

# Package ‘PBSmodelling’

May 7, 2026

**Version** 2.70.2

**Date** 2025-11-13

**Title** GUI Tools Made Easy: Interact with Models and Explore Data

**Author** Jon T. Schnute [aut],  
Alex Couture-Beil [aut],  
Rowan Haigh [aut],  
Nicholas Boers [aut],  
Anisa Egeli [aut],  
A. R. Kronlund [ctb],  
Steve Martell [ctb],  
Norm Olsen [ctb],  
Nick Fisch [cre]

**Maintainer** Nick Fisch <nick.fisch@dfo-mpo.gc.ca>

**Copyright** 2005-2025, Fisheries and Oceans Canada

**Depends** R (>= 4.2.0)

**Imports** methods, tcltk, XML

**Suggests** PBSmapping, deSolve, KernSmooth

**NeedsCompilation** yes

**SystemRequirements** BWidget

**Description** Provides software to facilitate the design, testing, and operation of computer models. It focuses particularly on tools that make it easy to construct and edit a customized graphical user interface ('GUI'). Although our simplified 'GUI' language depends heavily on the R interface to the 'Tcl/Tk' package, a user does not need to know 'Tcl/Tk'. Examples illustrate models built with other R packages, including 'PBSmapping', 'PBSdresolve', and 'BRugs'. A complete user's guide 'PBSmodelling-UG.pdf' shows how to use this package effectively.

**License** GPL (>= 2)

**URL** <https://github.com/pbs-software/pbs-modelling>

**Repository** CRAN

**Date/Publication** 2025-11-15 12:00:25 UTC

## Contents

addArrows . . . . .	4
addLabel . . . . .	5
addLegend . . . . .	6
calcFib . . . . .	7
calcGM . . . . .	7
calcMin . . . . .	8
CCA.qbr . . . . .	10
chooseWinVal . . . . .	12
cleanProj . . . . .	14
cleanWD . . . . .	15
clearAll . . . . .	16
clearPBSext . . . . .	16
clearRcon . . . . .	17
clearWinVal . . . . .	18
clipVector . . . . .	18
closeWin . . . . .	19
compileC . . . . .	20
compileDescription . . . . .	21
convSlashes . . . . .	21
createVector . . . . .	22
createWin . . . . .	23
declareGUIoptions . . . . .	25
doAction . . . . .	26
dot-c.support.funs . . . . .	27
dot-convert.funs . . . . .	28
dot-create.funs . . . . .	29
dot-gui.funs . . . . .	31
dot-history.funs . . . . .	35
dot-list.funs . . . . .	36
dot-PBSmodEnv . . . . .	38
dot-project.funs . . . . .	38
dot-support.funs . . . . .	41
dot-talk.funs . . . . .	42
dot-widget.defs . . . . .	44
dot-win.funs . . . . .	45
drawBars . . . . .	46
evalCall . . . . .	46
expandGraph . . . . .	48
exportHistory . . . . .	49
findPat . . . . .	50
findPrefix . . . . .	50
findProgram . . . . .	51
focusWin . . . . .	52
genMatrix . . . . .	53
getChoice . . . . .	54
getGUIoptions . . . . .	55

getOptions . . . . .	56
getOptionsFileName . . . . .	57
getOptionsPrefix . . . . .	58
getPBSext . . . . .	59
getPBSoptions . . . . .	60
getWinAct . . . . .	60
getWinFun . . . . .	61
getWinVal . . . . .	61
getYes . . . . .	62
GT0 . . . . .	63
importHistory . . . . .	64
initHistory . . . . .	65
isWhat . . . . .	67
lisp . . . . .	68
loadC . . . . .	69
loadOptions . . . . .	70
loadOptionsGUI . . . . .	71
lucent . . . . .	71
openExamples . . . . .	72
openFile . . . . .	73
openUG . . . . .	74
packList . . . . .	75
pad0 . . . . .	77
parseWinFile . . . . .	78
pause . . . . .	79
PBSmodelling . . . . .	79
PBSoptions-class . . . . .	80
pickCol . . . . .	82
plotACF . . . . .	83
plotAsp . . . . .	84
plotBubbles . . . . .	85
plotCsum . . . . .	86
plotDens . . . . .	87
plotFriedEggs . . . . .	88
plotSidebars . . . . .	89
plotTrace . . . . .	90
presentTalk . . . . .	91
promptWriteOptions . . . . .	92
readList . . . . .	93
readPBSoptions . . . . .	94
resetGraph . . . . .	95
restorePar . . . . .	96
runDemos . . . . .	97
runExample . . . . .	97
runExamples . . . . .	98
scalePar . . . . .	99
selectDir . . . . .	100
selectFile . . . . .	101

setFileOption . . . . .	102
setGUIoptions . . . . .	103
setPathOption . . . . .	104
setPBSext . . . . .	105
setPBSoptions . . . . .	106
setwdGUI . . . . .	107
setWidgetColor . . . . .	108
setWidgetState . . . . .	109
setWinAct . . . . .	111
setWinVal . . . . .	111
show0 . . . . .	112
showAlert . . . . .	114
showArgs . . . . .	114
showHelp . . . . .	115
showPacks . . . . .	116
showRes . . . . .	117
showVignettes . . . . .	118
sortHistory . . . . .	118
talk-class . . . . .	119
testAlpha . . . . .	121
testCol . . . . .	122
testLty . . . . .	123
testLwd . . . . .	124
testWidgets . . . . .	125
tget . . . . .	127
unpackList . . . . .	129
updateGUI . . . . .	130
vbdata . . . . .	131
vbpars . . . . .	131
view . . . . .	132
viewCode . . . . .	133
writeList . . . . .	134
writePBSoptions . . . . .	136

**Index****137**

---

**addArrows***Add Arrows to a Plot Using Relative (0:1) Coordinates*

---

**Description**

Call the arrows function using relative (0:1) coordinates.

**Usage**

```
addArrows(x1, y1, x2, y2, ...)
```

**Arguments**

x1	x-coordinate (0:1) at base of arrow.
y1	y-coordinate (0:1) at base of arrow.
x2	x-coordinate (0:1) at tip of arrow.
y2	y-coordinate (0:1) at tip of arrow.
...	additional parameters for the function arrows.

**Details**

Lines will be drawn from  $(x1[i], y1[i])$  to  $(x2[i], y2[i])$

**Author(s)**

Jon T. Schnute, Pacific Biological Station, Fisheries and Oceans Canada , Nanaimo BC

**See Also**

[addLabel](#), [addLegend](#)

**Examples**

```
local(envir=.PBSmodEnv,expr={
  oldpar = par(no.readonly=TRUE)
  tt=seq(from=-5,to=5,by=0.01)
  plot(sin(tt), cos(tt)*(1-sin(tt)), type="l")
  addArrows(0.2,0.5,0.8,0.5)
  addArrows(0.8,0.95,0.95,0.55, col="#FF0066")
  par(oldpar)
})
```

---

addLabel

*Add a Label to a Plot Using Relative (0:1) Coordinates*

---

**Description**

Place a label in a plot using relative (0:1) coordinates

**Usage**

```
addLabel(x, y, txt, ...)
```

**Arguments**

x	x-axis coordinate in the range (0:1); can step outside.
y	y-axis coordinate in the range (0:1); can step outside.
txt	desired label at (x, y).
...	additional arguments passed to the function text.

**Author(s)**

Jon T. Schnute, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**See Also**

[addArrows](#), [addLegend](#)

**Examples**

```
local(envir=.PBSmodEnv,expr={
  oldpar = par(no.readonly=TRUE)
  resetGraph()
  addLabel(0.75,seq(from=0.9,to=0.1,by=-0.10),c('a','b','c'), col="#0033AA")
  par(oldpar)
})
```

---

addLegend

*Add a Legend to a Plot Using Relative (0:1) Coordinates*

---

**Description**

Place a legend in a plot using relative (0:1) coordinates.

**Usage**

```
addLegend(x, y, ...)
```

**Arguments**

x	x-axis coordinate in the range (0:1); can step outside.
y	y-axis coordinate in the range (0:1); can step outside.
...	arguments used by the function legend, such as lines, text, or rectangle.

**Author(s)**

Jon T. Schnute, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**See Also**

[addArrows](#), [addLabel](#)

**Examples**

```
local(envir=.PBSmodEnv,expr={
  oldpar = par(no.readonly=TRUE)
  resetGraph()
  n <- sample(1:length(colors()),15); clrns <- colors()[n]
  addLegend(.2,1,fill=clrns,leg=clrns,cex=1.5)
  par(oldpar)
})
```

---

`calcFib`*Calculate Fibonacci Numbers by Several Methods*

---

**Description**

Compute Fibonacci numbers using four different methods: 1) iteratively using R code, 2) via the closed function in R code, 3) iteratively in C using the `.C` function, and 4) iteratively in C using the `.Call` function.

**Usage**

```
calcFib(n, len=1, method="C")
```

**Arguments**

<code>n</code>	nth fibonacci number to calculate
<code>len</code>	a vector of length <code>len</code> showing previous fibonacci numbers
<code>method</code>	select method to use: C, Call, R, closed

**Value**

Vector of the last `len` Fibonacci numbers calculated.

**Author(s)**

Jon T. Schnute, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

---

`calcGM`*Calculate the Geometric Mean, Allowing for Zeroes*

---

**Description**

Calculate the geometric mean of a numeric vector, possibly excluding zeroes and/or adding an offset to compensate for zero values.

**Usage**

```
calcGM(x, offset = 0, exzero = TRUE)
```

**Arguments**

<code>x</code>	vector of numbers
<code>offset</code>	value to add to all components, including zeroes
<code>exzero</code>	if TRUE, exclude zeroes (but still add the offset)

**Value**

Geometric mean of the modified vector  $x + \text{offset}$

**Note**

NA values are automatically removed from  $x$

**Author(s)**

Rowan Haigh, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**Examples**

```
local(envir=.PBSmodEnv,expr={
  calcGM(c(0,1,100))
  calcGM(c(0,1,100),offset=0.01,exzero=FALSE)
})
```

---

 calcMin

---

*Calculate the Minimum of a User-Defined Function*


---

**Description**

Minimization based on the R-stat functions `nlm`, `nlminb`, and `optim`. Model parameters are scaled and can be active or not in the minimization.

**Usage**

```
calcMin(pvec, func, method="nlm", trace=0, maxit=1000, reltol=1e-8,
  steptol=1e-6, temp=10, repN=0, ...)
```

**Arguments**

<code>pvec</code>	Initial values of the model parameters to be optimized. <code>pvec</code> is a data frame comprising four columns ( <code>"val"</code> , <code>"min"</code> , <code>"max"</code> , <code>"active"</code> ) and as many rows as there are model parameters. The <code>"active"</code> field (logical) determines whether the parameters are estimated (T) or remain fixed (F).
<code>func</code>	The user-defined function to be minimized (or maximized). The function should return a scalar result.
<code>method</code>	The minimization method to use: one of <code>nlm</code> , <code>nlminb</code> , Nelder-Mead, BFGS, CG, L-BFGS-B, or SANN. Default is <code>nlm</code> .
<code>trace</code>	Non-negative integer. If positive, tracing information on the progress of the minimization is produced. Higher values may produce more tracing information: for method "L-BFGS-B" there are six levels of tracing. Default is 0.
<code>maxit</code>	The maximum number of iterations. Default is 1000.

reltol	Relative convergence tolerance. The algorithm stops if it is unable to reduce the value by a factor of $\text{reltol} * (\text{abs}(\text{val}) + \text{reltol})$ at a step. Default is $1e-8$ .
steptol	A positive scalar providing the minimum allowable relative step length. Default is $1e-6$ .
temp	Temperature controlling the "SANN" method. It is the starting temperature for the cooling schedule. Default is $10$ .
repN	Reports the parameter and objective function values on the R-console every repN evaluations. Default is $0$ for no reporting.
...	Further arguments to be passed to the optimizing function chosen: <code>nlm</code> , <code>nlminb</code> , or <code>optim</code> . Beware of partial matching to earlier arguments.

### Details

See `optim` for details on the following methods: Nelder-Mead, BFGS, CG, L-BFGS-B, and SANN.

### Value

A list with components:

<code>fout</code>	The output list from the optimizer function chosen through <code>method</code> .
<code>iters</code>	Number of iterations.
<code>evals</code>	Number of evaluations.
<code>cpuTime</code>	The user CPU time to execute the minimization.
<code>elapsedTime</code>	The total elapsed time to execute the minimization.
<code>fminS</code>	The objective function value calculated at the start of the minimization.
<code>fminE</code>	The objective function value calculated at the end of the minimization.
<code>Pstart</code>	Starting values for the model parameters.
<code>Pend</code>	Final values estimated for the model parameters from the minimization.
<code>AIC</code>	Akaike's Information Criterion
<code>message</code>	Convergence message from the minimization routine.

### Note

Some arguments to `calcMin` have no effect depending on the method chosen.

### Author(s)

Jon T. Schnute, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

### See Also

[scalePar](#), [restorePar](#), [calcMin](#), [GT0](#)  
In the `stats` package: `nlm`, `nlminb`, and `optim`.

**Examples**

```

local(envir=.PBSmodEnv,expr={
Ufun <- function(P) {
  Linf <- P[1]; K <- P[2]; t0 <- P[3]; obs <- afile$len;
  pred <- Linf * (1 - exp(-K*(afile$age-t0)));
  n <- length(obs); ssq <- sum((obs-pred)^2 );
  return(n*log(ssq)); };

oldpar = par(no.readonly = TRUE)
afile <- data.frame(age=1:16,len=c(7.36,14.3,21.8,27.6,31.5,35.3,39,
  41.1,43.8,45.1,47.4,48.9,50.1,51.7,51.7,54.1));
pvec <- data.frame(val=c(70,0.5,0),min=c(40,0.01,-2),max=c(100,2,2),
  active=c(TRUE,TRUE,TRUE),row.names=c("Linf","K","t0"),
  stringsAsFactors=FALSE);
alist <- calcMin(pvec=pvec,func=Ufun,method="nlm",steptol=1e-4,repN=10);
print(alist[-1]); P <- alist$Pend;
#resetGraph();
expandGraph();
xnew <- seq(afile$age[1],afile$age[nrow(afile)],len=100);
ynew <- P[1] * (1 - exp(-P[2]*(xnew-P[3])) );
plot(afile); lines(xnew,ynew,col="red",lwd=2);
addLabel(.05,.88,paste(paste(c("Linf","K","t0"),round(P,c(2,4,4))),
  sep=" = "),collapse="\n"),adj=0,cex=0.9);
par(oldpar)
})

```

CCA.qbr

*Data: Sampled Counts of Quillback Rockfish (Sebastes maliger)***Description**

Count of sampled fish-at-age for quillback rockfish (*Sebastes maliger*) in Johnstone Strait, British Columbia, from 1984 to 2004.

**Usage**

```
data(CCA.qbr)
```

**Format**

A matrix with 70 rows (ages) and 14 columns (years). Attributes “syr” and “cyr” specify years of survey and commercial data, respectively.

```

[,c(3:5,9,13,14)]    Counts-at-age from research survey samples
[,c(1,2,6:8,10:12)]  Counts-at-age from commercial fishery samples

```

All elements represent sampled counts-at-age in year. Zero-value entries indicate no observations.

## Details

Handline surveys for rockfish have been conducted in Johnstone Strait (British Columbia) and adjacent waterways (126°37'W to 126°53'W, 50°32'N to 50°39'N) since 1986. Yamanaka and Richards (1993) describe surveys conducted in 1986, 1987, 1988, and 1992. In 2001, the Rockfish Selective Fishery Study (Berry 2001) targeted quillback rockfish *Sebastes maliger* for experiments on improving survival after capture by hook and line gear. The resulting data subsequently have been incorporated into the survey data series. The most recent survey in 2004 essentially repeated the 1992 survey design. Fish samples from surveys have been supplemented by commercial hand-line fishery samples taken from a larger region (126°35'W to 127°39'W, 50°32'N to 50°59'N) in the years 1984-1985, 1989-1991, 1993, 1996, and 2000 (Schnute and Haigh 2007).

## Note

Years 1994, 1997-1999, and 2002-2003 do not have data.

## Source

Fisheries and Oceans Canada - GFBio database:

<http://www.pac.dfo-mpo.gc.ca/science/species-especes/groundfish-poissonsdesfonds/stats-eng.html>

## References

Berry, M.D. (2001) *Area 12 (Inside) Rockfish Selective Fishery Study*. Science Council of British Columbia, Project Number **FS00-05**.

Schnute, J.T. and Haigh, R. (2007) Compositional analysis of catch curve data with an application to *Sebastes maliger*. *ICES Journal of Marine Science* **64**, 218–233.

Yamanaka, K.L. and Richards, L.J. (1993) 1992 Research catch and effort data on nearshore reef-fishes in British Columbia Statistical Area 12. *Canadian Manuscript Report of Fisheries and Aquatic Sciences* **2184**, 77 pp.

## Examples

```
local(envir=.PBSmodEnv,expr={
  oldpar = par(no.readonly=TRUE)
  # Plot age proportions (blue bubbles = survey data, red = commercial)
  data(CCA.qbr,envir=.PBSmodEnv); clr=c("cornflowerblue","orangered")
  z <- CCA.qbr; cyr <- attributes(z)$cyr;
  z <- apply(z,2,function(x){x/sum(x)}); z[,cyr] <- -z[,cyr];
  x <- as.numeric(dimnames(z)[[2]]); xlim <- range(x) + c(-.5,.5);
  y <- as.numeric(dimnames(z)[[1]]); ylim <- range(y) + c(-1,1);
  expandGraph(mgp=c(2,.5,0),las=1)
  plotBubbles(z,xval=x,yval=y,powr=.5,size=0.15,clr=clr,
    xlim=xlim,ylim=ylim,xlab="Year",ylab="Age",cex.lab=1.5)
  addLegend(.5,1,bty="n",pch=1,cex=1.2,col=clr,
    legend=c("Survey","Commercial"),horiz=TRUE,xjust=.5)
  par(oldpar)
})
```

---

`chooseWinVal`*Choose and Set a String Item in a GUI*

---

**Description**

Prompts the user to choose one string item from a list of choices displayed in a GUI, then sets a specified variable in a target GUI.

**Usage**

```
chooseWinVal(choice, varname, winname="window")
```

**Arguments**

<code>choice</code>	vector of strings from which to choose
<code>varname</code>	variable name to which choice is assigned in the target GUI
<code>winname</code>	window name for the target GUI

**Details**

`chooseWinVal` activates a `setWinVal` command through an `onClose` function created by the `getChoice` command and modified by `chooseWinVal`.

**Value**

No value is returned directly. The choice is written to the PBS options workspace, accessible through `getPBSoptions("getChoice")`. Also set in PBS options is the window name from which the choice was activated.

