Accelerator Keys in LibreOffice

Write Accelerator Key settings to text 2025-12-22

Accelerator keys (short keys) are editable in “Tools - .Customize”. But they are not easily visible there. The here preferred tool converts the output of shortcuts ([Save] button) to a human proper enough readable text file.

Table of Contents

1 Approach, why yet another Accelerator Key presentation

Accelerator keys are key combinations which forces execution of complete commands. They are able to customize in the dialog box opened with *"Tools – Customize"*.

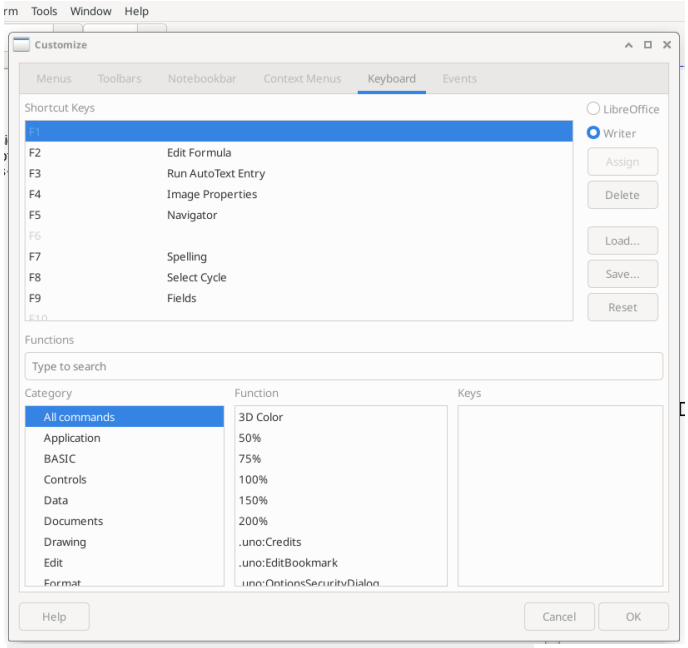

Figure 1: DialogBoxCustomizeKeyboard.png

Note: In opposite the term ***short cut*** is used for **menu entries** in combination with Alt, whereas ***hot key*** is usual used for a key combination on operation system level to enforce an action.

Accelerator Keys are a proper opportunity to writing a document in a fast way. The alternative is: Using the mouse for formatting, for insert special characters etc. Using the mouse is straight forward – you see the menu entries, explanations etc. But you need more time doing it.

For example format a part of the text in an often used special character style. Doing that with the mouse means, select the styles sidebar, search and click the character styles tab (per default paragraph styles are opened), search with the eyes or unfold in the hierarchical view the dedicated style group, found, double click, but now go with the cursor back to the text, select the next to associate the style, and go forward.

Using a known accelerator key is very more simple. You do not need the mouse, do it while typing, select the text with shift-cursor, press Shift+Alt-C which may be associated to your desired character style, and go forward.

But you should set your desired keys by yourself (or do it in a team with same accelerator keys one time for all, if you get consense).

There is a problem: Because the developer of a software also likes working with hot keys, many keys are assigned to a lot of opportunities, and you do not know all of them − hence you do not use it.

Another problem: You type a key combination on accident − you don't know what's happen. You are confused about the result.

The next image shows how to associate a character style to an accelerator key − only as example. The opportunities are great.

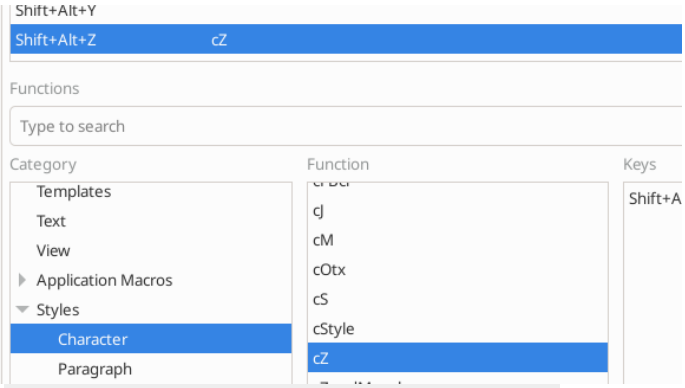

Figure 2: DialogBoxCustom_KeyCharStyle.png

But the overview in this Dialog box is not the best. It is not possible because all keys are presented. The order of keys is well sorted, first without modifying key, then with Shift, with Ctrl only, then with Shift+Ctrl, with Alt only, with Alt+Shift, Ctrl+Alt and at least with all three Shift+Ctrl+Alt modifying keys. But also this is not proper able for overview, there is to less space on screen.

