How to organize an interactive document

Martin Karrenbach [1]

/usr/local/src/our/word/tutorials/idocorg

ABSTRACT

An interactive document is a self contained unit; it can either exist by itself or as part of a report or book. Everything necessary to recreate your paper must be within the paper directory. This document outlines the organization of the paper directory. This document follows the same organizational rules; you can view it, by going to `our/word/tutorials/idocorg` and typing `"cake"` . To rebuild all figures type `cake figures`.

PAPERS AND REPORTS

The report directory `..../sep80` consists of the directory `Adm` and paper directories. **Each paper lives in its own directory!** The `Adm` directory contains the report file and administrative files, and front and back matter. Each author is responsible for his own paper directory. The next section describes the structure of a paper directory and is suggested for all authors. At SEP we check the compliance with our minimum set of requirements and authors who dare to be different, contribute "voluntarily" to the *Friday Afternoon Fund*.

ORGANIZATION

Each paper directory must at its top level contain at least the following:

a) A file called `paper.tex` (which is your TEXfile).

b) A `Cakefile` or cakefile that knows how to generate the figures, and how to clean the entire paper directory. In order words you need at least in that Cakefile:

[1] **email:** jon@sep.stanford.edu

  - "clean &", a clean rule
  - "#define FIGDIR ...", a where the Figure directory is.
  - "#define FIGLIST ...", a list of all the rebuildable figures in the paper.

c) A subdirectory `Fig`, that contains vplot, PostScript figures, and movie files.

You might only have the above bare minimum in your paper directory if you write a theoretical paper where all your figures are manually drawn.

The following subdirectories are recommended for larger papers:

d) Subdirectories for each complex figure or group of related figures. Those directories might contain local source code necessary for those figures only. It also can contain parameter files or files necessary for interactivity.

e) A subdirectory `Src`, that contains all paperwide program source code.

f) A subdirectory `Data`, that contains original data files.

g) A subdirectory `Bin`, which will contain the executables that are produced by the `Cakefile` or `cakefile` when regenerating the figures. (Make sure `cake clean` removes its contents completely, otherwise your paper might get stuck when running from CDROM.)

h) A subdirectory `Lib`, that contains libraries you may need to build when regenerating the figures. (make sure `cake clean` removes its contents completely)

FIGURE TYPES

Figures can have different properties, such as being rebuildable, non rebuildable, conditionally rebuildable, interactive and non interactive. Those properties naturally separate into reproducibility and interactivity issues. There is no overlap of those two domains.

> Figure naming conventions always relate to reproducibility and not to interactivity.

Definitions of Reproducibility

- **rebuildable**: a figure can be rebuilt from scratch in a short amount of time. Such figures are created from data programs or parameter files. The figure name has the extension `.ps` or `.v`.
  The figure is flagged **[R]** after the figure caption.

- **conditionally rebuildable**: a figure can be rebuilt, but it needs a large amount of time or some special hardware or proprietary software that is not contained in our distribution. Figures created using super computers or special graphics are of this type. There should be a text file with the name `figname.warning` in the figure directory explaining why the figure is not rebuildable. The figure name has the extension `.ps` and `.v`
  If there is a file with the name `figname.warning` in the figure directory, then the figure is flagged [**CR**] after the figure caption.

- **non rebuildable**: a figure cannot be recreated. An example is a picture drawn by hand or a scanned photo. The figure name has the extension `.ps.save` and `.v.save` . (This prevents them from getting destroyed in the clean up process, where all rebuildable `.ps` and `.v` files are removed.)
  The figure is flagged [**NR**] after the figure caption.

Testing reproducibility

The word reproducibility implies that figures can be destroyed and automatically rebuilt. To test the authors claim of reproducible figures one has to destroy existing figures. We use `cake` to accomplish that test. `Cake` knows about different figure types via their filenames. The command `cake burnRps` will remove all PostScript files `figname.ps` that have no warning file attached. So it only removes figures which are flagged as [**R**] in the document. The command `cake burnCRps` will remove all PostScript files `figname.ps` that have a file `figname.warning` in the figure directory. This command should only be used if the necessary requirements written in `figname.warning` are met. All figures `figname.ps.save` are not removed.