**Note**

Microsoft Windows users may experience difficulties switching focus between the R console and GUI windows. The latter frequently disappear from the screen and need to be reselected (either clicking on the task bar or pressing `<Alt><Tab>`). This issue can be resolved by switching from MDI to SDI mode. From the R console menu bar, select `<Edit>` and `<GUI preferences>`, then change the value of "single or multiple windows" to SDI.

**Author(s)**

Rowan Haigh, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**See Also**

[getChoice](#), [getWinVal](#), [setWinVal](#)

**Examples**

```

## Not run:
local(envir=.PBSmodEnv,expr={
dfnam <-
  c("airquality","attitude","ChickWeight","faithful","freeny",
    "iris","LifeCycleSavings","longley","morley","Orange",
    "quakes","randu","rock","stackloss","swiss","trees")

wlist <- c(
  "window name=choisir title=\"Test chooseWinVal\"",
  "label text=\"Press <ENTER> in the green entry box
  \nto choose a file, then press <GO>\" sticky=W pady=5",
  "grid 1 3 sticky=W",
  "label text=File: sticky=W",
  "entry name=fnam mode=character width=23 value=\"\"",
  "func=chFile entrybg=darkolivegreen1 pady=5",
  "button text=GO bg=green sticky=W func=test",
  "")

chFile <- function(ch=dfnam,fn="fnam")
  {chooseWinVal(ch,fn,winname="choisir")};

#-- Example 1 GUI test
test <- function() {
  oldpar = par(no.readonly=TRUE); on.exit(par(oldpar))
  getWinVal(winName="choisir",scope="L")
  if (fnam!="" && any(fnam==dfnam)) {
    file <- get(fnam);
    pairs(file,gap=0); }
  else {
    resetGraph();
    addLabel(.5,.5,"Press <ENTER> in the green entry box
    \nto choose a file, then press <GO>", col="red",cex=1.5)
  }
}

#-- Example 2 Non-GUI test
#To try the non-GUI version, type 'test2()' on the command line
test2 <- function(fnames=dfnam) {
  oldpar = par(no.readonly=TRUE); on.exit(par(oldpar))
  frame();resetGraph()
  again <- TRUE;
  while (again) {
    fnam <- sample(fnames,1); file <- get(fnam);
    flds <- names(file);
    xfld <- getChoice(paste("Pick x-field from",fnam),flds,gui=FALSE);
    yfld <- getChoice(paste("Pick y-field from",fnam),flds,gui=FALSE)
    plot(file[,xfld],file[,yfld],xlab=xfld,ylab=yfld,
         pch=16,cex=1.2,col="red");
    again <- getChoice("Plot another pair?",gui=FALSE)
  }
}
require(PBSmodelling)

```

```
createWin(wlist,astext=TRUE); test();
})

## End(Not run)
```

---

`cleanProj`*Launch a GUI for Project File Deletion*

---

### Description

Launches a new window which contains an interface for deleting junk files associated with a prefix and a set of suffixes (e.g., PBSadmb project) from the working directory.

### Usage

```
cleanProj(prefix, suffix, files)
```

### Arguments

prefix	default prefix for file names.
suffix	character vector of suffixes used for clean options.
files	character vector of file names used for clean options.

### Details

All arguments may contain wildcard characters ("\*" to match 0 or more characters, "?" to match any single character).

The GUI includes the following:

- 1 An entry box for the prefix.  
The default value of this entry box is taken from `prefix`.
- 2 Check boxes for each suffix in the `suffix` argument and for each file name in the `files` argument.
- 3 Buttons marked "Select All" and "Select None" for selecting and clearing all the check boxes, respectively.
- 4 A "Clean" button that deletes files in the working directory matching one of the following criteria:
  - (i) file name matches both an expansion of a concatenation of a prefix in the entry box and a suffix chosen with a check box; or
  - (ii) file name matches an expansion of a file chosen with a check box.

### Author(s)

Anisa Egeli, Vancouver Island University, Nanaimo BC

**Examples**

```
## Not run:
local(envir=.PBSmodEnv,expr={
  cleanProj(prefix="foo",suffix=c(".a*",".b?",".c","-old.d"),files=c("red","blue"))
})

## End(Not run)
```

---

`cleanWD`*Launch a GUI for File Deletion*

---

**Description**

Launches a new window which contains an interface for deleting specified files from the working directory.

**Usage**

```
cleanWD(files)
```

**Arguments**

`files` character vector of file names used for clean options.

**Details**

All arguments may contain wildcard characters ("`*`" to match 0 or more characters, "?" to match any single character).

The GUI includes the following:

- 1 Check boxes for each suffix in the `suffix` argument and for each file name in the `files` argument.
- 2 Buttons marked "Select All" and "Select None" for selecting and clearing all the check boxes, respectively.
- 3 A "Clean" button that deletes files in the working directory matching file name expansion of files chosen with a check box.

**Author(s)**

Rowan Haigh, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**Examples**

```
## Not run:
local(envir=.PBSmodEnv,expr={
  cleanWD(c("*.bak","*.tmp","junk*"))
})

## End(Not run)
```

---

clearAll *Remove all R Objects From a Specified Environment*

---

**Description**

Generic function to clear all objects from .RData in R

**Usage**

```
clearAll(hidden=TRUE, verbose=TRUE, PBSSave=TRUE, pos=".PBSmodEnv")
```

**Arguments**

hidden	if TRUE, remove variables that start with a dot(.).
verbose	if TRUE, report all removed items.
PBSSave	if TRUE, do not remove .PBSmod.
pos	The pos argument can specify the environment in which to look for the object in any of several ways: as an integer (the position in the search list); as the character string name of an element in the search list; or as an environment (including using <code>sys.frame</code> to access the currently active function calls).

**Author(s)**

Jon T. Schnute, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

---

clearPBSext *Clear File Extension Associations*

---

**Description**

Disassociate any number of file extensions from commands previously saved with `setPBSext`.

**Usage**

```
clearPBSext(ext)
```

**Arguments**

ext	optional character vector of file extensions to clear; if unspecified, all associations are removed
-----	---

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[setPBSext](#), [getPBSext](#), [openFile](#)

---

`clearRcon`*Clear the R Console / Focus on the RGui Window*

---

## Description

Clear the R console window or focus on the RGui window using Visual Basic shell scripts.

## Usage

```
clearRcon(os=.Platform$OS.type)
focusRgui(os=.Platform$OS.type)
```

## Arguments

`os` operating system (e.g., "windows", "unix").

## Details

Creates a VB shell script file called `clearRcon.vbs` or `focusRgui.vbs` in R's temporary working directory, then executes the script using the shell command.

While `clearRcon` clears the R console, `focusRgui` returns the desktop focus back to the RGui window.

These commands will only work on Windows operating platforms, using the system's executable `%SystemRoot%\system32\cscript.exe`.

## Author(s)

Norm Olsen, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

## See Also

[cleanWD](#), [clearPBSext](#), [clearWinVal](#)

## Examples

```
## Not run:
local(envir=.PBSmodEnv,expr={
  createWin( c("window title=Focus",
    "button text=\"Go to RGui\" width=20 bg=aliceblue func=focusRgui"), astext=T)
})

## End(Not run)
```

---

clearWinVal	<i>Remove all Current Widget Variables</i>
-------------	--

---

**Description**

Remove all global variables that share a name in common with any widget variable name defined in `names(getWinVal())`. Use this function with caution.

**Usage**

```
clearWinVal()
```

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[getWinVal](#)

---

clipVector	<i>Clip a Vector at One or Both Ends</i>
------------	--

---

**Description**

Clip a vector at one or both ends using the specified clip pattern to match.

**Usage**

```
clipVector(vec, clip, end=0)
```

**Arguments**

<code>vec</code>	vector object to clip
<code>clip</code>	value or string specifying repeated values to clip from ends
<code>end</code>	end to clip <code>clip</code> from: 0=both, 1=front, 2=back

**Details**

If the vector is named, the names are retained. Otherwise, element positions are assigned as the vector's names.

**Value**

Clipped vector with names.

**Author(s)**

Rowan Haigh, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**See Also**

[createVector](#)

**Examples**

```
local(envir=.PBSmodEnv,expr={
  x=c(0,0,0,0,1,1,1,1,0,0)
  print(clipVector(x,0))

  x=c(TRUE,TRUE,FALSE,TRUE)
  print(clipVector(x,TRUE))

  x=c("red","tide","red","red")
  print(clipVector(x,"red",2))
})
```

---

closeWin

*Close GUI Window(s)*

---

**Description**

Close (destroy) one or more windows made with createWin.

**Usage**

```
closeWin(name)
```

**Arguments**

name            a vector of window names that indicate which windows to close. These names appear in the *window description file(s)* on the line(s) defining WINDOW widgets. If name is omitted, all active windows will be closed.

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[createWin](#)

---

`compileC`*Compile a C File into a Shared Library Object*

---

**Description**

This function provides an alternative to using R's SHLIB command to compile C code into a shared library object.

**Usage**

```
compileC(file, lib="", options="", logWindow=TRUE, logFile=TRUE)
```

**Arguments**

<code>file</code>	name of the file to compile.
<code>lib</code>	name of shared library object (without extension).
<code>options</code>	linker options (in one string) to prepend to a compilation command.
<code>logWindow</code>	if TRUE, a log window containing the compiler output will be displayed.
<code>logFile</code>	if TRUE, a log file containing the compiler output will be created.

**Details**

If `lib=""`, it will take the same name as `file` (with a different extension).

If an object with the same name has already been dynamically loaded in R, it will be unloaded automatically for recompilation.

The name of the log file, if created, uses the string value from `lib` concatenated with `".log"`.

**Author(s)**

Anisa Egeli, Vancouver Island University, Nanaimo BC

**See Also**

[loadC](#)

**Examples**

```
## Not run:
local(envir=.PBSmodEnv,expr={
  cwd = getwd()
  edir <- system.file("examples", package = "PBSmodelling" )
  file.copy(paste(edir,"fib.c",sep="/"), tempdir(), overwrite=TRUE)
  setwd(tempdir())
  compileC("fib.c", lib="myLib", options="myObj.o", logWindow=FALSE)
  print(list.files())
  setwd(cwd)
})
```

```
## End(Not run)
```

---

compileDescription      *Convert and Save a Window Description as a List*

---

### Description

Convert a *window description file* (ASCII markup file) to an equivalent *window description list*. The output list (an ASCII file containing R-source code) is complete, i.e., all default values have been added.

### Usage

```
compileDescription(descFile, outFile)
```

### Arguments

descFile      name of *window description file* (markup file).  
outFile      name of output file containing R source code.

### Details

The *window description file* descFile is converted to a list, which is then converted to R code, and saved to outFile.

### Author(s)

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

### See Also

[parseWinFile](#), [createWin](#)

---

convSlashes      *Convert Slashes from UNIX to DOS*

---

### Description

Convert slashes in a string from '/' to '\\\' if the operating system is 'windows'. Do the reverse if the OS is 'unix'.

### Usage

```
convSlashes(expr, os=.Platform$OS.type, addQuotes=FALSE)
```

**Arguments**

expr	String value (usually a system pathway).
os	operating system (either "windows" or "unix").
addQuotes	logical: if TRUE, enclose the string expression in escaped double quotation marks.

**Value**

Returns the input string modified to have the appropriate slashes for the specified operating system.

**Author(s)**

Rowan Haigh, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

---

createVector                      *Create a GUI with a Vector Widget*

---

**Description**

Create a basic window containing a vector and a submit button. This provides a quick way to create a window without the need for a *window description file*.

**Usage**

```
createVector(vec, vectorLabels=NULL, func="",
            windowname="vectorwindow", env=NULL)
```

**Arguments**

vec	a vector of strings representing widget variables. The values in vec become the default values for the widget. If vec is named, the names are used as the variable names.
vectorLabels	an optional vector of strings to use as labels above each widget.
func	string name of function to call when new data are entered in widget boxes or when "GO" is pressed.
windowname	unique window name, required if multiple vector windows are created.
env	an environment in which to evaluate widget callback functions.

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[createWin](#)

**Examples**

```
## Not run:
local(envir=.PBSmodEnv,expr={
#user defined function which is called on new data
drawLiss <- function() {
  oldpar = par(no.readonly=TRUE); on.exit(par(oldpar))
  getWinVal(scope="L");
  tt <- 2*pi*(0:k)/k; x <- sin(2*pi*m*tt); y <- sin(2*pi*(n*tt+phi));
  plot(x,y,type="p"); invisible(NULL); };

#create the vector window
createVector(c(m=2, n=3, phi=0, k=1000), func="drawLiss",
  vectorLabels=c("x cycles","y cycles", "y phase", "points"));
})

## End(Not run)
```

---

createWin

*Create a GUI Window*


---

**Description**

Create a GUI window with widgets using instructions from a Window Description File (aka markup file).

**Usage**

```
createWin( fname, astext=FALSE, env=NULL )
```

**Arguments**

fname	name of <i>window description file</i> or list returned from parseWinFile.
astext	logical: if TRUE, interpret fname as a vector of strings with each element representing a line in a <i>window description file</i> .
env	an environment in which to evaluate widget callback functions; see example.

**Details**

Generally, the markup file contains a single widget per line. However, widgets can span multiple lines by including a backslash ('\') character at the end of a line, prompting the suppression of the newline character.

For more details on widget types and markup file, see "PBSModelling-UG.pdf" in the R directory `.../library/PBSmodelling/doc`.

It is possible to use a Window Description List produced by compileDescription rather than a file name for fname.

Another alternative is to pass a vector of characters to fname and set astext=T. This vector represents the file contents where each element is equivalent to a new line in the *window description file*.

**Note**

Microsoft Windows users may experience difficulties switching focus between the R console and GUI windows. The latter frequently disappear from the screen and need to be reselected (either clicking on the task bar or pressing <Alt><Tab>. This issue can be resolved by switching from MDI to SDI mode. From the R console menu bar, select <Edit> and <GUI preferences>, then change the value of “single or multiple windows” to SDI.

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[parseWinFile](#), [getWinVal](#), [setWinVal](#)  
[closeWin](#), [compileDescription](#), [createVector](#)  
[initHistory](#) for an example of using `astext=TRUE`  
[environment](#)

**Examples**

```
## Not run:
# See file ../library/PBSmodelling/testWidgets/LissWin.txt
# Calculate and draw the Lissajous figure
local(envir=.PBSmodEnv,expr={
  drawLiss <- function() {
    oldpar = par(no.readonly=TRUE); on.exit(par(oldpar))
    getWinVal(scope="L"); ti=2*pi*(0:k)/k;
    x=sin(2*pi*m*ti); y=sin(2*pi*(n*ti+phi));
    plot(x,y,type=ptype); invisible(NULL); };
  createWin(system.file("testWidgets/LissWin.txt",package="PBSmodelling"));
})

#####
# Environment example:
# function in global
local(envir=.PBSmodEnv,expr={
  hello <- function() {
    stop( "I shouldn't be called" )
  }

newNameGreeter <- function( name ) {
  # method to display window
  greet <- function() {
    createWin(c("button \"Say hello\" func=hello"), astext=TRUE,
              env=parent.env(environment()))
  }
  # hello method will refer to the name in this local scope
  hello <- function() {
    cat( "Hello", name, "\n" )
  }
}
```

```

    # return functions which the user can call directly
    return( list( greet=greet, hello=hello ) )
}
alex <- newNameGreeter( "Alex" )
jon  <- newNameGreeter( "Jon" )

alex$hello() # prints hello Alex
jon$hello()  # prints hello Jon
alex$greet() # creates a GUI with a button, which will print "hello Alex" when pushed
})

## End(Not run)

```

---

declareGUIOptions      *Declare Option Names that Correspond with Widget Names*

---

### Description

This function allows a GUI creator to specify widget names that correspond to names in PBS options. These widgets can then be used to load and set PBS options using `getGUIOptions` and `setGUIOptions`.

### Usage

```
declareGUIOptions(newOptions)
```

### Arguments

`newOptions`      a character vector of option names

### Details

`declareGUIOptions` is typically called in a GUI initialization function. The option names are remembered and used for the functions `getGUIOptions`, `setGUIOptions`, and `promptSave`.

### Author(s)

Anisa Egeli, Vancouver Island University, Nanaimo BC

### See Also

[getGUIOptions](#), [setGUIOptions](#), [promptWriteOptions](#)

### Examples

```

## Not run:
local(envir=.PBSmodEnv,expr={
  declareGUIOptions("editor")
})

## End(Not run)

```

doAction

*Execute Action Created by a Widget***Description**

Executes the action expression formulated by the user and written as an ‘action’ by a widget.

**Usage**

```
doAction(act)
```

**Arguments**

act                    string representing an expression that can be executed

**Details**

If act is missing, doAction looks for it in the action directory of the window’s widget directory in .PBSmod. This action can be accessed through `getWinAct()[1]`.

Due to parsing complications, the expression act translates various symbols. The function translates:

1. The back tick character ‘`’ to a double quote ‘”’ character. For example,

```
"openFile(paste(getWinVal())$prefix,`.tpl`,sep=``))"
```

2. Underscore period ‘\_.’ to four back slashes and one period ‘\\\\\\.’. For example,

```
"poop=strsplit(`some.thing.else`,split=`_.`)"
```

**Value**

Invisibly returns the string expression expr.

**Author(s)**

Rowan Haigh, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**See Also**

[createWin](#), [evalCall](#), [getWinAct](#)

**Examples**

```
## Not run:
local(envir=.PBSmodEnv,expr={
  createWin("button text=`list objects`" func=doAction action=print(ls(all.names=TRUE))),
  astext=TRUE)
})

## End(Not run)
```

---

dot-c.support.funs      *Dot C Support Functions*

---

## Description

Various dot C-support functions used by PBSmodelling code internally.

## Usage

```
.libName(lib="")  
.guiSource()  
.guiCompileC()  
.guiDyn()  
.cleanLoadC()  
.loadCRunComparison()
```

## Arguments

lib                      character – vector of file names without extensions

## Details

.libName	given a character vector of shared library object names, returns the file names with the appropriate extension for the user's platform (.dll for Windows or .so for Unix)
.guiSource	sources the .r file in the working directory indicated by the prefix entry widget in the GUI
.guiCompileC	gets the prefix and libPrefix arguments from the GUI and uses them to call .compileC
.guiDyn	based on the previous GUI action, either tries to load or unload the library with the lib prefix specified in the GUI (or the project file prefix if this is left blank)
.cleanLoadC	clean function
.loadCRunComparison	runs the provided C and R functions a number of times specified in the GUI and writes into text boxes the elapsed time for each

## Value

Varied lists and other objects to facilitate GUI manipulation.