Right side in this dialog box there are two buttons **[Load]** and **[Save]**, beside **[Reset]**. **[Save]** asks a file name, and creates a file with the default extension `.cfg`. This file can be used for **[Load]** for example on another PC to enforce the same key settings.

The problem is the overview over all accelerator key settings. For that also the comparability of a plain text output and also the sorting order of such a list is important.

2 The solution, accelerator key setting translation to text

This documentation offers a small tool written in Java which translates the saved `.cfg` file of the accelerator key settings in a plain text human readable and comparable list.

The tool is given in form of the downloadable zip file https://vishia.org/LibreOffc/deploy/vishiaKeyAcclText-2025-12.zip or possible later versions.

2.1 What does the vishiaKeyAcclText.zip file contain

The file `vishiaKeyAcclText-yyy-mm-dd.zip` does contain only a very small jar file to support download the necessary Java jar files to work, and some script files, `.sh` for Linux/UNIX-Systems and **.bat** for Windows to support a simple execution:

```
vishiaKeyAcclText/tools
 +-tools
 |  +-tools-vishiaLibOffc.bom
 |  +-+checkLoadTools-vishiaLibOffc.bat
 |  +-+checkLoadTools-vishiaLibOffc.sh
 |  +-vishiaGetWebfile-2025-12-21.jar
 +-KeyAcclText.bat
 +-KeyAcclText.sh
 +-KeyAcclText.gTxt
```

The necessary jar files are determined in the given `tools....bom`, a *Bill Of software-Material* file. This is a simple text file containing the URL to load the file, the destination (it is `./`, the `tools` directory itself) and a MD5 check sum. The MD5 check sum is checked while downloading, to assure non modified files.

General, the jar files can be stored also on a more central position because they contain also some other usable capabilities. But this quest is to clarify later. First make a simple test.

2.2 How to use the KeyAcclText translation tool the first time

What should you do:

- Unpack the zip file to any directory, recommended a `...\Programs\` on Windows or in the `/home/.../` on Linux.
- Look in the `tools/tools-vishiaLibOffc.bom` for interest, and start for downloading the jar files one of the given `+checkLoadTools-vishiaLibOffc.sh` or `...bat`.

  The download should be performed with correct MD5 check (output shortened):

```
Resolve dependencies, check tools/tools.bom
check MD5 from: ./vishiaGetWebfile-2025-12-
  : read 4816 bytes, MD5=bb8cc034f65edab202
copy from URL: https://www.vishia.org/Java/
  to: ./vishiaBase-2025-12-20.jar  (/tmp/ra
  ..................: read 1330432 bytes,
copy from URL: https://www.vishia.org/Java/
  to: ./src/vishiaBase-2025-12-20-source.zi
  ........................: read 1732030
copy from URL: https://www.vishia.org/Java/
  to: ./vishiaLOffcTools-2025-12-20.jar  (/
  ....: read 296998 bytes, MD5=045c6307f059
copy from URL: https://www.vishia.org/Java/
  to: ./src/vishiaLOffcTools-2025-12-20-sou
  ..: read 149729 bytes without check sum
 ...Press ENTER...
```

- After this action the tools are completed, the jar files are loaded, and additionally. also the sources of the jar files are loaded into a `tools/srcTools` directory. This allows reproducible build of the jar files (see https://vishia.org/Java/html/source+build/src_Archive.html
- Then you can write accelerator key settings immediately into this `vishiaKeyAcclText` directory (from the unpacked zip) using the [Save] button in the Accelerator key dialog box as `NAME.cfg`
- And then calling the `KeyAcclText.sh` or `~.bat` to convert. The standard arguments for input are `-i:*.cfg`.
- This generates the textual output file `NAME.shift.txt` proper to the `NAME.cfg` file.

But see hints in the next chapter.

