`demo.box` – zur Kommentierung und Ausführung von R-Anweisungen

Peter Wolf, File: cutandshowchunk.rev
in: /home/pwolf/R/test

June 4, 2013

Contents

1 Was kann das Demo-Box-Werkzeug?

Für Präsentationen gibt es Wünsche an die Technik, die sich nach ihrem Zweck unterscheiden. Folgende Settings sind plausibel:

- klassische Folien zeigen: PowerPoint oder LaTeX in Verbindung mit der `beamer`-Klasse bieten sich hierfür an.
- einfache R-Anweisungen vorführen: Dies geht mit einer einfachen R-Console.
- vorbereiteter R-Anweisungssequenzen starten: `ConstructDemoFunction()` und `chunk.player()` aus dem `relax`-Paket leisten hierzu gute Dienste.
- Einflüsse von Verfahrensparametern demonstrieren: Solche interaktiven Beispiele lassen sich gut mit den Funktionen `slider()` und `args.player()` aus der `relax`-Welt umsetzen.

Damit stehen reine Präsentationsbedürfnisse wie auch reine Experimentierumgebungen bereit. Es fehlen jedoch noch Werkzeuge, die kleine Präsentationen und R-Demonstrationen vereinigen. Mit dieser Frage beschäftigt sich dieses Papier, das Idee und Implementierung von *Demo-Boxen* präsentiert.

2 Demo-Box aus Präsentationssicht

Im Folgenden wird eine *Demo-Box* als ein Werkzeug bezeichnet, das einen kurzen Text zeigen kann, gleichzeitig aber auch R-Sequenzen zur Aktivierung anbietet. Wir wollen uns zunächst ein solches Werkzeug aus Sicht einer Präsentation vorstellen. Für den Einsatz muss ein Zusammenhang textlich oder graphisch dargestellt werden, und es muss die Möglichkeit geben, zugehörige R-Anweisungen zu starten. Der Start fest vorgegebener Anweisungssequenzen mit festen Ergebnissen ist unspannend und verlangt keine weiteren Auseinandersetzungen. Interessant wird es erst, wenn man während einer Präsentation den Code modifizieren und ausführen kann. Also benötigt man ein Feld mit R-Anweisungen, das zur Bearbeitung freigegeben ist.

Notwendige Elemente: In der einfachsten Form besteht die Oberfläche einer Demo-Box drei Teilen:

- einen Teil zur Textdarstellung,
- einen Teil zur Code-Darstellung und
- einen Teil zur Steuerung mit einem Aktivierungskopf.

Einsatz des Werkzeugs: Für den Präsentator sollte der Einsatz einer Demo-Box möglichst bequem sein. Es lässt sich vorstellen, dass sich aus R heraus ein R-Objekt laden lässt und dann die Box automatisch geöffnet wird. Eine zweite Möglichkeit besteht darin, Demo-Box-Management und Box-Inhalte zu trennen.

In der einfachsten Form können wir uns vorstellen, dass eine Demo-Box nur ein einziges Beispiel zeigt. Dagegen könnten aber auch mehrere Beispiele in einer Sammlung zusammengebunden sein. Dann müsste der Anwender

zwischen verschiedenen Inhalten wählen können, wie wir es vom `chunk.player()` her kennen.

Parameter: Zentral muss der Zugriff auf die gewünschten Demonstrationen fixiert werden. Weitere Parameter könnten das Layout der Demo-Box (wie die Größe des Fensters) oder eine geeignete Überschrift definieren.

Offene Fragen: Zu Beginn der Entwicklung sind verschiedene Fragen noch offen.

- Müssen im R-Teil alle notwendigen Dinge enthalten sein? Oder sollte es Initialisierungen geben, die eventuell unsichtbar sind und dadurch nicht vom Präsentationszweck ablenken?
- Sollte der R-Teil andere R-Chunks verwenden dürfen, wie wir es von `relax` her kennen?
- Sollte man die Evaluierungen in einer eigenen Umgebung vornehmen?
- Sollten Code oder Kommentar ausgeblendet werden können?

Mit einer Entscheidung über zusätzliche Möglichkeiten ergeben sich Umsetzungsfragen, die sehr sorgfältig in ihrer Syntax überlegt werden müssen.

3 Entwurfsentscheidungen für Material und Mechanismus

Das Material wird auch aus Portabilitätsgründen in Form von Dateien gespeichert werden müssen. Zur Präsentation müssen diese gelesen und auswertet werden. In der kompaktesten Form besteht eine Demo-Box aus einer einzigen Datei mit den (formatierten) Textinhalten und R-Anweisungen. Für die Texte ist denkbar, dass diese in Graphiken überführt und als Integer gespeichert werden. Der Mechanismus selbst kann als R-Funktion umgesetzt sein, die die Bild- und Anweisungsinfos enthält und die Demo-Box konstruiert.

Für Übersicht und Flexibilität ist es jedoch vorteilhaft, die Informationen auf verschiedene Dateien aufzuteilen und in einem Verzeichnis abzulegen. Dateien mit folgenden Inhalten sind plausibel:

- Demo-Box-Player
- R-Anweisungen
- Graphiken mit den Kommentartexten
- Konfigurationen
- Hilfetexte

Es bietet sich an, diese Dateien in einem eigenen Unterverzeichnis zusammenzufassen. Damit jederzeit eine Nutzung der Demo-Boxen möglich ist, erscheint es zweckmäßig, in ein solches Verzeichnis eine Readme-Datei und eine Demo-Box-Abspielfunktion aufnehmen.

Aus Sicht des Anwenders ist nicht viel zu tun:

- Der Anwender kann die Readme-Datei lesen, in der erklärt wird, dass die

- Demo-Box-Funktion eingelesen und gestartet werden muss,
- die den Anwender nach Namen / Ort der gewünschten Demo fragen wird.
- Dannach kann er sich der Bedienung widmen.

Aus Entwicklungssicht muss die Funktion zur Aktivierung der Demo-Boxen entwickelt werden, weiter muss aber auch ein Mechanismus her, um das Material von Demo-Boxen geordnet in Dateien abzulegen.

4 Die Erstellung von Demo-Boxen

Für die Erstellung sollten verschiedene Situationen berücksichtigt werden.

- Der Ersteller arbeitet im `relax`-Editor und möchte einen speziellen oder mehrere Chunks des Dokumentes mit zugehörigen Texten zu einer Demo-Box zusammenbinden. Vorstellbar ist, dass das aktuelle Dokument einen Satz Folien beschreibt.
- Der bzw. die zu verwendenden Text-Chunks und zugehörige Code-Chunks befinden sich in einer `rev`-Datei.
- In einer Datei befindet sich eine einzige Anweisungssequenz, die Verwendung finden soll.
- Ein Bild liegt vor wie auch eine Anweisungssequenz. Aus diesen beiden Teilen soll eine Demo-Box konstruiert werden.

Solange noch keine Klarheit über Detailfragen herrscht, versuchen wir Funktionen zu entwerfen, die elementare Teilaufgaben erledigen.

- Extraktion von Text und Code aus einer Relax-Anwendung oder aus einer rev-Datei.
- Konstruktion eines LaTeX-Textes aus einem Text-Schnipsel.
- Formatierung einer LaTeX-Datei und Überführung des Formatierungsergebnisses in ein Bild. Zur Unterstützung bieten sich externe Programme wie pdflatex und convert an.
- Konstruktion einer R-Code-Datei.
- Erstellung einer Konfigurationsdatei.
- Definition einer Player-Funktion.

4.1 Extraktion von Text

Die Funktion `extract.text.and.code()` hat die Aufgabe Texte und Code-Sequenzen aus einem einem File oder einem relax-Fenster zu extrahieren. Zwar sind Operationen wie *hole einen einzelnen Text* noch elementarer, jedoch müssen so Initialisierungen nur einmal ablaufen.