## Author(s)

Anisa Egeli, Software Developer, Alumnus of Vancouver Island University, Nanaimo BC

Maintainer: **Rowan Haigh**, Program Head – Offshore Rockfish

Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC

*locus opus*: remote office, Vancouver BC

Last modified Rd: 2024-11-04

**See Also**

In package **PBSmodelling**:  
[loadC](#), [compileC](#)

---

dot-convert.funs      *Dot Convert Functions*

---

**Description**

Various dot-convert functions used by PBSmodelling code internally.

**Usage**

```
.convertMatrixListToDataFrame(mList, colName="Y", rowNames="X")
.convertMatrixListToMatrix(mList)
.convertMode(x, mode)
.convertOldGridToNewGrid(grid)
.convertParamStrToList(x, fname="", line.start=0, line.end=0, sourcefile=list())
.convertParamStrToVector(x, fname="", line=0)
.convertVecToArray(x, d, byright=FALSE, byrow=TRUE)
```

**Arguments**

mList	list – matrix list object
colName	character – name of column
rowNames	character – vector of row names
x	character numeric – string object or vector of data to use to create array
mode	various – infer: mode of object
grid	numeric – infer: grid Polyset
fname	character – name of file for warning messages
line.start	numeric – line number where widget is first found
line.end	numeric – line number of last line of widget (ie extended line)
sourcefile	list – list object
line	numeric – line number for warning messages
d	numeric – dimensions of the array
byright	logical – if TRUE, vary indices by the most right number first
byrow	logical – if TRUE, process indices by row?

## Details

Various convert functions used by PBSmodelling code.

`.convertMatrixListToMatrix` – Convert a list into an N-dim array

`.convertMatrixListToDataFrame` – Convert a list into a data frame

`.convertMode` – Convert a variable into a mode without showing any warnings (deprecated?)

`.convertOldGridToNewGrid` – Convert old style (where `grid` has a `grid$.widgets[[i]][[j]]` data structure) to new format: `list( grid, child, child, ..., child )`; this function should never be used by new code (hopefully it will be removed one day)

`.convertParamStrToList` – Convert a given string of values separated by spaces into a list while preserving space and escaped quotes within quotes (kindof – the value must still be stripped depending if its a single string, or vector of strings)

`.convertParamStrToVector` – Convert a string, `x`, into a vector of elements separated by whitespace; whitespace can be interrupted as values if it is enclosed by quotes; special characters (newline, tab, `\`, `'`, `"`) must be escaped with `\`

`.convertVecToArray` – Convert a vector to an array

## Author(s)

**Alex Couture-Beil**, Software Engineer  
Earthly Technologies, Victoria BC

Maintainer: **Rowan Haigh**, Program Head – Offshore Rockfish  
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC  
*locus opus*: remote office, Vancouver BC  
Last modified Rd: 2024-10-30

## See Also

In **PBSmodelling**:

[.createWidget.data](#), [.createWidget.history](#), [.createWidget.matrix](#), [.extractVar](#), [.getParamFromStr](#),  
[.readList.P.convertData](#), [.superobject.saveValues](#)

---

dot-create.funs

*Dot Create Functions*

---

## Description

Various dot-create functions used by PBSmodelling code internally.

**Usage**

```

.createTkFont(fontstr)
.createWidget(tk, widgetList, winName)
.createWidget.button(tk, widgetList, winName)
.createWidget.check(tk, widgetList, winName)
.createWidget.data(tk, widgetList, winName)
.createWidget.droplist(tk, widgetList, winName)
.createWidget.entry(tk, widgetList, winName)
.createWidget.grid(tk, widgetList, winName)
.createWidget.image(tk, widgetList, winName)
.createWidget.include(tk, widgetList, winName)
.createWidget.label(tk, widgetList, winName)
.createWidget.matrix(tk, widgetList, winName)
.createWidget.notebook(tk, widgetList, winName)
.createWidget.null(tk, widgetList, winName)
.createWidget.object(tk, widgetList, winName, userObject=NULL )
.createWidget.object.scrolling(tk, widgetList, winName)
.createWidget.progressbar(tk, widgetList, winName)
.createWidget.radio(tk, widgetList, winName)
.createWidget.slide(tk, widgetList, winName)
.createWidget.slideplus(tk, widgetList, winName)
.createWidget.spinbox(tk, widgetList, winName)
.createWidget.table(tk, widgetList, winName)
.createWidget.text(tk, widgetList, winName)
.createWidget.vector(tk, widgetList, winName)

```

**Arguments**

fontstr	character – string describing a font and colour
tk	character – frame to attach widget to
widgetList	character – widget list
winName	character – active window name
userObject	numeric character – matrix or data frame

**Details**

Various create functions used by PBSmodelling code.

```

.createTkFont – create a usable TK font from a given string
.createWidget – Generic function to create most widgets, which calls appropriate
.createWidget.<widgetType>() function

```

**Author(s)**

**Alex Couture-Beil**, Software Engineer, Earthly Technologies, Victoria BC

Maintainer: **Rowan Haigh**, Program Head – Offshore Rockfish

Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC

*locus opus*: remote office, Vancouver BC  
 Last modified Rd: 2024-10-31

## See Also

In **PBSmodelling**:  
[testWidgets](#), [showArgs](#), [.createWidget.history](#)

---

 dot-gui.funs

*Dot GUI Functions*


---

## Description

Various dot functions used by PBSmodelling code internally to support the GUI functions.

## Usage

```
.buildgrid(tk, grid, winName, childWidgets)
.map.add(winName, key, ...)
.map.get(winName, key)
.map.getAll(winName)
.map.init(winName)
.map.set(winName, key, ...)
.setMatrixElement(m, a, ind=NULL)
.getMatrixListSize(m, d=NULL, big=0)
.matrixHelp(matrixList, ind, value)
.autoConvertMode(x)
.extractData(command, action, winName)
.extractFuns(data)
.extractVar(winName)
.getParamFromStr(inputStr, fname="", line.start=0, line.end=0,
sourcefile=list(), paramOrder=.widgetDefs)
.table.getvalue(winName, widgetName)
.table.setvalue(winName, widgetName, value)
.getValueForWidgetSetup(varname, widget, winName)
.getSimilarColour(col)
.adjustAllColours(tObj)
.do.gui.call(what, args, quote=FALSE, envir=parent.frame())
.catError(err, fname, line.start, line.end, sourcefile=list(),
errorType="GUI parse error")
.catError2(err, fname, line.start)
.check.object.exists(tk, widget, winName)
.inCollection(haystack, needle)
.packWidgetsIntoGrid(widgets, vertical=TRUE)
.parsegrid(data, nRow, nCol)
.parsemenu(data, nItems)
.PBSdimnameHelper(rownames, colnames, dim)
```

```

.searchCollection(haystack, needle)
.setWinValHelper(varname, value, winName)
.stopWidget(err, wid.debug, winName)
.stripComments(x)
.stripSlashes(x, fname="", line.start=0, line.end=0, sourcefile=list())
.stripSlashesVec(x, fname="", line.start=0, line.end=0, sourcefile=list())
.superobject.redraw(winName, widget_name)
.superobject.saveValues(winName, widget_name)
.trimWhiteSpace(x)
.validateWindowDescList(x)
.validateWindowDescWidgets(x)

```

### Arguments

tk	tcltk – parent tk frame to attach widget to
grid	character – widget list describing the grid
winName	character – active or target window name   map to process
childWidgets	character – infer: widgets used by other widgets
key	character – name of item to extract (i.e., widget name)
...	dots – named items to save (in a list)
m	numeric – infer: a matrix
a	numeric – infer: an array
ind	numeric – infer: index value
d	numeric – infer: a dimension
big	numeric – infer: minimum size of array dimension
matrixList	numeric – infer: matrix list object
value	numeric – infer: value to populate the matrix   list
x	character – string resembling a numeric   string with or without comments   string to trim   list to validate
command	character – user command to call (i.e., function argument of widget)
action	character – action value
data	character – list of widget lists
inputStr	character – string from win desc file describing a widget
fname	character – file name to display with error messages
line.start	numeric – line number where widget is first found
line.end	numeric – line number of last line of widget (i.e., extended line)
sourcefile	character – source code of the file in question
paramOrder	unknown – infer: parameter order
widgetName	character – infer: name of a widget
varname	character – infer: variable name
widget	list – widget list passed to .createWidget.<widget>

col	character numeric – the colour to adjust
tObj	tcltk – Tk object
what	unknown – infer: what what what
args	character – infer: a list of arguments
quote	logical – infer: if TRUE, add   use quotation marks
envir	environment – infer: environment where whatever awaits 'what' works
err	character – error string to display
errorType	character – type of error to display
haystack	character – a vector to search
needle	character – a single element to search for
widgets	list – infer: widget list
vertical	logical – infer: if TRUE, pack widgets vertically (top to bottom)
nRow	numeric – number of grid rows
nCol	numeric – number of grid columns
nItems	numeric – number of children to select for the menu
rownames	character – vector of size 1, or dim[1] naming the rows; if only one name is given, a number (1..dim[1]) will be appended to the name
colnames	character – vector of size 1 or dim[2] naming columns; if only one name is given, then (1..dim[2]) is appended
dim	character – vector of size 2, dim[1] is nRows, dim[2] is nCols
wid.debug	list – list of widget codes (created in parsing process)
widget_name	character – infer: name of a widget

## Details

### Create——

`.buildgrid` used to create a grid on a window

### Map——

`.map.add` save a new value for a given key; if a previous exists, ignore the new value and return previous value

`.map.get` returns a value associated with a key

`.map.getAll` returns all visible items of a map of a certain window

`.map.init` initialize the data structure that holds the map(s)

`.map.set` save a new value for a given key, even if it involves overwriting a previously stored value

### Convert——

`.setMatrixElement` helper function used by `.convertMatrixListToMatrix` to assign values from the matrix list into the array

`.getMatrixListSize` helper function used by `.convertMatrixListToMatrix` to determine the minimum required size of the array needed to create or convert the list into an array

`.matrixHelp` used to help `.extractVar` deal with N-dim matrices; firstly it is

<code>.autoConvertMode</code>	converted into a “matrix list” once the matrix list is completed (and size known), it should be converted into a true array converts x into a numeric mode, if it looks like a valid number
<b>Extract</b>	
<code>.extractData</code>	called by TK on button presses (or binds, on changes, slides, ...)
<code>.extractFuns</code>	get a list of called functions
<code>.extractVar</code>	extracts values from the tclvar pointers
<code>.getParamFromStr</code>	returns a list with all parameters extracted from a list
<code>.table.getvalue</code>	infer: get a value from a table
<code>.table.setvalue</code>	infer: set a value in a table
<code>.getValueForWidgetSetup</code>	returns the value(s) to use for initializing a widget from: -> widget\$value (if set) or -> global var matching widget\$name when value is NULL
<b>Helper</b>	
<code>.getSimilarColour</code>	obtains a similar colour (slightly brighter/darker) than the argument
<code>.adjustAllColours</code>	for the passed Tk widget, attempts to slightly modify all applicable colours
<code>.do.gui.call</code>	extends do.call, which is used to create most Tk widgets, to immediately adjust a new widget’s colours (to get around how tk_setPalette changes the colour of existing widgets)
<code>.catError</code>	used to display parsing errors
<code>.catError2</code>	infer: a variation of .catError
<code>.check.object.exists</code>	call to test for existence of dynamically loaded object for “object” and “table”; returns NULL on no errors; otherwise a tk display error is created (can be embedded in window)
<code>.inCollection</code>	returns true if needle occurs in haystack
<code>.packWidgetsIntoGrid</code>	pack all widgets into a grid with ncol=1 nrow=<number of widgets>
<code>.parsegrid</code>	returns two items in a list: -> \$gridData, which is a list of lists representing columns -> \$unparsedData, which is left over from the grid and still needs parsing
<code>.parsemenu</code>	very similar to .parsegrid but for menus; set up a menu with children menus or menuitems
<code>.PBSdimnameHelper</code>	adds dimnames to stuff (matrix, data.frame)
<code>.searchCollection</code>	searches a haystack for a needle, or a similar longer needle
<code>.setWinValHelper</code>	infer: help to set GUI Window values
<code>.stopWidget</code>	stop due to fatal error during window creation (not parse)
<code>.stripComments</code>	removes any trailing comments from a line, but ignores '#' in quoted strings
<code>.stripComments</code>	infer: strips out comments from a set of strings
<code>.stripSlashes</code>	removes slashes from a string
<code>.stripSlashesVec</code>	given a string, split it into a vector of words, which were separated by spaces; however, if single quotes are used, space is preserved; e.g., x="a b 'c d'" converts into "a" "b" "c d"
<code>.superobject.redraw</code>	infer: redraw a superobject (whatever that is)
<code>.superobject.saveValues</code>	must be outside of .createWidget.superobject since getWinVal must be able to call this
<code>.trimWhiteSpace</code>	remove leading and trailing whitespace
<code>.validateWindowDescList</code>	determines if the list represents a valid PBS Modelling description list; if any required fields are missing, it will halt via stop(); if any fields are omitted which have default values defined in the

`.validateWindowDescWidgets` `.widgetDefs` list, then those fields and values will be set used by `.validateWindowDescList` to validate each widget

### Value

Stuff and such.

### Author(s)

**Alex Couture-Beil**, Software Engineer, Earthly Technologies, Victoria BC

**Nicholas M. Boers**, Software Engineer, Jobber, Edmonton AB

Maintainer: **Rowan Haigh**, Program Head – Offshore Rockfish  
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC  
*locus opus*: remote office, Vancouver BC  
Last modified Rd: 2025-05-26

### See Also

In package **PBSmodelling**:  
[dot-create.funs](#), [dot-convert.funs](#)

---

dot-history.funs      *Dot History Functions*

---

### Description

Various dot functions used by PBSmodelling code internally for the history widget.

### Usage

```
.createWidget.history(tk, widgetList, winName)
.sortHelper()
.sortHelperActive(hisname)
.sortHelperFile(openfile, savefile)
.sortWidget(d, hisname)
.updateFile()
.updateHistory(hisname)
.updateHistoryButtons(hisname)
```

## Arguments

tk	character – frame to attach widget to
widgetList	character – widget list
winName	character – active window name
hisname	character – history instance name if multiple are active
openfile	character – file name of repository containing windows GUI settings
savefile	character – file name of repository for windows GUI settings
d	numeric character – data.frame object

## Details

Various dot functions used by PBSmodelling code to control the history widget.

.createWidget.history – create a history widget  
.sortHelper – sort entries in an existing history instance  
.updateHistory – update widget values

## Author(s)

**Alex Couture-Beil**, Software Engineer  
Earthly Technologies, Victoria BC

Maintainer: **Rowan Haigh**, Program Head – Offshore Rockfish  
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC  
*locus opus*: remote office, Vancouver BC  
Last modified Rd: 2024-10-31

## See Also

In **PBSmodelling**:  
[testWidgets](#), [showArgs](#)

---

dot-list.funs

*Dot List Functions*

---

## Description

Various dot-list functions used by PBSmodelling code internally.

## Usage

```
.mergeLists(x, y)
.readList.C(fname)
.readList.P(fname)
.readList.P.convertData(varOptions, varData, fname="", sourcefile=list())
.writelist.P(x, fname="", comments, prefix="")
```

**Arguments**

x	character – element x in a list object   data to write to file
y	character – element y in a list object
fname	character – name of file to read or write
varOptions	compound – options for a variable
varData	character – data for a variable
sourcefile	character – using in cat statements (not sure)
comments	character – include string as comment at the top of file
prefix	character – prefix for line entries

**Details**

Various dot functions used by PBSmodelling code to control list objects.

`.mergelists` – taken from R Curl : merge x and y, if elements match in both x and y, then elements from y are stored (overwriting x's elements).

`.readList.C` – read ADMB-like input file and create a list; adapted from 'readAD' in PBSawatea.

`.readList.P` – read list in 'P' format (PBS format).

`.readList.P.convertData` – helper function to convert data into proper mode.

`.writelList.P` – saves list x to disk using 'P' format (PBS format).

**Author(s)**

**Alex Couture-Beil**, Software Engineer  
Earthly Technologies, Victoria BC

Maintainer: **Rowan Haigh**, Program Head – Offshore Rockfish  
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC  
*locus opus*: remote office, Vancouver BC  
Last modified Rd: 2024-10-31

**See Also**

In **PBSmodelling**:

[lisp](#), [packList](#), [readList](#), [unpackList](#), [writeList](#), [.pFormatDefs](#)

---

`dot-PBSmodEnv`*PBSmodelling Environment*

---

**Description**

An environment set aside for PBSmodelling.

**Usage**

```
.PBSmodEnv
```

**Format**

A new environment with a `.GlobalEnv` parent.

**Details**

The environment is created in `'zzz.r'` and is used by PBSmodelling functions `'lisp'`, `'tget'`, `'tput'`, `'tprint'`, and `'tcall'`.

**Source**

Generated by a call to the base function `new.env()`.

**See Also**

In **PBSmodelling**:  
[lisp](#), [tget](#), [packList](#),

**Examples**

```
lisp(.PBSmodEnv)
```

---

`dot-project.funs`*Dot Project Functions*

---

**Description**

Various dot functions used by PBSmodelling code internally to support project functions.

**Usage**

```

.doClean()
.doCleanWD()
.getHiddenEnv()
.getHome()
.getPrefix(quiet=FALSE)
.makeCleanVec(vecName, items, rowLen)
.mergeLists2(list1, list2)
.mergeVectors(v1, v2)
.optionsNotUpdated()
.removeFromList(l, items)
.selectCleanBoxes()
.setOption(option, type)
.showLog(logText, fname, noWindow=FALSE, width=80, height=30)
.stripExt(x)
.tryOpen(filename, quiet=FALSE)

```

**Arguments**

quiet	logical – if TRUE, no errors/alerts will be displayed
vecName	character – infer: name of vector
items	character – items in vector to be cleaned   vector of names of components to remove
rowLen	numeric – infer: number of rows in vector if dimensioned
list1	list – first list
list2	list – second list
v1	numeric – first vector
v2	numeric – second vector
l	list – a list
option	character – infer: name of option
type	character – either "dir" or "file" for each function
logText	character – the text for the log
fname	character – name for log file (optional)
noWindow	logical – if TRUE, log window will not be shown
width	numeric – width of log window
height	numeric – height of log window
x	character – vector of filenames, e.g., "foo.c"
filename	character – the file to open

**Details**

<code>.doClean</code>	used by <code>cleanProj()</code> ; function called when Clean button is pressed
<code>.doCleanWD</code>	Anisa's <code>.doClean</code> function modified for file names only
<code>.getHiddenEnv</code>	allow GUIs from outside PBSmodelling to call PBSmodelling's internal hidden functions
<code>.getHome</code>	returns platform dependent home drive (Windows) or user home (Unix)
<code>.getPrefix</code>	get the project prefix from the focused GUI; used to standardise error message
<code>.makeCleanVec</code>	used by <code>cleanProj()</code> to create strings describing checkbox vectors
<code>.mergeLists2</code>	add a second list to a first list; if both lists share any keys, the values from the second list overwrite those in the first list; order of components in list is preserved
<code>.mergeVectors</code>	add a second vector to a first vector; if both vectors share any values, the resulting vector will only contain this value once
<code>.optionsNotUpdated</code>	check if any of the options given by <code>declareGUIoptions</code> are different in the GUI from their stored values; a blank GUI entry is equivalent to the option not being set
<code>.removeFromList</code>	remove components of a list with the given names; "NA" can be used to remove NA names
<code>.selectCleanBoxes</code>	used by <code>cleanProj</code> ; selects or deselects all of the checkboxes
<code>.setOption</code>	used for <code>setPathOption</code> and <code>setFileOption</code> ; see these functions for a descriptions
<code>.showLog</code>	given output for a log, will make pop-up log window containing this output and/or write this output to a logfile; "\r" is stripped from the log text for display
<code>.stripExt</code>	remove file extension from end of filename
<code>.tryOpen</code>	tries to open a given file using an editor entered by the GUI; if an editor wasn't set, tries to open using <code>openFile</code> ; appropriate alerts are shown if quiet isn't turned on

**Value**

Varied lists and other objects to facilitate GUI manipulation.

