TrIO: Transparent file I/O

Version of 2022-04-30

Udo Wermuth

The macros are written for the **original TEX** running with the format **plain.tex**. It assumes that a path to a file uses directory names that are separated by slashes. Except letters, digits, and the slash only a period is used to separate the main part of the file name and its extension.

The operation is done on a **command line** and makes use of a **bash script** and the stream editor **sed**. The installation scripts assume a **UNIX-like operating system**.

The application of the macros requires **concentration**; don't execute TEX with an instrumented source file hastily. Moreover, to use the macros successfully you must have a **non-beginner's knowledge** of TEX. Strictly speaking, **the more you know about plain TEX the better**; see section 10.

Note: My article in *TUGboat* 43:1 (2022), 59–72, contains a lot of additional information, for example, about my motivation to develop these macros. But it omits some parts like the start message. I suggest to read the article and this manual if you want to apply the macros.

a) Meaning of "transparent"

The word "transparent" means that the instrumented source cannot do any `\input`, `\openin`, or `\openout` without an approval of the user. Well, `\input` is an exception as it might be possible to input a file without approval—*might* because it depends on your operating system and your TEX program—but that a file is input is always shown on the terminal and the user must stop the processing. An evil-doer might try to circumvent the reporting of the macros. In such a case the user must check the source carefully for malicious code before it should be compiled again by TEX.

b) Rules

As indicated in the previous paragraph the application of the macros requires that the user follows a couple of rules. Otherwise the macros of this package might not do what is intended.

1. RULE: Create in the directory with the source that you want to test a new subdirectory, say, `trioo`, and redirect all `\openout` commands to this subdirectory.

2. RULE: Check that the bundle of the source files doesn't include a file that carries a name that's identical to a file name of this package. Otherwise rename such a source file.

3. RULE: Never approve that a file of this package is processed by the source file. Enter the new name of a source file if you renamed one of them because of rule 2. Otherwise stop the run and check the source.

4. RULE: Follow all terminal messages carefully; don't allow that the source inputs a file without your approval. Otherwise stop the run and check the source. (Maybe you need to apply an option; see section 4.)

1. Files in the package

The package contains the following files:

1. `TrIOmacros.org` – main file to instrument a source file
2. `TrIOsupport.org` – common macros to `TrIOmacros.tex` and `TrIOauto.tex`
3. `TrIOprivate.org` – copies of control sequences and test routines
4. `TrIOinput.org` – called for every `\input`
5. `TrIOopenin.org` – entered by the user for every `\openin`
6. `TrIOopenout.org` – entered by the user for every `\openout`
7. `TrIOextract.org` – `sed` script to create `TrIOnames.tex`
8. `TrIOlineno.org` – another `sed` script to create `TrIOnames.tex`
9. `TrIOauto.org` – replacement for `TrIOmacros.tex` in repeated executions
10. `TrIOopen.org` – called for every I/O command in `TrIOnames.tex`
11. `TrIO.org` – `bash` script to install the package
12. `TrIOinstall.org` – `sed` script to set passwords and private messages

The last two files are only used for the installation. They must be renamed and edited, see section 2. The other files get a new extension during the installation.

The first six files are required to instrument a source; see section 3. All are renamed during the installation so that all have the extension `tex` instead of `org`; in the names of files 5 and 6 the "TrIO" prefix is deleted too. Files 7–10 are needed for repeated executions; see section 8. The first two become `sed` scripts the other two `tex` files.

2. Installation

`TrIOinstall.org` and `TrIO.org` are used for the installation.

The other files are changed during the installation. They must be either moved into the directory that contains the source to be tested or they are installed in a directory that is searched for input files by the program TEX.

