The **patchDVI** package

Duncan Murdoch

December 16, 2013

Abstract

The **patchDVI** package works with Sweave [3] and document previewers to facilitate editing: it modifies the links that LaTeX puts into the output so that they refer to the original source. It also includes a few project management functions to make large multi-file Sweave documents easier to handle.

Contents

1 Introduction

Most implementations of LaTeX allow source references to be emitted, so that previewers of the `.dvi` or `.pdf` output file can link back to the original source line. This has been a feature of the `yap` previewer for `.dvi` files in MikTeX [4] for many years. Support for source references has appeared more recently for `.pdf` output, first in `pdfsync`. Most recently Synctex [2] links have been implemented in `pdflatex` and other LaTeX processors.

Unfortunately for Sweave users, these links point to the `.tex` source that was processed, which is not the true source code in the Sweave `.Rnw` or `.Snw` or other input file. (I will refer to all of these as `.Rnw` files.) Clicking on "go to source" in a previewer will jump to the `.tex` file; changes made there will be lost the next time the Sweave input is processed.

I wrote the **patchDVI** package to address this problem. It works as follows. If the Sweave file is processed with the option `concordance=TRUE`, Sweave will output a record of the concordance between the lines in the input file and the output file. When the file is processed by LaTeX, this information is embedded in the output. (Details of the embedding are described in sections 4 to 6 below.) After producing the `.dvi` or `.pdf` file, a **patchDVI** function is run to read the concordance information and to patch the source reference information produced by LaTeX. Once this has been done, previewers will see references to the original Sweave source rather than the LaTeX intermediate file.

Besides the technical details mentioned above, this paper describes the history of **patchDVI** in the next section and in section 7 some project management functions. It concludes with a short discussion.

2 Quick Start Instructions

There are several ways to make use of **patchDVI**. This section describes some common ones.

In all cases the package needs to be installed first; the current release is on CRAN and can be installed using

```
> install.packages("patchDVI")
```

Source code is maintained on R-forge, and the latest development version can be installed using

```
> install.packages("patchDVI", repos="http://R-forge.r-project.org")
```

The document also needs to have an option set to produce the "concordances" (links between the `.Rnw` source and the `.tex` output of Sweave). Include these lines near the start of your document:

```
\usepackage{Sweave}
\SweaveOpts{concordance=TRUE}
```

The simplest way to proceed is from within R. Assuming `doc.Rnw` is the Sweave document to process and it is in the current working directory, run

```
> library(patchDVI)
> SweavePDF("doc.Rnw")
```

This runs `doc.Rnw` through Sweave, runs any other chapters in the project through Sweave, then runs the main `.tex` file (typically `doc.tex`, but not necessarily; see section 7 below) through `pdflatex`, and patches the source links. To produce DVI output instead of PDF substitute `SweaveDVI` for `SweavePDF`, and to use `latex` and `dvipdfm` to produce `PDF` output, use `SweaveDVIPDFM`. If you are using MikTeX on Windows, the functions are `SweavePDFMiktex` and `SweaveMiktex` correspond to the first two of these respectively, and use a few MikTeX-specific features.

These functions all have an optional argument `preview`, which can contain a command line to run a `.pdf` or `.dvi` previewer (with the filename replaced by `%s`). The `.pdf` previewer should be one that can handle Synctex links; unfortunately, Acrobat Reader and MacOS Preview are both deficient in this area. On Windows, `SumatraPDF` works, as do the built-in previewers in TeXShop and TeXWorks on MacOS X and other platforms.

MikTeX includes the `yap` previewer for `.dvi` files; the `SweaveMiktex` command sets it as the default.

Another way to proceed is directly from within your text editor. The instructions here depend on your editor; I have included a few in the Appendices: TeXShop in Appendix A, WinEdt in Appendix C, and TeXWorks in Appendix B. Some editors (e.g. TeXShop and TeXWorks) include a previewer that can handle the source links.

Finally, you may want to run Sweave from the command line, outside of R. This line (or the obvious variants with replacements for `SweavePDF`) should do it:

```
Rscript -e 'patchDVI::SweavePDF("doc.Rnw")'
```

3 patchDVI History

Initially **patchDVI** only worked for `.dvi` files (hence the name). It required changes to the Sweave function in R, which first appeared around the release of R version 2.5.0. with incompatible changes in R version 2.8.0 when `.pdf` support was added to **patchDVI**.