**Author(s)**

Anisa Egeli, Software Developer, Alumnus of Vancouver Island University, Nanaimo BC

Maintainer: **Rowan Haigh**, Program Head – Offshore Rockfish

Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC

*locus opus*: remote office, Vancouver BC

Last modified Rd: 2024-11-01

**See Also**

In package **PBSmodelling**:

[cleanProj](#), [cleanWD](#), [findPrefix](#), [getGUIoptions](#), [setGUIoptions](#)

**Description**

Various dot support functions used by PBSmodelling code internally.

**Usage**

```
.fibCall(n, len=1)
.fibC(n, len=1)
.fibR(n, len=1)
.fibClosedForm(n, len=1)
.viewPkgDemo()
.dClose()
.dUpdateDesc()
.viewPkgVignette()
.addslashes(x, asvector=FALSE )
.findSquare(nc)
.forceMode(x, mode)
.getArrayPts(d, byright=FALSE, byrow=TRUE)
.initPBSoptions()
.mapArrayToVec(x, d, byright=FALSE)
.tclArrayToVector(str)
```

**Arguments**

n	numeric – calculate the nth number
len	numeric – return the last len calculated numbers
x	character - string to escape   variable to convert   array index (numeric vector)
asvector	logical – if TRUE, always force strings to be printed as character vector, e.g., "h w" becomes "'h w'"
nc	numeric – number of panels (figures) in a multi-panel plot
mode	various – mode to convert to
d	numeric – dimensions of the array
byright	logical – if TRUE, vary indices by the most right number first
byrow	logical – if TRUE, populate array by row
str	character – infer: a tcl string

**Details**

.fibCall	used by calcFib() to calculate a vector containing fibonacci numbers
.viewPkgDemo	display a GUI to display something equivalent to R's demo()

<code>.dClose</code>	function to execute on closing <code>runDemos()</code>
<code>.dUpdateDesc</code>	update demo window with demo descriptions
<code>.viewPkgVignettes</code>	display a GUI to display something equivalent to R's <code>vignette()</code>
<code>.addslashes</code>	escape special characters from a string, which can then be used in the "P" format; if <code>x</code> has more than one element, then it will returned as a nested <code>characterVector</code> e.g., <code>c("it's", "O K")</code> becomes <code>"'it\'s' 'O K'"</code>
<code>.convertVecToArray</code>	convert a vector to an array
<code>.findSquare</code>	calculate number of rows and columns from a single number of panels
<code>.forceMode</code>	force a variable into a mode without showing any warnings
<code>.getArrayPts</code>	return all possible indices of an array
<code>.initPBSoptions</code>	called from <code>zzz.r's .First.lib()</code> initialization function
<code>.mapArrayToVec</code>	determine which index to use for a vector, when given an N-dim index of an array
<code>.tclArrayToVector</code>	infer: change tcl code to represent array as a vector

## Value

Varied lists and other objects to facilitate GUI manipulation.

## Author(s)

**Alex Couture-Beil**, Software Engineer, Earthly Technologies, Victoria BC

Maintainer: **Rowan Haigh**, Program Head – Offshore Rockfish

Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC

*locus opus*: remote office, Vancouver BC

Last modified Rd: 2024-11-04

## See Also

In package **PBSmodelling**:

[calcFib](#), [runDemos](#), [showVignettes](#), [readPBSoptions](#), [runExamples](#)

---

dot-talk.funs

*Dot Talk Functions*

---

## Description

Various dot-talk functions used by `PBSmodelling` code internally for `'presentTalk()'`.

## Usage

```
.getButton(talk_name, obj)
.getButtons(talk)
.getIndexForSection(talk, section_id)
.getMenus(talk)
.getSectionNames(talk)
```

```

.getTalkIndexes(talk)
.nextSlide()
.parseTalkFile(talk_fname)
.presentTalkOpenFile()
.prevsSlide()
.sectiondrop()
.setsection()
.slidedrop()
.startSlide(talk)
.updateSlide(talk)

```

### Arguments

talk_name	character – name of talk (e.g., 'Swiss')
obj	character – talk component (e.g., 'section', 'file')
talk	compound – s4 object called 'talk'
section_id	character – section identification marker
talk_fname	character – name of talk file (e.g., 'swisstalk.xml')

### Details

Various dot functions used by PBSmodelling code to control the talk GUI for presentations.

```

.getButton – returns win description for a button for a file or section.
.getButtons – gets widget descriptions for file and section buttons.
.getIndexForSection – get the slide index which corresponds to the first slide in a given section.
.getMenus – get widget description for menus.
.getSectionNames – given a talk, return a vector of all section names.
.getTalkIndexes – returns a list of 2-element vectors (i, j), where i is the section index, and j is
the items index; each element of the list corresponds to a break point.
.parseTalkFile – returns a talk s4 class, given a filename to an xml talk
.presentTalkOpenFile – open files from the win act, and supports multiple files.

```

### Note

The function 'presentTalk' offers a cumbersome system for presenting talking points using R. It is likely superceded by more recent appliances like [Shiny](#).

### Author(s)

[Alex Couture-Beil](#), Software Engineer  
Earthly Technologies, Victoria BC

Maintainer: [Rowan Haigh](#), Program Head – Offshore Rockfish  
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC  
*locus opus*: remote office, Vancouver BC  
Last modified Rd: 2024-10-31

**See Also**

In **PBSmodelling**:  
[presentTalk](#), [.tagDefs](#), [.tagOptionChecks](#)

dot-widget.defs

*Dot Widget Definitions***Description**

Primary controller of the various widgets in PBSmodelling.

**Usage**

```
.widgetDefs
.pFormatDefs
.tagDefs
.tagOptionChecks
.maxActionSize
.regex.complex
.regex.numeric
.regex.logical
```

**Format**

List of 25 widgets:

window	list 8	grid	list 19	notebook	list 18	image	list 7	progressbar	list 15
menu	list 6	menuitem	list 7	label	list 14	droplist	list 19	spinbox	list 21
table	list 15	null	list 4	include	list 3	entry	list 22	button	list 13
check	list 15	radio	list 15	slide	list 15	slideplus	list 18	vector	list 25
matrix	list 28	data	list 28	object	list 23	history	list 13	text	list 16

**Details**

Storage list that specifies characteristics of each widget's arguments.

.widgetDefs	defines allowed widget parameters names and default values
.pFormatDefs	defines allowed P format parameters names and default values to be used in 'readList' (when P format is detected)
.tagDefs	defines formatting for each valid tag used in 'presentTalk', but presumably could be applied to other appliances
.tagOptionChecks	provides validation check for each tag option
.maxActionSize	maximum amount of actions to save in memory (default=50)
.regex.complex	catches all valid complex except it also catches "--"
.regex.numeric	catches numeric strings, but also catches "--"
.regex.logical	catches all logical values

**See Also**

In package **PBSmodelling**:  
[testWidgets](#), [showArgs](#), [presentTalk](#), [readList](#)

**Examples**

```
str(.widgetDefs,1)
```

---

dot-win.funs

*GUI Windows Wrappers*

---

**Description**

Wrapper dot functions to run existing or temporary functions from a GUI's function call.

**Usage**

```
.win.chFile()  
.win.chTest()  
.win.closeALL()  
.win.closeChoice()  
.win.closeSDE()  
.win.makeChoice()  
.win.restoreCWD()  
.win.runExHelperQuit()  
.win.tcall()  
.win.tget()  
.win.tprint()
```

**Details**

PBSmodelling Windows (Graphical User Interfaces or GUIs) call wrapper functions that call non-GUI functions in the package.

**Author(s)**

**Rowan Haigh**, Program Head – Offshore Rockfish  
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC  
*locus opus*: remote office, Vancouver BC  
Last modified Rd: 2023-10-25

**See Also**

In **PBSmodelling**:  
[getWinAct](#), [getWinVal](#), [runExamples](#), [tget](#)

---

drawBars	<i>Draw a Linear Barplot on the Current Plot</i>
----------	--

---

**Description**

Draw a linear barplot on the current plot. Fill if desired.

**Usage**

```
drawBars(x, y, width, base=0, fill=NULL, ...)
```

**Arguments**

x	x-coordinates
y	y-coordinates
width	bar width, computed if missing
base	y-value of the base of each bar
fill	valid R colour to fill the bars
...	further graphical parameters (see par) for the lines function.

**Author(s)**

Jon T. Schnute, Scientist Emeritus,  
Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**Examples**

```
local(envir=.PBSmodEnv,expr={
  if (dev.cur(>1) {
    oldpar=par(no.readonly=TRUE); on.exit(par(oldpar)) }
  plot(0:10,0:10,type="n")
  drawBars(x=1:9,y=9:1,col="deepskyblue4",fill="cyan",lwd=3)
})
```

---

evalCall	<i>Evaluate a Function Call</i>
----------	---------------------------------

---

**Description**

Evaluates a function call after resolving potential argument conflicts.

**Usage**

```
evalCall(fn, argu, ..., envir = parent.frame(),
  checkdef=FALSE, checkpar=FALSE)
```

**Arguments**

fn	R function
argu	list of explicitly named arguments and their values to pass to fn.
...	additional arguments that a user might wish to pass to fn.
envir	environment from which the call originates (currently has no use or effect).
checkdef	logical: if TRUE, gather additional formal arguments from the functions default function.
checkpar	logical: if TRUE, gather additional graphical arguments from the list object par.

**Details**

This function builds a call to the specified function and executes it. During the build, optional arguments (...) are checked for

- (i) duplication with explicit arguments argu: if any are duplicated, the user-supplied arguments supersede the explicit ones;
- (ii) availability as usable arguments in fn, fn.default if checkdef=TRUE, and par if checkpar=TRUE.

**Value**

Invisibly returns the string expression of the function call that is passed to eval(parse(text=expr)).

**Note**

Sometimes the user may wish to pass arguments into a function to be used by other functions within, but may not want all the arguments to be used, depending on the functions subsequently called. In this case, the user needs to create a list object called dots, which is passed to evalCall.

For instance, if the user passes lwd=4 but only wants this used in a call to lines but not in a call to points, the function might look like this:

```
myfunc = function(x=seq(0,360,5), ...) {
  pdots = ldots = list(...)
  pdots[["lwd"]] = NULL
  ldots[["col"]] = "cyan"
  xrad = x*pi/180
  plot(sin(xrad), type="n")
  evalCall(lines, argu=list(x=sin(xrad)), dots=ldots, checkpar=TRUE)
  evalCall(points, argu=list(x=sin(xrad)), dots=pdots, checkpar=TRUE)
}
myfunc(lwd=4, pch=20, col=" blue")
```

**Author(s)**

Rowan Haigh, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**See Also**

[doAction](#), [plotAsp](#)

**Examples**

```

local(envir=.PBSmodEnv,expr={
  oldpar = par(no.readonly=TRUE)
  # A user may have a function that calls other functions
  # using specific defaults (e.g., blue triangles)
  #-----
  pbsfun = function(..., use.evalCall=TRUE) {
    plotAsp(0,0,type="n",xlim=c(-1.5,1.5),ylim=c(-1.5,1.5),
            axes=FALSE, frame.plot=TRUE, xlab="",ylab="")
    if (use.evalCall)
      evalCall(polygon, ...,
              argu=list(x=c(-1,1,0),y=c(1,1,-1), col="dodgerblue", border="grey"))
    else
      polygon(x=c(-1,1,0),y=c(1,1,-1),col="dodgerblue",border="grey",...)
  }
  par(mfrow=c(2,1))
  pbsfun(lwd=4,use.evalCall=FALSE)
  #-----
  # But what if the user wants pink triangles?
  pbsfun(col="pink",lwd=4,use.evalCall=TRUE,checkpar=TRUE)
  par(oldpar)
})

# Without 'evalCall' an error occurs due to duplicated arguments
## Not run: pbsfun(col="pink",lwd=4,use.evalCall=FALSE)

```

---

 expandGraph

*Expand the Plot Area by Adjusting Margins*


---

**Description**

Optimize the plotting region(s) by minimizing margins.

**Usage**

```
expandGraph(mar=c(4,3,1.2,0.5), mgp=c(1.6,.5,0),...)
```

**Arguments**

mar	numerical vector of the form 'c(bottom, left, top, right)' specifying the margins of the plot
mgp	numerical vector of the form 'c(axis title, axis labels, axis line)' specifying the margins for axis title, axis labels, and axis line
...	additional graphical parameters to be passed to par

**Author(s)**

Jon T. Schnute, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**See Also**[resetGraph](#)**Examples**

```

local(envir=.PBSmodEnv,expr={
  oldpar = par(no.readonly=TRUE)
  expandGraph(mfrow=c(2,1));
  tt=seq(from=-10, to=10, by=0.05);

  plot(tt,sin(tt), xlab="this is the x label", ylab="this is the y label",
        main="main title", sub="sometimes there is a \"sub\" title")
  plot(cos(tt),sin(tt*2), xlab="cos(t)", ylab="sin(2 t)", main="main title",
        sub="sometimes there is a \"sub\" title")
  par(oldpar)
})

```

---

 exportHistory

*Export a Saved History*


---

**Description**

Export the current history list.

**Usage**

```
exportHistory(hisname="", fname="")
```

**Arguments**

hisname	name of the history list to export. If set to "", the value from <code>getWinAct()[1]</code> will be used instead.
fname	file name where history will be saved. If it is set to "", a <Save As> window will be displayed.

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**
[importHistory](#), [initHistory](#)

---

`findPat`*Search a Character Vector to Find Multiple Patterns*

---

**Description**

Use all available patterns in `pat` to search in `vec`, and return the matched elements in `vec`.

**Usage**

```
findPat(pat, vec)
```

**Arguments**

<code>pat</code>	character vector of patterns to match in <code>vec</code>
<code>vec</code>	character vector where matches are sought

**Value**

A character vector of all matched strings.

**Author(s)**

Rowan Haigh, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**Examples**

```
local(envir=.PBSmodEnv,expr={
  #find all strings with a vowel, or that start with a number
  print(findPat(c("[aeiou]", "^[0-9]"), c("hello", "WORLD", "1over")))
})
```

---

`findPrefix`*Find a Prefix Based on Names of Existing Files*

---

**Description**

Find the prefixes or suffixes of files with a given suffix or prefix in a directory.

**Usage**

```
findPrefix(suffix,path=".")
findSuffix(prefix,path=".")
```

**Arguments**

suffix	character vector of suffixes
prefix	character vector of prefixes
path	directory to look for files in

**Details**

The function `findPrefix` locates all files in a directory that end with one of the provided suffixes; where as `findSuffix` locates all files that start with the given prefixes.

**Value**

A character vector of all the prefixes or suffixes of files in the working directory that matched to one of the given suffixes.

**Author(s)**

Anisa Egeli, Vancouver Island University, Nanaimo BC

**Examples**

```
local(envir=.PBSmodEnv,expr={
  edir = system.file("examples", package="PBSmodelling")
  print(findPrefix(suffix=c(".txt", ".r"),path=edir)); cat("\n")

  #or use R's dir for similar functionality
  print(dir(pattern="txt$",path=edir)); cat("\n")
  print(dir(pattern="^[a-h]",path=edir)); cat("\n")
})
```

---

findProgram

*Locates a program in the PATH environment variable*

---

**Description**

Returns the complete filename and path of a program in the PATH environment variable. This is a wrapper for `Sys.which`, and may be deprecated in the future.

**Usage**

```
findProgram( name, includename=FALSE )
```

**Arguments**

name	name of a program to locate
includename	boolean: if true, include the filename in the path returned, otherwise just the directory.

**Value**

A string containing the location of the program. NULL is returned if the program is not located.

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[Sys.which](#)

**Examples**

```
local(envir=.PBSmodEnv,expr={
  print(list(
    gcc = findProgram( "gcc" ),
    notepad = findProgram( "notepad" ),
    R      = findProgram( "R", includename=TRUE ) ))
})
```

---

focusWin

*Set the Focus on a Particular Window*

---

**Description**

Bring the specified window into focus, and set it as the active window. `focusWin` will fail to bring the window into focus if it is called from the R console, since the R console returns focus to itself once a function returns. However, it will work if `focusWin` is called as a result of calling a function from the GUI window. (i.e., pushing a button or any other widget that has a function argument).

**Usage**

```
focusWin(winName, winVal=TRUE)
```

**Arguments**

`winName`            name of window to focus  
`winVal`             if TRUE, associate `winName` with the default window for `setWinVal` and `getWinVal`

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**Examples**

```
## Not run:
local(envir=.PBSmodEnv,expr={
focus <- function() {
  winName <- getWinVal()$select;
  focusWin(winName);
  mess = paste("Calling focusWin(\"", winName, "\")\n",
    "getWinVal()$myvar = ", getWinVal()$myvar, "\n\n", sep="",collapse="")
  cat(mess); invisible()
}
#create three windows named win1, win2, win3
#each having three radio buttons, which are used to change the focus
for(i in 1:3) {
  winDesc <- c(
    paste('window onclose=closeWin name=win',i,' title="Win',i,'" ', sep=''),
    paste('entry myvar ', i, sep=''),
    'radio name=select value=win1 text="one" function=focus mode=character',
    'radio name=select value=win2 text="two" function=focus mode=character',
    'radio name=select value=win3 text="three" function=focus mode=character');
  createWin(winDesc, astext=TRUE); }
})

## End(Not run)
```

genMatrix

*Generate Test Matrices for plotBubbles***Description**

Generate a test matrix of random numbers ( $\mu$  = mean and  $\sigma$  = standard deviation), primarily for plotBubbles.