Step 1: Rename and edit `TrIOinstall.org`

`TrIOinstall.org` must be copied to `TrIOinstall.sed`. This file must then be edited to have **individual password-protected macro names** and **private messages**. Note there is no valid default active. The macros fail to do what they should do if this file isn't carefully edited.

a) Passwords

A password-protected name contains a string of at least six letters. It must be a mix of lower- and uppercase, one letter should be from the first and one from the last third of the alphabet. It should be neither an English word nor one in your language. The general idea is that this string cannot be guessed or computed by someone else. A user hasn't to enter these password-protected macro names (except one) so a user has not to remember them. Thus, they must not be simple.

One of the passwords occur in a command that is displayed together with a private message. Although not required it would be nice to recognize this password. A second password is part of a control word that must be entered by the user into the source if the source contains extremely weird constructions; see section 5.

In total eight passwords must be created. `TrIOinstall.sed` contains lines like

```
s/aAmNzZ/new password/g
```

and you have to replace "`new password`" with the new password, i.e., a string of at least six letters with the above mentioned distribution. And please, never use the known default, i.e., the string `aAmNzZ`, etc.

b) Private messages

Next, you have to choose two private messages. The first one explains what to do for `\openin` or `\openout`, the second signals that the correct file for `\input` was executed.

The rest of `TrIOinstall.sed` can be ignored for the moment. These lines must be used if the source uses `TrIO` or `TRIO` as a prefix for its own macros; see c). Then one can change these prefixes in the macros of this package.

c) Optional: Prefix "`T[r|R]IO`"

The last lines in the installation script are used to change the prefix "`TrIO`" and the prefix "`TRIO`" in the case that some source uses one of them too. The macros detect if one of their control sequences gets redefined by the source and reports that with an error message: `! TrIO ALERT !!! Don't trust the source (<name of the redefined control word>).`

5. RULE: Stop the run if the "`TrIO ALERT`" error message is displayed. The source has redefined one of the control words that occur in the macros of this package. Check the source carefully to determine if that happens by accident or intentionally.

Note, the easiest way to fix a single use of the prefix "`TrIO`" is to change the source. Otherwise a user can apply the installation script and change the prefix for all macros of the package. The prefix isn't private, i.e., the source might contain code that determines which prefix is used by the macro package.

Step 2: Rename and edit `TrIO.org`; then execute

Rename `TrIO.org` to `TrIO.sh`. Make sure that the renamed file can be executed, for example, apply `chmod 744`. The file must be edited too: Enter a valid path for `inst_dir`.

The execution of the script `TrIO.sh`, i.e., enter `./TrIO.sh` on a command line in the directory of the `TrIO*.org` files, installs the files by default in this directory. This might be okay if the package should be placed there. But if you use a TEX distribution the installation directory should probably be changed; consult the documentation of your distribution to figure out where configured packages should be stored.

My suggestion: Use the current directory for experiments. If you like the macros then install them in your TEX file system and make sure that the correct `openin.tex` and `openout.tex` are found as these files carry quite generic names. (Maybe you must **rename** them in `TrIO.sh`; see also section 9.)

3. Using `TrIOmacros.tex`

To activate the macros create a new first line in the main file of the source that should be instrumented. Enter "`<options> \input TrIOmacros`"; where `<options>` is a combination of

i) empty;
ii) `\let\twonosubdirs=y`;
iii) `\let\threenosubdirs=y`;
iv) `\let\disablespecial=n`;
v) `\let\disablespecial=y`.

Note: here iii) beats ii) and v) beats iv). The options are explained in the next section.

a) An example

Let's do a trial run with `Transparent-IO-example.tex`; here with line numbers.

```
1.\input TrIOmacros
2.
3.\input Transparent-IO-hello
4.
5.\openin0=Transparent-IO-hello
6.
7.\openout5=Transparent-IO-goodbye
8.
9.\bye
```

After TEX's banner line and the name of the main file, TEX inputs three files (I omit the license message):

```
(./TrIOmacros.tex (./TrIOsupport.tex) (./TrIOprivate.tex)
<<< TrIO >>> \special: primitive
<<< TrIO >>> \TrIOnosubdir: nosubdir/
```

You should check that the correct files of section 1, items 1–3, are input. Here I assume that the files are in the current directory indicated by "`./`". Then TEX displays two lines about the current values of control sequences that are influenced by options. Here both show the default value: `\special` is kept as TEX's primitive and `\TrIOnosubdir` is a single directory with the name `nosubdir/`.