Using **patchDVI** requires a pre-processing step (Sweave), LaTeX processing, and a post-processing step (patching). This is usually followed by a preview of the resulting output file. It quickly became apparent that it was convenient to package these steps into a single R function, so the user only needed to make a single call. But the details of LaTeX processing vary from platform to platform, so I wrote functions `SweaveMiktex` and `SweavePDFMiktex` specific to the MikTex platform, with the intention of adding others as I used them or users told me what needed adding. This never happened, but in the meantime, Brian Ripley made the `tools::texi2dvi` function in R much more flexible, and in version 1.7 of **patchDVI** I included a modified version of it with the hope that **patchDVI** should be more nearly platform neutral.

The 1.7 release was motivated by an attempt to support TeXWorks [1], a cross-platform LaTeX targetted editor. TeXWorks was still in its early days (I was working with version 0.2 on Windows), and it did not have enough flexibility to handle large Sweave projects, where for example, each chapter of a book requires separate Sweave processing, but LaTeX processes only a main wrapper file. This prompted me to include more `make`-style capabilities into **patchDVI**. It is now possible to specify a list of Sweave input files to process (optionally only if they have changed since the last processing) and the main wrapper file, all within Sweave chunks in a single file, using the `SweaveAll` function.

The `SweaveDVIPDFM` function is the most recent addition. For English language processing, I find `pdflatex` to be the most convenient processor, but it does not work well in languages like Japanese. During a visit to the Institute of Statistical Mathematics in Tokyo I learned of the issues, and with the help of Prof. H. Okumura and Junji Nakano I worked out `SweaveDVIPDFM` to handle the two step conversion to PDF.

4 Sweave Concordances

Sweave processes the code chunks in the `.Rnw` file, replacing each with the requested output from the command. This means that the output

Table 1: Input file for simple example.

Line number	Input text
1	`\SweaveOpts{concordance=TRUE}`
2	This is text
3	`<<>>=`
4	`123`
5	`@`
6	This is more text

`.tex` file alternates between copied LaTeX source and newly produced blocks of output. Each line in the `.tex` file can thus be mapped to one or more lines of input, and that is what the concordance does.

The concordance records are text records in the following format. There are four parts, separated by colons:

1. The label `concordance` to indicate the type of record.
2. The output `.tex` filename.
3. The input `.Rnw` filename.
4. The input line numbers corresponding to each output line.

The third component is compressed using a simple encoding: The first number is the first line number; the remainder of line numbers are a run-length encoding of the differences. Thus if the input file is as shown in Table 1, the output file would be as shown in Table 2, with the concordance as shown there in the second column. This concordance would be recorded in the file `sample-concordance.tex` as

```
\Sconcordance{concordance:sample.tex:sample.Rnw:%
1 1 1 1 2 7 0 1 2}
```

The numeric part of this file may be interpreted as shown in Table 3.

5 Patching `.dvi` Files

The `\Sconcordance` macro expands to a `\special` macro when producing a `.dvi` file. This is included verbatim in the `.dvi` file. The

Table 2: Output file for simple example.

Output line	Input line	Output text
1	1	`\input{sample-concordance}`
2	2	This is text.
3	4	`\begin{Schunk}`
4	4	`\begin{Sinput}`
5	4	`> 123`
6	4	`\end{Sinput}`
7	4	`\begin{Soutput}`
8	4	`[1] 123`
9	4	`\end{Soutput}`
10	4	`\end{Schunk}`
11	6	This is more text

Table 3: Encoding of numeric part of concordance record.

Values	Interpretation	Expansion
`1`	line 1	1
`1 1`	1 increase of 1	2
`1 2`	1 increase of 2	4
`7 0`	7 increases of 0	4 4 4 4 4 4 4
`1 2`	1 increase of 2	6

"concordance:" prefix identifies it as a **patchDVI** concordance. The `patchDVI` function scans the whole file until it finds this sort of record. (There may be more than one, if multiple files make up the document.) Source references are also recorded by LaTeX in `\special` records; their prefix is "src:". The `patchDVI` function reads each "src:" special and if it refers to a file in a "concordance:" special, makes the substitution. At the end, it rewrites the whole `.dvi` file.