**Usage**

```
genMatrix(m,n,mu=0,sigma=1)
```

**Arguments**

m	number of rows
n	number of columns
mu	mean of normal distribution
sigma	standard deviation of normal distribution

**Value**

An  $m$  by  $n$  matrix with normally distributed random values.

**Author(s)**

Jon T. Schnute, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**See Also**

[plotBubbles](#)

**Examples**

```
local(envir=.PBSmodEnv,expr={
  plotBubbles(genMatrix(20,6))
})
```

---

getChoice

*Choose One String Item from a List of Choices*

---

**Description**

Prompts the user to choose one string item from a list of choices displayed in a GUI. The simplest case `getChoice()` yields TRUE or FALSE.

**Usage**

```
getChoice(choice=c("Yes","No"), question="Make a choice: ",
  winname="getChoice", horizontal=TRUE, radio=FALSE,
  qcolor="blue", gui=FALSE, quiet=FALSE)
```

**Arguments**

choice	vector of strings from which to choose.
question	question or prompting statement.
winname	window name for the <code>getChoice</code> GUI.
horizontal	logical: if TRUE, display the choices horizontally, else vertically.
radio	logical: if TRUE, display the choices as radio buttons, else as buttons.
qcolor	colour for question.
gui	logical: if TRUE, <code>getChoice</code> is functional when called from a GUI, else it is functional from command line programs.
quiet	logical: if TRUE, don't print the choice on the command line.

**Details**

The user's choice is stored in `.PBSmod$options$getChoice` (or whatever `winname` is supplied). `getChoice` generates an `onClose` function that returns focus to the calling window (if applicable) and prints out the choice.

**Value**

If called from a GUI (`gui=TRUE`), no value is returned directly. Rather, the choice is written to the PBS options workspace, accessible through `getPBSoptions("getChoice")` (or whatever winname was supplied).

If called from a command line program (`gui=FALSE`), the choice is returned directly as a string scalar (e.g., `answer <- getChoice(gui=F)`).

**Note**

Microsoft Windows users may experience difficulties switching focus between the R console and GUI windows. The latter frequently disappear from the screen and need to be reselected (either clicking on the task bar or pressing `<Alt><Tab>`). This issue can be resolved by switching from MDI to SDI mode. From the R console menu bar, select `<Edit>` and `<GUI preferences>`, then change the value of "single or multiple windows" to SDI.

**Author(s)**

Rowan Haigh, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**See Also**

[chooseWinVal](#), [getWinVal](#), [setWinVal](#)

**Examples**

```
## Not run:
##-- Example 1
local(envir=.PBSmodEnv,expr={
  getChoice(c("Fame", "Fortune", "Health", "Beauty", "Lunch"),
    "What do you want?",qcolor="red",gui=FALSE)
})

##-- Example 2
local(envir=.PBSmodEnv,expr={
  getChoice(c("Homer Simpson", "Wilberforce Humphries", "Miss Marple", "Gary Numan"),
    "Who`s your idol?",horiz=FALSE,radio=TRUE,gui=FALSE)
})

## End(Not run)
```

---

getGUIoptions

*Get PBS Options for Widgets*

---

**Description**

Get the PBS options declared for GUI usage and set their corresponding widget values.

**Usage**

```
getGUIOptions()
```

**Details**

The options declared using `declareGUIOptions` are copied from the R environment into widget values. These widgets should have names that match the names of their corresponding options.

**Author(s)**

Anisa Egeli, Vancouver Island University, Nanaimo BC

**See Also**

[declareGUIOptions](#), [setGUIOptions](#), [promptWriteOptions](#), [readPBSoptions](#)

**Examples**

```
## Not run:
local(envir=.PBSmodEnv,expr={
  getPBSoptions() #loads from default PBSoptions.txt
})

## End(Not run)
```

---

getOptions

*Get and Set User Options*

---

**Description**

Functions to get and set user options within an option class object.

**Usage**

```
getOptions(option.object, key)
setOptions(option.object, ...)
.showOptions(object)
```

**Arguments**

<code>option.object</code>	options class object used for storing package options
<code>key</code>	name of option to retrieve; if missing, all options are returned
<code>...</code>	any number of user options to set where either (a) the named argument is the option key and the value is the option value or (b) the single unnamed argument is a list object where each named list element is the option key and the value is the element's value
<code>object</code>	options class object used for storing package options

**Value**

Value of the option specified by key (if specified) or a list of all options (if missing).

**Author(s)**

**Alex Couture-Beil**, Software Engineer, Earthly Technologies, Victoria BC

Maintainer: **Rowan Haigh**, Program Head – Offshore Rockfish

Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC

*locus opus*: remote office, Vancouver BC

Last modified Rd: 2024-11-04

**See Also**

See [PBSoptions-class](#) for more details and an example that uses **PBSmodelling**'s option management functions.

---

`getOptionsFileName`      *Get and Set File Name for Saving and Loading of Options*

---

**Description**

Functions for retrieving and setting the default file name used by `loadOptions` and `saveOptions`.

**Usage**

```
getOptionsFileName(option.object)
setOptionsFileName(option.object, name)
```

**Arguments**

<code>option.object</code>	options class object used for storing package options
<code>name</code>	new name for default file name

**Value**

`getOptionsFileName`: the default file name

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[loadOptions](#), [saveOptions](#)

See [PBSoptions-class](#) for more details and an example using **PBSmodelling**'s option management functions.

---

getOptionsPrefix      *Get and Set GUI Prefix of Options Class*

---

### Description

The GUI prefix is used for determining which GUI variables are associated with a user option.

### Usage

```
getOptionsPrefix(option.object)
setOptionsPrefix(option.object, prefix)
```

### Arguments

option.object	options class object used for storing package options
prefix	new prefix to use

### Value

getOptionsPrefix: a prefix string used to reference GUI variables

### Author(s)

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

### See Also

See [PBSoptions-class](#) for more details and an example using **PBSmodelling**'s option management functions.

### Examples

```
## Not run:
local(envir=.PBSmodEnv,expr={
  # For the example, the options object is saved to the temporary environment;
  # however, a user would normally create the object in his/her workspace.
  .mypkg <- new( "PBSoptions", filename="my_pkg.txt",
    initial.options=list(pi=3.14), gui.prefix="opt" )

  #prefix the option "pi" with "opt" to get "optpi"
  createWin( "entry name=optpi", astatic = TRUE )

  #the GUI variable "optpi" will be loaded with the option "pi"
  loadOptionsGUI( .mypkg )
})

## End(Not run)
```

---

`getPBSext`*Get Command Associated With File Name Extension*

---

**Description**

Display all locally defined file extensions and their associated commands, or search for the command associated with a specific file extension `ext`.

**Usage**

```
getPBSext(ext)
```

**Arguments**

`ext` character – optional string specifying a file extension.

**Value**

Command associated with file extension.

**Note**

These file associations are not saved from one *PBS Modelling* session to the next unless explicitly saved and loaded (see `writePBSoptions` and `readPBSoptions`).

**Author(s)**

**Alex Couture-Beil**, Software Engineer  
Earthly Technologies, Victoria BC

Maintainer: **Rowan Haigh**, Program Head – Offshore Rockfish  
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC  
*locus opus*: remote office, Vancouver BC  
Last modified Rd: 2023-01-05

**See Also**

In package **PBSmodelling**:  
[setPBSext](#), [openFile](#), [clearPBSext](#)

---

getPBSoptions	<i>Retrieve A User Option</i>
---------------	-------------------------------

---

**Description**

Get a previously defined user option.

**Usage**

```
getPBSoptions(option)
```

**Arguments**

option                    name of option to retrieve. If omitted, a list containing all options is returned.

**Value**

Value of the specified option, or NULL if the specified option is not found.

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[getPBSext](#), [readPBSoptions](#)

---

getWinAct	<i>Retrieve the Last Window Action</i>
-----------	--

---

**Description**

Get a string vector of actions (latest to earliest).

**Usage**

```
getWinAct(winName)
```

**Arguments**

winName                    name of window to retrieve action from

**Details**

When a function is called from a GUI, a string descriptor associated with the action of the function is stored internally (appended to the first position of the action vector). A user can utilize this action as a type of argument for programming purposes. The command `getWinAct()[1]` yields the latest action.

**Value**

String vector of recorded actions (latest first).

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

---

*getWinFun*                      *Retrieve Names of Functions Referenced in a Window*

---

**Description**

Get a vector of all function names referenced by a window.

**Usage**

```
getWinFun(winName)
```

**Arguments**

winName                      name of window, to retrieve its function list

**Value**

A vector of function names referenced by a window.

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

---

*getWinVal*                      *Retrieve Widget Values for Use in R Code*

---

**Description**

Get a list of variables defined and set by the GUI widgets. An optional argument scope directs the function to create local or global variables based on the list that is returned.

**Usage**

```
getWinVal(v=NULL, scope="", asvector=FALSE, winName="")
```

**Arguments**

v	vector of variable names to retrieve from the GUI widgets. If NULL, v retrieves all variables from all GUI widgets.
scope	scope of the retrieval. The default sets no variables in the non-GUI environment; scope="L" creates variables locally in relation to the parent frame that called the function; scope="P" creates variables in the temporary package workspace called .PBSmodEnv; and scope="G" creates global variables (pos=1).
asvector	return a vector instead of a list. <b>WARNING:</b> if a widget variable defines a true vector or matrix, this will not work.
winName	window from which to select GUI widget values. The default takes the window that has most recently received new user input.

**Details**

TODO: talk about scope=G/P/L and side effects of overwriting existing variables

**Value**

A list (or vector) with named components, where names and values are defined by GUI widgets.

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[parseWinFile](#), [setWinVal](#), [clearWinVal](#)

---

getYes

*Prompt the User to Choose Yes or No*

---

**Description**

Display a message prompt with "Yes" and "No" buttons.

**Usage**

```
getYes(message, title="Choice", icon="question")
```

**Arguments**

message	message to display in prompt window.
title	title of prompt window.
icon	icon to display in prompt window; options are "error", "info", "question", or "warning".

**Value**

Returns TRUE if the "Yes" button is clicked, FALSE if the "No" button is clicked.

**Author(s)**

Anisa Egeli, Vancouver Island University, Nanaimo BC

**See Also**

[showAlert](#), [getChoice](#), [chooseWinVal](#)

**Examples**

```
## Not run:
local(envir=.PBSmodEnv,expr={
  if(getYes("Print the number 1?"))
    print(1) else print("hallucination")
})

## End(Not run)
```

---

 GT0

---

*Restrict a Numeric Variable to a Positive Value*


---

**Description**

Restrict a numeric value  $x$  to a positive value using a differentiable function. GT0 stands for “greater than zero”.

**Usage**

```
GT0(x, eps=1e-4)
```

**Arguments**

$x$	vector of values
$eps$	minimum value greater than zero.

**Details**

```
if (x >= eps).....GT0 = x
if (0 < x < eps).....GT0 = (eps/2) * (1 + (x/eps)^2)
if (x <= 0).....GT0 = eps/2
```

**Author(s)**

Jon T. Schnute, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**See Also**

[scalePar](#), [restorePar](#), [calcMin](#)

**Examples**

```
local(envir=.PBSmodEnv,expr={
  oldpar = par(no.readonly=TRUE)
  plotGT0 <- function(eps=1,x1=-2,x2=10,n=1000,col="black") {
    x <- seq(x1,x2,len=n); y <- GT0(x,eps);
    lines(x,y,col=col,lwd=2); invisible(list(x=x,y=y)); }

  testGT0 <- function(eps=c(7,5,3,1,.1),x1=-2,x2=10,n=1000) {
    x <- seq(x1,x2,len=n); y <- x;
    plot(x,y,type="l");
    mycol <- c("red","blue","green","brown","violet","orange","pink");
    for (i in 1:length(eps))
      plotGT0(eps=eps[i],x1=x1,x2=x2,n=n,col=mycol[i]);
    invisible(); };

  testGT0()
  par(oldpar)
})
```

---

importHistory

---

*Import a History List from a File*


---

**Description**

Import a history list from file `fname`, and place it into the history list `hisname`.

**Usage**

```
importHistory(hisname="", fname="", updateHis=TRUE)
```

**Arguments**

<code>hisname</code>	name of the history list to be populated. The default ("") uses the value from <code>getWinAct()[1]</code> .
<code>fname</code>	file name of history file to import. The default ("") causes an open-file window to be displayed.
<code>updateHis</code>	logical: if TRUE, update the history widget to reflect the change in size and index.

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[exporthistory](#), [initHistory](#)

---

initHistory

*Create Structures for a New History Widget*


---

## Description

PBS history functions (below) are available to those who would like to use the package's history functionality, without using the pre-defined history widget. These functions allow users to create customized history widgets.

## Usage

```

initHistory(hisname, indexname=NULL, sizename=NULL,
            buttonnames=NULL, modename=NULL, func=NULL, overwrite=TRUE)
rmHistory(hisname="", index="")
addHistory(hisname="")
forwHistory(hisname="")
backHistory(hisname="")
lastHistory(hisname="")
firstHistory(hisname="")
jumpHistory(hisname="", index="")
clearHistory(hisname="")

```

## Arguments

hisname	name of the history "list" to manipulate. If it is omitted, the function uses the value of <code>getWinAct()[1]</code> as the history name. This allows the calling of functions directly from the <i>window description file</i> (except <code>initHistory</code> , which must be called before <code>createWin()</code> ).
indexname	name of the index entry widget in the <i>window description file</i> . If NULL, then the current index feature will be disabled.
sizename	name of the current size entry widget. If NULL, then the current size feature will be disabled.
buttonnames	named list of names of the first, prev, next, and last buttons. If NULL, then the buttons are not disabled ever
modename	name of the radio widgets used to change <code>addHistory</code> 's mode. If NULL, then the default mode will be to insert after the current index.
index	index to the history item. The default ("" ) causes the value to be extracted from the widget identified by <code>indexname</code> .
func	name of user supplied function to call when viewing history items.
overwrite	if TRUE, history (matching <code>hisname</code> ) will be cleared. Otherwise, the imported history will be merged with the current one.

## Details

PBS Modelling includes a pre-built history widget designed to collect interesting choices of GUI variables so that they can be redisplayed later, rather like a slide show.

Normally, a user would invoke a history widget simply by including a reference to it in the *window description file*. However, PBS Modelling includes support functions (above) for customized applications.

To create a customized history, each button must be described separately in the *window description file* rather than making reference to the history widget.

The history "List" must be initialized before any other functions may be called. The use of a unique history name (hisname) is used to associate a unique history session with the supporting functions.

The indexname and sizename arguments correspond to the given names of entry widgets in the *window description file*, which will be used to display the current index and total size of the list. The indexname entry widget can also be used by jumpHistory to retrieve a target index.

## Author(s)

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

## See Also

[importHistory](#), [exportHistory](#)

## Examples

```
## Not run:
# ***** THIS CODE DOES NOT RUN. NEEDS FIXING *****
# Example of creating a custom history widget that saves values
# whenever the "Plot" button is pressed. The user can tweak the
# inputs "a", "b", and "points" before each "Plot" and see the
# "Index" increase. After sufficient archiving, the user can review
# scenarios using the "Back" and "Next" buttons.
# A custom history is needed to achieve this functionality since
# the packages pre-defined history widget does not update plots.

# To start, create a Window Description to be used with createWin
# using astart=TRUE. P.S. Watch out for special characters which
# must be "escaped" twice (first for R, then PBSmodelling).

local(envir=.PBSmodEnv,expr={
  oldpar = par(no.readonly=TRUE)

winDesc <- '
  window title="Custom History"
  vector names="a b k" labels="a b points" font="bold" \\
  values="1 1 1000" function=myPlot
  grid 1 3
    button function=myHistoryBack text="<- Back"
    button function=myPlot text="Plot"
    button function=myHistoryForw text="Next ->"
```

```

grid 2 2
  label "Index"
  entry name="myHistoryIndex" width=5
  label "Size"
  entry name="myHistorySize" width=5
,
# Convert text to vector with each line represented as a new element
winDesc <- strsplit(winDesc, "\n")[[1]]

# Custom functions to update plots after restoring history values
myHistoryBack <- function() {
  backHistory("myHistory");
  myPlot(saveVal=FALSE); # show the plot with saved values
}
myHistoryForw <- function() {
  forwHistory("myHistory");
  myPlot(saveVal=FALSE); # show the plot with saved values
}
myPlot <- function(saveVal=TRUE) {
  # save all data whenever plot is called (directly)
  if (saveVal) addHistory("myHistory");
  getWinVal(scope="L");
  tt <- 2*pi*(0:k)/k;
  x <- (1+sin(a*tt)); y <- cos(tt)*(1+sin(b*tt));
  plot(x, y);
}
iHistory("myHistory", "myHistoryIndex", "myHistorySize")
createWin(winDesc, astext=TRUE)
par(oldpar)
})

## End(Not run)

```

---

isWhat

*Identify an Object and Print Information*


---

### Description

Identify an object by class, mode, typeof, and attributes.

### Usage

```
isWhat(x)
```

### Arguments

x                    an R object

### Value

No value is returned. The function prints the object's characteristics on the command line.

**Author(s)**

Jon T. Schnute, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

---

lisp

*List Objects in .PBSmodEnv Workspace*

---

**Description**

The function `lisp` returns a vector of character strings giving the names of the objects in `.PBSmodEnv`. It is only a wrapper for the base function `ls`.

**Usage**

```
lisp(name, pos = .PBSmodEnv, envir = as.environment(pos),  
     all.names = TRUE, pattern)
```

**Arguments**

name	which environment to use in listing the available objects. See the details section of <a href="#">ls</a> .
pos	an alternative argument to name for specifying the environment as a position in the search list.
envir	an alternative argument to name for specifying the environment.
all.names	a logical value. If TRUE, all object names are returned. If FALSE, names which begin with a '.' are omitted.
pattern	an optional <a href="#">regular expression</a> . Only names matching <code>pattern</code> are returned. See <a href="#">ls</a> for additional details.

**Details**

See the base function [ls](#) for details.

**Author(s)**

Copyright 1995–2012 R Core Development Team; distributed under GPL 2 or later.

**See Also**

[ls](#), [tget](#)  
[glob2rx](#) for converting wildcard patterns to regular expressions.

loadC

*Launch a GUI for Compiling and Loading C Code***Description**

A GUI interface allows users to edit, compile, and embed C functions in the R environment.

**Usage**

```
loadC()
```

**Details**

The function `loadC()` launches an interactive GUI that can be used to manage the construction of C functions intended to be called from R. The GUI provides tools to edit, compile, load, and run C functions in the R environment.

The `loadC` GUI also includes a tool for comparison between the running times and return values of R and C functions. It is assumed that the R and C functions are named `prefix.r` and `prefix.c`, respectively, where `prefix` can be any user-chosen prefix. If an initialization function `prefix.init` exists, it is called before the start of the comparison.

