The checklistings package*

Timothy Bourke and Marc Pouzet

July 8, 2017

1 Introduction

User manuals and papers about programming languages usually contain many code samples, often with accompanying compiler messages giving the types of declarations or error messages explaining why certain declarations are invalid. This package extends `fancyvrb`[1] and `listings`[2], which are ideal for displaying code, with a way to pass code samples through a compiler and include the resulting messages in a LaTeX document. The motivation is to check the code samples in a document for syntax and typing errors and to facilitate the inclusion of inferred types and compiler warnings or errors in a text. This package is intentionally very lightweight and unlike packages like `python`,[3] it is not intended for interacting with an interpretor or including the execution traces of code. While it does not focus on a specific programming language, it is designed to work well with ML-like languages.

As an example, the text at left below is generated by the LaTeX code at right:

Code samples are included verbatim and the results of compilation can be included automatically:

```
let inc x = x + 1
let y = inc 3
```

```
val inc : int -> int
val y : int
```

```
Code samples are included
verbatim and the results of
compilation can be included
automatically:
\begin{chklisting}[withresult]
let inc x = x + 1
let y = inc 3
\end{chklisting}
```

A first pass through `latex` generates both a `.chkl` file, with parameters for the compiler, and an `.ml` file containing the source code (i.e., the two lines in the example above). Running the `checklistings.sh` script processes these files to produce a `.tex` file with the results of compilation. A second pass through `latex` updates the compiler message.

*This document corresponds to checklistings v1.0, dated 2015/09/01.

[1] http://www.ctan.org/pkg/fancyvrb

[2] http://www.ctan.org/pkg/listings

[3] http://www.ctan.org/pkg/python

It is possible to continue examples and to label them (to be continued at some later point):

```
These definitions follow on from the previous ones:
\begin{chklisting}[continue,withresult,label=early]
let z = y + inc y
\end{chklisting}
```

These definitions follow on from the previous ones:

```
let z = y + inc y
```

```
val z : int
```

Examples need not necessarily succeed:

```
This code does not compile:
\begin{chklisting}[continue,fail,withresult,skipone]
let u = 3
let w = u + "four"
\end{chklisting}
```

This code does not compile:

```
let u = 3
let w = u + "four"
```

```
File "chklisting.ml", line 1, characters 12-18:
Error: This expression has type string but an expression was expected of type
int
```

Note that the line number in the error message is 1 even though the `continue` option added a line to include the previous definitions and the `skipone` option hid the `let u = 3` line. The `checklistings.sh` script performs this adjustment by looking for the regular expression '`line  *[0-9][0-9]*`' and decrementing the number appropriately. This approach is simple and effective but obviously not foolproof. Some manual tuning of the script may be required for correct results.

2 Use

Using the package involves three elements:

1. The declaration `\usepackage{checklistings}`.
   Section 2.1 describes the options for configuring package behaviour.

2. The environment `chklisting`.
   This environment is used like any other verbatim environment. Section 2.2 describes options that may be given to control its behaviour.

3. The script `checklistings.sh`.
   Running this script passes the contents of each `chklisting` environment through a compiler or interpreter and copies the resulting output into a file.

Additionally, the file `checklistings.hva` can be used to incorporate the output of `checklistings` in HTML documents generated by HeVeA.[4]

2.1 Package options

`\checklistings`

Package options are either given as optional arguments to `\usepackage` or via one or more calls to `\checklistings`. The advantage of the latter is that macros are not expanded (for a detailed explanation see the documentation for `kvoptions`,[5] Section 4.1, *Package kvoptions-patch*). Options are passed as a comma separated list of ⟨*key*⟩=⟨*value*⟩ pairs and single ⟨*key*⟩s.

There are three classes of options: options controlling the default behaviour of `chklisting`, options for configuring the `checklistings.sh` script, and options controlling the display of code and results.

2.1.1 Behavioural options

These options control the default behaviour of the `chklisting` environment.