2.3 Set path to LibreOffice installation

The `KeyAcclText` execution needs information about translation between the ***uno*** commands which are frequently used in the written XML file on *Acceleration Key Customize-Dialog − [Save]*. To get this information, some XML files which are a part of the LibreOffice installation are used. For that the LibreOffice-Installation directory should be known. With the option `-LOffcPrg:...` it is offered.

The installation directory for LibreOffice have a default path for Windows and Linux, as used in the both command scripts `KeyAcclText.sh` and `~.bat`, But it is possible that the LibreOffice installation path is another one. This should be adjusted in the scripts changing the option `-LOffcPrg:...`, A faulty path forces an error message but the translation runs anyhow, only doesn't translate the uno commands.

2.4 Call of the translation from cfg to human readable text

This is a view to the console, the current directory is the unpacked zip file

The called command is `./KeyAcclText.sh`:

```
hartmut@hartmut-70:~/Settings/LibreOffice$ ./KeyAcclText.sh
```

The following output is:

```
called: ./KeyAcclText.sh
SCRIPTDIR=/home/hartmut/Settings/LibreOffice
currdir = /home/hartmut/Settings/LibreOffice
-i:/home/hartmut/Settings/LibreOffice/*.cfg
/home/hartmut/Settings/LibreOffice> java -cp /home/D/jars/vishiaLOffcTools-2025-12-22.jar:/...

reads XML: /opt/libreoffice25.8/share/registry/main.xcd
reads XML: /opt/libreoffice25.8/share/registry/writer.xcd
reads XML: /opt/libreoffice25.8/share/registry/draw.xcd
reads XML: /opt/libreoffice25.8/share/registry/calc.xcd
reads accelerator key XML from /home/hartmut/Settings/LibreOffice/AcclKeyWriter.cfg,
  internal in zip: 'Configurations2/accelerator/current.xml' ... done
writes accelerator key text: AcclKeyWriter.shift.txt in /home/hartmut/Settings/LibreOff ... done

/home/hartmut/Settings/LibreOffice/ ... done
RETDIR=/home/hartmut/Settings/LibreOffice
 ...Press ENTER...
```

The file `AcclKeyWriter.cfg` was written as output from the accelerator key dialog box, or copied to here.

The `KeyAcclText.sh` was called.

The result is the new written `drawKeyboardDraw.shift.txt`.

After script execution the command `ls -all` was called:

```
hartmut@hartmut-70:~/Settings/LibreOffice$ ls -all
```

The result shows the situation in this directory. This is Linux.

```
total 56
drwxr-xr-x 2 hartmut hartmut  4096 Dec 31 16:54 .
drwxr-xr-x 3 hartmut hartmut  4096 Dec 23 00:09 ..
-rw-r--r-- 1 hartmut hartmut  1390 Dec 23 00:15 AcclKeyLOffcWriter.cfg
-rw-r--r-- 1 hartmut hartmut 15447 Dec 31 16:49 AcclKeyLOffcWriter.shift.txt
-rw-r--r-- 1 hartmut hartmut  2583 Dec 23 00:15 AcclKeyWriter.cfg
-rw-r--r-- 1 hartmut hartmut 12734 Dec 31 16:49 AcclKeyWriter.shift.txt
-rw-r--r-- 1 hartmut hartmut   534 Dec 22 15:46 KeyAcclText.gTxt
-rwxrwxrwx 1 hartmut hartmut  1745 Dec 31 16:55 KeyAcclText.sh
hartmut@hartmut-70:~/Settings/LibreOffice$
```

The situation on Windows with call of `KeyAcclText.bat` is similar.