**The GUI controls:**

<b>File Prefix</b>	Prefix for <code>.c</code> and <code>.r</code> files.
<b>Lib Prefix</b>	Prefix for shared library object.
<b>Set WD</b>	Set the working directory.
<b>Open Log</b>	Open the log file.
<b>Open.c File</b>	Open the file <code>prefix.c</code> from the working directory.
<b>Open.r File</b>	Open the file <code>prefix.r</code> from the working directory.
<b>COMPILE</b>	Compile <code>prefix.c</code> into a shared library object.
<b>LOAD</b>	Load the shared library object.
<b>SOURCE R</b>	Source the file <code>prefix.r</code> .
<b>UNLOAD</b>	Unload the shared library object.
<b>Options</b>	
<b>Editor</b>	Text editor to use.
<b>Update</b>	Commit option changes.
<b>Browse</b>	Browse for a text editor.
<b>Clean Options</b>	
<b>Select All</b>	Select all check boxes specifying file types.
<b>Select None</b>	Select none of the check boxes.
<b>Clean Proj</b>	Clean the project of selected file types.
<b>Clean All</b>	Clean the directory of selected file types.
<b>Comparison</b>	
<b>Times to Run</b>	Number of times to run the R and C functions.
<b>RUN</b>	Run the comparison between R and C functions.
<b>R Time</b>	Computing time to run the R function multiple times.
<b>C Time</b>	Computing time to run the C function multiple times.

**Ratio**                      Ratio of R/C run times.

### Author(s)

Anisa Egeli, Vancouver Island University, Nanaimo BC

### See Also

[compileC](#)

---

loadOptions	<i>Save and Load Options to and from Disk</i>
-------------	---

---

### Description

Save and load options for use from one R session to another. If no file name is given, then the default file name (specified when the option object was created) is used.

### Usage

```
loadOptions(option.object, fname, prompt = FALSE)
saveOptions(option.object, fname, prompt = FALSE)
```

### Arguments

option.object	options class object used for storing package options
fname	file name to use: if missing the default file name is used; if given, file name becomes the default.
prompt	logical: if TRUE, prompt the user to select a file from an interactive GUI. If fname is given, then the value appears as the default selected file.

### Details

If fname is given (or selected when prompt=TRUE), then that file becomes the default file name for subsequent loading and saving.

### See Also

See [PBSoptions-class](#) for more details and an example using **PBSmodelling**'s option management functions.

---

loadOptionsGUI	<i>Load and Save Options Values to and from a GUI</i>
----------------	---

---

**Description**

These functions are used to move option values to and from a GUI. Option values are stored within an R object (as referenced by the `option.object`).

`loadOptionsGUI` copies the values from the R object to the GUI.

`saveOptionsGUI` copies the GUI values from the tcltk GUI to the R object.

**Usage**

```
loadOptionsGUI(option.object)
```

```
saveOptionsGUI(option.object)
```

**Arguments**

`option.object` options class object used for storing package options

**See Also**

See [PBSoptions-class](#) for more details and an example using **PBSmodelling**'s option management functions.

---

lucent	<i>Convert Solid Colours to Translucence</i>
--------	--

---

**Description**

Convert a vector of solid colours to a vector of translucent ones (or vice versa)

**Usage**

```
lucent(col.pal=1, a=1)
```

**Arguments**

`col.pal` vector of colours

`a` alpha transparency value ( $0$  = fully transparent,  $1$  = opaque)

**Details**

The function acts as a small wrapper to the `rgb` function.

**Value**

Vector of transformed colours depending on the alpha transparency value `a`.

**Author(s)**

Steve Martell, International Pacific Halibut Commission, Seattle WA

**See Also**

[pickCol](#), [testCol](#), [col2rgb](#), [rgb](#)

**Examples**

```
local(envir=.PBSmodEnv,expr={
  oldpar = par(no.readonly=TRUE)
  pbsfun = function(clrs=c("moccasin",rainbow(3))){
    clrs = c(clrs,lucent(clrs,a=0.25))
    testCol(clrs); invisible() }
  pbsfun()
  par(oldpar)
})
```

---

openExamples

*Open Example Files from a Package*

---

**Description**

Open examples from the examples subdirectory of a given package.

**Usage**

```
openExamples(package, prefix, suffix)
```

**Arguments**

package	name of the package that contains the examples.
prefix	prefix of the example file(s).
suffix	character vector of suffixes for the example files.

**Details**

Copies of each example file are placed in the working directory and opened. If files with the same name already exist, the user is prompted with a choice to overwrite.

To use this function in a *window description file*, the package, prefix and suffix arguments must be specified as the action of the widget that calls openExamples. Furthermore, package, prefix, and each suffix must be separated by commas. For example, `action=myPackage,example1,.r,.c` will copy `example1.r` and `example2.c` from the examples directory of the package **myPackage** to the working directory and open these files. If the function was called by a widget, a widget named `prefix` will be set to the specified prefix.

**Note**

If all the required arguments are missing, it is assumed that the function is being called by a GUI widget.

**Author(s)**

Anisa Egeli, Vancouver Island University, Nanaimo BC

**See Also**

[openFile](#)

**Examples**

```
## Not run:
# Copies fib.c and fib.r from the examples directory in
# PBSmodelling to the temporary working directory, and opens these files.
local(envir=.PBSmodEnv,expr={
  cwd = getwd(); setwd(tempdir())
  openExamples("PBSmodelling", c("fib"), c(".r", ".c"))
  setwd(cwd)
})

## End(Not run)
```

---

openFile

*Open File with Associated Program*

---

**Description**

Open a file using the program that the operating system (Windows / Mac OS X / Linux) associates with its type. Users wishing to override the default application can specify a program association using 'setPBSext'.

**Usage**

```
openFile(fname="", package=NULL, select=FALSE)
```

**Arguments**

fname	character – vector containing file names to open.
package	character – (optional) package name; open files relative to this package.
select	logical – if TRUE, force the use of 'selectFile'.

**Value**

An invisible string vector of the file names and/or commands with file names.

**Note**

If a command is registered with `setPBSext`, then `openFile` will replace all occurrences of `"%f"` with the absolute path of the filename before executing the command.

**Author(s)**

**Alex Couture-Beil**, Software Engineer  
Earthly Technologies, Victoria BC

Maintainer: **Rowan Haigh**, Program Head – Offshore Rockfish  
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC  
*locus opus*: remote office, Vancouver BC  
Last modified Rd: 2023-01-05

**See Also**

In package **PBSmodelling**:  
[getPBSext](#), [setPBSext](#), [clearPBSext](#), [writePBSoptions](#)

**Examples**

```
## Not run:
local(envir=.PBSmodEnv,expr={
  # use openFile directly:
  openFile( "doc/PBSmodelling-UG.pdf", package="PBSmodelling" )
})
local(envir=.PBSmodEnv,expr={
  # via doAction in a window description file:
  createWin( "button text=help func=doAction width=20 pady=25 bg=green
  action=\"openFile(`doc/PBSmodelling-UG.pdf`,package=`PBSmodelling`)\", astext=TRUE)
})
local(envir=.PBSmodEnv,expr={
  # Set up 'Firefox' to open '.html' files (only applicable if Firefox is NOT default web browser)
  setPBSext("html", "'c:/Program Files/Mozilla Firefox/firefox.exe" file://%f')
  openFile("foo.html")
})

## End(Not run)
```

**Description**

Open package User's Guide '`<pkg>-UG.pdf`' if it exists. This function is essentially a wrapper for `openFile`.

**Usage**

```
openUG(pkg = "PBSmodelling")
```

**Arguments**

pkg                    character – full name (with or without quotes) of a package installed on the user's system.

**Details**

The user's guide is assumed to be PDF format with extension pdf. The name of the PDF file will be '<pkg>-UG.pdf' (e.g., PBSmodelling-UG.pdf).

**Author(s)**

**Rowan Haigh**, Program Head – Offshore Rockfish  
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC  
*locus opus*: remote office, Vancouver BC  
Last modified Rd: 2023-10-18

**See Also**

In package **PBSmodelling**:  
[openFile](#), [showHelp](#), [viewCode](#)

---

packList                    *Pack a List with Objects*

---

**Description**

Pack a list with existing objects using names only.

**Usage**

```
packList(stuff, target="PBSlist", value, penv=NULL, tenv=.PBSmodEnv)
```

**Arguments**

stuff                    character – vector of object names  
target                    character – name of target list object  
value                    numeric|character – an optional explicit value to assign to stuff  
penv                    environment – source environment (default=parent) or user-specified environment where stuff resides  
tenv                    environment – target environment where target list exists or will be located

## Details

A list object called `target` will be located in the `tenv` environment. The objects named in `stuff` and located in the `penv` environment will appear as named components within the list object `target`.

If an explicit value is specified, the function uses this value instead of looking for local objects. Essentially, `stuff = value` which is then packed into `target`.

## Value

No value is returned

## Note

The function determines the parent environment from within. This environment contains the objects from which the function copies to the target environment. Alternatively, the user can specify the environment where `stuff` resides.

## Author(s)

**Rowan Haigh**, Program Head – Offshore Rockfish  
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC  
*locus opus*: remote office, Vancouver BC  
Last modified Rd: 2019-03-12

## See Also

In package **PBSmodelling**:  
[unpackList](#), [readList](#), [writeList](#)  
Accessor functions:  
[tget](#), [tcall](#), [tprint](#), and [tput](#)

## Examples

```
local(envir=.PBSmodEnv,expr={
  fn = function() {
    alpha=rnorm(10)
    beta=letters
    gamma=mean
    delta=longley
    packList(c("alpha","beta","gamma","delta")) }
  fn(); tprint(PBSlist)
})
```

**Description**

Pad numbers and/or text with leading and/or trailing zeroes.

**Usage**

```
pad0(x, n, f = 0)
```

**Arguments**

x	vector of numbers and/or strings
n	number of text characters representing a padded integer
f	factor of 10 transformation on x before padding

**Details**

Converts numbers (or text coerced to numeric) to integers and then to text, and pads them with leading zeroes. If the factor f is >0, then trailing zeroes are also added.

**Value**

If `length(f)==1` or `length(x)==1`, the function returns a character vector representing x with leading zeroes.

If both f and x have lengths >1, then a list of character vectors indexed by f is returned.

**Author(s)**

Rowan Haigh, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**See Also**

[show0](#), [GT0](#)

**Examples**

```
local(envir=.PBSmodEnv,expr={
  resetGraph(); x <- pad0(x=123,n=10,f=0:7);
  addLabel(.5,.5,paste(x,collapse="\n"),cex=1.5);
})
```

---

`parseWinFile`*Convert a Window Description File into a List Object*

---

**Description**

Parse a *window description file* (markup file) into the list format expected by `createWin`.

**Usage**

```
parseWinFile(fname, astext=FALSE)
```

**Arguments**

<code>fname</code>	file name of the <i>window description file</i> .
<code>astext</code>	if TRUE, <code>fname</code> is interpreted as a vector of strings, with each element representing a line of code in a <i>window description file</i> .

**Value**

A list representing a parsed *window description file* that can be directly passed to `createWin`.

**Note**

All widgets are forced into a 1-column by N-row grid.

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[createWin](#), [compileDescription](#)

**Examples**

```
## Not run:
local(envir=.PBSmodEnv,expr={
  x <- parseWinFile(system.file("examples/LissFigWin.txt",package="PBSmodelling"))
  createWin(x)
})

## End(Not run)
```

---

pause

*Pause Between Graphics Displays or Other Calculations*

---

**Description**

Pause, typically between graphics displays. Useful for demo purposes.

**Usage**

```
pause(s = "Press <Enter> to continue")
```

**Arguments**

s                   text issued on the command line when pause is invoked.

**Author(s)**

Jon T. Schnute, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

---

PBSmodelling

*PBS Modelling*

---

**Description**

*PBS Modelling* provides software to facilitate the design, testing, and operation of computer models. It focuses particularly on tools that make it easy to construct and edit a customized graphical user interface (GUI). Although it depends heavily on the R interface to the Tcl/Tk package, a user does not need to know Tcl/Tk.

PBSmodelling contains examples that illustrate models built using other R packages, including PBSmapping, deSolve, PBSddesolve, and BRugs. It also serves as a convenient prototype for building new R packages, along with instructions and batch files to facilitate that process.

The R directory `.../library/PBSmodelling/doc` includes a complete user guide ‘PBSmodelling-UG.pdf’. To use this package effectively, please consult the guide.

*PBS Modelling* comes packaged with interesting examples accessed through the function `runExamples()`. Additionally, users can view *PBS Modelling* widgets through the function `testWidgets()`. More generally, a user can run any available demos in his/her locally installed packages through the function `runDemos()`.

## Description

Projects commonly involve various settings or options such as paths to C compilers or other third-party tools. **PBSmodelling** provides a set of option management functions for managing user specific options. Options can be modified through the provided set of functions on the command line, or through a custom GUI. These options can be saved to disk for use in subsequent R sessions.

To use **PBSmodelling**'s suite of option management functions, a PBSoptions object must be created for each of your projects. Each PBSoptions object contains a distinct R environment where option values are stored; this allows different projects to use overlapping option names without conflicts (provided each project has its own PBSoptions class object).

## Details

When a PBSoptions object is created with the new function, the `initial.options` list, if supplied, is stored as initial user options in the object. The initialization routine then attempts to load user set options from the `filename` file. If such a file exists, these values are stored in the PBSoptions object overwriting any initial values as specified by `initial.options`

Option values are not directly stored in the object, but rather in an environment stored in the instance slot. Using an environment rather than slots for storing options allows us to pass option object by reference rather than value; that is, we can save options in the object without the need of returning a new modified class object. It is therefore necessary that users use the functions listed in the "see also" section to effectively manage user options.

## Objects from the Class

Objects can be created by calls of the form

```
new("PBSoptions", filename, initial.options=list(), gui.prefix="option").
```

`filename`: default file name to use when saving and loading options to and from disk

`initial.options`: a list with distinctly named initial options to use if no previously saved file exists

`gui.prefix`: a prefix used to identify GUI variables which correspond to user options

## Slots

`instance`: The R environment used to store options. Please do not use this directly; use the functions listed under the "see also" section.

## Methods

**print** `signature(x = "PBSoptions")`: prints the list of options

**Warning**

Do not use the slots directly – use the access functions instead.

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[getOptions](#) for retrieving and modifying user options

[getOptionsFileName](#) for retrieving and modifying the default options file name

[loadOptions](#) for loading and saving options from and to disk

[getOptionsPrefix](#) for retrieving and modifying the GUI prefix (for custom GUI interfaces)

[loadOptionsGUI](#) for setting GUI values to reflect user options and vice-versa

**Examples**

```
## Not run:
local(envir=.PBSmodEnv,expr={
  #initialize an option manager with a single logical option
  .mypkg <- new( "PBSoptions", filename="my_pkg.txt",
    initial.options=list( sillyhatday=FALSE ) )

  #retrieving an option
  silly <- getOptions( .mypkg, "sillyhatday" )
  cat( "today is", ifelse( silly, "silly hat day!", "monday" ), "\n" )

  #set an option
  setOptions( .mypkg, sillyhatday = TRUE, photos = "/shares/silly_hat_photos" )

  #create a GUI which works with options
  createWin( c(
    "check name=options&#x26;sillyhatday text=\&quot;silly hat day&quot;",
    "entry name=optionphotos width=22 mode=character label=\&quot;photos directory&quot;",
    "button func=doAction text=save action=saveOptionsGUI(.mypkg)" ), astext = TRUE )

  #update GUI values based on values stored in .mypkg's options
  loadOptionsGUI( .mypkg )
  print(getOptions( .mypkg ))
})

## End(Not run)
```

---

`pickCol`*Pick a Colour From a Palette and get the Hexadecimal Code*

---

**Description**

Display an interactive colour palette from which the user can choose a colour.

**Usage**

```
pickCol(returnValue=TRUE)
```

**Arguments**

<code>returnValue</code>	If TRUE, display the full colour palette, choose a colour, and return the hex value to the R session. If FALSE, use an intermediate GUI to interact with the palette and display the hex value of the chosen colour.
--------------------------	---

**Value**

A hexadecimal colour value.

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[testCol](#)

**Examples**

```
## Not run:  
local(envir=.PBSmodEnv,expr={  
  junk<-pickCol(); resetGraph()  
  addLabel(.5,.5,junk,cex=4,col=junk)  
})  
  
## End(Not run)
```

---

`plotACF`*Plot Autocorrelation Bars From a Data Frame, Matrix, or Vector*

---

**Description**

Plot autocorrelation bars (ACF) from a data frame, matrix, or vector.

**Usage**

```
plotACF(file, lags=20,  
        clr=c("blue", "red", "green", "magenta", "navy"), ...)
```

**Arguments**

<code>file</code>	data frame, matrix, or vector of numeric values.
<code>lags</code>	maximum number of lags to use in the ACF calculation.
<code>clr</code>	vector of colours. Patterns are repeated if the number of fields exceed the length of <code>clr</code> .
<code>...</code>	additional arguments for plot or lines.

**Details**

This function is designed primarily to give greater flexibility when viewing results from the R-package BRugs. Use `plotACF` in conjunction with `samplesHistory("*", beg=0, plot=FALSE)` rather than `samplesAutoC` which calls `plotAutoC`.

**Author(s)**

Rowan Haigh, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**Examples**

```
local(envir=.PBSmodEnv, expr={  
  oldpar = par(no.readonly=TRUE)  
  resetGraph(); plotACF(trees, lwd=2, lags=30)  
  par(oldpar)  
})
```

---

`plotAsp`*Construct a Plot with a Specified Aspect Ratio*

---

**Description**

Plot x and y coordinates using a specified aspect ratio.

**Usage**

```
plotAsp(x, y, asp=1, ...)
```

**Arguments**

<code>x</code>	vector of x-coordinate points in the plot.
<code>y</code>	vector of y-coordinate points in the plot.
<code>asp</code>	y/x aspect ratio.
<code>...</code>	additional arguments for plot.

**Details**

The function `plotAsp` differs from `plot(x,y,asp=1)` in the way axis limits are handled. Rather than expand the range, `plotAsp` expands the margins through padding to keep the aspect ratio accurate.

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**Examples**

```
local(envir=.PBSmodEnv,expr={
  oldpar = par(no.readonly=TRUE)
  x <- seq(0,10,0.1)
  y <- sin(x)
  par(mfrow=2:1)
  plotAsp(x,y,asp=1,xlim=c(0,10),ylim=c(-2,2), main="sin(x)")
  plotAsp(x,y^2,asp=1,xlim=c(0,10),ylim=c(-2,2), main="sin^2(x)")
  par(oldpar)
})
```

---

plotBubbles

---

*Construct a Bubble Plot from a Matrix*


---

### Description

Construct a bubble plot for a matrix *z*.