`option`	description	`default`
`withresult`	Automatically show compilation results.	`false`
`skipone`	Do not display the first line of code (see the description under the `chklisting` environment).	`false`
`skiptwo`	Do not display the first two lines of code (see the description under the `chklisting` environment).	`false`
`hideproblems`	Globally disable the display of failure messages (within a red box) when compilation fails or succeeds unexpectedly.	`false`

2.1.2 Configuring compilation

These options are used for naming and placing the files generated by `chklisting` environments. They are passed to the `checklistings.sh` script and thus control its behaviour.

`option`	description	`default`
`prefix`	Prefix for naming source files. It must not contain underscores (_).	`chklisting`
`ext`	Extension of source files.	`.ml`

[4] http://hevea.inria.fr
[5] http://www.ctan.org/pkg/kvoptions

`subdir`	If defined, source files are created in the given subdirectory, which must already exist. A final slash (`/`) should not be given. The name must not contain underscores (`_`).	`.`
`prompt`	The prompt displayed by `\chklistingcmd`.	`#`
`compiler`	Path of the compiler to execute.	`ocamlc`
`compilerflags`	Flags passed to the compiler. These are not revealed by `\chklistingcmd`.	
`lastflags`	Flags passed to the compiler before the main source file, that is, the last one given.	`-i`
`includecmd`	The source language command for importing the definitions of another file.	`open`
`html`	The path of a program that renders source code as html. This value can be useful when using HeVeA to produce web pages from LaTeX documents. In any case, it is optional.	

Each `chklisting` environment is assigned a number n, from zero, and its contents are written to the file: ⟨*subdir*⟩`/`⟨*prefix*⟩⟨n⟩`.`⟨*ext*⟩, where ⟨n⟩ is zero-padded to four characters. For example, by default, the fourth environment is written to the file `chklisting0003.ml` in the current directory.

Source lines are added for each dependency, and those files are compiled using the ⟨*compiler*⟩, ⟨*compilerflags*⟩, and ⟨*lastflags*⟩ options. For example, if the fourth environment depends on the first and the second, a line is added:

⟨*includecmd*⟩ `Withopen0000` ⟨*includecmd*⟩ `Withopen0001`,

where `Withopen` is the prefix used for such augmented files, and the compiler is invoked with:

⟨*compiler*⟩ ⟨*compilerflags*⟩ `Withopen0000 Withopen0001` ⟨*lastflags*⟩ `Withopen0003`

2.1.3 Controlling the display

This package exploits the display options given by the `fancyvrb` package.

`option`	description	`default`
`codestyle`	`fancyvrb` options for code	
`msgstyle`	`fancyvrb` options for compiler messages	`formatcom=\em`
`errstyle`	`fancyvrb` options for error messages	`formatcom=\em`
`codelst`	`listings` options for source code, passed with `\lstset`. When this option is not empty, `fancyvrb=true` is included automatically.	
`msglst`	As for `codelst`, but applied to compiler messages.	
`errlst`	As for `codelst`, but applied to error messages.	

Other options are passed directly through to the `fancyvrb` package and applied to all `chklisting` code blocks (but not to messages or errors). For example, `frame=single`. These options must typically be set using `\checklistings`, since they will usually contain commands that should not be expanded immediately (like `\em` or `\bf`).

2.2 The `chklisting` environment

`chklisting` As an optional argument, this environment takes a comma separated list of ⟨*key*⟩=⟨*value*⟩ and single ⟨*key*⟩s.