2.5 Explanation of the translation script with its options

The called shell script `KeyAcclText.sh` is the following, note that the bat file for Windows is similar:

```
#!/bin/sh
## This file is the shell script to call convert accelerator key settings from cfg to human
readable text
echo called: $0 $1 $2 $3 $4
export SCRIPTFILE="$(realpath $0)"          ## The absolute path of this script file from the
invocation dir
export SCRIPTDIR="$(dirname $SCRIPTFILE)"   ## The absolute path of this script dir
echo SCRIPTDIR=$SCRIPTDIR
## set the input
if test "$1" != ""; then export INFILES="$(realpath $1)"
else export INFILES="*.cfg"
fi
echo -i:$INFILES
##
if test "$OS" = "Windows_NT"; then ##Windows with shell script capability for example MinGw
  JCPSEP=";"
  JAVA=java
else                          ## real Linux/UNIX
  JCPSEP=":"
  JAVA=java
fi
TOOLSDIR="tools"   ## maybe changed, maybe absolute path to the jar file directory
JCP="$TOOLSDIR/vishiaLOffcTools-2025-12-22.jar${JCPSEP}$TOOLSDIR/vishiaBase-2025-12-31.jar"
echo "$PWD>" java -cp $JCP org.vishia.libOffc.cfgui.WriteKeyAccl --@$SCRIPTFILE:args
$JAVA -cp $JCP org.vishia.libOffc.cfgui.WriteKeyAccl --@$SCRIPTFILE:args
### args ##
### -i:$INFILES
### -LOffcPrg:/opt/libreoffice25.8             ## Libre Office installation here
### -LOffcUser:~/.config/libreoffice/4/user    ## User settings here
### -gTxt:$SCRIPTDIR/KeyAcclText.gTxt          ## output script
### -shfollow            ## sorted shift key variant in order of key names
### -o:*.shift.txt       ## output is INFILES name with given extension
if test "$1" != "NOPAUSE" -a "$2" != "NOPAUSE"; then read -p " ...Press ENTER..." VAR; fi
```

The shell script `KeyAcclText.sh` is partially commented and regards calling from another directory. For that the variable `SCRIPTDIR` is set.

If no input file(s) are given (argument on shell script call, `$1`), then as default in line 10, `*.cfg` is taken, regarded to the current directory. The `*` is admissible to convert all files with this pattern.

The arguments of the java call are given in the shell script file itself, as comment for the shell script execution. The java call command line contains only `--@$SCRIPTFILE:args`. It means the script file is the argument file, one argument line per line, starting after the label `args` on a left side position after possible (necessary for the shell script) comment characters, regarding `##` as end line comment string itself.

`-LOffcPrg:/opt/libreoffice25.8`: Here the paths to the LibreOffice installation is given.

`-LOffcUser:...` The path to the user configuration is given here, but it is not used yet.

The option `-o:*shift.txt` is the pattern to build the output file, whereas the `*` is replaced by the name of the input file.

The option `-gTxt:$SCRIPTDIR/KeyAcclText.gTxt` references to the file `KeyAcclText.gTxt`. This file contains the complete pattern how the output is written, and also which internal data are accessed. See *3.4 Output the text* page *9*

For windows usage the `KeyAcclText.bat` file is adequate.

2.6 Persistently using this translation tool

The first quest is, why it isn't a part of the LibreOffice installation itself. The answer for that is: It may be possible, but it is a fundamental decision. The capability to run tools outside of LibreOffice using standardized LibreOffice file formats is a proper capability.

One possibility is, start the translation from inside LibreOffice by a icon button, which starts is assigned to a LibreOffice Macro. The macro then starts the translater.

Secondly, the jar files which are used here contains more capabilities. The here given jar files contains also the translation between a plain text format ZmL for Writer documents to LibreOffice-Writer odt, see https://vishia.org/LibreOffc/html/Videos_LOffcZmL.html.

Adding two more vishia-jar files, and **the.File.commander** runs also on Linux, which is an alternative to the known Midnight Commander or the Total-Commander and its derivations, only for example. Or, with one other jar file the translation from specific LibreOffice Draw graphics to embedded system target code runs (https://vishia.org/fbg/html/Videos_OFB_VishiaDiagrams.html)

It means it is worthwhile to spend a central position on your hard disk for these files (especially the tool directory), preferred on Windows in `C:\Programs\vishia\tools` or in Linux in `/usr/share/vishia/tools`. or locally stored for my usage: `/home/hartmut/jars`. Then only the `KeyAcclText.sh` or `~.bat`. scripts should be adapted for the `TOOLSDIR`, and stored in a proper working directory also possible in the system's `PATH`. .

I have organized a directory in `/home/hartmut/Settings/LibreOffice`. In this directory the `+checkLoadTools.sh` is stored. This directory is also used as output for [save] from the Accelerator key dialog box for the `*.cfg` files.