1 ⟨*definiere* `extract.text.and.code` 1⟩ ≡ ⊂ 12

```
extract.text.and.code <- function(SrcFile="",
                                  SrcFilePath="",
                                  CodeNo=1){
  # get R rev file ------------------------------------------------
  if(""==SrcFile){ # assuming relax is running
    if(!exists("revive.env")) return("Warning: no text defined!?") else
```

```
    revive.sys <- get("revive.sys",env=revive.env)
  tw <- get("tworkwin", revive.sys)
  text.of.demo <- tclvalue(tkget(tw,"0.0","end"))
  text.of.demo <- strsplit(text.of.demo,"\n")[[1]]
} else { # get source from file
  if("" != SrcFilePath) SrcFile <- file.path(SrcFilePath,SrcFile)
  if(!file.exists(SrcFile)) SrcFile <- paste(SrcFile,".rev",sep="")
  if(!file.exists(SrcFile)) return("Warning: source file not found!?")
  text.of.demo <- base::scan(file=SrcFile,what="",sep="\n")
}  # cat("get SrcFile"); print(text.of.demo)

# find begin lines of code chunks ------------------------------
idx.code.begin <- grep("^[<][<].*[>][>][=]",text.of.demo)
if(0 == length(idx.code.begin)){
  # 'no code begin' => file is a file of code
  text.of.demo <- c(paste("<","<*>",">=",sep=""), text.of.demo)
  idx.code.begin <- 1
}
lines <- idx.code.begin[CodeNo]
CodeNo <- CodeNo[h <- !is.na(lines)]; lines <- lines[h]
if(0 == length(lines)) return("Warning: code not found")

# loop over code chunks that have been choosen -----------------
res.code <- res.text <- NULL
for(idx in seq(along=CodeNo)){
  line <- lines[idx]

  # extract code from source ..................................
  code <- if(1<line) text.of.demo[-(1:(line-1))] else text.of.demo
  end <- grep("^@",code)
  if(0 < length(end)) code <- code[1:(end[1]-1)]
  code <- list(code); names(code) <- as.character(CodeNo[idx])
  res.code <- c(res.code,code)

  # extract text from source ..................................
  demo.text <- text.of.demo[ 1:(line-1) ]
  begin <- grep("^@",demo.text)
  if(0 < length(begin)) demo.text <- demo.text[-(1:max(begin))]
  begin <- grep("^[\\]begin[{]document[}]",demo.text)
  if(0 < length(begin)) demo.text <- demo.text[-(1:begin[1])]
  demo.text <- list(demo.text)
  names(demo.text) <- as.character(CodeNo[idx])
  res.text <- c(res.text,demo.text)
}

# pack and return result ---------------------------------------
return(list(text=res.text, code=res.code))
}
# ex.res <- extract.text.and.code(CodeNo=c(5,9,10))
```

4.2 Präambel-Ergänzung und Größenfestlegung

Nachdem Texte und Code-Sequenzen extrahiert sind, müssen wir die Textschipsel zu TeX-Dokumenten ausbauen. Dafür entwerfen wir eine einfache Funktion, die an einen Textvektor die notwendigen Modifikationen vornimmt. Zu den Modifikationen gehört die Ergänzung einer Präambel, in der wesentlich die Größe des Textfeldes festgelegt werden muss.

2 ⟨*definiere* `construct.latex.text` 2⟩ ≡ ⊂ 12

```
construct.latex.text <- function(demo.text,PraeambelCmds="",
                                 TextBeginCmds="",
                                 SlideWidthFactor=2,
                                 SlideHeightFactor=1,
                                 DemoNumber=1){
  # find latex paper sizes ----------------------------------------
  paperheight <- round(2*SlideHeightFactor*6.5,1) # paperheight layout multiplied by 2
  paperwidth  <- round(SlideWidthFactor* 6.5,1)

  # append praeambel ----------------------------------------------
  praeambel <- c( "\\documentclass{article}",
                  "\\usepackage[pass]{geometry}",
                  paste(sep="","\\paperheight=",paperheight,"cm"),
                  paste(sep="","\\paperwidth=", paperwidth ,"cm"),
                  "\\oddsidemargin=1cm\\topmargin=0cm",
                  "\\addtolength{\\oddsidemargin}{-1in}",
                  "\\addtolength{\\topmargin}{-1in}",
                  "\\setlength{\\textheight}{\\paperheight}",
                  "\\addtolength{\\textheight}{-\\topmargin}",
                  "\\addtolength{\\textheight}{-\\headsep}",
                  "\\addtolength{\\textheight}{-\\headheight}",
                  "\\setlength{\\textwidth}{\\paperwidth}",
                  "\\addtolength{\\textwidth}{-\\oddsidemargin}",
                  "\\addtolength{\\textwidth}{-3cm}",
                  "\\parindent=0mm",
                  paste("\\newcommand{\\DemoNumber}{",DemoNumber,"}",sep=""),
                  paste("\\newcounter{DemoNumber}\\setcounter{DemoNumber}{",
                        DemoNumber,"}",sep=""),
                  paste("\\newcommand{\\DemoSection}[1]{",
                        "\\setcounter{section}{\\value{DemoNumber}}",
                        "\\addtocounter{section}{-1}","\\section{#1}}",
                        sep=""),
                  PraeambelCmds,
                  "\\begin{document}\\raggedright",
                  TextBeginCmds,
                  ""
                )

  # construct text and remove some comment lines ------------------
  demo.text <- c(praeambel, demo.text, "\\end{document}")
  demo.text <- demo.text[ "%" != substring(demo.text,1,1) ]
}
# construct.latex.text(ex.res$text[[3]])
```

4.3 Die Umwandlung der Texte

Für den Produktionsprozess sind mehrere Punkte zu erledigen:

- ggf. Erstellung eines Verzeichnisses
- Speicherung der LaTeX-Texte als temporäre Dateien
- Formatierung der LaTeX-Dateien
- Überführung der formatierten Texte in Bilder.

3 ⟨*definiere* `construct.pic.files` 3⟩ ≡ ⊂ 12

```
construct.pic.files <-
  function(DemoBoxName="dbox",
```

```
            DemoBoxPath=".",
            ex.res=ex.res, # text and code to find start chunk
            GraphicsType="gif",
            FontSizeFactor=1,
            SlideWidthFactor=2,
            SlideHeightFactor=1,
            PraeambelCmds="",
            TextBeginCmds="",
            pdflatex="", latex="",
            convert="", gs=""){
  # create dir if not found ----------------------------------------
  dir <- file.path(DemoBoxPath,DemoBoxName)
  if(!file.exists(dir)) dir.create(dir)

  # construct tmp files names --------------------------------------
  tmp.name <- paste("tmp-",DemoBoxName,sep="")

  # check if linux version is working ------------------------------
  # conditions: (pdf)latex and (convert or gs):
  if(linux <- (0 < length(grep("linux", version))) ||
               (0 < length(grep("darwin", version)))) {
    ⟨find LaTeX-Formatter and convert or ghostscript 4⟩
    if(!pdflatex.found && !latex.found){
      print("Warning: pdflatex or latex needed"); return()
    }
    if(!convert.found && !gs.found){
      print("Warning: convert or ghostscript needed"); return()
    }
  } else { print("Warning: linux system needed"); return() }
  # loop along the demos -------------------------------------------
  demo.text <- ex.res$text; code <- ex.res$code
  if(!is.list(demo.text)) demo.text <- list(demo.text)
  idx <- 0
  for(element in seq(along=demo.text)){ # 130215 start chunk
    is.not.start <- 0 == length(grep("^[<][<]start[>][>][=]",code[[element]][1]))
    if(is.not.start){ # 130219
      idx <- idx + 1
      ⟨konstruiere idx-te Text-Graphik aus Text-Element element 5⟩
    } else next
  }
}
# construct.pic.files(demo.text=ex.res$text,GraphicsType="gif")
```

4 ⟨find LaTeX-Formatter and convert or ghostscript 4⟩ ≡ ⊂ 3

```
  pdflatex.found <- ""!=pdflatex; latex.found <- ""!=latex
  if( !pdflatex.found && !latex.found ){ # 1.search:
    pdflatex.found <- 0 == system("which pdflatex")
    if(pdflatex.found){
      pdflatex <- system("which pdflatex",TRUE)
    } else { # 2.search:
      cat("looking for pdflatex, please wait")
      pdflatex.found <- 0 == system('find /usr | grep "/pdflatex$" ')
      if(pdflatex.found){
        pdflatex <- system('find /usr | grep "/pdflatex$" ',TRUE)[1]
      } else { # 3.search:
        latex.found <- 0 == system("which latex")
        if(latex.found){
          latex <- system("which latex",TRUE)
        } else { # 4.search:
          cat("looking for latex, please wait")
```

```
        latex.found <- 0 == system('find /usr | grep "/latex$" ')
        if(latex.found){
          latex <- system('find /usr | grep "/latex$" ',TRUE)[1]
        } else {
          cat("Warning: (pdf)latex not found, use argument 'pdflatex' or format by hand!")
        }
      }
    }
  }
 }
gs <- convert <- ""
convert.found <- ""!=convert; gs.found <- ""!=gs
 if( !convert.found && !gs.found ){ # 1.search:
   convert.found <- 0 == system("which convert")
   if(convert.found){
     convert <- system("which convert",TRUE)
   } else { # 2.search:
     cat("looking for convert, please wait")
     convert.found <- 0 == system('find /usr | grep "/convert$" ')
     if(convert.found){
       convert <- system('find /usr | grep "/convert$" ',TRUE)[1]
     } else { # 3.search:
       gs.found <- 0 == system("which gs")
       if(gs.found){
         gs <- system("which gs",TRUE)
       } else { # 4.search:
         cat("looking for gs, please wait")
         gs.found <- 0 == system('find /usr | grep "/gs$" ')
         if(gs.found){
           gs <- system('find /usr | grep "/gs$" ',TRUE)[1]
         } else {
           cat("Warning: 'convert' not found, please convert pdf to gif or ppm files by hand!"
         }
       }
     }
   }
 }
```