6 Patching `.synctex` Files

When using `pdflatex`, the `\Sconcordance` macro expands to a `\pdfobj` macro containing the concordance, which eventually is embedded in the `.pdf` file. However, the Synctex scheme of source references does not write the references to the `.pdf` file directly. Instead, they are written to a separate file with extension `.synctex`, or a compressed version of that file, with extension `.synctex.gz`. The `patchSynctex` function reads the concordances from either the `.pdf` file (when `pdflatex` was used) or the `.dvi` file, and the source references from the Synctex file. It rewrites only the Synctex file when it makes its changes.

7 Project Management Function `SweaveAll`

As mentioned above, there are a number of steps involved in running **patchDVI** with a complex Sweave project:

1. Run Sweave on each input file.
2. Run LaTeX on the main wrapper file.
3. Run the appropriate **patchDVI** function on the output file.
4. Preview or print the result.

Moreover, step 1 needs to be repeated once for each Sweave file, but only if the content has changed since the last run, while the other steps need only be done once.

To manage this complication, the **patchDVI** package includes a simple project management function, `SweaveAll`. This function runs Sweave on multiple files and determines the name of the main wrapper file. It is used internally by the functions described in Section 7.1 below, but can also be called directly by the user.

Here is how it works. `SweaveAll` takes a vector of filenames as input, and runs Sweave on each. After each run, it examines the global environment for four variables: `.PostSweaveHook`, `.SweaveFiles`, `.SweaveMake` and `.TexRoot`.

A code chunk in a `.Rnw` file may produce a function (or the name of a function; `match.fun` is used to look it up) named `.PostSweaveHook`. If present, this should be a function taking a single argument. Immediately after running `Sweave`, `SweaveAll` will call this function, passing the name of the `.tex` output file as the only argument. The hook can do any required postprocessing, for example, it could remove local pathnames from output strings.

The optional parameter `PostSweaveHook` to the `SweaveAll` function can provide a default hook function. Hooks specified via `.PostSweaveHook` take precedence in any given input file.

`SweaveAll` will also check for a character vector named `.SweaveFiles`. It should contain the names of `.Rnw` files in the project. If no corresponding `.tex` file exists, or the `.Rnw` file is newer, they will be run through Sweave. They may in turn name additional `.Rnw` files to process; each file is processed only once, even if it is named several times.

There is an optional parameter named `make` to the `SweaveAll` function. If `make=1` (the default), things proceed as described above. If `make=0`, the `.SweaveFiles` variable is ignored, and only the explicitly named files in the call to `SweaveAll` are processed. If `make=2`, then all files are processed, whether they are newer than their `.tex` file or not. The `.SweaveMake` variable will override the value of `make`.

An `.Rnw` file may also set the value of `.TexRoot` to the name of a `.tex` file. If it does, then that is the file that should be passed to LaTeX for processing. If none is given, then the first file in the call to `SweaveAll` will be assumed to be the root file. (If multiple different `.TexRoot` variables are specified by different `.Rnw` files, one of them will be used, but it is hard to predict which: so don't do that.) Whichever file is determined to be the root file is the name returned by the `SweaveAll` call.

`SweaveAll` is called by all of the functions described in subsection 7.1 below to do step 1 of the **patchDVI** steps.

The workflow this is designed for is as follows. Each `.Rnw` chapter (named for example "chapter.Rnw") in a large project should specify the `.TexRoot`, e.g. using the code chunk

```
<<echo=FALSE>>=
```

```
.TexRoot <- "wrapper.tex"
@
```

Similarly, the wrapper file (named for example "wrapper.Rnw") should be a `.Rnw` file that sets `.SweaveFiles` to the complete list of files in the project. Then one can build an initial copy of the entire document by calling `SweavePDF` or `SweaveDVI` (or the MikTeX versions) with argument `"wrapper.Rnw"`. Later, while one is working on "chapter.Rnw", one can call one of those functions with argument `"chapter.Rnw"` and the chapter will be processed through the full sequence, without running Sweave on the other chapters.

More complicated schemes are possible. For example:

- Each chapter can have subsections in separate files; then the chapter would name the subsections, but the main wrapper would only need to name the chapters if you can assume that only the chapter being edited was changed.
- If one wants to "make" the full project every time, then include `"wrapper.Rnw"` in `.SweaveFiles` in each chapter.

7.1 The Complete Process

The **patchDVI** package contains five functions designed to run all four of the steps listed at the start of this section. The functions `SweaveDVI` and `SweaveMiktex` produce `.dvi` output in the general case and for MikTeX respectively; `SweavePDF` and `SweavePDFMiktex` do the same for direct `.pdf` output from `pdflatex`. Finally, `SweaveDVIPDFM` runs the two-step conversion using first `latex` and then `dvipdfm`.