`option`	description
`fail`	This code is expected to fail; an error is reported if it succeeds.
`continue`	This code is continued from the previous `chklisting` environment; all of the definitions available there are imported.
`label`	Label this code for later inclusion.
`include`	All of the definitions available after the environment with the given label are imported.
`withresult`	The result of compiling the code is displayed (see also `\chklistingmsg` and `\chklistingerr`). This is normally either the types of declared values or the results of evaluation. For environments marked `fail`, it will be an error message.
`withoutresult`	The result of compiling code is not displayed automatically. This is the default behaviour, but it can be overridden by the package options.
`hide`	The code is not displayed, but it is still compiled, compiler messages are still displayed (if `withresult` is active), and definitions are still available for continuation (`continue`), labelling (`label`), and later inclusion (`include`).
`skipone`	Do not display the first line of the code. This line is still sent to the compiler and may thus be used to open other modules, or to pass execution options (via comments).
`skiptwo`	As pre the previous option, but two lines are skipped.
`skipnone`	Do not skip any lines; this option overrides any package-level skip setting.
`hideproblems`	Do not display an error message within the document when compilation fails or succeeds unexpectedly.
`showproblems`	Display an error message within the document when compilation fails or succeeds unexpectedly. This is the default behaviour unless the `hideproblems` package option was set.
`compilerflags`	Flags passed to the compiler for a specific example. These flags are given after those of the package-level `compilerflags` setting and before those of the package-level `lastflags` setting. These *are* revealed by `\chklistingcmd`.

The results of compiling the code of a `chklisting` environment are made

available in the following macros until the next `chklisting` which will redefine them.

`\chklistingcmd` `\chklistingcmd` contains an idealised version of the command line used to compile the code sample. It includes the `prompt`, the basename of `compiler`, and `lastflags`, but not `compilerflags` or the list of included files. Furthermore, the `subdir` and serial number are removed from the filename of the code sample, which becomes simply ⟨*prefix*⟩.⟨*ext*⟩.

`\chklistingmsg` `\chklistingmsg{`⟨*label*⟩`}` inserts the verbatim text emitted by the compiler, provided compilation succeeded, for the `chklisting` environment labelled ⟨*label*⟩. When ⟨*label*⟩ is left empty, the message for the last environment is inserted. It should not be used after environments marked `fail`.

`\chklistingerr` `\chklistingerr{`⟨*label*⟩`}` inserts the verbatim text emitted by the compiler, provided compilation failed, for the `chklisting` environment labelled ⟨*label*⟩. When ⟨*label*⟩ is left empty, the message for the last environment is inserted. It should only be used after environments marked `fail`.

`\chklistingcurrfile` `\chklistingcurrfile` contains the path to the file created for the code sample.

The following macros are provided for working with the contents of `chklisting` environments.

`\chklistinginput` `\chklistinginput{`⟨*filename*⟩`}` : loads the specified file as verbatim text according to the `codestyle` and `codelst` display options. One use for this macro is to define a block of code in one place so that it can be included in other code blocks, but to actually present the definitions at a later point. For instance,

We define g as follows,

```
let g x = f x + f x

val g : int -> int
```

where f was defined as

```
let f x = x + 1

val f : int -> int
```

```
\begin{chklisting}[hide,label=f]
let f x = x + 1
\end{chklisting}

\edef\fcode{\chklistingcurrfile}

We define \verb|g| as follows,
\begin{chklisting}[withresult, continue]
let g x = f x + f x
\end{chklisting}
where \verb|f| was defined as
\chklistinginput{\fcode}
\chklistingmsg{f}
```

2.3 The `checklistings.sh` script

Processing a document that uses the `checklistings` package produces a `.chkl` file containing compiler options and a list of source files together with their interdependencies. The `checklistings.sh` script processes `.chkl` files by executing

the specified compiler against each listed source file ⟨*subdir*⟩`/`⟨*prefix*⟩⟨*n*⟩`.`⟨*ext*⟩ and copying the results—the command-line used, whether it succeeded or failed, the messages on `stdout`, and the messages on `stderr`—into a corresponding file, ⟨*subdir*⟩`/`⟨*prefix*⟩⟨*n*⟩`.tex`, for inclusion in the original document.

The `checklistings.sh` script is written for the Bourne shell (`sh`). It takes a list of `.chkl` files as arguments (with or without the exentions), but if none are given it processes all such files in the current working directory.

The compilation options specified within a LaTeX source file, see Section 2.1.2, can be manually overridden by processing a `.chkl` file, before any others, containing a '`lock`' directive, for example: `lock compiler=/usr/local/bin/ocamlc`. This feature is useful when working with others to develop the compiler being documented.

3 Remarks

3.1 Known limitations

The package and script have some known limitations.