### Usage

```
plotBubbles(z, xval=FALSE, yval=FALSE, dnam=FALSE, rpro=FALSE,
            cpro=FALSE, rres=FALSE, cres=FALSE, powr=0.5, size=0.2, lwd=1,
            clr=c("black","red","blue"), hide0=FALSE, frange=0.05, prettyaxis=FALSE, ...)
```

### Arguments

<i>z</i>	numeric – input matrix, array (2 dimensions) or data frame.
<i>xval</i>	numeric – x-values and/or labels for the columns of <i>z</i> . if <i>xval</i> =TRUE, the first row contains x-values for the columns.
<i>yval</i>	numeric – y-values and/or labels for the rows of <i>z</i> . If <i>yval</i> =TRUE, the first column contains y-values for the rows.
<i>dnam</i>	logical – if TRUE, attempt to use dimnames of input matrix <i>z</i> as <i>xval</i> and <i>yval</i> . The dimnames are converted to numeric values and must be strictly increasing or decreasing. If successful, these values will overwrite previously specified values of <i>xval</i> and <i>yval</i> or any default indices.
<i>rpro</i>	logical – if TRUE, convert rows to proportions.
<i>cpro</i>	logical – if TRUE, convert columns to proportions.
<i>rres</i>	logical – if TRUE, use row residuals (subtract row means).
<i>cres</i>	logical – if TRUE, use column residuals (subtract column means).
<i>powr</i>	numeric – power transform; radii are proportional to $z^{\text{powr}}$ . Note: <i>powr</i> =0.5 yields bubble areas proportional to <i>z</i> .
<i>size</i>	numeric – size (inches) of the largest bubble.
<i>lwd</i>	numeric – line width for drawing circles.
<i>clr</i>	character – colours (3-element vector) used for positive, negative, and zero values, respectively.
<i>hide0</i>	logical – if TRUE, hide zero-value bubbles.
<i>frange</i>	numeric – number specifying the fraction by which the range of the axes should be extended.
<i>prettyaxis</i>	logical – if TRUE, apply the pretty function to both axes.
...	dots – additional arguments for plotting functions.

**Details**

The function `plotBubbles` essentially flips the `z` matrix visually. The columns of `z` become the `x`-values while the rows of `z` become the `y`-values, where the first row is displayed as the bottom `y`-value and the last row is displayed as the top `y`-value. The function's original intention was to display proportions-at-age vs. year.

**Author(s)**

**Jon T. Schnute**, Research Scientist Emeritus  
Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

Maintainer: **Rowan Haigh**, Program Head – Offshore Rockfish  
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC  
*locus opus*: remote office, Vancouver BC  
Last modified Rd: 2023-10-18

**See Also**

In package **PBSmodelling**:  
[genMatrix](#)

**Examples**

```
local(envir=.PBSmodEnv,expr={
  oldpar = par(no.readonly=TRUE)
  plotBubbles(round(genMatrix(40,20),0),clrs=c("green","grey","red"));
  data(CCA.qbr,envir=.PBSmodEnv)
  plotBubbles(CCA.qbr,cpro=TRUE,powr=.5,dnam=TRUE,size=.15,
    ylim=c(0,70),xlab="Year",ylab="Quillback Rockfish Age")
  par(oldpar)
})
```

---

plotCsum

*Plot Cumulative Sum of Data*

---

**Description**

Plot the cumulative frequency of a data vector or matrix, showing the median and mean of the distribution.

**Usage**

```
plotCsum(x, add = FALSE, ylim = c(0, 1), xlab = "Measure",
  ylab = "Cumulative Proportion", ...)
```

**Arguments**

x	vector or matrix of numeric values.
add	logical: if TRUE, add the cumulative frequency curve to a current plot.
ylim	limits for the y-axis.
xlab	label for the x-axis.
ylab	label for the y-axis.
...	additional arguments for the plot function.

**Author(s)**

Rowan Haigh, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**Examples**

```
local(envir=.PBSmodEnv,expr={
  oldpar = par(no.readonly=TRUE)
  x <- rgamma(n=1000,shape=2)
  plotCsum(x)
  par(oldpar)
})
```

---

plotDens

---

*Plot Density Curves from a Data Frame, Matrix, or Vector*


---

**Description**

Plot the density curves from a data frame, matrix, or vector. The mean density curve of the data combined is also shown.

**Usage**

```
plotDens(file, clr=c("blue","red","green","magenta","navy"), ...)
```

**Arguments**

file	data frame, matrix, or vector of numeric values.
clr	vector of colours. Patterns are repeated if the number of fields exceed the length of clr.
...	additional arguments for plot or lines.

**Details**

This function is designed primarily to give greater flexibility when viewing results from the R-package BRugs. Use plotDens in conjunction with samplesHistory("\*",beg=0,plot=FALSE) rather than samplesDensity which calls plotDensity.

**Author(s)**

Rowan Haigh, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**Examples**

```
local(envir=.PBSmodEnv,expr={
  oldpar = par(no.readonly=TRUE)
  z <- data.frame(y1=rnorm(50,sd=2),y2=rnorm(50,sd=1),y3=rnorm(50,sd=.5))
  plotDens(z,lwd=3)
  par(oldpar)
})
```

---

plotFriedEggs

*Render a Pairs Plot as Fried Eggs and Beer*

---

**Description**

Create a pairs plot where the lower left half comprises either fried egg contours or smoke ring contours, the upper right half comprises glasses of beer filled to the correlation point, and the diagonals show frequency histograms of the input data.

**Usage**

```
plotFriedEggs(A, eggs=TRUE, rings=TRUE, levs=c(0.01,0.1,0.5,0.75,0.95),
  pepper=200, replace=FALSE, jitt=c(1,1), bw=25, histclr=NULL)
```

**Arguments**

A	data frame or matrix for use in a pairs plot.
eggs	logical: if TRUE, fry eggs in the lower panels.
rings	logical: if TRUE, blow smoke rings in the lower panels.
levs	explicit contour levels expressed as quantiles.
pepper	number of samples to draw from A to pepper the plots.
replace	logical: if TRUE, sample A with replacement.
jitt	argument factor used by function <code>base::jitter</code> when peppering. If user supplies two numbers, the first will jitter x, the second will jitter y.
bw	argument bandwidth used by function <code>KernSmooth::bkde2D</code> .
histclr	user-specified colour(s) for histogram bars along the diagonal.

**Details**

This function comes to us from Dr. Steve Martell of the Fisheries Science Centre at UBC. Obviously many hours of contemplation with his students at the local pub have contributed to this unique rendition of a pairs plot.

**Note**

If eggs=TRUE and rings=FALSE, fried eggs are served.  
 If eggs=FALSE and rings=TRUE, smoke rings are blown.  
 If eggs=TRUE and rings=TRUE, only fried eggs are served.  
 If eggs=FALSE and rings=FALSE, only pepper is sprinkled.

**Author(s)**

Steve Martell, International Pacific Halibut Commission, Seattle WA

**See Also**

[plotBubbles](#), [scalePar](#)

[KernSmooth::bkde2D](#), [grDevices::contourLines](#), [graphics::contour](#)

**Examples**

```
local(envir=.PBSmodEnv,expr={
  oldpar = par(no.readonly=TRUE)
  x=rnorm(5000,10,3); y=-x+rnorm(5000,1,4); z=x+rnorm(5000,1,3)
  A=data.frame(x=x,y=y,z=z)
  for (i in 1:3)
    switch(i,
      {plotFriedEggs(A,eggs=TRUE,rings=FALSE);
        pause("Here are the eggs...(Press Enter for next)"}),
      {plotFriedEggs(A,eggs=FALSE,rings=TRUE);
        pause("Here are the rings...(Press Enter for next)"}),
      {plotFriedEggs(A,eggs=FALSE,rings=FALSE);
        cat("Here is the pepper alone.\n")} )
  par(oldpar)
})
```

---

plotSidebars

*Plot Table as Horizontal Sidebars*

---

**Description**

Plot (x,y) table (matrix or data frame) as horizontal sidebars.

**Usage**

```
plotSidebars(z, scale = 1, col = lucent("blue", 0.25), ...)
```

**Arguments**

<code>z</code>	data frame or matrix of z-values (e.g., age frequencies) where rows form the plot's y-values and columns describe the grouping variable along the x-axis.
<code>scale</code>	numeric scale factor controlling the leftward expansion of z-value bars.
<code>col</code>	colour to fill bars.
<code>...</code>	additional parameters used by <code>par</code> and <code>polygon</code> . The user can also pass in two non-formal arguments to control the function: <code>lbl</code> – labels for the x- and y-axis; <code>margin</code> – function to report margin summaries.

**Details**

Plots z-data as horizontal bars arising from an x-coordinate controlled by the column names of `z`. The bars extend left along the y-coordinate by `z*scale` from the central x-coordinate.

**Author(s)**

Steve Martell, International Pacific Halibut Commission, Seattle WA

**See Also**

[plotBubbles](#), [plotFriedEggs](#), [evalCall](#)

**Examples**

```
local(envir=.PBSmodEnv,expr={
  oldpar = par(no.readonly=TRUE)
  pbsfun = function () {
    meanmarg = function(x){x=x[!is.na(x)];
      if (length(x)==0 | all(x==0)) NA else sum((x/sum(x))*as.numeric(names(x)))}
    data(CCA.qbr,envir=.PBSmodEnv)
    plotSidebars(CCA.qbr,scale=4,las=1,border="navyblue",mar=c(4,4,1,1),
      lbl=c("Year","Quillback Rockfish Age"),margin=function(x){round(meanmarg(x),0)})
    invisible() }
  pbsfun()
  par(oldpar)
})
```

---

plotTrace

*Plot Trace Lines from a Data Frame, Matrix, or Vector*

---

**Description**

Plot trace lines from a data frame or matrix where the first field contains x-values, and subsequent fields give y-values to be traced over x. If input is a vector, this is traced over the number of observations.

**Usage**

```
plotTrace(file, clr=c("blue","red","green","magenta","navy"), ...)
```

**Arguments**

file	data frame or matrix of x and y-values, or a vector of y-values.
clr	vector of colours. Patterns are repeated if the number of traces (y-fields) exceed the length of clr.
...	additional arguments for plot or lines.

**Details**

This function is designed primarily to give greater flexibility when viewing results from the R-package BRugs. Use plotTrace in conjunction with samplesHistory("\*", beg=0, plot=FALSE) rather than samplesHistory which calls plotHistory.

**Author(s)**

Rowan Haigh, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**Examples**

```
local(envir=.PBSmodEnv, expr={
  oldpar = par(no.readonly=TRUE)
  z <- data.frame(x=1:50, y1=rnorm(50, sd=3), y2=rnorm(50, sd=1), y3=rnorm(50, sd=.25))
  plotTrace(z, lwd=3)
  par(oldpar)
})
```

---

presentTalk

*Run a Presentation in R*

---

**Description**

Start an R talk from a *talk description file* that launches a control GUI.

**Usage**

```
presentTalk(talk)
```

**Arguments**

talk	name of file containing XML code (e.g., swisstalk.xml).
------	---

## Details

The function `presentTalk` is a tool that facilitates lectures and workshops in R. The function allows the presenter to show code snippets alongside their execution, making use of R's graphical capabilities. When `presentTalk` is called, a graphical user interface (GUI) is launched that allows the user to control the flow of the talk (e.g., switching between talks or skipping to various sections of a talk).

The automatic control buttons allow the user to move forward or backward in the talk. The GO button moves forward one tag segment, the Back button moves back to the previous tag segment. The blue buttons allow movement among sections – Prev to the previous section, Restart to the start of the current section, and Next to the next section. Drop down lists are provided for both indicating the current section and slide number and as an additional interface for jumping between different sections or slide numbers.

In addition to the automatic menu items, a user can add buttons to the GUI that accomplish similar purposes.

## Note

See the `PBSmodelling` User's Guide for more information.

## Author(s)

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

## See Also

See `PBStalk-class` for more details on `PBSmodelling`'s talk presentation classes.

## Examples

```
## Not run:
local(envir=.PBSmodEnv,expr={
  cwd = getwd()
  talk_dir <- system.file("examples", package = "PBSmodelling" )
  setwd(talk_dir)
  presentTalk( "swisstalk.xml" ) # closing the GUI should restore cwd
})

## End(Not run)
```

---

promptWriteOptions      *Prompt the User to Write Changed Options*

---

## Description

If changes have been made to PBS options, this function allows the user to choose whether to write PBS options to an external file that can be loaded later by `readPBSOptions`.

**Usage**

```
promptWriteOptions(fname="")
```

**Arguments**

fname            name of file where options will be saved.

**Details**

If there are options that have been changed in the GUI but have not been committed to PBSmodelling memory in the global R environment, the user is prompted to choose whether or not to commit these options.

Then, if any PBS options have been changed, the user is prompted to choose whether to save these options to the file fname. (When a new R session is started or when a call to readPBSoptions or writePBSoptions is made, PBS options are considered to be unchanged; when an option is set, the options are considered to be changed).

If fname="", the user is prompted to save under the file name last used by a call to readPBSoptions or writePBSoptions if available. Otherwise, the default file name "PBSoptions.txt" is used.

**Author(s)**

Anisa Egeli, Vancouver Island University, Nanaimo BC

**See Also**

[writePBSoptions](#), [readPBSoptions](#), [setPBSoptions](#)

**Examples**

```
## Not run:
local(envir=.PBSmodEnv,expr={
  promptWriteOptions() #uses default filename PBSoptions.txt
})

## End(Not run)
```

---

readList

*Read a List from an ASCII File*

---

**Description**

Read in a list previously saved to an ASCII file by various R functions such as dput, dump, and writeList. The latter can render lists in a convenient **PBSmodelling** format. The function readList detects the format automatically.

For information about the **PBSmodelling** format, see writeList.

**Usage**

```
readList(fname)
```

**Arguments**

fname            file name of the text file containing the list.

**Value**

Returns a list object from ASCII files originally formatted in one of the following ways:

"D" = created by the R functions `dput` or `dump`;

"R" = R list object that uses 'structure' (e.g., Windows History file);

"P" = PBS-formatted file (see `writeList`);

"C" = comment-delimited file (e.g., Awatea/Coleraine input files).

**Warning**

When importing a list in the PBSmodelling ("P") format, if two list elements share the same name, the list will import incorrectly.

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[packList](#), [unpackList](#), [writeList](#)

---

readPBSOptions

*Read PBS Options from an External File*

---

**Description**

Load options that were saved using `writePBSOptions`, for use with `openFile`, `getPBSOptions` or interfaces such as `loadC`.

**Usage**

```
readPBSOptions(fname="PBSOptions.txt")
```

**Arguments**

fname            file name or full path of file from which the options will be loaded.

**Note**

If an option exists in R memory but not in the saved file, the option is not cleared from memory.

**Author(s)**

Anisa Egeli, Vancouver Island University, Nanaimo BC

**See Also**

[writePBSoptions](#), [getGUIoptions](#), [openFile](#), [getPBSoptions](#)

---

resetGraph

*Reset par Values for a Plot*

---

**Description**

Reset `par()` to default values to ensure that a new plot utilizes a full figure region. This function helps manage the device surface, especially after previous plotting has altered it.

**Usage**

```
resetGraph(reset.mf=TRUE)
```

**Arguments**

`reset.mf` if TRUE reset the multi-frame status; otherwise preserve `mfrow`, `mfc01`, and `mfg`

**Details**

This function resets `par()` to its default values. If `reset.mf=TRUE`, it also clears the graphics device with `frame()`. Otherwise, the values of `mfrow`, `mfc01`, and `mfg` are preserved, and graphics continues as usual in the current plot. Use `resetGraph` only before a high level command that would routinely advance to a new frame.

**Value**

invisible return of the reset value `par()`

**Author(s)**

Jon T. Schnute, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

---

 restorePar

*Get Actual Parameters from Scaled Values*


---

**Description**

Restore scaled parameters to their original units. Used in minimization by calcMin.

**Usage**

```
restorePar(S,pvec)
```

**Arguments**

S	scaled parameter vector.
pvec	a data frame comprising four columns - c("val", "min", "max", "active") and as many rows as there are model parameters. The "active" field (logical) determines whether the parameters are estimated (TRUE) or remain fixed (FALSE).

**Details**

Restoration algorithm:  $P = P_{min} + (P_{max} - P_{min})(\sin(\frac{\pi S}{2}))^2$

**Value**

Parameter vector converted from scaled units to original units specified by pvec.

**Author(s)**

Jon T. Schnute, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**See Also**

[scalePar](#), [calcMin](#), [GT0](#)

**Examples**

```
local(envir=.PBSmodEnv,expr={
  pvec <- data.frame(val=c(1,100,10000),min=c(0,0,0),max=c(5,500,50000),
    active=c(TRUE,TRUE,TRUE))
  S    <- c(.5,.5,.5)
  P    <- restorePar(S,pvec)
  print(cbind(pvec,S,P))
})
```

---

`runDemos`*Interactive GUI for R Demos*

---

**Description**

An interactive GUI for accessing demos from any R package installed on the user's system. `runDemos` is a convenient alternative to R's `demo` function.

**Usage**

```
runDemos(package)
```

**Arguments**

`package`            display demos from a particular package (optional).

**Details**

If the argument `package` is not specified, the function will look for demos in all packages installed on the user's system.

**Note**

The `runDemos` GUI attempts to retain the user's objects and restore the working directory. However, pre-existing objects will be overwritten if their names coincide with names used by the various demos. Also, depending on conditions, the user may lose working directory focus. We suggest that cautious users run this demo from a project where data objects are not critical.

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[runExamples](#) for examples specific to **PBSmodelling**.

---

`runExample`*Run a Single GUI Example Included with PBS Modelling*

---

**Description**

Display a GUI to demonstrate one PBS Modelling example.

The example source files can be found in the R directory `.../library/PBSmodelling/examples`.

**Usage**

```
runExample(ex, pkg="PBSmodelling")
```

**Arguments**

`ex` string specifying an example in the pkg directory examples.  
`pkg` package with an examples subdirectory.

**Details**

If no example is specified or if the example does not exist, a GUI pops up informing you of potential choices. Note that the string choice is case-sensitive.

Some examples use external packages which must be installed to work correctly:

BRugs - LinReg, MarkRec, and CCA;

deSolve/PBSddesolve - FishRes;

PBSmapping - FishTows.

**Note**

The examples are copied from `.../library/PBSmodelling/examples` to R's current temporary working directory and run from there.

**Author(s)**

Rowan Haigh, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**See Also**

[runDemos](#), [runExamples](#)

---

runExamples

*Run GUI Examples Included with PBS Modelling*

---

**Description**

Display an interactive GUI to demonstrate PBS Modelling examples.

The example source files can be found in the R directory `.../library/PBSmodelling/examples`.