In each case, the TEXprocessing functions are customizable.

For example, the text editor that I use allows me to call external functions with arguments depending on the name of the current file and the line number within it. I have it call a Windows batch file with the line set as argument `%1` and the filename set as argument `%2`; the batch file invokes R using the command line

```
echo patchDVI::SweaveMiktex('%2',
    preview='yap -1 -s"%1%2" "\x25s"')
    | Rterm --slave
```

(all on one long line). This passes the current file to `SweaveMiktex`, and sets the preview command to use the `yap` options `-1` to update the current view (rather than opening a new window), and to jump to

the line corresponding to the editor line. The code `"\x25s"` is simply `"%s"` encoded without an explicit percent sign, which would be misinterpreted by the Windows command processor. When **patchDVI** calls the previewer, the main `.dvi` filename will be substituted for `%s`.

8 Conclusion

As described in this paper, the **patchDVI** package is a convenient way to work with Sweave in a modern setting, allowing fast switching from source input to preview. It also offers some features to make the management of larger projects easier.

Other possibilities may exist to make use of the code in this package. In order to read and patch `.dvi`, `.pdf` and `.synctex` files, **patchDVI** includes code to work with each of those formats. Users may find imaginative uses for this capability, which I've tried to leave in general form. The low-level `.dvi` editing is done by C functions called from R, while the PDF related work is done in pure R code.

References

[1] Jonathan Kew. TEXworks: Lowering the barrier to entry. *TUGBoat*, 29:362–364, 2008.

[2] Jerôme Laurens. Direct and reverse synchronization with SyncTEX. *TUGBoat*, 29:365–371, 2008.

[3] Friedrich Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. In Wolfgang Härdle and Bernd Rönz, editors, *Compstat 2002 — Proceedings in Computational Statistics*, pages 575–580. Physica Verlag, Heidelberg, 2002. ISBN 3-7908-1517-9.

[4] Christian Schenk. About MikTeX, 2010. Web page `http://www.miktex.org/about`, retrieved August 13, 2010.

A Using patchDVI with TeXShop

TeXShop is a nice TEX editor on MacOS. Dave Gabrielson of the University of Manitoba helped me to work out these instructions. I have updated them in December, 2013 for TeXShop 2.47.

1. In Preferences – Typesetting – Sync Method, choose "SyncTeX".
2. Create a file called `Library/TeXShop/Engines/Sweave.engine` containing the lines

   ```
   #!/bin/tcsh
   Rscript -e "patchDVI::SweavePDF(  '$1' )"
   ```

   in your home directory, and give it executable permissions.
3. Install the `patchDVI` package into R.
4. When editing a `.Rnw` file in TeXShop, choose the Sweave engine from the menu.
5. If you have multiple files in your project, your main file must be a Sweave file (e.g. `Main.Rnw`) which lists all Sweave files in a `.SweaveFiles` variable, and you need to add the line

   ```
   %!TEX root = Main.Rnw
   ```

   to each subordinate file.
6. Add the `\SweaveOpts{concordance=TRUE}` line to your document.

The TeXShop previewer supports SyncTeX; you right click in the preview, and choose Sync from the menu to jump to your source location.

B Using patchDVI with TeXWorks

TeXWorks is an editor for multiple platforms, somewhat similar to TeXShop. These instructions have been tested in version 0.4.5, with MikTeX 2.9 on Windows, and MacTeX on MacOS.

TeXWorks can work with the `patchDVI` project management features using a script to tell it to process the current file through Sweave, but preview the main file. See the instructions below for my current best attempt at such a script. It can also use the TeXShop approach of specifying the TEX root file to be a Sweave file.

1. Add a new SweavePDF command: In

   Edit | Preferences | Typesetting

   click on the "+" sign near the bottom. Set the name of the tool to be SweavePDF. Set the program to Rscript.

   Add two arguments, one per line:

   (a) `-e`

   (b) `patchDVI::SweavePDF('$fullname')`

2. Install the `patchDVI` package into R.

3. Tell TeXWorks to open Sweave files by editing the file pattern configuration file `texworks-config.txt`. This file is in the `configuration` folder of the TeXWorks home directory. For example, I have this line in my file:

   ```
   file-open-filter: Sweave and TeX documents (*.Rnw *.tex)
   ```

4. When editing a `.Rnw` file in TeXWorks, choose the SweavePDF engine from the menu.

5. Add the `\SweaveOpts{concordance=TRUE}` line to your document.

6. If you are using the project management features of `patchDVI` and are editing a subordinate file, TeXWorks will not open or update the PDF preview after it processes changes. There are four workarounds for this.