Next, the macros inform the user that they are active.

```
! TrIO activated. Type h to get instructions.
l.115 ...IO activated. Type h to get instructions}
                                                  \endgroup
? h
File I/O ( \input \openin \openout ) is now under TrIO's control.
Be alerted if a file is loaded without the proper TrIO message
and follow always the instruction of this message.
Make sure that the sequence number is incremented sequentially.
?
)
```

The help text gives two important hints. Follow the instructions that are displayed at every stop initiated by these macros. As above these messages contain a combination of the word "TrIO" and three '<' or three '>' symbols. The second hint defines a rule.

6. RULE: Check the counter for I/O commands. It must increase sequentially; otherwise stop the run and check the source.

The counter appears as soon as TEX stops for an I/O command. For `\input` is looks like this.

```
<<<
(./TrIOinput.tex
>>> TRIO: Check passed!
)
TrIO >>> ( 1 ) Line 3: input
>>> enter shown file name without 'nosubdir/'.
<<<
! I can't find file 'nosubdir/Transparent-IO-hello.tex'.
l.3 \input Transparent-IO-hello

Please type another input file name:
```

The first four lines appear every time a file gets input. It contains one of the private messages; here it's the default message "`>>> TRIO: Check successful!`" with an uppercase 'R' in the word "TRIO". During the installation this message was changed and you should always check that your private message is displayed.

7. RULE: Check that for every `\input` the file `TrIOinput.tex` is input and that your private message follows. Otherwise stop the run and check the source.

Then the macros inform that an I/O command was found. It is the first command in the file: It occurs in line 3, and it's the command `\input`. The stop of the run was initiated by an error message that was intentionally raised through the use of an nonexistent subdirectory in front of the file name. The source doesn't contain this subdirectory; it is added by the macros. Thus TEX reports that it cannot find "`nosubdir/Transparent-IO-hello.tex`" although the source states "`\input Transparent-IO-hello`".

Next, the user has to enter a new file name. As the file `Transparent-IO-hello.tex` doesn't belong to the macro package (see rule 4) we follow the instruction (see rule 4) and enter "Transparent-IO-hello" as new file name.

```
Please type another input file name: Transparent-IO-hello
(./Transparent-IO-hello.tex)
```

The next block shows a stop for the I/O command `\openin` with stream number 0.

```
<<<
TrIO >>> ( 2 ) Line 5: openin 0
>>> If you accept that the file (without nosubdir/) is read
>>> enter 'openin' and follow the instructions.
<<<
! I can't find file 'nosubdir/Transparent-IO-hello.tex'.
l.5 \openin0=Transparent-IO-hello

Please type another input file name:
```

After the check that the counter was increased we follow the instruction and enter the file name of the package (`openin.tex`) and not the name that appears in the source (here: `Transparent-IO-hello`).

```
Please type another input file name: openin
(./openin.tex
Enter 1> return 2> file name}\TRIOgGKptTpausing=0=>
```

This block shows the second private message and one of our private passwords. Again, we follow the instructions, that is, we press `<return>` and at the next prompt we enter the name of the file that should be opened:

```
\FilenameOPENIN=Transparent-IO-hello
)
```

The treatment of `\openout` is similar to `\openin`.

```
<<<
TrIO >>> ( 3 ) Line 7: openout 5
>>> If you accept that the file (without nosubdir/) is created
>>> enter 'openout' and follow the instructions.
<<<
! I can't find file 'nosubdir/Transparent-IO-goodbye.tex'.
l.7 \openout5=Transparent-IO-goodbye

Please type another input file name: openout
(./openout.tex
Enter 1> return 2> file name}\TRIOgGKptTpausing=0=>

\FilenameOPENOUT=trioo/Transparent-IO-goodbye
)
```

There is one new technique based on rule 1. The file isn't created in the current directory but in a special subdirectory called `trioo/` as seen in the second last line. Of course, one must remember which files were redirected to this subdirectory as the source might want to input it later again and then the name of the subdirectory must be used too.