5 ⟨*konstruiere* idx*-te Text-Graphik aus Text-Element* element 5⟩ ≡ ⊂ 3

```
# prepare and save latex files ..............................
tex <- construct.latex.text(demo.text[[element]],
                            PraeambelCmds=PraeambelCmds,
                            TextBeginCmds=TextBeginCmds,
                            SlideWidthFactor=SlideWidthFactor,
                            SlideHeightFactor=SlideHeightFactor,
                            DemoNumber=idx)
f.name <- paste(tmp.name,"-",idx,".tex",sep="")
pdf.name <- sub(".tex$",".pdf",f.name)
base::cat(tex,sep="\n",file=file.path(dir,f.name))

# format latex files by pdflatex ............................
if(pdflatex.found)
  system(paste("cd ",dir,"; echo q | ",pdflatex," ",f.name))
else {
  if(latex.found)
    system(paste("cd ",dir,"; echo q | ",latex," ",f.name,";",
                 sub("latex","dvipdf",latex)," ",sub(".tex$","",f.name)))
}
```

```
  # convert formated to ppm or gif graphics files ..............
  # not by pdftoppm / pdftogif
  pic.name <- sub(".pdf$",paste(".",GraphicsType,sep=""),pdf.name)
  if(file.exists(file.path(dir,pdf.name))){
    if( convert.found & (GraphicsType=="ppm" || GraphicsType=="gif")){
      ⟨convert pdf to pic.name by convert 6⟩
    } else {
      if( gs.found & GraphicsType=="ppm"){
        ⟨convert pdf to pic.name by ghostscript 7⟩
      }
    }
    if(!file.exists(file.path(dir,pic.name)))
      print(paste("please convert",pdf.name,"in directory",dir,
            "to",pic.name,"by hand"))
  }
```

6 ⟨*convert pdf to* pic.name *by convert* 6⟩ ≡ ⊂ 5

```
  pic.name <- sub("^tmp-","",pic.name)
  ret.code <- system(paste("cd ",dir,"; ",convert,
                    paste(" -verbose -white-threshold 100% -density ",
                          round(FontSizeFactor*100),"x",
                          round(FontSizeFactor*100), sep=""),
                    pdf.name, pic.name))
  if(0 == ret.code) cat(paste(pic.name,"generated")) else {
    cat(paste(pic.name,"NOT generated"))
  }
```

Das Konsolenkommando zur Konvertierung von pdf nach ppm lautet beispielsweise:

```
gs -dNOPAUSE -dBATCH -sDEVICE=ppmraw -sOutputFile="test.ppm"
test.pdf
```

Für eine Portierung nach Windows sei erwähnt, dass ghostscript oft an folgender Stelle zu finden ist:

```
gs <- "C:/Programme/gs/gs8.61/bin/gswin32c"
```

7 ⟨*convert pdf to* pic.name *by ghostscript* 7⟩ ≡ ⊂ 5

```
 XRES <- YRES <- round(FontSizeFactor*100)
 ret.code <- system(paste("cd ",dir,"; ",
                    gs," -dNOPAUSE -dBATCH -sDEVICE=ppmraw -r",XRES,"x",YRES,
                       " -sOutputFile=",pic.name," ",pdf.name,sep=""))
 if(0 == ret.code) cat(paste(pic.name,"generated")) else {
   cat(paste(pic.name,"NOT generated"))
 }
```

4.4 Die Abspeicherung der Code-Sequenzen

8 ⟨*definiere* construct.code.file 8⟩ ≡ ⊂ 12

```
 construct.code.file <- function(DemoBoxName="dbox",
                                 DemoBoxPath=".",
                                 demo.code){
   if( 0 == (n <- length(demo.code))) return("Warning: no code chunks found")
   # loop along the chunks to integrate some markers ---------------
   demo.no <- 0
   for(i in 1:n){
     x <- demo.code[[i]]
     is.not.start <- 0 == length(grep("^[<][<]start[>][>][=]",x[1]))
     if(is.not.start) demo.no <- demo.no + 1
```

```
    x[1] <- sub("^[<]([<].*[>])[>][=].*$","##\\1=##",x[1])
    x <- c(paste("#",demo.no*is.not.start,":",sep=""),x,
           paste("#:",demo.no*is.not.start,sep=""))
    demo.code[[i]] <- x
  }
  demo.code <- unlist(demo.code)
  f.name <- file.path(DemoBoxPath,DemoBoxName,
                      paste(DemoBoxName,".R",sep=""))
  base::cat(file=f.name, demo.code, sep="\n"); cat(f.name,"generated")
}
# construct.code.file(demo.code=ex.res$code)
```

4.5 Aufräumen, Konfiguration und Readme

9 ⟨*definiere* clean.demo.box 9⟩ ≡ ⊂ 12

```
clean.demo.box <- function(DemoBoxName="dbox",
                           tmptypes=c("log","aux"),
                           DemoBoxPath="."){
  for(ext in tmptypes){
    f.list <- list.files(path=DemoBoxName,
                         pattern=paste("^tmp-.*",ext,"$",sep=""))
    f.list <- file.path(DemoBoxName,f.list)
    if(0 < length(f.list)){
      file.remove(f.list); print(paste("file",f.list,"removed"))
    }
  }
}
# clean.demo.box()
```

10 ⟨*definiere* construct.config.file 10⟩ ≡ ⊂ 12

```
construct.config.file <- function(DemoBoxName = "dbox",
                                  SlideWidthFactor = 2,
                                  SlideHeightFactor = 1,
                                  CodeHeightFactor = 1,
                                  DemoBoxPath = "."){
  f.name <- file.path(DemoBoxName,paste(DemoBoxName,".config",sep=""))
  base::cat(file = f.name, sep="\n",
            paste("SlideWidthFactor <-", SlideWidthFactor),
            paste("SlideHeightFactor <-",SlideHeightFactor),
            paste("CodeHeightFactor <-", CodeHeightFactor))
}
```

11 ⟨*definiere* construct.Readme.file 11⟩ ≡ ⊂ 12

```
construct.Readme.file <- function(DemoBoxName="dbox",DemoBoxPath="."){
  txt <- c(
   l <- "#=============================================================================================",
   paste("This directory contains the demo box '",DemoBoxName,"' -- created: ",
         date(),".\n",sep=""),
   paste("It should comes the following files:\n\n '",DemoBoxName,".R'\n '",
         DemoBoxName,"-<no>.gif' or '",DemoBoxName,"-<no>.gif' for some numbers <no>\n '",
         DemoBoxName,".config'\n 'DemoBoxPlayer.R'\n 'Readme.txt'\n ",sep=""),
   paste("'",DemoBoxName,".R' consists of the code sequences of the demos of the box.",
         sep=""),
   "Textual comments have been converted to pictures as gif or ppm files.",
   paste("Some configuration infos are stored in '",DemoBoxName,".config'.\n",sep=""),
   "To open this demo.box you have to call the function 'DemoBoxPlayer()' which will",
   "asked you for a demo box directory. The definition of this function and",
```

```
    "a call of it is also found in this directory -- see file: 'DemoBoxPlayer.R'.\n",
    "To start the DemoBoxPlayer 'source' the file 'DemoBoxPlayer.R'.",
    "For example, you can open the demo box by the R statement (after inserting the right path):",
    paste("\n\n  > source('>>>PATH<<</",DemoBoxName,"/DemoBoxPlayer.R)\n",sep=""),
    "\n... and the demo session will begin!",1)
  # if(debug) cat(txt,sep="\n")
  cat(txt,file=file.path(DemoBoxPath,DemoBoxName,"Readme.txt"),sep="\n")
}
# construct.Readme.file()
```