   The simplest is to manually open the `.pdf` file the first time. After that it will be updated automatically. Unfortunately, if you happen to be editing the main file, the `.pdf` will be opened automatically, and then updates won't happen if you later edit a subordinate file.

   The next simplest is the TeXShop approach: include a line

   ```
   %!TEX root = Main.Rnw
   ```

   near the top of the file, and make sure that `Main.Rnw` refers to all subordinate Sweave files.

   You can force updates without the TEX root specification using the following hook script (written by Stefan Löffler).

```
// TeXworksScript
// Title: Refresh PDF
// Description: Refreshes PDFs after typesetting finishes
// Author: Stefan Löffler
// Version: 0.1
// Date: 2013-11-20
// Script-Type: hook
// Hook: AfterTypeset

windows = TW.app.getOpenWindows()

for (i = 0; i < windows.length; i++) {
    win = windows[i]
    if (win.objectName == "PDFDocument") {
        win.reload()
    }
}
```

Put this script in a file `refreshpdf.js` in your TeXWorks script directory (available through the menu using Scripts | Scripting TeXWorks | Show Scripts Folder). After that, PDF previews will be automatically updated after every Typeset operation.

I have written another script (included in the scripts directory of `patchDVI` as `sweavePreview.js`); it attempts to determine the PDF filename from the console output, and reloads that file. I have tested it on my own systems but not elsewhere, so I am not sure that it will work on other systems. With this script (shown below) TeXWorks should open the preview without using the tricks above. The TeXWorks previewer will jump back to the source if you right click and choose Jump to Source.

```
// TeXworksScript
// Title: Sweave Preview
// Description: Looks for PDF filename to preview
// Author: Duncan Murdoch
// Version: 0.1
// Date: 2013-12-01
// Script-Type: hook
// Hook: AfterTypeset

/*
```

```
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

outputRE = new RegExp("^Output written on ([^(]*)[.](dvi|pdf) \\(");

function getBasePath(path) {
    var i = Math.max(path.lastIndexOf('/'),
                     path.lastIndexOf('\\'));
    return (i == -1) ? path : path.slice(0, i+1);
}

// get the text from the standard console output
txt = TW.target.consoleOutput;
lines = txt.split('\n');

filename = "Not found";
done = 0;

for (i = lines.length - 1; i >= 0; --i) {
    line = lines[i];

    matched = outputRE.exec(line);
    if (matched) {
        filename = getBasePath(TW.target.rootFileName) +
                   matched[1] + ".pdf";
        windows = TW.app.getOpenWindows();

        for (j = 0; j < windows.length; j++) {
            win = windows[j];
```

```
            if (win.fileName == filename) {
                win.reload();
                done = 1;
            }
        }
        if (!done)
            TW.app.openFileFromScript(filename, TW);
        break;
    }
}

if (done)
  TW.result = "Attempted to reload " + filename;
else
  TW.result = "Attempted to load " + filename;
```

C Using patchDVI with WinEdt

WinEdt is a Windows editor with TEX support. The configuration options have changed a number of times; I do not know how to implement these instructions in the latest version. These instructions apply to version 5.5, and assume you are using it with MikTeX.

1. In Options – Execution Modes choose Texify, and click on Browse for Executable. Find the `Rscript` executable in your R installation, directory `bin/i386` or `bin/x64`, and choose it. In the Switches line, put

   ```
   -e
   ```

   and in the Parameters line, put

   ```
   "patchDVI::SweaveMiktex('%n%t', '%N.tex')"
   ```

   The quotes are necessary!
2. Do the same for the PDF Texify command, replacing `SweaveMiktex` with `SweavePDFMiktex`.
3. In Options – Execution modes, make sure Start Viewer and Forward Search are selected for LaTeX and PDF LaTeX.

When you preview a file in `yap`, double clicking should jump back to the editor. If it doesn't (or it opens the wrong editor), while you're

in `yap` choose View – Options – Inverse DVI search. You should see "WinEdt (auto-detected)" as an option; if so, select it. If not, create a new entry for WinEdt, and for the command line, put in

```
"path\to\winedt.exe" "[Open(|%f|);SelPar(%l,8)]"
```

after editing the path as necessary.