Of course, `\immediate` is handled by the macros too; the user sees "immediate openout" instead of "openout" in the `TrIO >>>` line. But someone who knows the macros can fake the occurrence of `\immediate`. Although there isn't much harm in a wrong `\immediate` it is recommended to follow this rule.

8. RULE: Always check that an `\immediate` appears in front of an `\openout` if the macros report "immediate openout". Check the source carefully and reject it if `\immediate` doesn't appear.

4. Options

In section 3 the four options for the macros were introduced.

a) Path with nonexistent directories

As explained in section 3 a single nonexistent directory "`nosubdir/`" is used in the macros by default. But if the source contains a file given with a path that starts with a move to the parent directory, i.e., `../`, then the effect of `nosubdir/../` might specify the current directory; this depends on several factors, i.e., your TEX might not do it. If it's done a file in the current directory carrying the same name as the file that the source wants to input from the parent directory is directly input without a stop of the run. That violates rule 4.

Therefore you should use either the option `\let\twonosubdirs=y` or `\let\threenosubdirs=y`. In the first case the single nonexistent directory is replaced by two nonexistent directories and in the second there are three nonexistent directories. The start message of the package shows how many `nosubdir/`'s are present.

9. RULE: Stop the processing if the source that failed with a single nonexistent directory fails again at the same I/O command although the number of nonexistent subdirectories was increased. The source reacts on this number.

b) Treatment of `\special`

The second start message is about `\special`. This command can be dangerous as it writes arbitrary data to TEX's main output file. The macros don't check what is written to the DVI file and they assume that a user switches on the safety features of the program that display, prints, or transforms the DVI file. Therefore `\special` is kept as primitive.

10. RULE: The programs that interpret the DVI file can be the target by with an untrusted source attacks your system. Thus switch on all options to avoid that the DVI file executes unfriendly code if you keep the primitive `\special` active.

The option `\let\disablespecial=y` redirects all data of a `\special` into the LOG file of the TEX run. That is, `\special` is no longer a primitive; nothing is written to the DVI file.

The option `\let\disablespecial=n` tries to be a mixture of the described behaviours. The primitive `\special` is kept but its data (or a part of it) is also written to the LOG file; the integer parameters `\showboxdepth` and `\showboxbreadth` are increased. To look at the data enter on a command-line the command "`grep -e'^\.\.*\\special' <logfile>`".

5. Errors

Errors in the original source are of course kept in the instrumented source. But a few eccentric cases can occur that produce new errors because of the fact that the I/O primitives are now macros.

I don't mean a case where a source tests if the I/O commands are macros or primitives. The source must do something after this test and whatever it does I don't count this as an error.

Here is the file `Transparent-IO-eccentric.tex` that contains some eccentric cases:

```
\input TrIOmacros
\global\input Transparent-IO-hello
\expandafter\show\input Transparent-IO-hello
\edef\csone{\input Transparent-IO-hello }
\expandafter\show\openin0=Transparent-IO-hello
\bye
```

a) Command in front of `\input` meant for the first token of the file that is input

Some errors happen with `\input` in situations like this one where `\global` is applied to `\input`.

```
! You can't use a prefix with `\begingroup'.
<to be read again>
                   \TRIObegingroup
\input ->\TRIObegingroup
                         \TRIOafterassignment \TRIOnoexpand \TrIOempty \TRIO...
l.2 \global\input
                  Transparent-IO-hello
?
```

This is the typical picture for an error where TEX wants to read again the first token of the macro `\input`. It is shown not only for `\global` but also for `\number` and many other commands.

The solution is to type `42`—you have to remember this number—and then to insert the command that appears in front of `\input`; here it's `\global`.

```
? 42

\input ...put.tex \TrIOinputmessage \TrIOendgroup
                                                \TRIOinput \TrIOnosubdir
l.2 \global\input
                  Transparent-IO-hello
? I\global
! I can't find file `nosubdir/Transparent-IO-hello.tex'.
```

```
l.2 \global\input Transparent-IO-hello

Please type another input file name:
```

Note this is a very strange use of `\input` as the acceptance of the command in front of `\input` depends on the first token in the file that's input.

b) `\input` as second token after `\expandafter`

Another kind of error occurs if the commands in front of `\input` digests tokens. Here is an example.

```
> \TRIObegingroup=\begingroup.
\input ->\TRIObegingroup
                          \TRIOafterassignment \TRIOnoexpand \TrIOempty \TRIO...
l.3 \expandafter\show\input
                             Transparent-IO-hello
?
```

The solution is to type `41` (as $42 - 1 = 41$) and then to insert the two commands in front of `\input`.

```
? 41

\input ...put.tex \TrIOinputmessage \TrIOendgroup
                                                    \TRIOinput \TrIOnosubdir
l.3 \expandafter\show\input
                             Transparent-IO-hello
? I\expandafter\show
! I can't find file `nosubdir/Transparent-IO-hello.tex'.
l.3 \expandafter\show\input Transparent-IO-hello

Please type another input file name:
```

c) Use of `\input` in an `\edef`

Expansion in an `\edef` or `\xdef` generates a new kind of error. This time the error message is the well-known `! Undefined control sequence`. This message is raised intentionally by a token that explains what to do next.

```
! Undefined control sequence.
\input ...y \TRIOdef \TrIOskipXXXVIinsTrIOfixedef
                                                    \TrIOempty {}\TrIOempty \T...
l.4 \edef\csone{\input
                        Transparent-IO-hello }
?
```

The undefined control word states to skip `36` tokens and then to insert `\TrIOfixedef`. You could also enter first `\show\TrIOfixedef` and check that the macro wasn't changed.

```
? 36

\input ...put.tex \TrIOinputmessage \TrIOendgroup
                                                    \TRIOinput \TrIOnosubdir
l.4 \edef\csone{\input
                        Transparent-IO-hello }
? I\TrIOfixedef
! I can't find file `nosubdir/Transparent-IO-hello.tex'.
l.4 \edef\csone{\input Transparent-IO-hello
                                 }
Please type another input file name:
```

Note the source throws more errors as `\input` cannot be used in an `\edef` except the file that's input ends with a certain command. Again a unusual scenario.

d) Errors with `\openin` and `\openout`

For `\openin` and `\openout` there is only one error situation. It occurs with expansion if a sequence of tokens like the nonsense `\expandafter\show\openin0=Transparent-IO-hello` appears in the source. In this case the expansion of `\openin`, which contains two tokens, looses its first token (that's a `\the`). One has to insert this token to continue the processing.

```
> \TRIOthe=\the.
\openin ->\TRIOthe
                    \TrIOtropenin
1.5 \expandafter\show\openin
                               0=Transparent-IO-hello
?
```

Again, a very strange code pattern. Why would someone want to expand an usually unexpandable token before `\show` is applied?

Summary

Most of the described cases don't occur in normal code. Some cases represent a TEX coding error, i.e., they throw one or more errors during normal execution. These cases might be inserted by someone who wants to fool you—maybe after the detection that `TrIOmacros` is used. Thus, I recommend to either inspect the code carefully in order to find out why the unusual construction is used or to ignore the source.

11. RULE: Inspect the source carefully if an error occurs that involves one of the I/O commands. These errors can only appear with weird code without any valid use case. Consider to ignore the source.