- Line numbers in error messages may not correspond correctly with the line numbers of sample files, due to either the `skip*` options, or because of lines added to import code.
- The system has been designed to work with ML-style compilers. It has not been tested with other compilers and interpreters. Please contact `tim@tbrk.org` if you would like to support other systems. Patches are most welcome, but the intent is to keep this package relatively simple rather than to try to do everything.
- Multiple languages in a single document are not supported.
- Care must be taken when using `checklistings` with the overlays of the Beamer package. In particular, `chklisting` environment commands are to be avoided within commands that completely omit material from slides, like `only`, `alt`, or `temporal`, or with 'closed' overlay specifications, like `2` or `1-3`. As these commands do not execute the material, the environment sequence numbers do not increase monotonically, and the compilation results may not be properly synchronized with the verbatim text. Commands that only hide material, or that introduce it successively, like `uncover` or `visible`, or 'open' overlay specifications, like `2-`, should function as expected.

4 Implementation

All internal macros have names of the form `\CHKL@`⟨*name*⟩.

`\chklisting` Generate the sequence of source code identifiers used in per-environment filenames and to manage dependencies. If `beamer` is being used, reset the counter on each overlay to avoid generating multiple output files for the same program.

```
\newcounter{chklisting}
\ifdefined\resetcounteronoverlays
\resetcounteronoverlays{chklisting}
\fi
```

`\ifCHKL@fileexists` An internal boolean variable for remembering whether an input `.tex` file, corresponding to the compilation of source code, was found.

```
\newif\ifCHKL@fileexists
```

4.1 Package Options

The package options are processed using the `kvoptions` package.

`\CHKL@pkg@verbopts` This list accumulates package-level options for the `verbatim` environments.

```
\def\CHKL@pkg@verbopts{}
```

`\CHKL@pkg@globalskip` Defines the number of lines to be skipped when displaying a code sample as defined by the global options `skipone` and `skiptwo`.

```
\def\CHKL@pkg@globalskip{0}
```

Declare the package options and their default values:

```
\DeclareBoolOption{withresult}
\DeclareComplementaryOption{withoutresult}{withresult}
\DeclareBoolOption[true]{showproblems}
\DeclareComplementaryOption{hideproblems}{showproblems}
\DeclareVoidOption{skipone}
    {\edef\CHKL@pkg@globalskip{1}%
     \edef\CHKL@pkg@verbopts{\CHKL@pkg@verbopts,firstline=2}}
\DeclareVoidOption{skiptwo}
    {\edef\CHKL@pkg@globalskip{2}%
     \edef\CHKL@pkg@verbopts{\CHKL@pkg@verbopts,firstline=3}}
\DeclareDefaultOption
    {\edef\CHKL@pkg@verbopts{\CHKL@pkg@verbopts,\CurrentOption}}
\DeclareStringOption[]{codestyle}
\DeclareStringOption[formatcom=\em]{msgstyle}
\DeclareStringOption[formatcom=\em]{errstyle}
\DeclareStringOption{codelst}
\DeclareStringOption{msglst}
\DeclareStringOption{errlst}
\DeclareStringOption{emptyoption}
\DeclareStringOption[.]{subdir}
\DeclareStringOption[chklisting]{prefix}
\DeclareStringOption[.ml]{ext}
\DeclareStringOption[\#]{prompt}
\DeclareStringOption[ocamlc]{compiler}
```

```
\DeclareStringOption{compilerflags}
\DeclareStringOption[-i]{lastflags}
\DeclareStringOption[open]{includecmd}
\DeclareStringOption[]{html}
\ProcessKeyvalOptions*
```

`\checklistings` This macro offers another way of setting package options with the advantage that values are not expanded.

```
\def\checklistings{\kvsetkeys{CHKL}}
```

4.2 Logging Files to be Processed

Several definitions and commands are used to create and write to the `.chkl` file.