4.6 Die Konstruktion der Konstruktionsfunktion

12 ⟨*definiere* ConstructDemoBox 12⟩ ≡ ⊂ 15, 17

```
ConstructDemoBox <- function(DemoBoxName="1",
                             DemoBoxPath=".",
                             SrcFile="",
                             SrcFilePath="",
                             CodeNo=1,
                             GraphicsType = "gif",
                             FontSizeFactor = 1,
                             SlideWidthFactor = 2,
                             SlideHeightFactor = 1, # for latex layout
                             CodeHeightFactor = 1,
                             PraeambelCmds = "",
                             TextBeginCmds = "",
                             pdflatex="",latex="",
                             convert="",gs="",
                             debug = FALSE){
  DemoBoxName <- paste("R_DemoBox",DemoBoxName,sep="_")
  ⟨definiere extract.text.and.code 1⟩
  ⟨definiere construct.latex.text 2⟩
  ⟨definiere construct.pic.files 3⟩
  ⟨definiere construct.code.file 8⟩
  ⟨definiere clean.demo.box 9⟩
  ⟨definiere construct.config.file 10⟩
  ⟨definiere construct.Readme.file 11⟩
  ⟨definiere dump.DemoBoxPlayer 13⟩
  ex.res <- extract.text.and.code(SrcFile=SrcFile,
                                  SrcFilePath=SrcFilePath,
                                  CodeNo=CodeNo)
  if(debug) cat("extract.text.and.code: done") # ------------------------------

  construct.pic.files(DemoBoxName=DemoBoxName,
                      DemoBoxPath=DemoBoxPath,
                      ex.res=ex.res,
                      GraphicsType=GraphicsType,
                      FontSizeFactor = FontSizeFactor,
                      SlideWidthFactor = SlideWidthFactor,
                      SlideHeightFactor = SlideHeightFactor,
                      PraeambelCmds = PraeambelCmds,
                      TextBeginCmds = TextBeginCmds,
                      pdflatex = pdflatex, latex = latex,
                      convert = convert, gs = gs)
  if(debug) cat("construct.pic.files: done") # --------------------------------

  construct.code.file(DemoBoxName=DemoBoxName,
                      demo.code=ex.res$code,
                      DemoBoxPath=DemoBoxPath)
  if(debug) cat("construct.code.files: done") # -------------------------------
```

```
  construct.config.file(DemoBoxName=DemoBoxName,
                        SlideWidthFactor = SlideWidthFactor,
                        SlideHeightFactor = SlideHeightFactor,
                        CodeHeightFactor = CodeHeightFactor,
                        DemoBoxPath=DemoBoxPath)
  if(debug) cat("construct.config.file: done") # ----------------------------

  construct.Readme.file(DemoBoxName=DemoBoxName,
                        DemoBoxPath=DemoBoxPath)
  if(debug) cat("construct.Readme.file: done") # ----------------------------

  dump.DemoBoxPlayer(DemoBoxName=DemoBoxName,
                     DemoBoxPath=DemoBoxPath,
                     SlideWidthFactor = SlideWidthFactor,
                     SlideHeightFactor = SlideHeightFactor,
                     CodeHeightFactor = CodeHeightFactor)
  if(debug) cat("dump.DemoBoxPlayer: done") # ----------------------------

  if(!debug) clean.demo.box(DemoBoxName=DemoBoxName,
                            tmptypes=c("log","aux","pdf","tex"),
                            DemoBoxPath=DemoBoxPath) # -------------------
}
# ConstructDemoBox(DemoBoxName="dbox",CodeNo=c(5,9,10),GraphicsType = "gif")
```

5 DemoBoxPlayer

13 ⟨*definiere* dump.DemoBoxPlayer 13⟩ ≡ ⊂ 12

```
dump.DemoBoxPlayer <- function(DemoBoxName="dbox",
                               DemoBoxPath=".",
                               SlideWidthFactor = 2,
                               SlideHeightFactor = 1,
                               CodeHeightFactor = 1){
  ⟨definiere DemoBoxPlayer 14⟩
  txt <- deparse(DemoBoxPlayer)
  txt <- c(paste("cat('choose demo box with name >",DemoBoxName,"< !\\n')"),
           "DemoBoxPlayer <-",txt,
           paste("DemoBoxPlayer(",
                 "SlideWidthFactor =", SlideWidthFactor,",",
                 "SlideHeightFactor =", SlideHeightFactor
                 ,",",
                 "CodeHeightFactor =", CodeHeightFactor
                 ,")"))
  cat(txt,file=file.path(DemoBoxPath,DemoBoxName,"DemoBoxPlayer.R"),sep="\n")
}
```

14 ⟨*definiere* DemoBoxPlayer 14⟩ ≡ ⊂ 13, 16, 17

```
DemoBoxPlayer <- function(DemoBoxName,
                          DemoBoxPath=".",
                          main,
                          SlideWidthFactor,
                          SlideHeightFactor=1,
                          CodeHeightFactor=1,
                          start=TRUE,
                          debug=!TRUE){
  require("tcltk"); where<-environment()   # demo head
  ⟨find demo box 18⟩
  ⟨find configuration 19⟩
```

```
  ⟨read code chunks 20⟩
  ⟨prepare size variables 21⟩
  ⟨initialize history 32⟩
  ⟨evaluate start chunk 22⟩
  ⟨initialize secno and define functions for button commands 23⟩
  ⟨open new top level widget 33⟩
  ⟨pack widget for textual explanations / image 34⟩
  ⟨pack control buttons 35⟩
  ⟨construct a lower and upper button for text widget 36⟩
  ⟨construct text widget for code 37⟩
  ⟨save no and set to no 1 38⟩
print(where)
  ⟨construct and implement ShowHistory function 25⟩
}
```

15 ⟨*teste* `ConstructDemoBox` 15⟩ ≡

```
⟨definiere ConstructDemoBox 12⟩
ConstructDemoBox(DemoBoxName="2",CodeNo=39:45,GraphicsType = "gif",
                 PraeambelCmds="\\usepackage[utf8]{inputenc}",
                 FontSizeFactor=1,
                 SlideHeightFactor = 1, debug=!FALSE)
```

16 ⟨*teste* `DemoBoxPlayer` 16⟩ ≡ ⊂ 29

```
⟨definiere DemoBoxPlayer 14⟩
#DemoBoxPlayer()
DemoBoxPlayer(DemoBoxName="R_DemoBox_2")
#DemoBoxPlayer(DemoBoxName="cash_sim","/home/pwolf/projekte/biehler/biehler",debug=TRUE)
```

6 Speichere relevante Funktionen als .R-File

17 ⟨*speichere relevante Funktionen* 17⟩ ≡

```
⟨definiere ConstructDemoBox 12⟩
⟨definiere DemoBoxPlayer 14⟩
dump(c("ConstructDemoBox","DemoBoxPlayer"),file="DB.R")
```

7 Verfeinerungen von `DemoBoxPlayer`

18 ⟨*find demo box* 18⟩ ≡ ⊂ 14

```
  if(missing(DemoBoxName)) DemoBoxName <- tkchooseDirectory(
                             title="Select directory with DemoBox", initialdir=getwd()) else {
     if(0 == length(grep("^R_DemoBox_",DemoBoxName))) # 130215
       DemoBoxName <- paste("R_DemoBox_",DemoBoxName,sep="")
  ##    if(DemoBoxPath != "." && 0 == length(grep("^R_DemoBox_",DemoBoxPath)))
  ##      DemoBoxPath <- file.path(DemoBoxPath, DemoBoxName)
  ##paste("R_DemoBox_",DemoBoxPath,sep="") # 130215
     DemoBoxName <- file.path(DemoBoxPath, DemoBoxName)
  }
  DemoBoxName <- as.character(DemoBoxName)
  if(debug) cat("path:",DemoBoxPath,"-- name:",DemoBoxName,"\n")
  if(0 == length(DemoBoxName))  return("Warning: no demo box file choosen")
  if(!file.exists(DemoBoxName))
    return(paste("Warning: sorry,",DemoBoxName,"not found"))
  # if(0 < grep(".zip$")) unzip(demobox)
  DemoBoxPath <- DemoBoxName#sub(paste("(.*",.Platform$file.sep,").*",sep=""),"\\1",DemoBoxName)
  DemoBoxName <- sub(paste(".*",.Platform$file.sep,sep=""),"",DemoBoxName)
```

```
  if(debug) cat("path:",DemoBoxPath,"-- name:",DemoBoxName,"\n")
  print(paste("DemoBox '",DemoBoxName,"' will be opened",sep=""))
```

19 ⟨*find configuration* 19⟩ ≡ ⊂ 14

```
  cfg <- readLines(file.path(DemoBoxPath,paste(DemoBoxName,".config",sep="")))
  if(missing(SlideWidthFactor))
    SlideWidthFactor <-  as.numeric(sub(".*[-] ","",grep("SlideWidth",cfg,value=TRUE)))
  if(missing(SlideHeightFactor))
    SlideHeightFactor <- as.numeric(sub(".*[-] ","",grep("SlideHeight",cfg,value=TRUE)))
  if(missing(CodeHeightFactor))
    CodeHeightFactor <-  as.numeric(sub(".*[-] ","",grep("CodeHeight",cfg,value=TRUE)))
  if(debug) cat("factors:",SlideWidthFactor, SlideHeightFactor, CodeHeightFactor)
```

20 ⟨*read code chunks* 20⟩ ≡ ⊂ 14

```
 chunks <- scan(file.path(DemoBoxPath,paste(DemoBoxName,".R",sep="")),"",sep="\n")
 if(debug) print(chunks[outer(0:2,grep("^[#][0-9]+[:]",chunks),FUN="+")])
```

21 ⟨*prepare size variables* 21⟩ ≡ ⊂ 14

```
  tw.height <- 17*SlideHeightFactor; cw.height <- 17*CodeHeightFactor
  geo.height <- round(15*(tw.height + cw.height) + 33)
  tw.height <- as.character(tw.height,1); cw.height <- as.character(round(cw.height))
  w.width <- 52*SlideWidthFactor; geo.width <- round(w.width*5)
  w.width <- as.character(round(w.width))
 if(debug) cat("tw.height, geo.height, cw.height, w.width, geo.width:",
               tw.height, geo.height, cw.height, w.width, geo.width,"\n")
```