It means, on changing keys the `cfg` is saved with a proper name, for example `KeyAcclWriterLOffc.cfg`, then this shell script is started, and then all `*.cfg` files there are translated to text, and can be compared or viewed.

What is the difference if the Key Acceleration Setting is store for "LibreOffice" or for "Writer" or "Draw" or any what? Compare both textual scripts with a textual diff viewer, you see it.

2.7 Output, Key settings with shift follow

The output looks like (only a snippet)

```
        Left    "To Character Left"                ==>.uno:GoLeft
Shift-  Left    "Select Character Left"            ==>.uno:CharLeftSel
        PgDn    "Next Page"                        ==>.uno:PageDown
Shift-  PgDn    "Select to Next Page"              ==>.uno:PageDownSelctrl-Bspc "Delete to Sta...
   Ctrl- -      "Insert Soft Hyphen"               ==>.uno:InsertSoftHyphen
Sh+Ctrl- -      "Insert Non-breaking Hyphen"       ==>.uno:InsertHardHyphen
Sh+Ctrl- 0      "Default Paragraph"                ==>.uno:StyleApply?Style:string=Standard&Fa...
   Ctrl- 1      "Heading 1"                        ==>.uno:StyleApply?Style:string=Heading 1&F...
```

It may be interesting, that the key binding with shift follows immediately the key binding without shift. This is meanwhile not due to the option `-shfollow`, it is clarified by usage of the variable `idxKeyAcclShift` instead `idxKeyAccl` in the script `KeyAcclText.gTxt`. Often the shift key is used or seen as *'modification'*. Typical (for example in Eclipse), `ctrl-F` is *"search"*, and `ctrl-sh-F` is *"search in all files"*. Or as seen here: `ctrl-Bspc` is *"Delete to ... word"* and `ctrl-sh-Bspc` is *"Delete to ... sentence"* as original LibreOffice setting. It is helpful when this key settings are written one after another.

For comparability the sorting of the keys is essential. It is possible to compare two files, settings from another PC, or from an older time ... and see and discuss differences.

The question, which accelerator key settings are sensible or recommended, this is not discussed here. But it is to discuss.

2.8 Update translation tools

It is also possible to go to the archive: https://www.vishia.org/Java/deploy/ and https://vishia.org/LibreOffc/deploy/ (see `tools/tools-vishiaLibOffc.bom`) and pick up proper files from there. The both files needs a time stamp from and after 2025-12-22. The directory contains also the correspond `...md5` file to each jar file. This file contains as text the check sum and the information which `vishiaBase-yy-mm-dd.jar` was used for translation to jar. But also newer `vishiaBase-yy-mm-dd.jar` files can be used as mix, should be usual compatible.

2.9 What about back translation to accelerator key settings

It seems to be that this is also interesting, because changing and adding accelerator key settings in the plain text may be done faster than in the dialog box in LibreOffice. But is this worthwhile? It may need only a little bit more time to search the mentioned operations and keys in the dialog. In opposite, writing an immediately simple text should be checked afterwards, with error messages − lets see in future whether it may be required.

2.10 Which accelerator keys are sensible, should be used

The question, which accelerator key settings are sensible or recommended, this is not discussed here. But it is to discuss.

3 How does it work internally

This are information for insider or interests.

3.1 XML reading

The important functionality is reading an XML file and analyze its content. This is done by the https://vishia.org/Java/html/RWTrans/XmlJzReader.html. This is done also for the input XML file, as also for some more files from the LibreOffice installation.

The quest is, why using another XML reader, not using proven solutions. One answer is, the core libraries in standard java contain the concept for XML reading, but do not support the immediately translation to a user specific data set. For that the >>`XmlJzReader` class was developed by me, and hence used here. It is a part of the standard `vishiaBase.jar` file.

3.2 Translate to human readable commands instead uno

The `KeyAcclText` execution needs information about translation between the ***uno*** commands which are frequently used in the written XML file on *Acceleration Key Customize-Dialog − Save*. Such information are part of the sources of LibreOffice, for example in `libreoffice/officecfg/registry/data/org/openoffice/Office/UI/*.xcu`. Using this needs transfer of this information in a Java accessible form, means create a source file which fills a `TreeMap` with this information ready to use. This is a high effort non acceptable opportunity.