`\CHKL@samplefile` The file generated when a LaTeX document that uses the `checklistings` package is processed.

```
\newwrite\CHKL@samplefile
\openout\CHKL@samplefile=\jobname.chkl
\AtEndDocument{\closeout\CHKL@samplefile}
```

Package options are logged to the file.

```
\write\CHKL@samplefile{subdir=\CHKL@subdir/}
\write\CHKL@samplefile{prefix=\CHKL@prefix}
\write\CHKL@samplefile{ext=\CHKL@ext}
\write\CHKL@samplefile{compiler=\CHKL@compiler}
\write\CHKL@samplefile{compilerflags=\CHKL@compilerflags}
\write\CHKL@samplefile{lastflags=\CHKL@lastflags}
\write\CHKL@samplefile{includecmd=\CHKL@includecmd}
\write\CHKL@samplefile{htmlfilter=\CHKL@html}
```

`\CHKL@logsample` An entry is logged for each `chklisting` environment. It contains the sequence number for the example, followed by a colon, an ordered list of other sample files to import, and the page and line numbers (to include in error messages).

```
\DeclareRobustCommand{\CHKL@logsample}[2]{%
  \edef\CHKL@tolog{#1:#2 [page=\noexpand\thepage] [line=\the\inputlineno]}%
  \expandafter\write\expandafter\CHKL@samplefile\expandafter{\CHKL@tolog}%
}
```

4.3 Insertion of Compilation Results

Several macros are defined for use by the `checklistings.sh` script (and any similar program). These macros are called from within the `.tex` file generated for each `chklisting` environment.

`\ifchklisting` A successful compilation is signalled by `\chklistingtrue`, and a failed compilation by `\chklistingfalse`.

```
\newif\ifchklisting
```

`\setchklistingcmd` The command used to compile a sample is recorded by `\setchklistingcmd` which (re)defines the internal `\CHKL@prompt` value.

```
\DeclareRobustCommand{\setchklistingcmd}[1]{%
    \global\def\chklistingcmd{\emph{\CHKL@prompt{#1}}}}
```

`ChkListingMsg` Normal compiler messages (written on `stdout`) should be communicated between `\begin{ChkListingMsg}` and `\end{ChkListingMsg}`. This verbatim text is saved using the `SaveVerbatim` feature of `fancyvrb`.

```
\def\ChkListingMsg{\FV@Environment{}{ChkListingMsg}}
\def\FVB@ChkListingMsg{\FVB@SaveVerbatim{ChkListingMsg}}
\let\FVE@ChkListingMsg\FVE@SaveVerbatim
\DefineVerbatimEnvironment{ChkListingMsg}{ChkListingMsg}{}
```

`ChkListingErr` Compiler error messages (usually written on `stderr`) should be communicated between `\begin{ChkListingErr}` and `\end{ChkListingErr}`. This verbatim text is saved using the `SaveVerbatim` feature of `fancyvrb`.

```
\def\ChkListingErr{\FV@Environment{}{ChkListingErr}}
\def\FVB@ChkListingErr{\FVB@SaveVerbatim{ChkListingErr}}
\let\FVE@ChkListingErr\FVE@SaveVerbatim
\DefineVerbatimEnvironment{ChkListingErr}{ChkListingErr}{}
```

`\chklistingfile` This is the filename used by `checklistings.sh` to refer to the file containing sample code when `\setchklistingcmd` is called.

```
\DeclareRobustCommand{\chklistingfile}{\CHKL@prefix\CHKL@ext}
```

4.4 Main Environment

Several auxiliary definitions are needed to track per-environment configuration options.

`\ifCHKL@shouldfail` This boolean variable records whether sample code is expected to fail.

```
\newif\ifCHKL@shouldfail
```

`\ifCHKL@showcode` This boolean variable records whether the compilation result should be shown.

```
\newif\ifCHKL@showcode
```

`\CHKL@skip` Defines the number of lines to be skipped when displaying a code sample. It defaults to the value of `\CHKL@pkg@globalskip`, but may be altered by the environment options `skipone` and `skiptwo`. This value is only used to generate the `skip=` field in the `.chkl` file. The actual skipping is done by the `firstline` option of `verbopts`.

```
\def\CHKL@skip{0}
```

`\CHKL@lcompilerflags` Defines compiler flags to be used for a specific example (before those of the global `\CHKL@compilerflags`). It is empty by default, but may be altered by the environment option `compilerflags`. This value is only used to generate the `flags=` field in the `.chkl` file.

```
\edef\CHKL@lcompilerflags{\null}
```

The `keyval` package[6] is used to parse environment options. The following macros setup parameters used by the `chklisting` environment.