22 ⟨*evaluate start chunk* 22⟩ ≡ ⊂ 14

```
 if(start==TRUE){
   no<-0
   no.start<-grep(paste("^#",no,":$",sep=""),chunks)
   no.end<-grep(paste("^#:",no,"$",sep=""),chunks)
   if(length(no.end)==0 || is.na(no.end) || length(no.start)==0 || is.na(no.start)||
       is.nan(no.end)||is.nan(no.start)){
       if(debug) cat("# no start chunk to evaluate\n")
   } else {
     code<-chunks[no.start:no.end]
     eval(parse(text=code),envir=where)
     if(debug) cat("start chunk evaluated")
     code<-paste(code,collapse="\n")
     h<-paste(rep("#",60),collapse="")
     code<-sub("#0:",h,code); code<-sub("#:0",h,code)
     allcodechunks<-c(allcodechunks,paste("\n@\n@<","<start@>",">=",sep=""),code,"\n@")
     assign("allcodechunks",allcodechunks,envir=where)
   }
 }
 if(debug) cat("start chunk: done")
```

23 ⟨*initialize* `secno` *and define functions for button commands* 23⟩ ≡ ⊂ 14

```
  secno<-tclVar("1") # 0
  show.next.number<-function(...){ # cat("show.next.number")
   no <- as.character(as.numeric(tclvalue(secno))+1)
   ⟨get chunk and pic 31⟩
  }
  show.back.number<-function(...){ # cat("show.back.number")
   no <- as.character(as.numeric(tclvalue(secno))-1)
```

```
  ⟨get chunk and pic 31⟩
}
show.number<-function(...){ # cat("show.number")
 no <- as.character(as.numeric(tclvalue(secno)))
 ⟨get chunk and pic 31⟩
}
```

24 ⟨*initialize* secno *and define functions for button commands* 23⟩+ ≡ ⊂ 14

```
eval.code <- function(...){
  code<-tclvalue(tkget(ttext,"0.0","end"))
  code.orig<-code<-unlist(strsplit(code,"\n"))
  code<-code[!substring(code,1,1)=="#"]
  ##?## code<-unlist(strsplit(code,";"))  ##110429 Fehler bei sep=";"
  if(length(code)==0){ cat("ok\n"); return() }
  result<-try(eval(parse(text=code),envir=where))
  code.orig<-sub("#([0-9]+):","##wnt-Code-Chunk:\\1-begin#",code.orig)
  code.orig<-sub("#:([0-9]+)","##wnt-Code-Chunk:\\1-end#",code.orig)
  h<-get("allcodechunks",envir=where)
  h<-c(h,paste("<","<*>",">=",sep=""),code.orig,"\n@\n")
  assign("allcodechunks",h,envir=where)
  code<-sub("^ *","",code); code<-code[nchar(code)>0]
  lexpr<-rev(code)[1]; lexpr<-substring(lexpr,1,4)
  if(length(code)==0||is.null(lexpr)||is.na(lexpr)) return()
  plot.res<-c("plot","boxp","par(","abli","pie(","hist","axis","show", #))
         "lsfi","pair","ylab","help",
         "qqli","qqno","qqpl","rug(","lege","segm","text","xlab",      #)
         "poin","line","titl","eda(","imag","vgl.","curv")             #)
  if(any(plot.res==lexpr)){ cat("plot generated\n"); return() }
  if(is.null(result)||is.na(result)||lexpr=="prin"||lexpr=="cat(") #)
    { cat("ok\n"); return() }
  if(is.list(result)&& length(names(result))> 0 &&
                          names(result)[1]=="ID") return()
  ## if(is.list(result)&& TRUE) return()
  no<-as.character(as.numeric(tclvalue(secno)))
  cat("result of code chunk",no,":\n")
 if(class(result)=="try-error"){ class(result)<-"character"; cat(result,"\n")
 }else{ print(result) } #; cat("ok\n")
}
exit.function<-function(...){
   tkdestroy(top)
   h<-get("allcodechunks",envir=where)
   if( length(h) < 2 ) return()
   filename<-tkgetSaveFile(filetypes="{{Paper Files} {.rev}}",
                           title="Do you want to save the activated R statements?")
   if(!is.character(filename)) filename<-tclvalue(filename)
   if(filename==""){
     cat("Demo function stopped without saving\n"); return()
   }
   if(0==length(grep("rev$",filename))) filename<-paste(filename,".rev",sep="")
   try(cat(h,sep="\n",file=filename))
   cat(paste("Remark: activated statements saved in\n   ",filename,"\n"))
   return()
}
```

8 History

Neu:

25 ⟨*construct and implement ShowHistory function* 25⟩ ≡ ⊂ 14

```
tfH<-tkframe(topHF)
⟨define functions of History control elements 26⟩
ttextH<-tktext(topHF,background="#f7fffF", #height=19,
               font="-Adobe-courier-Medium-R-Normal--18-180-*")
tkpack(tfH,   side="bottom",fill="x")
# tkpack(tfH,   side="bottom",fill="both",expand="y") #??true   "both"  "y"
tkpack(ttextH,side="bottom",fill="both",expand="y") # 091026
bShowHistory<-tkbutton(tf,text="History",width=butw)
⟨define control elements of History 30⟩
⟨define ShowHistory for evaluation 28⟩
```

26 ⟨*define functions of History control elements* 26⟩ ≡ ⊂ 25

```
  secnoH<-tclVar("1") #  just for initialization
  Show.next.number<-function(...){
   no<-as.character(as.numeric(tclvalue(secnoH))+1)
   ⟨hole chunk Nummer 27⟩
   if(0<length(code)) {
    tkdelete(ttextH,"0.0","end"); tkinsert(ttextH,"0.0",code)
    tclvalue(secnoH)<-as.character(no)
   }
  }
  Show.back.number<-function(...){
   no<-as.character(as.numeric(tclvalue(secnoH))-1)
   ⟨hole chunk Nummer 27⟩
   if(0<length(code)) {
    tkdelete(ttextH,"0.0","end"); tkinsert(ttextH,"0.0",code)
    tclvalue(secnoH)<-as.character(no)
   }
  }
  Show.number<-function(...){
   no<-as.character(as.numeric(tclvalue(secnoH)))
   ⟨hole chunk Nummer 27⟩
   if(0<length(code)) {
    tkdelete(ttextH,"0.0","end"); tkinsert(ttextH,"0.0",code)
    tclvalue(secnoH)<-as.character(no)
   }
  }
  Eval.code<-function(...){
### print("H:Eval.function")
    code<-tclvalue(tkget(ttextH,"0.0","end"))
    code.orig<-code<-unlist(strsplit(code,"\n"))
    code<-code[!substring(code,1,1)=="#"]
    ##?## code<-unlist(strsplit(code,";"))  ##110429 Fehler bei sep=";"
    if(length(code)==0){ cat("ok\n"); return() }
    result<-try(eval(parse(text=code),envir=where))
    code.orig<-sub("#([0-9]+):","##wnt-Code-Chunk:\\1-begin#",code.orig)
    code.orig<-sub("#:([0-9]+)","##wnt-Code-Chunk:\\1-end#",code.orig)
    h<-get("allcodechunks",envir=where)
    h<-c(h,paste("<","<*>",">=",sep=""),code.orig,"\n@\n")
    assign("allcodechunks",h,envir=where)
    code<-sub("^ *","",code)
    code<-code[nchar(code)>0]
    lexpr<-rev(code)[1]; lexpr<-substring(lexpr,1,4)
    if(length(code)==0||is.null(lexpr)||is.na(lexpr)) return()
    plot.res<-c("plot","boxp","par(","abli","pie(","hist","axis","show", #))
           "lsfi","pair","ylab","help",
           "qqli","qqno","qqpl","rug(","lege","segm","text","xlab", #)
           "poin","line","titl","eda(","imag","vgl.","curv")     #)
    if(any(plot.res==lexpr)){
```

```
      cat("Plot erstellt\n"); return()
    }
    if(is.null(result)||is.na(result)||lexpr=="prin"||lexpr=="cat("){  #)
      cat("ok\n"); return() }
    if(is.list(result)&& length(names(result))> 0 &&
       names(result)[1]=="ID") return()  ## if(is.list(result)&& TRUE) return()
    no<-as.character(as.numeric(tclvalue(secnoH)))
    cat("Result of code chunk",no,":\n")
   if(class(result)=="try-error"){
     class(result)<-"character";  cat(result,"\n")
   }else{
     print(result)
   }
   cat("ok\n")
  }
  Exit.function<-function(){
### print("H:Exit.function")
     tkpack(topf,fill="both",expand="y"); tkwm.title(top, main)
     tkpack.forget(topHF)
    return()
    filename<-tkgetSaveFile(filetypes="{{Paper Files} {.rev}}",
                            title="Do you want to save the activated R statements?")
    if(!is.character(filename)) filename<-tclvalue(filename)
    if(filename==""){
      cat("History function stopped without saving\n")
      return()
    }
    if(0==length(grep("rev$",filename))) filename<-paste(filename,".rev",sep="")
    h<-get("allcodechunks",envir=where)
    try(cat(h,sep="\n",file=filename))
    cat(paste("Remark: activated statements saved in\n   ",filename,"\n"))
    return()
  }
```