Instead, files given in the LibreOffice installation are used:

```
LOffcProgram/share/registry:main.xcd; writer.xcd; draw.xcd; calc.xcd
```

This files contain associations between the uno calls which are referenced in the `KeySettings.cfg` file, and the possible human readable commands which is also usual used in the *“Customize”* dialog box for the Accelerator Key Settings. The additional read XML files are not intrinsically for translation, they contain certain control information, but just with the uno call and the appearance in the user interface dialogues, and are hence usable for this approach.

A general file (XML) which is a **user language specific human readable translation from uno c**ommands **is missing (not found)**.

The `registry/*xml` files are completely read, but only the uno commands and the “label” information are dissolved, stored in a key map (index) to associate a given uno call with the human readable meaning. Last one is outputted in the text file.

3.3 Translate keys from internal to human readable

The name of the keys in the XML file is such as `KEY_C` or `KEY_DIVIDE`. The `KEY_DIVIDE` is the `/` key which is intrinsic pressed with the shift key. The more complicated key designations are necessary in the XML file, because for example `KEY_/` is not an identifier. But for the list of acceleration keys it is desired, that the keys are always human readable and have always a size of 4 characters to offer a regular column structure in a mono spaces presented text.

The translation of the key is done by a simple key map index with the content: >>`WriteKeyAcclPrc#initKeyTranslate()`.

3.4 Output the text

The output is prepared using the concept of >>`OutTextPreparer` or also on https://vishia.org/Java/html/RWTrans/RWTrans.html#_outtextpreparer. A fallback scripts is internally defined in Java. But with the option `-gTxt:path/to/gTxtScript.gTxt` the output can be formed anyhow. Only the access to the correct internal data should be regarded, elsewhere error messages in the output script are created.

The offered script is the following, only comments are here shortened:

```
<:gTxt:KeyAcclText: thiz, NEWLINE:----:>
<:type:thiz:org.vishia.libOffc.cfgui.WriteKeyAcclPrc>    ## That helps with Javadoc to see what
is the meaning of the data
== Accelerator Key settings LibreOffice Translated via KeyAcclText-2025-12-23 ==
<:if:thiz.reg.idxUnoLabel.size()==0>
  :----:PROBLEM uno translation not done because faulty
-LOffcPrg:<&thiz.cmdArgs.dirLOffcPrg.getAbsolutePath()><.if>
<:for:e:thiz.idxKeyAcclShift>
  :----:<&e.keyString1><:tab:18>"<&e.sLabel_en.replace('~', '')>"<:tab:50>==><&e.sHref>
<.for>"
<:n>
<.gTxt>
```

The essential lines are the `<:for:....<.for>` lines:

Using the `idxKeyAcclShift` selects the index of key settings which are sorted with following the Shift key variant. The other index is `idxKeyAccl`, both are available and usable.

`e` is the entry of the index. With this one line is produces with access to the given fields of the index entry. With the `replace('~', '')` a standard Java operation `String#replace(...)` is called. This removes a `~` in the label text, which is contained in the read XML files for translation.

The meaning and the accessible fields in the given class `org.vishia.libOffc.cfgui.WriteKeyAcclPrc` and its referenced classes can be seen in the Javadoc, >>`WriteKeyAcclPrc`

See >>`WriteKeyAcclPrc#outKeys(...)`.

3.5 jar files

The jar file contains by the way also the translation between LibreOffice and the so named ZmL textual markup format for writer documents, described on https://vishia.org/LibreOffc/html/Videos_LOffcZmL.html. But that is another more sophisticated approach not described here. That translator is also not too complex, works also with the XML read and write utilities and hence is delivered with the same jar file.

The both used jar files are:

- `vishiaBase-20yy-mm-dd.jar`: General jar file with common usable capabilities, with currently 1.3 Mbyte not too large.
- `vishiaLOffcTools-20yy-mm-dd.jar`: contains all surround LibreOffice, not only this translation, but has only currently ~0.3 MByte.

3.6 Software archive

The software as java sources is also versioned as git in

* https://gitlab.com/jzhartmut/srcJava_vishiaBase
* https://gitlab.com/jzhartmut/srcJava_vishiaLibreOffc