`\CHKL@continue` `\CHKL@precontinue`

These two macros hold lists of source code identifiers: `\CHKL@precontinue` tracks the dependencies of the previous `chklisting` environment and `\CHKL@continue` tracks those of the current one. The `continue` option appends the previous dependencies onto the list of current ones. The dependencies used at each labelled environment are remembered in `\RBRB@deps`⟨*label*⟩. The `include` option causes them to be added to the list of current dependencies.

```
\edef\CHKL@precontinue{}
\define@key{CHKL@envkeys}{continue}[]{\edef\CHKL@continue{\CHKL@precontinue}}
\define@key{CHKL@envkeys}{include}{%
  \edef\CHKL@continue{\CHKL@continue\space\@ifundefined{CHKL@deps@#1}%
    {#1}{\csname CHKL@deps@#1\endcsname}}}

\define@key{CHKL@envkeys}{fail}[]{\CHKL@shouldfailtrue}
\define@key{CHKL@envkeys}{label}{\edef\CHKL@label{#1}}
\define@key{CHKL@envkeys}{skipnone}[]{%
  \edef\CHKL@skip{0}\edef\CHKL@verbopts{\CHKL@verbopts,firstline=1}}
\define@key{CHKL@envkeys}{skipone}[]{%
  \edef\CHKL@skip{1}\edef\CHKL@verbopts{\CHKL@verbopts,firstline=2}}
\define@key{CHKL@envkeys}{skiptwo}[]{%
  \edef\CHKL@skip{2}\edef\CHKL@verbopts{\CHKL@verbopts,firstline=3}}
\define@key{CHKL@envkeys}{hide}[]{\CHKL@showcodefalse}
\define@key{CHKL@envkeys}{withresult}[]{\CHKL@withresulttrue}
\define@key{CHKL@envkeys}{withoutresult}[]{\CHKL@withresultfalse}
\define@key{CHKL@envkeys}{compilerflags}[]{\edef\CHKL@lcompilerflags{#1}}
\define@key{CHKL@envkeys}{showproblems}[]{\CHKL@showproblemstrue}
\define@key{CHKL@envkeys}{hideproblems}[]{\CHKL@showproblemsfalse}
```

`\chklistingmsg`

This macro takes a single argument ⟨*label*⟩. It first configures the `listings` and `fancyvrb` packages with the current display options. It then checks the `fancyvrb` saved text namespace (‘`FV@SV@...`’) for an entry named ‘`...CHKL@MSG@`⟨*label*⟩’. If found, the associated verbatim text is inserted via the `\UseVerbatim` feature of `fancyvrb`, otherwise an error message is inserted. In the latter case, we prefer not to fail outright, because the user may not yet have had the chance to run the compiler on the extracted code, in which case the log will already contain warnings from `chklisting`. By convention, the `chklisting` environment creates an entry for the empty label (‘`FV@SV@CHKL@MSG@`’) when compilation succeeds.

```
\DeclareRobustCommand{\chklistingmsg}[1]{
  \bgroup%
  \ifx\CHKL@msglst\CHKL@emptyoption\else
     \expandafter\lstset\expandafter{\CHKL@msglst,fancyvrb=true}\fi%
  \@ifundefined{FV@SV@CHKL@MSG@#1}
    {\def\@tempa{#1}
     \ifx\@tempa\empty
       \CHKL@none
```

[6] http://www.ctan.org/pkg/keyval

```
      \else
        $\langle$No message found for the label `#1'!$\rangle$
      \fi}
     {\expandafter\UseVerbatim\expandafter[\CHKL@msgstyle]{CHKL@MSG@#1}}%
  \egroup}
```