**Usage**

```
runExamples()
```

**Details**

Some examples use external packages which must be installed to work correctly:

BRugs - LinReg, MarkRec, and CCA;

deSolve/PBSdresolve - FishRes;

PBSmapping - FishTows.

**Note**

The examples are copied from `.../library/PBSmodelling/examples` to R's current temporary working directory and run from there.

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[runDemos](#), [runExample](#)

---

scalePar

*Scale Parameters to [0,1]*

---

**Description**

Scale parameters for function minimization by calcMin.

**Usage**

```
scalePar(pvec)
```

**Arguments**

`pvec` a data frame comprising four columns - `c("val", "min", "max", "active")` and as many rows as there are model parameters. The "active" field (logical) determines whether the parameters are estimated (TRUE) or remain fixed (FALSE).

**Details**

Scaling algorithm:  $S = \frac{2}{\pi} a \sin \sqrt{\frac{P - P_{min}}{P_{max} - P_{min}}}$

**Value**

Parameter vector scaled between 0 and 1.

**Author(s)**

Jon T. Schnute, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**See Also**

[restorePar](#), [calcMin](#), [GT0](#)

**Examples**

```
local(envir=.PBSmodEnv,expr={
  pvec <- data.frame(val=c(1,100,10000),min=c(0,0,0),max=c(5,500,50000),
    active=c(TRUE,TRUE,TRUE))
  S <- scalePar(pvec)
  print(cbind(pvec,S))
})
```

---

selectDir

*Display Dialogue: Select directory*

---

**Description**

Display and select a system directory using a call to `tcltk::tkchooseDirectory`.

**Usage**

```
selectDir(initialdir=getwd(), mustexist=TRUE, title="", usewidget=NULL,
  targetdir=NULL, relative=FALSE)
```

**Arguments**

<code>initialdir</code>	character – initially selected directory
<code>mustexist</code>	logical – if TRUE, only an existing directory can be selected
<code>title</code>	character – title for the prompt window
<code>usewidget</code>	character – store the selected directory in the named entry widget
<code>targetdir</code>	character – a alternative directory to <code>initialdir</code> from which to start a search
<code>relative</code>	logical – if TRUE, express the selected directory relative to <code>initialdir</code> or <code>targetdir</code>

**Details**

When used in a GUI, the coupling of `selectDir` with `setGUIoptions` necessarily ties the `initialdir` to the `usewidget` directory. A new argument `targetdir` allows the user to specify an alternative directory from which to search that is independent of the `initialdir`-`usewidget` coupling.

**Value**

The directory path selected by the user.

**Author(s)**

**Alex Couture-Beil**, Software Engineer, Earthly Technologies, Victoria BC  
 Maintainer: **Rowan Haigh**, Program Head – Offshore Rockfish  
 Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC  
*locus opus*: remote office, Vancouver BC  
 Last modified Rd: 2025-05-26

**See Also**

In package **PBSmodelling**:  
[selectFile](#), [setGUIoptions](#)

**Examples**

```
## Not run:
local(envir=.PBSmodEnv,expr={
  dir(selectDir(title="select a directory to list contents of"))

  #integration with widget via doAction
  createWin( c( "entry foo mode=character width=25",
    "button text=\"select dir\"
    func=doAction action=\"selectDir(usewidget=`foo`)\")" ), astext=TRUE )
})

## End(Not run)
```

---

 selectFile

*Display Dialogue: Open or Save File*


---

**Description**

Display the default **Open** or **Save** prompt provided by the Operating System.

**Usage**

```
selectFile(initialfile="", initialdir=getwd(),
  filetype=list(c("*","All Files")), mode="open", multiple=FALSE,
  title="", defaulttextension="", usewidget=NULL)
```

**Arguments**

initialfile	initially selected file
initialdir	initially directory the dialog opens
filetype	a list of character vectors indicating file types made available to users of the GUI. Each vector is of length one or two. The first element specifies either the file extension or "*" for all file types. The second element gives an optional descriptor name for the file type. The supplied filetype list appears as a set of choices in the pull-down box labelled "Files of type:".

mode	string: if "save" display <b>Save As</b> prompt, if "open" display <b>Open</b> prompt.
multiple	if TRUE the open prompt can select multiple files. This has no effect for the save prompt.
title	title for the prompt window
defaultextension	default file extension if none is provided by the user
usewidget	store the selected file in the named entry widget

**Value**

The file name and path of the file(s) selected by the user.

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[selectDir](#)

**Examples**

```
## Not run:
local(envir=.PBSmodEnv,expr={
  #integration with widget via doAction
  createWin( c( "entry foo mode=character width=60",
    "button text=\"select file\"
    func=doAction action=\"selectFile(usewidget=`foo`)\")" ), astext=TRUE )
})
## End(Not run)
```

---

setFileOption

*Set a PBS File Path Option Interactively*

---

**Description**

Set a PBS option by browsing for a file. This function provides an alternative to using setPBSoptions when setting an option that has a path to a file as its value.

**Usage**

```
setFileOption(option)
```

**Arguments**

option            name PBS option to change

**Note**

If all the required arguments are missing, it is assumed that the function is being called by a GUI widget.

**Author(s)**

Anisa Egeli, Vancouver Island University, Nanaimo BC

**See Also**

[setPathOption](#), [setPBSoptions](#)

**Examples**

```
## Not run:
local(envir=.PBSmodEnv,expr={
  setPathOption("editor")
})
## End(Not run)
```

---

setGUIoptions

*Set PBS Options from Widget Values*

---

**Description**

Set PBS options from corresponding values of widgets in a GUI.

**Usage**

```
setGUIoptions(option)
```

**Arguments**

option            the name of a single option or the string "\*".

**Details**

A GUI may have PBS options that it uses, which have corresponding widgets that are used for entering values for these options. These are declared by `declareGUIoptions`.

If the option argument is the name of an option, `setGUIoptions` transfers the value of this option from a same-named widget into PBS options global R environment database.

If the option argument is "\*", then all the options that have been declared by `declareGUIoptions` will be transferred in this fashion.

To use this function in a *window description file*, the option argument must be specified as the action of the widget that calls `setGUIoptions` – `action=editor` or `action=*` for example.

**Note**

If all the required arguments are missing, it is assumed that the function is being called by a GUI widget.

**Author(s)**

Anisa Egeli, Vancouver Island University, Nanaimo BC

**See Also**

[declareGUIOptions](#), [getGUIOptions](#), [setPBSOptions](#),

**Examples**

```
## Not run:
local(envir=.PBSmodEnv,expr={
  setGUIOptions("editor")
})

## End(Not run)
```

---

setPathOption

*Set a PBS Path Option Interactively*

---

**Description**

Set a PBS option by browsing for a directory. This function provides an alternative to using setPBSOptions when setting an option that has a path as its value.

**Usage**

```
setPathOption(option)
```

**Arguments**

option            name PBS option to change

**Note**

If all the required arguments are missing, it is assumed that the function is being called by a GUI widget.

**Author(s)**

Anisa Egeli, Vancouver Island University, Nanaimo BC

**See Also**

[setFileOption](#), [setPBSOptions](#)

## Examples

```
## Not run:  
local(envir=.PBSmodEnv,expr={  
  setPathOption("myPath")  
})  
  
## End(Not run)
```

---

setPBSext

*Set Command to Associate with File Name Extension*

---

## Description

Set a command with an associated extension, for use in `openFile`. The command must specify where the target file name is inserted by indicating a `'%f'`.

## Usage

```
setPBSext(ext, cmd)
```

## Arguments

`ext` character – string specifying the extension suffix.  
`cmd` character – command string to associate with the extension.

## Note

These values are not saved from one *PBS Modelling* session to the next.

## Author(s)

**Alex Couture-Beil**, Software Engineer  
Earthly Technologies, Victoria BC

Maintainer: **Rowan Haigh**, Program Head – Offshore Rockfish  
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC  
*locus opus*: remote office, Vancouver BC  
Last modified Rd: 2023-01-05

## See Also

In package **PBSmodelling**:  
[getPBSext](#), [openFile](#), [clearPBSext](#)

**Examples**

```

## Not run:
local(envir=.PBSmodEnv,expr={
  ## Set 'Firefox' to open '.html' files (only applicable if Firefox is NOT default web browser)
  setPBSext("html", "'c:/Program Files/Mozilla Firefox/firefox.exe" file://%f')
  openFile("foo.html")

  ## Set 'notepad' to open '.txt' files (only applicable if notepad is NOT default editor)
  ## Note that commands to editors (at least in Windows) do not use 'file://'.
  setPBSext('txt', "'C:/Windows/notepad.exe" %f')
  openFile("foo.txt")
})

## End(Not run)

```

---

setPBSOptions	<i>Set A User Option</i>
---------------	--------------------------

---

**Description**

Options set by the user for use by other functions.

**Usage**

```
setPBSOptions(option, value, sublist=FALSE)
```

**Arguments**

option	name of the option to set.
value	new value to assign this option.
sublist	if value is a sublist (list component) of option, this list component can be changed individually using sublist=TRUE.

**Details**

Objects can be placed into the PBS options manager (see [PBSOptions-class](#)).  
If the user wishes to change the object associated with an option, issue the command:

```
setPBSOptions("someOldOption",someNewOption)
```

If an option comprises a list object, a user can alter specific components of the list by activating the sublist argument:

```
setPBSOptions(option="myList", value=list(gamma=130), sublist=TRUE)
```

See example below.

**Note**

A value `.PBSmod$.options$.optionsChanged` is set to TRUE when an option is changed, so that the user doesn't always have to be prompted to save the options file.  
 By default, `.PBSmod$.options$.optionsChanged` is not set or NULL.  
 Also, if an option is set to "" or NULL then it is removed.  
`.initPBSoptions()` is now called first (options starting with a dot "." do not set `.optionsChanged`).

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[getPBSoptions](#), [writePBSoptions](#), [readPBSoptions](#)

**Examples**

```
local(envir=.PBSmodEnv,expr={
  myList=list(alpha=1,beta=2,gamma=3,delta=4)
  setPBSoptions(option="myList", myList)
  cat("Original myList:\n-----\n")
  print(getPBSoptions("myList"))
  setPBSoptions(option="myList", value=list(gamma=130), sublist=TRUE)
  cat("Revised myList:\n-----\n")
  print(getPBSoptions("myList"))
})
```

---

setwdGUI

*Browse for Working Directory and Optionally Find Prefix*

---

**Description**

Allows the user to browse a directory tree to set the working directory. Optionally, files with given suffixes can be located in the new directory.

**Usage**

```
setwdGUI()
```

**Examples**

```
## Not run:
local(envir=.PBSmodEnv,expr={
  createWin( "button text=\\"Change working directory\\" func=setwdGUI", astext=TRUE )
})

## End(Not run)
```

---

setWidgetColor	<i>Update Widget Color</i>
----------------	----------------------------

---

### Description

Update the foreground and background colors of a widget

### Usage

```
setWidgetColor(name, radioValue, winName = .PBSmodEnv$.PBSmod$.activeWin, ...)
```

### Arguments

name	the name of the widget
radioValue	if specified, modify a particular radio option, as identified by the value, rather than the complete set (identified by the common name)
winName	window from which to select the GUI widget. The window that most recently receive user input is used by default if winname is not supplied
...	any combination of "fg", "bg", "disablefg", "disablebg", "entryfg", "entrybg", "noeditfg", "noeditbg" arguments, depending on type of widget - see details

### Details

The setWidgetColor function allows dynamic updating of widget colors during program execution. However, two factors determine whether dynamic color updating is possible for a particular widget: (i) the type of widget, and (ii) the nature of the Tk implementation in the underlying widget library. Thus, a given widget may not support all combinations of colour variables. The following widgets support the corresponding options:

**button:** fg, bg, disablefg

**check:** fg, bg, disablefg, entryfg, entrybg

**data:** entryfg, entrybg, noeditfg, noeditbg

**droplist:** fg, bg

**entry:** entryfg, entrybg, noeditfg, noeditbg

**label:** fg, bg

**matrix:** entryfg, entrybg, noeditfg, noeditbg

**object:** entryfg, entrybg, noeditfg, noeditbg

**progressbar:** fg, bg

**radio:** fg, bg

**slide:** fg, bg

**spinbox:** entryfg, entrybg

**text:** fg, bg

**vector:** entryfg, entrybg, noeditfg, noeditbg

These options are described in the PBSmodelling User Guide under Appendix A.

Be aware that Tk uses gray for the highlight color during a selection operation. This means that when the background colour is also gray, there is no visual clue that the value has been selected for a copy operation.

### Author(s)

Alex Couture-Beil (VIU, Nanaimo BC) and Allen R. Kronlund (PBS, Nanaimo BC)

### Examples

```
## Not run:
local(envir=.PBSmodEnv,expr={
  createWin("label \"hello world\" name=hello", astext=TRUE)
  setWidgetColor( "hello", bg="lightgreen", fg="purple" )
})
local(envir=.PBSmodEnv,expr={
  createWin("vector names=v length=3 values=\"1 2 3\"", astext=TRUE)
  setWidgetColor( "v[1]", entrybg="lightgreen", entryfg="purple" )
  setWidgetColor( "v[2]", entrybg="green", entryfg="purple" )
  setWidgetColor( "v[3]", entrybg="forestgreen", entryfg="purple" )
})

## End(Not run)
```

---

setWidgetState	<i>Update Widget State</i>
----------------	----------------------------

---

### Description

Update the read-only state of a widget.

### Usage

```
setWidgetState( varname, state, radiovalue, winname, warn=TRUE )
```

### Arguments

varname	the name of the widget
state	"normal" or "disabled" and for some widgets "readonly" as described under Details below.
radiovalue	if specified, disable a particular radio option, as identified by the value, rather than the complete set (identified by the common name)
winname	window from which to select the GUI widget. The window that most recently receive user input is used by default if winname is not supplied.
warn	if TRUE, display a warning if readonly is converted to disabled (only applies for widgets that don't accept readonly)

## Details

The `setWidgetState` function allows dynamic control of widget functioning during program execution. The function serves as a wrapper for the `tkconfigure` function available in the underlying Tk libraries used by PBS Modelling. Thus, `setWidgetState` is only available for those widgets that use Tk library widgets.

The state of the following PBS Modelling widgets can be set to "normal" or "disabled": button, check, data, droplist, entry, matrix, object, radio, slide, spinbox, table, text, and vector. When the state variable is set to "disabled", values displayed in the widget cannot be changed or copied except in the case of the object and table widgets which permit the values to be copied.

The data, entry, matrix, and vector widgets support a "readonly" state that allows values displayed in the widget to be copied but not changed. The displayed value can be selected using the keyboard or mouse. However, the copy and paste operations can only be accomplished via Ctrl-C and Ctrl-V, respectively, not the mouse.

Be aware that Tk uses gray for the highlight color during a selection operation. This means that when the background colour is also gray, there is no visual clue that the value has been selected for a copy operation.

Exceptions to the behaviour determined by state include the object, table and text widgets. There is no "readonly" state applicable to these widgets. Nevertheless, the values displayed can be copied even when the state is "disabled".

Individual radio widgets grouped by the name variable of a radio declaration can be updated by specifying `radiovalue` in the call to `setWidgetState`.

The state of individual elements in the data, matrix, and vector widgets can be updated by indexing. For the vector and matrix widgets any element can be addressed by appending the desired index to the widget name using square brackets (e.g., "myVec[2]" or "myMatrix[2,3]"). The data widget is indexed differently than the matrix widget by adding "d" after the brackets (e.g., "myData[1,1]d"). This change in syntax is required for internal coding of PBS Modelling.

## Author(s)

Alex Couture-Beil (VIU, Nanaimo BC) and Allen R. Kronlund (PBS, Nanaimo BC)

## Examples

```
## Not run:
local(envir=.PBSmodEnv,expr={
  winDesc <- c('vector length=3 name=vec labels="normal disabled readonly" values="1 2 3"',
              "matrix nrow=2 ncol=2 name=mat", "button name=but_name" );
  createWin(winDesc, astatic=TRUE)

  setWidgetState( "vec[1]", "normal" )
  setWidgetState( "vec[2]", "disabled" )
  setWidgetState( "vec[3]", "readonly" )

  setWidgetState( "mat", "readonly" )
  setWinVal( list( mat = matrix( 1:4, 2, 2 ) ) )

  #works for buttons too
  setWidgetState( "but_name", "disabled" )
}
```

```
})  
## End(Not run)
```

---

setWinAct	<i>Add a Window Action to the Saved Action Vector</i>
-----------	---

---

### Description

Append a string value specifying an action to the first position of an action vector.

### Usage

```
setWinAct(winName, action)
```

### Arguments

winName	window name where action is taking place.
action	string value describing an action.

### Details

When a function is called from a GUI, a string descriptor associated with the action of the function is stored internally (appended to the first position of the action vector). A user can utilize this action as a type of argument for programming purposes. The command `getWinAct()[1]` yields the latest action.

Sometimes it is useful to “fake” an action. Calling `setWinAct` allows the recording of an action, even if a button has not been pressed.

### Author(s)

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

---

setWinVal	<i>Update Widget Values</i>
-----------	-----------------------------

---

### Description

Update a widget with a new value.

### Usage

```
setWinVal(vars, winName)
```

**Arguments**

vars	a list or vector with named components.
winName	window from which to select GUI widget values. The default takes the window that has most recently received new user input.

**Details**

The vars argument expects a list or vector with named elements. Every element name corresponds to the widget name which will be updated with the supplied element value.

The vector, matrix, and data widgets can be updated in several ways. If more than one name is specified for the names argument of these widgets, each element is treated like an entry widget.

If however, a single name describes any of these three widgets, the entire widget can be updated by passing an appropriately sized object.

Alternatively, any element can be updated by appending its index in square brackets to the end of the name. The data widget is indexed differently than the matrix widget by adding "d" after the brackets. This tweak is necessary for the internal coding (bookkeeping) of *PBS Modelling*. Example: "foo[1,1]d".

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[getWinVal](#), [createWin](#)

**Examples**

```
## Not run:
local(envir=.PBSmodEnv,expr={
  winDesc <- c("vector length=3 name=vec",
    "matrix nrow=2 ncol=2 name=mat", "slideplus name=foo");
  createWin(winDesc, astext=TRUE)
  setWinVal(list(vec=1:3, "mat[1,1]"=123, foo.max=1.5, foo.min=0.25, foo=0.7))
})

## End(Not run)
```

---

show0

*Convert Numbers into Text with Specified Decimal Places*

---

**Description**

Return a character representation of a number with added zeroes out to a specified number of decimal places.

**Usage**

```
show0(x, n, add2int=FALSE, round2n=FALSE)
```

**Arguments**

x	numeric data (scalar, vector, or matrix).
n	number of decimal places to show, including zeroes.
add2int	if TRUE, add zeroes to integers after the decimal.
round2n	if TRUE, round x first to n decimal places.

**Value**

A scalar/vector of strings representing numbers. