (SGML) that have not been resolved so far. This includes characters like “&AMPL;.”
Usage: prog_1_2_replace_smgl_codes
Routines used: RLreplace

3.3. prog_1_3_replace_coded_chars.do

This program replaces unresolved specially coded characters with their simplified ASCII variants. For instance, this routine replaces “{OVERSCORE (A)}” with “A”.
Usage: prog_1_3_replace_coded_chars
Routines used: RLreplace

3.4. prog_1_4_replace_bracket_symbols.do

Here the bracket symbols [, { and < are replaced with (. The symbols ], } and > are replaced with ). Double angle brackets (« and ») are deleted.
Usage: prog_1_4_replace_bracket_symbols
Routines used: RLreplace

3.5. prog_1_5_get_bracket_content.do

This routine identifies bracketed expressions in RLName. If a bracketed expression is identified, it is extracted from RLName and saved in the variable brackets. Should RLName be completely enclosed in parentheses, only the bracket symbols are removed from RLName. In this case the variable brackets contains the text “Steht komplett in Klammern”.

The content in parentheses in RLName is deleted later on in step 2.6 (section 4.7).
Usage: prog_1_5_get_bracket_content
Routines used: RLtrim

3.6. prog_1_6_replace_ascended_chars.do

This program translates accented characters into their simplified ASCII variants. Example: “À” is replaced with “A”. The German umlauts Ä, Ö, Ü are replaced with AE, OE, UE.
Usage: prog_1_6_replace_ascended_chars
Routines used: RLreplace
3.7. prog_1_7_replace_and.do

This routine replaces terms for “AND” such as “+”, “AND”, “UND”, “U.” and “ET” with an ampersand (&).

Usage: prog_1_7_replace_and

Routines used: RLreplace

4. Routines of the RLPC steps - step 2

The second step of the RLPC procedure condenses characters, isolates corporate structures, adjusts notations and removes bracketed expressions. Figure 4 shows the sequence of this second step.

Figure 4: Flowchart: Functions of step 2 routines.
4.1. prog_2_1_condensing_part_1.do

This program removes ASCII special characters and other special characters by use of
the functions remove_ascii_special_chars and remove_special_chars. This pre-cleaning
measure speeds up the subsequent Regex-based routine significantly.

Afterwards all characters in RLName that are not A-Z, 0-9, (,) or <space> are replaced
with spaces by means of char_condensing_one.

Finally, spaces immediately following opening parentheses are deleted, while also making
sure that there are spaces immediately before all opening parentheses. The same is done
for closing parentheses.

Usage: prog_2_1_condensing_part_1

Routines used: remove_ascii_special_chars; remove_special_chars;
RLreplace; RLtrim

4.2. prog_2_2_identify_legalforms.do

This program identifies the legal forms of companies and replaces them with identifiable
short forms with the help of routine legform_detect. To do this the legal forms are searched
for sequentially, first at the beginning, then at the end and finally in the middle.

Since this process is relatively time-consuming, the time measurements for these sub-steps
are recorded separately and written into the timelog (begin_substep; end_substep).

After each of these sub-steps the legal form is written into the variable legal_form by
means of step_2_2_set_legform.

Usage: prog_2_2_identify_legalforms

Routines used: legform_detect; step_2_2_set_legform; RLtrim

4.3. prog_2_2_set_legform.do

This do-file defines the program step_2_2_set_legform, which writes all of the identified
short forms of legal forms from RLName into the (empty) variable legal_form using the
function set_legform.

Usage: step_2_2_set_legform

Routines used: set_legform
4.4. prog_2_3_clear_company_word.do

This program defines the routine clear_company_word, which replaces company words from RLName that are frequently used but not evaluated here with the place holder “«»”. These company words include, for example, “INTERNATIONAL CORPORATION” or “GESELLSCHAFT”. The double angle brackets “«»” are treated later on in step 2.4 (section 4.5).

It can be specified, whether during the course of the search the company words with a length of less than 5 characters should be separated with spaces (type = 0) or not (type = 1). The number and position of the spaces depends on the search position of the company words (beginning, middle, end) and are used in such a way to consider the various uses of company words.

The list of company words is:

```
step_2_2_common_words.tsv.
```

This file must be located in the program directory $progdir and should be sorted by the length of the company words in descending (!) order. This way it is easier to recognize special forms, since they are queried before the general forms.

The list contains two columns, which are separated by the <TAB>-character (file extension tsv). It has the following format:

```
<company word><TAB><type><CR LF>.
```

When searching for the company words, the routine can be told to search at the beginning, the end or in the middle of RLName. Therefore, “replace_at_end”, “replace_at_beginning” or “replace_in_middle” can be used as parameters for <replacetype>.

Usage: clear_company_word "<replacetype>"

Routines used: -

Subsequently, this step sequentially runs the program clear_company_word described above with the parameters “replace_at_end”, “replace_at_beginning” or “replace_in_middle”.