6. Permanent fixes for the errors

Note that the interactive fixes of the errors in the previous section have suppressed several messages, for example, the counter was not shown and it was not increased. This generates a problem for `TrIOauto.tex` as we will see later.

An `\expandafter` with `\openin` or `\openout` can be deleted in the source. Some constructions for `\input` require a different approach; to fix a `\global\input` permanently the source must be edited too.

The macros contain the password-protected control sequence `\TrIOcCkPxXmove` that has to be entered in front of the commands that would occur after the `?I` in the interactive fix. For example, the above case is permanently fixed with the code `\TrIOcCkPxXmove\global\input`. Of course, you must use the control word that contains the password that you picked during the installation; as mentioned above this is the only password one must remember.

12. RULE: Never return an edited source file to the author without removing the first line and all appearances of `\TrIOcCkPxXmove`.

13. RULE: Never return a LOG file of your runs to the author of the source. If you want to return it make sure that it contains no information about your password-protected macro names, your private messages, or any trace information of the TrIO macros.

14. RULE: Never return the DVI file of your runs to the author of the source. If you want to return it check the DVI output and make sure, for example, by using the program `dvitype` of the TEXware collection, that it contains no information about your password-protected macro names or your private messages.

15. RULE: Create new passwords and private messages if you violate one of the rules 12–14 by accident.

7. An unsolvable problem

There is one TEX primitive that can create errors which cannot be detected by the macros and, thus, which cannot be fixed. But the macros can detect if the problematic primitive is active. If the macros report this the user has to check the source carefully in order to confirm that the activation is required and that it happens in a way which doesn't influence the work of the macros.

The problematic primitive is the integer parameter `\globaldefs`. Usually its value is 0 and the macros can do their work without problems. If its value is less than 0 everything is fine too. But a value greater than 0 makes any assignment global even if it is not prefixed with `\global` and this can lead to a problem.

Our macros first reset the value to 0; this assignment is global. At the end the macros set the value locally back to 1 or −1 depending of the value of `\globaldefs` when a macro is called. Thus code like this doesn't work correctly: `\globaldefs=1 {\input Transparent-IO-hello }`. In this case the value of `\globaldefs` after the closing brace is 0 not 1. But this case cannot be distinguished by the macros from the good case `{\globaldefs=1 \input Transparent-IO-hello }`. The macros can only detect if `\globaldefs` has a value greater than 0 and they report this finding.

16. RULE: Stop the run if the macros report that `\globaldefs` is positive. Check the source carefully why this rarely used parameter must be set. If it must have a value greater 0 check that only the good case occurs. Otherwise reject the source.

8. Using `TrIOauto`

If a run with `TrIOmacros.tex` was successful, i.e., no interactive fix was necessary, repeated executions of the source can be made simpler.

First, the LOG file of the successful run should be saved; copy it, for example, and give it the extension `trio`. Second, one of the two `sed` files must be applied to this copy. For example, `sed -f TrIOlineno.sed <triofile> > TrIOnames.tex` where `<triofile>` is the copy of the LOG file of the successful run. The last step is to change `TrIOmacros` in the first line of the instrumented source by `TrIOauto`.

The file `TrIOlineno.sed` makes stricter comparisons than `TrIOextract.sed`. You can always recreate `TrIOnames.tex` if the source file needs to be edited and a comparison of line numbers would fail by using `TrIOextract.sed` instead of `TrIOlineno.sed` in the above `sed` command.

a) Rerun of the example of section 3

A run with `TrIOauto.tex` starts similar to a run with `TrIOmacros.tex` except that one more file is loaded but no information about a nonexistent subdirectory is given.

```
(./TrIOauto.tex (./TrIOsupport.tex) (./TrIOprivate.tex) (./TrIOnames.tex)
<<< TrIO >>> \special: primitive
! TrIO auto-mode activated. Type h to get instructions.
l.61 ...ode activated. Type h to get instructions}
                                                  \endgroup
? h
File I/O ( \input \openin \openout ) is now under TrIO's control.
Be alerted if a file is loaded without the proper TrIO message
and check that the file name without extension that raises
the font error matches the file name of the TrIO message.
?
```