`\chklistingerr` This macro is essentially the same as the previous one—only that the substring '`ERR`' is used instead of '`MSG`'.

```
\DeclareRobustCommand{\chklistingerr}[1]{
  \bgroup%
  \ifx\CHKL@errlst\CHKL@emptyoption\else
     \expandafter\lstset\expandafter{\CHKL@errlst,fancyvrb=true}\fi%
  \@ifundefined{FV@SV@CHKL@ERR@#1}
    {\def\@tempa{#1}
     \ifx\@tempa\empty
       \CHKL@none
     \else
       $\langle$No message found for the label `#1'!$\rangle$
     \fi}
    {\expandafter\UseVerbatim\expandafter[\CHKL@errstyle]{CHKL@ERR@#1}}%
  \egroup}
```

`\chklistinginput` Load the given filename, normally `\chklistingcurrfile` into a verbatim environment using `\CHKL@verbopts`.

```
\newcommand{\chklistinginput}[1]{%
   \bgroup%
   \ifx\CHKL@codelst\CHKL@emptyoption\else%
     \expandafter\lstset\expandafter{\CHKL@codelst,fancyvrb=true}%
   \fi%
   \expandafter\fvset\expandafter{\CHKL@pkg@verbopts}%
   \expandafter\VerbatimInput\expandafter[\CHKL@codestyle]{#1}%
   \egroup%
}
```

`chklisting` `\chklistingcurrfile` This is the main environment for including source code. This implementation has two parts:

1. It uses the listings package to write the code to a file,
2. It either loads the corresponding `.tex` file or logs an error message.

The `listings` package allows the definition of custom verbatim environments. This one has a single argument (a list of `keyval` options).

The `\chklistingcurrfile` macro is set to the filename of the most recent file created by the environment.

```
\lstnewenvironment{chklisting}[1][]
  {%
```

Set default parameter values before invoking `\setkeys`:

```
    \CHKL@shouldfailfalse%
    \CHKL@showcodetrue%
    \let\CHKL@label\@undefined%
    \edef\CHKL@continue{}%
    \def\CHKL@skip{\CHKL@pkg@globalskip}%
    \let\CHKL@verbopts\CHKL@pkg@verbopts%
    \def\@currentlabel{\thechklisting}%
    \setkeys{CHKL@envkeys}{#1}%
```

Log an entry to the `.chkl` file:

```
    \CHKL@logsample{\arabic{chklisting}}{\CHKL@continue%
      \space\ifnum\CHKL@skip>0[skip=\CHKL@skip]\fi%
      \ifx\CHKL@lcompilerflags\null\else\space[flags=\CHKL@lcompilerflags]\fi%
      \ifCHKL@shouldfail\space[fail]\fi}%
```

Update `\CHKL@precontinue` for the next source code block, and, if a label was defined, add an `\CHKL@deps@`⟨*label*⟩ entry.

```
    \global\edef\CHKL@precontinue{\CHKL@continue\space\arabic{chklisting}}%
    \@ifundefined{CHKL@label}{}{%
    \global\expandafter\edef\csname CHKL@deps@\CHKL@label\endcsname{\CHKL@precontinue}}%
```

A file will be created in the `\CHKL@subdir` subdirectory, with the name `\CHKL@prefix` followed by the value of the `chklisting` counter, padded out with zeroes to four digits, and the extension `\RBRB@ext`.

```
    \edef\CHKL@num{%
        \ifnum\value{chklisting}<1000 0\fi
        \ifnum\value{chklisting}<100  0\fi
        \ifnum\value{chklisting}<10   0\fi
        \arabic{chklisting}}%
    \stepcounter{chklisting}%
    \def\CHKL@file{\CHKL@subdir/\CHKL@prefix\CHKL@num}%
```