Usage: prog_2_3_clear_company_word

Routines used: clear_company_word
4.5. prog_2_4_clean_name.do

Firstly, this routine defines the program clean_name, which deletes the predefined string "«legal form»" from RLName.

Usage: clean_name "«legal form»"

Routines used: -

Secondly, clean_name is run for all known legal forms and "«»", in order to remove them from RLName.

Usage: prog_2_4_clean_name

Routines used: clean_name; RLtrim

4.6. prog_2_5_replace_spelling_variation.do

This routine identifies a search string in RLName and replaces it with a predefined replacement string. Here, this routine is used to unify notations for company extensions (e.g. INTERNATIONALE → INTERNATIONAL).

The list of company extensions and the respective replacement strings is:

prog_2_5_replace_spelling_variation.tsv.

This file must be located in the program directory $progdir. The list contains entries in two columns, which are separated by the <TAB>-character (file extension tsv). It has the following format:

<search string><TAB><replacement string><CR LF>.

Usage: prog_2_5_replace_spelling_variation

Routines used: -

4.7. prog_2_6_remove_brackets.do

Here, all contents of RLName that are enclosed in parentheses are deleted. The parentheses are removed as well. The contents of these parentheses have already been transferred into the variable brackets in step 1.5 (section 3.5).

Usage: prog_2_6_remove_brackets

Routines used: -
5. Routines of the RLPC steps - step 3

The third and last step of the RLPC procedure condenses all characters in \texttt{RLName}, finalizing the now RL compatible variable by doing so. Figure 5 depicts the sequence of this third and last step.

![Flowchart: functions of step 3 routines.](image)

5.1. \texttt{prog\_3\_1\_condensing\_part\_2.do}

This program deletes all spaces in \texttt{RLName}, as well as parentheses ("(" and ")"). As a precaution, all characters in \texttt{RLName} that are not A-Z or 0-9 are subsequently deleted by using \texttt{char\_condensing\_two}.

\textit{Usage: prog\_3\_1\_condensing\_part\_2}

\textit{Routines used: RLreplace; char\_condensing\_two}
References


### A. Appendix

#### A.1. Code Statistics

Stand: Juli 2015

<table>
<thead>
<tr>
<th>Modul</th>
<th>Anzahl Zeilen</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Basic routines</strong></td>
<td></td>
</tr>
<tr>
<td>PreCleaning.do</td>
<td>150</td>
</tr>
<tr>
<td>Programs_Load.do</td>
<td>34</td>
</tr>
<tr>
<td>Programs_Basic_Routines.do</td>
<td>71</td>
</tr>
<tr>
<td>Programs_Module_Management.do</td>
<td>126</td>
</tr>
<tr>
<td>Programs_ASCII_Routines.do</td>
<td>125</td>
</tr>
<tr>
<td>Programs_Legform_Routines.do</td>
<td>91</td>
</tr>
<tr>
<td><strong>Routines RLPC step 1</strong></td>
<td></td>
</tr>
<tr>
<td>prog_1_1_uppercase.do</td>
<td>98</td>
</tr>
<tr>
<td>prog_1_2_replace_smgl_codes.do</td>
<td>52</td>
</tr>
<tr>
<td>prog_1_3_replace_coded_chars.do</td>
<td>50</td>
</tr>
<tr>
<td>prog_1_4_replace_bracket_symbols.do</td>
<td>28</td>
</tr>
<tr>
<td>prog_1_5_get_bracket_content.do</td>
<td>38</td>
</tr>
<tr>
<td>prog_1_6_replace_ascended_chars.do</td>
<td>61</td>
</tr>
<tr>
<td>prog_1_7_replace_and.do</td>
<td>23</td>
</tr>
<tr>
<td><strong>Routines RLPC step 2</strong></td>
<td></td>
</tr>
<tr>
<td>prog_2_1_condensing_part_1.do</td>
<td>34</td>
</tr>
<tr>
<td>prog_2_2_identify_legalforms.do</td>
<td>37</td>
</tr>
<tr>
<td>prog_2_2_set_legform.do</td>
<td>57</td>
</tr>
<tr>
<td>prog_2_3_clear_company_word.do</td>
<td>82</td>
</tr>
<tr>
<td>prog_2_4_clean_name.do</td>
<td>70</td>
</tr>
<tr>
<td>prog_2_5_replace_spelling_variation.do</td>
<td>44</td>
</tr>
<tr>
<td>prog_2_6_remove_brackets.do</td>
<td>25</td>
</tr>
<tr>
<td><strong>Routines RLPC step 3</strong></td>
<td></td>
</tr>
<tr>
<td>prog_3_1_condensing_part_2.do</td>
<td>27</td>
</tr>
<tr>
<td><strong>Anzahl Module</strong></td>
<td>27</td>
</tr>
<tr>
<td><strong>Codezeilen Gesamt</strong></td>
<td><strong>1323</strong></td>
</tr>
</tbody>
</table>