The removed figures can be rebuilt by `cake figures`.

The following three figures are of the above categories. All figures in this paper are included in the LaTeX document with the line:
`\activesideplot{figurename}{width=3.in}{cakedir}{ This figure is ...}`

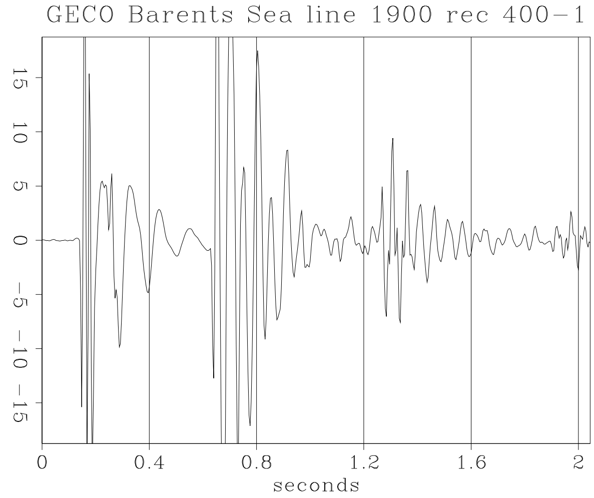

Figure 1: This figure is rebuildable and its name is `reprod.ps` . Note the flag **[R]** at the end of this caption. reprod [ER]

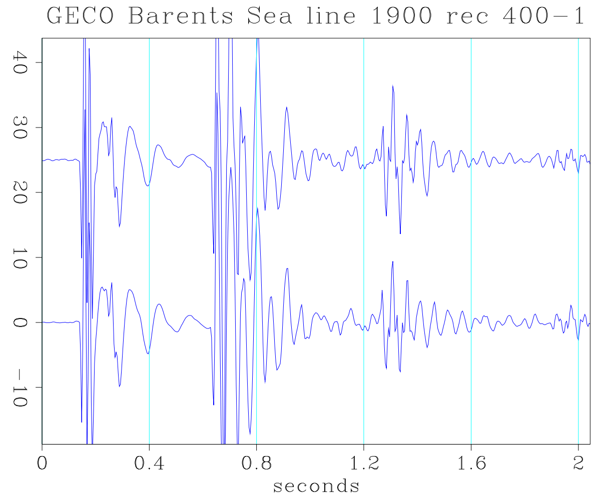

Figure 2: This figure is conditionally rebuildable and its name is `pseudo.ps`. There exists also a file called `pseudo.warning` in the directory where the figure resides, explaining why it is not immediately rebuildable. In case the file pseudo.warning exists in the figure directory, the figure is flagged **[CR]**, meaning it is conditionally reproducible. pseudo [CR]

To each of those figures I could have attached some interaction, but I save this for the next section which deals exclusively with interactions.

Definition of Interactivity

- **interactive**: the author has provided some mechanism to let reader explore the figure. The simplest interaction is a movie. Or the reader can modify parameters or source code and recompute figures.
- **non interactive**: no interaction is provided except the ability to view the existing picture in different sizes.

Figure 3: This figure is non-rebuildable and its name is **nonrep.ps.save** . Note the flag **[NR]** at the end of the caption. nonrep [NR]

Interactivity is handled exclusively through an interaction mechanism. By default it is the **Cakefile**. A figure is designated interactive if the author provides an **%.action** rule in the Cakefile or if certain files exist in the paper directory. Even without knowing anything about the Interaction Mechanism, it is very easy to make figures interactive by merely placing some file into your "cake directory", which is by default the top level of your paper directory. That can be changed by specifying a "cake directory" explicitly in the plot command.

Let's explore some easy ways to make figures interactive.