27 ⟨*hole chunk Nummer* 27⟩ ≡ ⊂ 26

```
### cat("gesucht no:",no)
chunksH <- get("chunksH",envir=where)
no.start<-grep(paste("^#",no,":$",sep=""),chunksH)
no.end<-grep(paste("^#:",no,"$",sep=""),chunksH)
if(length(no.end)==0||is.na(no.end) ||is.na(no.start)||
     is.nan(no.end)||is.nan(no.start)){
     cat("# sorry, chunk number '",no,"' wrong!\n"); return()
}
### cat("# aktueller chunk:",no,"\n")
code<-paste(chunksH[no.start:no.end],collapse="\n")
```

28 ⟨*define ShowHistory for evaluation* 28⟩ ≡ ⊂ 25

```
ShowHistory<-function(...){
###cat("ShowHistory")
  # get allcodechunks with chunks formated in rev-style
  chunksH<-get("allcodechunks",envir=where); if( length(chunksH) < 2 ) return()
  allcodechunks<-get("allcodechunks",envir=where)
  # change frame
  tkpack.forget(topf); tkpack(topHF,fill="both",expand="y")
  tkwm.title(top, "history of evaluations")
  # extract code chunks
  chunksH <- unlist(strsplit(chunksH,"\n")); chunksH <- sub("^Report","##Report",chunksH[-1])
  idx <- grep(paste("^<","<",sep=""),chunksH);
                       chunksH[idx] <- paste("#",1:length(idx),":",sep="")
```

```
  idx <- grep("^@",chunksH); chunksH[idx] <- paste("#:",1:length(idx),sep="")
###cat("no of chunksH",as.character(length(idx)))
  tclvalue(secnoH)<-as.character(length(idx))
  filename<-paste("tmp-demo",".R",sep=""); try(cat(chunksH,sep="\n",file=filename))
  assign("chunksH",chunksH,envir=where)
  cat(paste("Remark: statements of history have been saved in:  ",filename,"\n"))
  Show.number()
}
tkconfigure(bShowHistory, command=ShowHistory)
tkpack(bShowHistory)
```

29 ⟨*doit* 29⟩ ≡
⟨*teste* DemoBoxPlayer 16⟩

30 ⟨*define control elements of History* 30⟩ ≡ ⊂ 25

```
  bexitH<-tkbutton(tfH,text="Quit",width=butwh)
  bevalH<-tkbutton(tfH,text="Eval",width=butwh)
  bnextH<-tkbutton(tfH,text=">",width=butwa)
  bbackH<-tkbutton(tfH,text="<",width=butwa)
  tkbind(top, "<<EvalRCode>>", Eval.code)
  if( ! substring(version$os,1,6)=="darwin" ) {
    tkbind(top, "<<Next>>", Show.next.number)
    tkbind(top, "<<Back>>", Show.back.number)
  }
  lnoH  <-tkentry(tfH,textvariable=secnoH,width=butwh)
  linfoH<-tklabel(tfH,text="chunk:")
  tkpack(bevalH,linfoH,bbackH,lnoH,bnextH,side="left")
  tkpack(bexitH,side="right")
  tkconfigure(bexitH,command=Exit.function)
  tkconfigure(bnextH,command=Show.next.number)
  tkconfigure(bbackH,command=Show.back.number)
  tkconfigure(bevalH,command=Eval.code)

  tkevent.add("<<Paste>>",   "<Control_L><v>")
## ok:  tkbind(ttextH,"<<Paste>> { catch {%W insert insert [selection get -selection CLIPBOARD] }
  # tkbind(lnoH,"<Return>",Show.number)
  tkbind(lno,"<KeyRelease>",Show.number)
  tclvalue(secnoH)<-as.character(no)
if(substring(version$os,1,6)=="darwin" ){
  mac.paste<-function(...){  ## Ctrl-V  # mac-PASTE
     try({.Tcl("clipboard append hello"); .Tcl("clipboard clear")
          news<-base::scan(file=pipe("pbpaste","r"),what="",sep="\n",blank.lines.skip=FALSE)
          tkinsert(ttextH,"insert",paste(news,collapse="\n"))})
          tksee(ttextH,"insert - 7 lines");  tksee(ttextH,"insert + 7 lines") #090706
  }
  tkbind(ttextH,"<Control_L><v>",mac.paste)
  mac.copy<-function(...){  ## Ctrl-C  # mac-COPY
     news<-""
     try(news<-tclvalue(.Tcl("if {[catch {clipboard get}]} {set aa empty} {set aa full}")))
     if(news=="empty") return()
     try({news<-tclvalue(.Tcl("set aaa [selection get -selection CLIPBOARD]"))
          tmp.file.name <- tempfile("rt-tmp")
          base::cat(news,file=tmp.file.name); system(paste("pbcopy < ",tmp.file.name))
          .Tcl("clipboard append hello"); .Tcl("clipboard clear")})
  }
  tkbind(ttextH,"<Control_L><c>",mac.copy)
  tkevent.add("<<extract>>",  "<Control_L><c><KeyRelease>")   # mac-extract
  tkbind(ttextH,"<<extract>>",mac.copy)
}else{
  tkevent.add("<<Paste>>",   "<Control_L><v>")
```

```
  tkbind(ttextH,"<<Paste>> { catch {%W insert insert [selection get -selection CLIPBOARD] } }")
}
```

End of History-Feature.

31 ⟨*get chunk and pic* 31⟩ ≡ ⊂ 23

```
no.start<-grep(paste("^#",no,":$",sep=""),chunks)
no.end  <-grep(paste("^#:",no,"$",sep=""),chunks)
if(length(no.end)==0||is.na(no.end) ||is.na(no.start)||
  is.nan(no.end)||is.nan(no.start)){
  cat("sorry, chunk number '",no,"' wrong!\n"); return()
}
if(debug) cat("put code",no)
code<-paste(chunks[no.start:no.end],collapse="\n")
if(0<length(code)) {
  tkdelete(ttext,"0.0","end"); tkinsert(ttext,"0.0",code)
  tclvalue(secno)<-as.character(no)
}
if(debug) cat("put text",no)
tkdelete(tw,"0.0","end")
text.pic <- file.path(DemoBoxPath,paste(DemoBoxName,"-",no,".gif",sep=""))
if(0 < no){  print(text.pic)
  text.ok <- TRUE
  if( file.exists(text.pic) ) tcl("createandshowimagegif", tw, text.pic, "0.0") else {
    text.pic <- sub(".gif",".ppm",text.pic)
    if( file.exists(text.pic) ) tcl("createandshowimageppm", tw, text.pic, "0.0") else {
      # tkconfigure(tframe,height="0"); tkconfigure(tw,height="0"); text.ok <- FALSE
    }
  }
}
if(debug) cat("get chunk and pic: done")
```

32 ⟨*initialize history* 32⟩ ≡ ⊂ 14

```
allcodechunks <- paste("@\nReport of activated chunks from: ",date(),
                       "\n  ", sep="")
if(debug) cat("initialize history: done")
```

33 ⟨*open new top level widget* 33⟩ ≡ ⊂ 14

```
  top <- tktoplevel()
  topf <- tkframe(top); tkpack(topf,fill="both",expand="y")
  topHF <- tkframe(top) # frame for History
  if(missing(main)) main <- paste(DemoBoxName); tkwm.title(top,main)
  geo <- paste(geo.width,"x",geo.height,"+10+10",sep=""); tkwm.geometry(top,geo)
```

34 ⟨*pack widget for textual explanations / image* 34⟩ ≡ ⊂ 14

```
  tkpack(tframe <- tkframe(topf),fill="x")
  tkpack(tw <- tktext(tframe),expand="yes",fill="x")
  tkconfigure(tframe,height=tw.height,width=w.width)
  tw.height.act<-tclVar("0"); tclvalue(tw.height.act)<-as.character(tw.height)
  tkconfigure(tw,height=tw.height,width=w.width)
  .Tcl( paste(""
    ,"proc createandshowimagegif {w im place} {"
    ,"global imageno","set imageno [image create photo -format gif -file $im]"
    ,"$w image create $place -image $imageno","}"
    ,"proc createandshowimageppm {w im place} {"
    ,"global imageno","set imageno [image create photo -format ppm -file $im]"
    ,"$w image create $place -image $imageno","}", sep="\n")
  )
  if(debug) cat("pack widget for textual explanations / image: done")
```

