The runverbatim package*

Timothy Bourke and Marc Pouzet

February 24, 2014

1 Introduction

User manuals and papers about programming languages usually contain many code samples, often with accompanying compiler messages, giving the types or values of declarations, or errors explaining why certain declarations are invalid. Packages like `fancyvrb`[1] and `listings`[2] are ideal for displaying code—this package extends them slightly to facilitate passing this same code through a compiler and displaying the results. While it does not focus on a specific programming language, it is intended 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 x : int
val f : int -> int
```

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

A first pass through `latex` generates both an `.rvrb` 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 `runverbatim.sh` script processes these files to produce a `.tex` file with the results of compilation. A second pass through `latex` updates the compiler message.

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

*This document corresponds to runverbatim ?, dated ?.

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

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

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

These definitions follow on from the previous ones:

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

```
val y : int
```

Examples need not necessarily succeed:

```
This code does not compile:
\begin{runverbatim}[fail, withresult]
let w = 1 + "four"
\end{runverbatim}
```

This code does not compile:

```
let w = 1 + "four"
```

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

2 Use

Using the package involves three elements:

1. The declaration `\usepackage{runverbatim}`.
   Section 2.1 describes the options for configuring package behaviour.
2. The environment `runverbatim`.
   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 `runverbatim.sh`.
   Running this script passes the contents of each `runverbatim` environment through a compiler or interpreter and copies the resulting output into a file.

2.1 Package options

`\runverbatimsetup`

Package options are either given as optional arguments to `\usepackage` or via one or more calls to `\runverbatimsetup`. The advantage of the latter is that macros are not expanded (for a detailed explanation see the documentation for `kvoptions`,[3] Section 4.1, *Package kvoptions-patch*). Options are passed as a comma separated list of ⟨*key*⟩=⟨*value*⟩ pairs and single ⟨*key*⟩s.

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

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

2.1.1 Behavioural options

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

`option`	description	`default`
`withresult`	Automatically show compilation results.	`false`
`skipone`	Do not display the first line of code (see the description under the `runverbatim` environment).	
`skiptwo`	Do not display the first two lines of code (see the description under the `runverbatim` environment).	

2.1.2 Configuring compilation

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

`option`	description	`default`
`prefix`	Prefix for naming source files.	`runverbatim`
`ext`	Extension of source files.	`.ml`
`subdir`	If defined, source files are created in the given subdirectory, which must already exist. A final slash (`/`) should not be given.	`.`
`prompt`	The prompt displayed by "runverbatimcmd.	`#`
`compiler`	Path of the compiler to execute.	`ocamlc`
`compilerflags`	Flags passed to the compiler. These are not revealed by "runverbatimcmd.	
`lastflags`	Flags passed to the compiler before the main source file, i.e., the last one given.	`-i`
`includecmd`	The source language command for importing the definitions of another file.	`open`

Each `runverbatim` environment is assigned a number n, from zero, and its contents are written to the file: $\langle subdir \rangle / \langle prefix \rangle \langle n \rangle . \langle ext \rangle$, where $\langle n \rangle$ is zero-padded to four characters. For example, by default, the fourth environment is written to the file `runverbatim0003.ml` in the current directory.

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

$\langle includecmd \rangle$ `Withopen0000` $\langle includecmd \rangle$ `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 through to the `fancyvrb` package and applied to code blocks. For example, `frame=single`. These options must typically be set using `\runverbatimsetup`, since they will usually contain commands that should not be expanded immediately (like `\em` or `\bf`).

2.2 The `runverbatim` environment

`runverbatim` 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 `runverbatim` 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 `\runverbatimmsg` and `runverbatimerr`). 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`	Do not display the code itself. It is still compiled and displayed (if `withresult`) is active, and its definitions are still available for continuation (`continue`) or 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.

The results of compiling the code in a `runenvironment` are made available in the following macros until the next `runenvironment` which will redefine them.

`\runverbatimcmd` `\runverbatimcmd` 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*⟩.

`\runverbatimmsg` `\runverbatimmsg` contains the verbatim text emitted by the compiler when compilation succeeds. It is undefined after an environment marked `fail`.

`\runverbatimerr` `\runverbatimerr` contains the verbatim text emitted by the compiler when compilation fails. It is undefined after an environment not marked `fail`.

2.3 The `runverbatim.sh` script

Processing a document that uses the `runverbatim` package produces a `.rvrb` file containing compiler options and a list of source files to together with their interdependencies. The `runverbatim.sh` script processes `.rvrb` files by executing the specified compiler (or interpreter) 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 `runverbatim.sh` script is written for the Bourne shell (`sh`). It takes a list of `.rvrb` files as arguments (with or without the exentions), but if none are given it processes all such files in the current working directory.

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 and do everything.

4 Implementation

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

`\runverbatim` Generate the sequence of source code identifiers used in per-environment filenames and to manage dependencies.

```
\newcounter{runverbatim}
```

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

```
\newif\ifRVRB@fileexists
```

4.1 Package Options

The package options are processed using the `kvoptions` package.[4]

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

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

Declare the package options and their default values:

```
\DeclareBoolOption{withresult}
\DeclareComplementaryOption{withoutresult}{withresult}
\DeclareVoidOption{skipone}
    {\edef\RVRB@pkg@verbopts{\RVRB@pkg@verbopts,firstline=2}}
\DeclareVoidOption{skiptwo}
    {\edef\RVRB@pkg@verbopts{\RVRB@pkg@verbopts,firstline=3}}
\DeclareDefaultOption
    {\edef\RVRB@pkg@verbopts{\RVRB@pkg@verbopts,\CurrentOption}}
\DeclareStringOption[]{codestyle}
\DeclareStringOption[formatcom=\em]{msgstyle}
\DeclareStringOption[formatcom=\em]{errstyle}
\DeclareStringOption{codelst}
\DeclareStringOption{msglst}
\DeclareStringOption{errlst}
\DeclareStringOption{emptyoption}
\DeclareStringOption[.]{subdir}
```

[4] http://www.ctan.org/pkg/kvoptions

```
\DeclareStringOption[runverbatim]{prefix}
\DeclareStringOption[.ml]{ext}
\DeclareStringOption[\#]{prompt}
\DeclareStringOption[ocamlc]{compiler}
\DeclareStringOption{compilerflags}
\DeclareStringOption[-i]{lastflags}
\DeclareStringOption[open]{includecmd}
\ProcessKeyvalOptions*
```

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

```
\def\runverbatimsetup{\kvsetkeys{RVRB}}
```

4.2 Logging Files to Process

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

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

```
\newwrite\RVRB@samplefile
\openout\RVRB@samplefile=\jobname.rvrb
\AtEndDocument{\closeout\RVRB@samplefile}
```

Package options are logged to the file.

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

`\RVRB@logsample` An entry is logged for each `runverbatim` 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).

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

4.3 Insertion of Compilation Results

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

`\ifrunverbatim` A successful compilation is signalled by `\runverbatimtrue`, and a failed compilation by `\runverbatimfalse`.

```
\newif\ifrunverbatim
```

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

```
\newcommand{\setrunverbatimcmd}[1]{%
    \global\def\runverbatimcmd{\emph{\RVRB@prompt{#1}}}}
```

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

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

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

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

`\runverbatimfile` This is the filename used by `runverbatim.sh` to refer to the file containing sample code when `\setrunverbatimcmd` is called.

```
\newcommand{\runverbatimfile}{\RVRB@prefix\RVRB@ext}
```

4.4 Main Environment

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

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

```
\newif\ifRVRB@shouldfail
```

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

```
\newif\ifRVRB@showcode
```

The `keyval` package[5] is used to parse environment options. The following macros setup parameters used by the `runverbatim` environment.

`\RVRB@continue` `\RVRB@precontinue` These two macros hold lists of source code identifiers: `\RVRB@precontinue` tracks the dependencies of the previous `runverbatim` environment, and `\RVRB@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\RVRB@precontinue{}
```

[5] http://www.ctan.org/pkg/kvoptions

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

\define@key{RVRB@envkeys}{fail}[]{\RVRB@shouldfailtrue}
\define@key{RVRB@envkeys}{label}{\edef\RVRB@label{#1}}
\define@key{RVRB@envkeys}{skipnone}[]{\edef\RVRB@verbopts{\RVRB@verbopts,firstline=1}}
\define@key{RVRB@envkeys}{skipone}[]{\edef\RVRB@verbopts{\RVRB@verbopts,firstline=2}}
\define@key{RVRB@envkeys}{skiptwo}[]{\edef\RVRB@verbopts{\RVRB@verbopts,firstline=3}}
\define@key{RVRB@envkeys}{hide}[]{\RVRB@showcodefalse}
\define@key{RVRB@envkeys}{withresult}[]{\RVRB@withresulttrue}
\define@key{RVRB@envkeys}{withoutresult}[]{\RVRB@withresultfalse}
```

`RunVerbatim` This is the main environment for including source code. This macro works in 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).

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

Set default parameter values before invoking `\setkeys`:

```
    \RVRB@shouldfailfalse%
    \RVRB@showcodetrue%
    \let\RVRB@label\@undefined%
    \edef\RVRB@continue{}%
    \let\RVRB@verbopts\RVRB@pkg@verbopts%
    \def\@currentlabel{\therunverbatim}%
    \setkeys{RVRB@envkeys}{#1}%
```

Log an entry to the `.rvrb` file:

```
    \RVRB@logsample{\arabic{runverbatim}}{\RVRB@continue\ifRVRB@shouldfail\space[fail]\fi}%
```

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

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

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

```
    \edef\RVRB@num{%
        \ifnum\value{runverbatim}<1000 0\fi
        \ifnum\value{runverbatim}<100  0\fi
```

```
        \ifnum\value{runverbatim}<10   0\fi
        \arabic{runverbatim}}%
    \stepcounter{runverbatim}%
    \def\RVRB@file{\RVRB@subdir/\RVRB@prefix\RVRB@num}%
```

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\let\runverbatimcmd\@undefined%
    \global\let\FV@SV@RunVerbatimMsg\@undefined%
    \global\let\FV@SV@RunVerbatimErr\@undefined%
    \runverbatimtrue%
    \setbox\@tempboxa\hbox\bgroup%
    \lst@BeginWriteFile{\RVRB@file\RVRB@ext}%
}
```

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

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

If `hide` is not active, apply `\RVRB@verbopts` and reload the newly created file.

```
    \ifRVRB@showcode
        \bgroup%
        \ifx\RVRB@codelst\RVRB@emptyoption\else
          \expandafter\lstset\expandafter{\RVRB@codelst,fancyvrb=true}%
        \fi
        \expandafter\fvset\expandafter{\RVRB@verbopts}%
        \expandafter\VerbatimInput\expandafter[\RVRB@codestyle]{\RVRB@file\RVRB@ext}%
        \egroup%
    \fi
```

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

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

If the `.tex` file was loaded successfully, set the `\runverbatimmsg` and `\runverbatimerr` macros via the `\UseVerbatim` feature of `fancyvrb`.

```
    \ifRVRB@fileexists
        \@ifundefined{FV@SV@RunVerbatimMsg}
          {}{\global\def\runverbatimmsg{%
              \bgroup%
              \ifx\RVRB@msglst\RVRB@emptyoption\else
                 \expandafter\lstset\expandafter{\RVRB@msglst,fancyvrb=true}\fi%
              \expandafter\UseVerbatim\expandafter[\RVRB@msgstyle]{RunVerbatimMsg}%
              \egroup}}
        \@ifundefined{FV@SV@RunVerbatimErr}
          {}{\global\def\runverbatimerr{%
              \bgroup%
              \ifx\RVRB@errlst\RVRB@emptyoption\else%
```

```
                \expandafter\lstset\expandafter{\RVRB@errlst,fancyvrb=true}\fi%
             \expandafter\UseVerbatim\expandafter[\RVRB@errstyle]{RunVerbatimErr}%
             \egroup}}
```

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 include details in the document. Otherwise, if `withresult` was given, expand either `\runverbatimerr` or `\runverbatimmsg`.

```
        \ifRVRB@shouldfail
            \ifrunverbatim
                \PackageWarning{runverbatim}
                    {Compilation of \RVRB@file\RVRB@ext\space should have failed}
                \UseVerbatim[frame=single,
                             label=Unexpected success,
                             rulecolor=\color{red}]{RunVerbatimMsg}
            \else
                \ifRVRB@withresult
                   {\setlength{\partopsep}{0em}\runverbatimerr}
                \fi
            \fi
        \else
            \ifrunverbatim
                \ifRVRB@withresult
                   {\setlength{\partopsep}{0em}\runverbatimmsg}
                \fi
            \else
                \PackageWarning{runverbatim}
                    {Compilation of \RVRB@file\RVRB@ext\space should not have failed}
                \UseVerbatim[frame=single,
                             label=Unexpected failure,
                             rulecolor=\color{red}]{RunVerbatimErr}
            \fi
        \fi
    \else
```

If the `.tex` file was not loaded successfully, set `\runverbatimcmd`, `\runverbatimmsg`, and `\runverbatimerr` to an error message.

```
        \PackageWarning{runverbatim}{Cannot load \RVRB@file.tex}
        \global\let\runverbatimcmd\RVRB@none
        \global\let\runverbatimmsg\RVRB@none
        \global\let\runverbatimerr\RVRB@none
    \fi
}
```