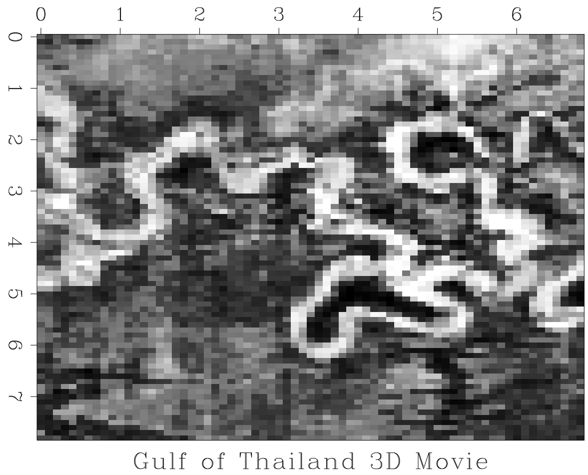

Figure 4: This figure shows a vplot movie and its name is **vpmovie.ps**. In the figure directory there exists a file called **vpmovie.v3** which is a concatenation of vplot files. You can flip through the movie frames with the tube program. vpmovie [ER]

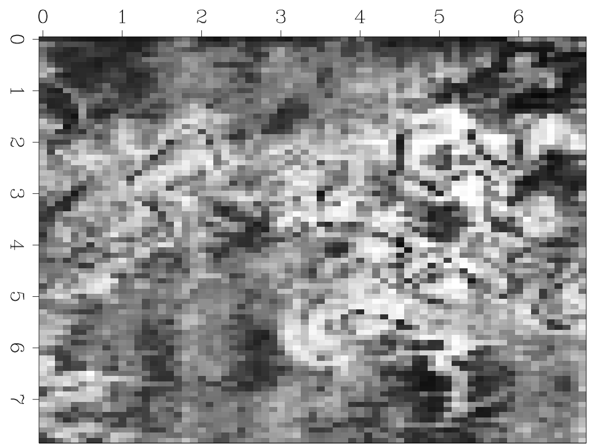

Figure 5: This interactive figure shows a raster movie of animated bytes and its name is `Amovie.ps`. In the figure directory there exists a file called `Amovie.A` which is a cube of bytes. You can explore the data cube with the `X11movie` program, which allows you to transpose on the fly. Amovie [ER]

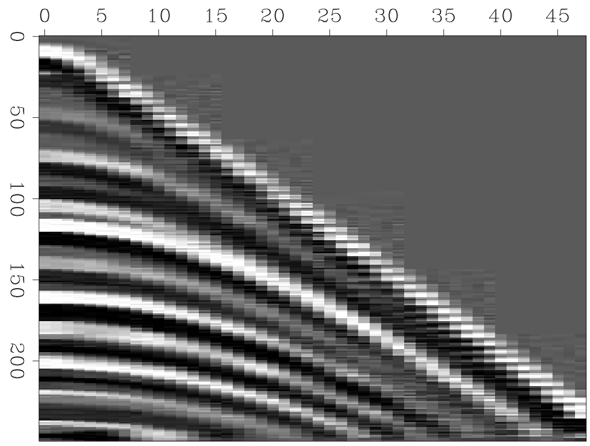

Figure 6: This interactive figure uses a graphical interface and its name is `process.ps`. In the directory `process` there exists a file called `process.panel` which is an `xtpanel` script file giving a graphical user interface for interaction. You can change the parameter settings and rebuild the figure. process [ER]

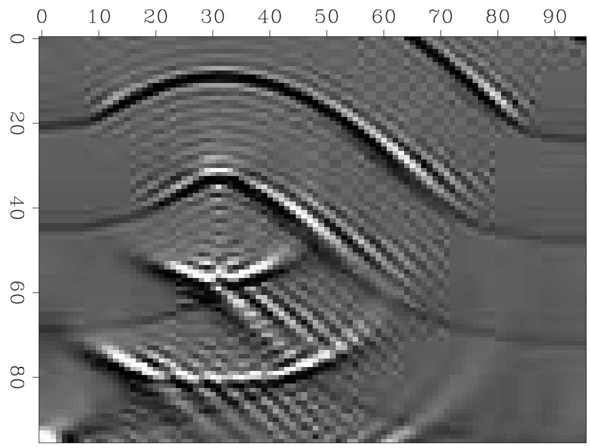

Figure 7: This interactive figure allows changing of parameters and the source code and its name is `wave.ps`. In the directory `wave` there exists a file called `wave.p` which is a text file containing possible parameter settings. There is also the Ratfor source code `wave.r`. You can edit the source, change parameters and rebuild the figure. wave [ER]

Figures 4, 5, 6 and 7 use default interaction rules provided for you. Those rules cover many applications. However, you can override them as shown in Figure 8.