35 ⟨*pack control buttons* 35⟩ ≡ ⊂ 14

```
tf<-tkframe(topf);  butw <- "6"; butwh <- "3"; butwa <- "1"
# cut and paste    # if(#<OS ist Mac-Mini-Tiger># ) ...
tkevent.add("<<Paste>>",   "<Control_L><v>")
bexit<-tkbutton(tf,text="Exit",width=butwh)
beval<-tkbutton(tf,text="Eval",width=butwh)
# bnext<-tkbutton(tf,text="\u21a6",width=butwa) # next
bnext<-tkbutton(tf,text=">",width=butwa) # next
tkconfigure(bnext,cursor="right_side")
# bback<-tkbutton(tf,text="\u21a4",width=butwa) # back
bback<-tkbutton(tf,text="<",width=butwa) # back
tkconfigure(bback,cursor="left_side")
tkbind(topf, "<<EvalRCode>>", eval.code)
# if( ! #<OS ist Mac-Mini-Tiger>> ) { } to do .. else:
  tkbind(topf, "<<Next>>", show.next.number)
  tkbind(topf, "<<Back>>", show.back.number)
#}
lno  <-tkentry(tf,textvariable=secno,width=butwh)
labno <- tklabel(tf, text="Demo:", width=butw)
# linfo<-tklabel(tf,text="chunk number:"); tkpack(linfo)
tkpack(beval,labno,bback,lno,bnext,side="left")
tkconfigure(bexit,command=exit.function)
tkconfigure(bnext,command=show.next.number)
tkconfigure(bback,command=show.back.number)
tkconfigure(beval,command=eval.code)
# tkbind(lno,"<Return>",show.number)
tkbind(lno,"<KeyRelease>",show.number)
if(debug) cat("pack control buttons: done")
```

36 ⟨*construct a lower and upper button for text widget* 36⟩ ≡ ⊂ 14

```
tkpack(bexit,
       # bigger.but  <- tkbutton(tf, text="\u21a7", width=butwa), # down
       # smaller.but <- tkbutton(tf, text="\u21a5", width=butwa), # up
       bigger.but  <- tkbutton(tf, text="v", width=butwa), # down
       smaller.but <- tkbutton(tf, text="^", width=butwa), # up
       side="right")
tkconfigure(bigger.but,cursor="bottom_side")
tkconfigure(smaller.but,cursor="top_side")
bigger.tw <- function(){
  # if(debug) print("bigger")
  tclvalue(tw.height.act) <- tw.height <-
    as.character(min(as.numeric(tw.height)+as.numeric(cw.height)-3,
                     as.numeric(tclvalue(tw.height.act))+5))
  tkconfigure(tw,height=tw.height) #(,width=w.width)
}
smaller.tw <- function(){
  # if(debug) print("smaller") # up
  tclvalue(tw.height.act) <- tw.height <-
    as.character(max(1,as.numeric(tclvalue(tw.height.act))-5))
  tkconfigure(tw,height=tw.height) #(,width=w.width)
}
tkconfigure(bigger.but,  command=bigger.tw)
tkconfigure(smaller.but, command=smaller.tw)
tkpack(tf,side="top",fill="x")  #  tkpack(tf,side="top",fill="y")
#  tkpack(tf,side="top",fill="both",expand="true")  #  tkpack(tf,side="top",fill="y")
if(debug) cat("construct a lower and upper button for text widget: done")
```

37 ⟨*construct text widget for code* 37⟩ ≡ ⊂ 14

```
ttext<-tktext(topf,height=19,background="#f7fffF",
```

```
                font="-Adobe-courier-Medium-R-Normal--18-180-*")
  tkpack(ttext,side="bottom",fill="both",expand="y") # 091026
  tkbind(ttext,"<<Paste>> { catch {%W insert insert [selection get -selection CLIPBOARD] } }")
  if(debug) cat("construct text widget for code: done")
```

38 ⟨*save no and set to no 1* 38⟩ ≡ ⊂ 14

```
tclvalue(secno)<-as.character("1"); show.number() #; show.next.number()
```

\DemoSectionAnwendungsbeispiele

Beispiele

39 ⟨*Testchunk* 39⟩ ≡

```
base::cat("x <- co2",file="tt.R")
```

\DemoSectionZRA

Eine Demo-Box zur Inspektion von Zeitreihen.

40 ⟨* 40⟩ ≡

```
ts.zoom<-function(x){
  refresh.code<-function(...){
  # Vorbereitung
    start<-slider(no=1)*1000; end<-slider(no=2)*1000
    if(start>end) start<-slider(set.no.value=c(1,round(end/1000-1)))
  # Plot
    plot(x,xlim=c(min(start,end), max(start,end)))
  }
  slider(refresh.code,
         sl.names=c("begin: index * 0.001",  "end: index * 0.001 "),
         sl.mins=c(0,0),
         sl.maxs=c(length(x)/1000,length(x)/1000),
         sl.deltas=c(1,1),
         sl.defaults=c(1,length(x)/1000)
  )
}
ts.zoom(runif(10000))
```

\DemoSectionEin Glücksradsimulator Was ist ein Jahrmarkt oder Sportfest ohne Glücksrad? Jeder kennt diese Zufallsspielerei, bei der man für einen Einsatz mehrmals – zum Beispiel 3 Male – drehen darf. Die Wahrscheinlichkeiten für die unterschiedlichen Ausgänge hängen vom jeweiligen Glücksrad ab. Eine zentrale Frage ist natürlich, welche Gewinnerwartungen mit einem speziellen Rad verbunden sind. Per Simulation gestattet es der hier vorgestellte Simulator, Gewinne zu simulieren und auszuwerten. Folgende Dinge charakterisieren ein Spiel:

types:	mögliche Gewinn- bzw. Verlusttypen
probs:	Wahrscheinlichkeiten für diese Typen
n.rots:	Anzahl der Drehungen pro Einsatz

Zur Simulation sind zusätzlich noch folgende Angaben erforderlich:

seed:	Zufallsstart
runs:	Anzahl der Simulationsdurchgänge

Zum Schluss bleibt, die Auswertung zu definieren. Es sollen ausgegeben werden:

- ein Boxplot der Auszahlungen und die Entwicklung der Mittel
- wie auch das Auszahlungsmittel sowie der Looser-Anteil

Umsetzung

41 ⟨* 40⟩+ ≡
```
# Initialisierungen
types <- c(0,       1,   50) # Auszahlungstypen
probs <- c(0.54, 0.45, 0.01) # Wahrscheinlichkeiten der Typen
n.rot <- 3                   # Anzahl der Drehungen
seed  <- 13                  # Zufallsstart
runs  <- 10000               # Versuche
par(mfrow=2:1)               # graphische Parameter
# Simulator definieren
simulator <- function(x) sample(types, n.rot, prob=probs, replace=TRUE)
# Ziehungen
set.seed(seed)
results <- sapply(1:runs, simulator)
# Auswertung
Auszahlungen <- colSums(results)
cat("Mittel der Auszahlungen:", h <- mean(Auszahlungen),"\n" )
cat("relative Looser-Anzahl: ", mean(Auszahlungen == 0),"\n" )
boxplot(Auszahlungen, horizontal=TRUE); abline(v=h,col="red")
plot(1:runs, cumsum(Auszahlungen)/(1:runs), type="l", bty="n")
```

ein start

42 ⟨*start* 42⟩ ≡
```
print("hello world")
```

\DemoSectionEin alternativer Zufallsradsimulator

Dies ist die Demo \DemoNumber (per \DemoNumber) oder (per \roman{DemoNumber})

43 ⟨* 40⟩+ ≡
```
# "Personen-Simulator" definierten
simulator <- function(x) sample(size = 3,  c(50,1,0), prob = c(1,45,54), replace = TRUE)
# Wiederholungszahl setzen
n <- 1000
# Zufallsstart setzen
set.seed(13)
# Simulation anstoÃ§en
auszahlungen <- sapply(1:n, simulator)
# Einzelauszahlungen auswerten /mitteln (Aufgabe 4a)
print(mean(colSums(auszahlungen)))
# relative HÃ¤ufigkeit fÃ¼r keinen Gewinn
mean(0==colSums(auszahlungen))
```

\DemoSectionDOKO-Fragen

Wie groß ist die Wahrscheinlichkeit, dass Herz durchgeht?

Diese Frage lässt sich mit Hilfe von Binomialkoeffizienten beantworten. Denn für einen reinen Herzstich muss jeder der Spieler eine Herzkarte gezogen haben.