The nonexistent subdirectory isn't required anymore as all I/O commands are taken from the file `TrIOnames.tex`. In this run a user cannot change a file name used in the source; the macros use the ones entered during the successful run. A surprise might be the text about "font errors" in the help text. Although no file name has to be entered the macros show the name of the successful run and the name that they read in the current run.

```
(./TrIOopen.tex
TrIO >>> ( 1 ) Line 3: input Transparent-IO-hello
)
! Font \TRIOunused=TrIO_Transparent-IO-hello not loadable: Metric (TFM) file not found.
<to be read again>
                   \par
l.4

?
```

Note that TEX has the complete file name only after it sees the next line which is empty.

Although only approved file names are processed a user should still be concentrated.

17. RULE: With `TrIOauto.tex` a user should check that `TrIOopen.tex` is loaded, the sequential counter is increased from stop to stop, and that the main part of the file name in the line that starts with "`TrIO >>>`" agrees with the name that appears after the underline in the error message (with `\openout` the path is replaced by `trioo`, see rule 1). Otherwise the source operates with varying file names; check the source carefully.

Now press return to input `Transparent-IO-hello.tex` and to go to the next I/O command. The messages for `\openin` and `\openout` look similar.

```
(./TrIOopen.tex
TrIO >>> ( 2 ) Line 5: openin 0 Transparent-IO-hello
)
! Font \TRIOunused=TrIO_Transparent-IO-hello not loadable: Metric (TFM) file not found.
<to be read again>
                   \par
l.6

?
(./TrIOopen.tex
TrIO >>> ( 3 ) Line 7: openout 5 trioo/Transparent-IO-goodbye
)
! Font \TRIOunused=TrIO_Transparent-IO-goodbye not loadable: Metric (TFM) file not found.
<to be read again>
                   \par
l.8

?
```

You can see that the subdirectory `trioo` (see rule 1) is mentioned in the case of `\openout`. Of course, an `\immediate\openout` is signaled by "immediate openout" instead of "openout" in the line that starts with `TrIO >>>`.

9. Remarks about `openin.tex` and `openout.tex`

The TEX commands `\openin` and `\openout` are less verbose than `\input` with respect to the file name they have to process. As they occur less frequently than `\input` the macros implement a two-step procedure to enter the file names for `\openin` and `\openout`. This slow-down in the workflow gives the user a chance to avoid hasty decisions.

But I assume some users want to have a faster processing. Here are two ideas how to speed-up the workflow. (1) The file `openin.tex` and `openout.tex` aren't called by other files of the package; they are only entered by the user. Thus, a user can rename them to make the names easier to type. (The names are used in messages (see `TrIOmacros.org`) so either change them there too or remember that you have changed the file names.) (2) The files stop with a private message whereas `TrIOinput.tex` just outputs such a message. If you check all private messages carefully you can change the code in `openin.tex` and `openout.tex` so that it displays the private message on the terminal without using `\pausing` to avoid to enter `<return>`.

10. Precautionary warning

As you saw the macros cannot run without user interaction. This means, your knowledge about plain TEX is a crucial success factor if problems occur. You must be able to distinguish if an error message is based on TrIO's actions, or if it's an aggressive attempt to circumvent TrIO, or if it's a simple error in the source.

Above I wrote that you should have a non-beginner's knowledge of plain TEX. Now you should be aware that your knowledge about the original TEX, plain TEX, the command `\special` and its risks, as well as tools like `dvitype`, `sed`, etc. are important.

As soon as you don't understand what happens in the untrusted source stop the execution by TEX. Stop the execution in an error situation if you don't understand what you have to do with TrIO.