The next figure demonstrates that the figure in the text and its associated action can be totally unrelated. The underlying action in this figure is to demonstrate the use of "`xtpanel`". `xtpanel` is an easy to use and very flexible graphical user interface for `UNIX` tasks . It is described in more detail in (Cole and Nichols, 1992) and all interactions in this electronic report are based on it.

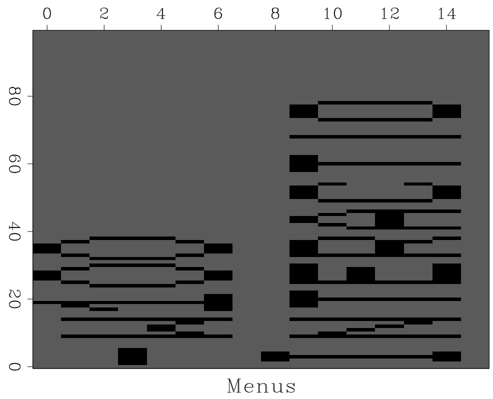

Figure 8: This interactive figure has a particular action specified and its name is `funpanel.ps`. In the Cakefile the target `funpanel.action` brings up example files of a graphical user interface for interaction. You can explore many possibilities for menus and buttons. funpanel [ER]

The suffix `.action` in your `Cakefile` determines what happens, when the reader presses the middle mouse button on an activeplot label.

See the `Cakefile` listing for more details.

COPYING DATA FILES INTO THE PAPER DIRECTORY

All your original data files should reside in the subdirectory `Data`. When you access that data files, you should use only relative pathnames.

If all data sets which are used in your paper are located within your paper directory, then your paper is totally rebuildable even if you move the entire paper directory to some other place.

To copy SEPlib data files to your `Data` directory, you have two choices.

Copy the data regularly without header and data concatenation. To copy the header file:
`cp /q2/data/cmp/wz00.H /wrk2/sep77/mypaper/Data/wz00.H`
and to copy the data file
`cp /q2/data/cmp/wz00.D /wrk2/sep77/mypaper/Data/wz00.D` After copying you have to edit the header file mypaper/Data/wz00.H and add the correct line `in="some-relative-path/wz00.D"`, where the path has to be relative within your paper directory. Make sure that works. Test it.

Globally used data (a practical issue)

Some data sets are used by many people and live already in public places like $Q2 or $CLASSDATA. If you are using big data sets from those official places, we can save a lot of space on the CD if we keep them in a central location. You can use it from that location without copying it to individual paper directories. Two examples of usage are "<$Q2/cmp/wz00.H Myprogram.x >junk.H" ,
or "<$CLASSDATA/cmp/wz00.H Myprogram.x >junk2.H" .

The environment variables $Q2 and $CLASSDATA will be set by the CD setup script.

All you have to do is to enter the file name in a file called DATA_REQUIRED in the administration directory Adm. All Data files listed in there will be copied on to the CD.

There is one drawback to this method. If you move a paper directory to another system you will not have everything to reproduce the entire paper, unless you also copy the necessary data files, now located in the some central place.

HOW TO MOVE AN ENTIRE PAPER DIRECTORY

Do not use absolute pathnames in your paper directory; always make relative references to files within your paper directory.

Ideally, if you followed all guidelines so far, you should have a paper directory that can be moved in its entirety without breaking anything. In that case it takes a mere `cp -r` of your paper directory `paper1` in order to include it in a report or book as `username1`, assuming that `username1` does not already exist.

```
cp -rp /myhome/paper1 /report_directory/username1
```

After copying go ahead and test if your `Cakefile` is able to generate all figures and make sure the actual **data** files are **within your paper directory** and not on some scratch directory. (You will not have to worry about that if all your file references (especially data files) are relative. Go to your paper directory and type `cake burn`. That will remove all rebuildable figures. Thereafter you can test if all figures rebuild ok, by typing `cake figures`.