Set the `\chklistingcurrfile` macro to the name of the file created for this environment, both for use internally and externally. Clear the definitions used to return information about the compilation run, and close the environment by opening a file, using the `listings` package, into which to write the ensuing contents.

```
    \global\edef\chklistingcurrfile{\CHKL@file\CHKL@ext}%
    \global\let\chklistingcmd\@undefined%
    \global\let\FV@SV@ChkListingMsg\@undefined%
    \global\let\FV@SV@ChkListingErr\@undefined%
    \chklistingtrue%
    \setbox\@tempboxa\hbox\bgroup%
    \lst@BeginWriteFile{\CHKL@file\CHKL@ext}%
  }
```

Start closing the environment by closing the previously opened file and group.

```
  {%
    \lst@EndWriteFile%
    \egroup%
```

If `hide` is not active, then load the newly created file.

```
\ifCHKL@showcode\chklistinginput{\CHKL@file\CHKL@ext}\fi%
```

Check whether a corresponding `.tex` file was created:

```
\global\edef\CHKL@none{$\langle$Cannot load \CHKL@file.tex!$\rangle$}%
\InputIfFileExists{\CHKL@file.tex}{\CHKL@fileexiststrue}{\CHKL@fileexistsfalse}%
```

If the `.tex` file was loaded successfully, create 'unlabelled' saved verbatim environments for the message and error texts. These are exploited, respectively, by the `\chklistingmsg` and `\chklistingerr` macros.

```
\ifCHKL@fileexists%
    \@ifundefined{FV@SV@ChkListingMsg}%
      {}{\global\let\FV@SV@CHKL@MSG@=\FV@SV@ChkListingMsg}%
    \@ifundefined{FV@SV@ChkListingErr}%
      {}{\global\let\FV@SV@CHKL@ERR@=\FV@SV@ChkListingErr}%
```

Then, if compilation failed and the `fail` option was not active, or if compilation succeeded and the `fail` option is active, log a warning message and, if errors are not being ignored, include details in the document. Otherwise, if `withresult` was given, expand either `\chklistingerr` or `\chklistingmsg`.

```
    \ifCHKL@shouldfail%
        \ifchklisting%
            \PackageWarning{checklistings}%
                {\CHKL@file\CHKL@ext\space success}%
            \ifCHKL@showproblems%
              \UseVerbatim[frame=single,
                           label=Unexpected success,
                           rulecolor=\color{red}]{ChkListingMsg}%
            \fi%
        \else%
            \ifCHKL@withresult%
               {\setlength{\partopsep}{0em}\chklistingerr{}}%
            \fi%
        \fi%
    \else%
        \ifchklisting%
            \ifCHKL@withresult%
               {%\vspace{\dimexpr-2\topsep-2\partopsep+.5em\relax}%
                \setlength{\partopsep}{0em}\chklistingmsg{}}%
            \fi%
        \else%
            \PackageWarning{checklistings}%
                {\CHKL@file\CHKL@ext\space failure}%
            \ifCHKL@showproblems%
              \UseVerbatim[frame=single,
                           label=Unexpected failure,
                           rulecolor=\color{red}]{ChkListingErr}%
            \fi%
        \fi%
```

```
        \fi%
    \else%
```

If the `.tex` file was not loaded successfully, clear the `\chklistingcmd` macros, and the 'unlabelled' saved verbatim environments for the message and error results.

```
        \PackageWarning{checklistings}{Cannot load \CHKL@file.tex}%
        \global\let\chklistingcmd\CHKL@none%
        \global\let\FV@SV@CHKL@MSG@\@undefined%
        \global\let\FV@SV@CHKL@ERR@\@undefined%
    \fi%
% If this environment is labelled, create persistent references to the saved
% verbatim environments for the message and error results.
% These are exploited, respectively, by the |\chklistingmsg| and
% |\chklistingerr| macros when their \meta{label} argument is not empty.
%
    \@ifundefined{CHKL@label}{}{%
      \global\expandafter\let%
        \csname FV@SV@CHKL@MSG@\CHKL@label\endcsname=\FV@SV@CHKL@MSG@%
      \global\expandafter\let%
        \csname FV@SV@CHKL@ERR@\CHKL@label\endcsname=\FV@SV@CHKL@ERR@%
    }%
  }%
```

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.