44 ⟨* 40⟩+ ≡
```
result <- 1:10
for(i in 1:10){
```

```
  stich.weg <- i-1
  # x, m, n, k :: white player, white total, black total, number choosen
  herz <- 4; not.herz <- 36 - stich.weg*4
  ws.spieler.1 <- dhyper(1, herz, not.herz, 10-stich.weg)
  herz <- herz-1; not.herz <- not.herz - (10-stich.weg) + 1
  ws.spieler.2 <- dhyper(1, herz, not.herz, 10-stich.weg)
  herz <- herz-1; not.herz <- not.herz - (10-stich.weg) + 1
  ws.spieler.3 <- dhyper(1, herz, not.herz, 10-stich.weg)
  herz <- herz-1; not.herz <- not.herz - (10-stich.weg) + 1
  ws.spieler.4 <- dhyper(1, herz, not.herz, 10-stich.weg) # zum Check
  ws <- ws.spieler.1*ws.spieler.2*ws.spieler.3*ws.spieler.4
  result[i] <- ws
}
plot(1:10, result, type="h", ylim=0:1)
```

\DemoSectionUnfälle in Bielefeld

Es sind glücklicherweise in Bielefeld im Jahr 2012 nur 3 Personen durch Verkehrsunfälle getötet worden. Wir wollen untersuchen, wie stark welche Variabilitäten die Verteilung eines unterstellten Poisson-Modells in Abhängigkeit des Parameters λ besitzt.

Fragen:

- Wenn das Poisson-Modell mit $\lambda = 3$ stimmt, wie wahrscheinlich sind dann mehr als 5 Tote?
- Wenn $\lambda = 7$ zutreffen würde, wie wahrscheinlich sind dann 3 oder weniger Unfallopfer?

Umsetzung:

45 ⟨* 40⟩+ ≡

```
lambda <- 3
x <- 0:20
f.x <- dpois(x, lambda)
F.x <- ppois(x, lambda)
par(mfrow=c(2,2))
plot(x, f.x, type="h", lwd=4, col="red")
title(paste("lambda =", round(lambda,2)))
plot(x, F.x, type="S", lwd=4, col="blue")
n <- 1000
stpr <- rpois(n, lambda)
boxplot(stpr, horizontal=TRUE, ylim=range(x))
# open.demo.box(file.name = "unfaelleBI", "90", "65")
```

\DemoSectionStrahlungsreduktion im Zeitablauf

Für radioaktive Körper halbiert sich die Strahlung nach der Halbwertzeit. Eine interessante Frage ist in diesem Kontext, wann die Strahlung unter eine vorgegebene Grenze gesunken ist. Dazu kann man im Zeitablauf die Strahlung messen und eine Exponentialfunktion anpassen. Durch Extrapolation des Modells können wir die Grenzzeit bestimmen.

Durch Wiederholung des Prozesses können wir die Auswirkung von Messfehler erkennen. In der Simulationssituation haben wir eine Fehlerstandardabweichung von sd unterstellt.

46 ⟨* 40⟩+ ≡

```
# Prozessparameter
```

```
a <- 100; b <- .3; sd <- .1
# Beobachtungszeitpunkte
time <- 1:4; t.max <- 20
# WDs
wd <- 10; set.seed(123)
# Simulation, Anpassung, Grenzfrage
strahlung.orig <- strahlung <- a*exp(-b*time)*rnorm(length(time),1,sd)
plot(time,strahlung, xlim= c(0,10*1.5), ylim=c(0,a), col="red", lwd=4); abline(h=0)
print(beta.dach <- lsfit(time, log(strahlung))$coef)
a <- exp(beta.dach[1]); b <- -beta.dach[2]; cat("a =",a.orig <- a,"b =",b.orig <- b)
modell <- function(x, a, b) y <- a*exp(-b*x)
imodell <- function(y.limit, a, b) t.limit <- -log(y.limit/a)/b
x <- seq(0,t.max,length=100); lines(x,modell(x,a,b), lwd=2)
y.limit <- 10; t.limit <- imodell(y.limit,a,b)
xy <- cbind(c(0,t.limit,t.limit),c(y.limit,y.limit,0)); lines(xy, col="blue", lwd=3)
for(w in 1:wd){
  strahlung <- a*exp(-b*time)*rnorm(length(time),1,sd)
  points(time, strahlung, col=w, cex=.5)
  beta.dach <- lsfit(time, log(strahlung))$coef
  a <- exp(beta.dach[1]); b <- -beta.dach[2]; cat("a =",a,"b =",b)
  modell <- function(x, a, b) y <- a*exp(-b*x)
  imodell <- function(y.limit, a, b) t.limit <- -log(y.limit/a)/b
  x <- seq(0,t.max,length=100); lines(x,modell(x,a,b), lwd=0.5,col=w)
  y.limit <- 10; t.limit <- imodell(y.limit,a,b)
  xy <- cbind(c(0,t.limit,t.limit),c(y.limit,y.limit,0)); lines(xy, col=w, lwd=0.5)
}
```

9 Anhang

Object Index

Code Chunk Index

10 Reste

10.1 Speicherung von Bilder

Speicherung als Bit-Objekt

47 ⟨*Bild als R-objekt speichern* 47⟩ ≡

```
pic.name <- "tszoom.demo.box.gif"
# Bild lesen
aa <- readBin(pic.name ,what="raw",n=100000); aa.org <- aa
# Bild in Zeichenkette umwandeln
aa <- as.character(rawToBits(aa))
# Zeichenkette in Textanweisung umwandeln bzw. hantieren
aa <- deparse(aa); dump("aa")
# neue Zuweisung zusammenbauen
txt <- c("bb <-", aa)
# Zuweisung realisieren
eval(parse(text=txt))
# Zeichenkette in gepackte Bits umwandeln
bb <- packBits(as.raw(bb),"raw")
# Bits wegschreiben
writeBin(bb,"a-gif.gif")
aa.org[1700:1709]
```

Speicherung als Integer

48 ⟨*Bild als R-objekt speichern* 47⟩+ ≡

```
pic.name <- "tszoom.demo.box.gif"
# Bild lesen
aa <- c(readBin(pic.name ,what="integer",n=1000)) #,useBytes=TRUE))
# Bild in Zeichenkette umwandeln
dump("aa") # 5566 Byte, aber hinten fehlt ein Byte im Bild
bb <- intToBits(aa)
bb <- packBits(as.raw(bb),"raw")
writeBin(bb,"a-gif3.gif")
```

Speicherung als Integer-Zahlen

49 ⟨*Bild als R-objekt speichern* 47⟩+ ≡

```
pic.name <- "tszoom.demo.box.gif"
# Bild lesen
aa <- readBin(pic.name ,what="raw",n=10000)#,endian="big"); aa.org <- aa
# Bild in Zeichenkette umwandeln
aa <- c(rawToBits(aa))#,rawToBits(aa[1:1]))
n <- ceiling(length(aa)/32)*32 # ceiling(length(aa)/32)*32
aa <- packBits(aa[1:n],"integer")
dump("aa")# 5571 Byte, aber hinten drei Bytes mehr!
bb <- intToBits(aa)
bb <- packBits(as.raw(bb),"raw")
writeBin(bb,"a-gif4.gif")
bb <- eval(parse(text=c(readLines("a-gif.R")))
```

10.2 Internet-Seiten

::::online converter jpg to gif http://www.coolutils.com/online/image-converter/

::::online converter pdf to gif http://www.zamzar.com/convert/pdf-to-gif/
http://www.pcwelt.de/downloads/PDF-Konverter-PDF-to-Image-Converter-1282495.html

::::CROP-Internetseiten: How to crop a pdf file via ghostscript
http://stackoverflow.com/questions/6183479/cropping-a-pdf-using-ghostscript-9-

01

pdfcrop bei ctan: http://www.ctan.org/tex-archive/support/pdfcrop/

pdfcrop // pdfcrop.exe http://hzqtc.github.com/2012/04/pdf-tools-merging-extracting-and-cropping.html

::::PDF to Image: via pstoimg: ps → ppmraw
`http://hepunx.rl.ac.uk/~adye/pstoim`

http://www.computerbild.de/download/1-2-3-PDFConverter-10625.htmlg.html

http://web.mit.edu/windows-graphics/bin/

gswin32c.exe -sDEVICE=ppmraw -sOutputFile="test.ppm" t2.pdf -dNOPAUSE -dBATCH

gs -sDEVICE=ppmraw -sOutputFile="test.ppm" t2.pdf -dNOPAUSE -dBATCH

pdf2gif:

http://www.download25.com/install/pdf-to-image-converter.html

http://www.software112.com/tags-program/1/pdf2image+exe.html

http://tex.stackexchange.com/questions/31973/latex-how-do-i-force-pdf-page-height-width

10.3 etc.

50 ⟨*ppm* 50⟩ ≡

```
system("dvipdf cutandshowchunk.dvi")
system("pdfcrop cutandshowchunk.pdf t2.pdf; convert t2.pdf cutandshowchunk.ppm")
```

51 ⟨*pdfcrop* 51⟩ ≡

```
# Rand abschneiden
#  system(paste("pdfcrop", sub(".tex$",".pdf",latex.name), sub(".tex$",".PDF",latex.name)))
#  file.rename(sub(".tex$",".PDF",latex.name),sub(".tex$",".pdf",latex.name))

#.Tcl("set ino [image create photo -format ppm -file
cutandshowchunk.ppm]")
```