Try to make your figures interactive (bring up movies, etc.). By default, when you click a button on any figure, a menu will come up that lets you choose some action.

REFERENCES

Claerbout, J. F., and Karrenbach, M., 1993, How to use cake with interactive documents: /usr/local/lib/cake/idoc.doc.

Cole, S., and Nichols, D., 1992, Xtpanel: An interactive panel builder: SEP–**75**, 497–520.

Knuth, D. E., 1984, The TEX book: Addison-Wesley.

Lamport, L., 1986, LATEX User's guide and reference manual: Addison-Wesley.

APPENDIX: CAKEFILE OF THIS DOCUMENT

```
#define FIGDIR ../idocorg/Fig
#define BINDIR /usr/local/bin
#include <SEP.defs>
$                            defining the list of figures
#define FIGLIST reprod pseudo vpmovie Amovie process wave funpanel

default&: read

install-to-read&: paper.dvi /usr/local/bin/IDOCORG
        cp paper.dvi ../Dvi/IDOCORG.dvi

$                                        install reading shell
BINDIR/IDOCORG IDOCORG&: IDOCORG.csh
        cp IDOCORG.csh BINDIR/IDOCORG
        chmod +x BINDIR/IDOCORG

$                                                Rules for making figures
FIGDIR/reprod.v: Data/Hbarents.HH
        Window n1=512 f2=47 n2=1 n3=1 <Data/Hbarents.HH >trace.H
        echo " d1=.004 d2=25. d3=0. o1=0. o2=0. o3=0." >>trace.H
        Wiggle < trace.H out=FIGDIR/reprod.v >/dev/null

FIGDIR/pseudo.v: Data/Hbarents.HH
        Window n1=512 f2=47 n2=1       <Data/Hbarents.HH >section.H
        echo " d1=.004 d2=25. d3=25. o1=0. o2=0. o3=0." >>section.H
        Wiggle < section.H out=FIGDIR/pseudo.v >/dev/null

FIGDIR/vpmovie.v: Data/thai.HH
        Window f3=15 n3=1 <Data/thai.HH >junk.H
        Ta2vplot <junk.H            out=FIGDIR/vpmovie.v  >/dev/null
        Ta2vplot <junk.H            out=FIGDIR/vpmovie.v3 >/dev/null

FIGDIR/Amovie.v: Data/thai.HH
        Window f3=2 n3=30 <Data/thai.HH >fullmovie.H
        Window f3=10 n3=1 <fullmovie.H >junk.H
        Ta2vplot <junk.H                   out=FIGDIR/Amovie.v  >/dev/null
        Cat  fullmovie.H >FIGDIR/Amovie.A

FIGDIR/funpanel.v:  if not exist FIGDIR/funpanel.v
        Banner 100 "Interaction" >junk.H
        Byte <junk.H |Reverse which=1 >junk2.H
        Ta2vplot <junk2.H title="Menus" out=FIGDIR/funpanel.v >/dev/null

FIGDIR/process.%:  if not {{ cd process ; cake -q ../Fig/process.% }}
        ! cd process ; cake ../Fig/process.%

FIGDIR/wave.%: if not {{ cd wave ; cake -q ../Fig/wave.% }}
        ! cd wave ; cake ../Fig/wave.%

$                                                override default actions
funpanel.action&:
        xtpanel-examples

$                                        Binaries live here temporarily
Bin/%:  if not {{ cd Bin ; cake -q % }}
```

```
        ! cd Bin; cake %

$                                       Libraries live here  temporarily
Lib/%:  if not {{ cd Lib ; cake -q % }}
        ! cd Lib ; cake %

$                                       Source code used through out the paper
Src/%:  if not {{ cd Bin ; cake -q % }}
        ! cd Src ; cake %

$                           make sure all directories are cleaned
clean&: texclean jclean
        ! cd Data ; cake clean
        ! cd Lib ; cake clean
        ! cd Src ; cake clean
        ! cd Bin ; cake clean
        ! cd process ; cake clean
        ! cd wave ; cake clean

#include <SEP.prog.rules>
#include <SEP.obj.rules>
#include <SEP.idoc.rules>
```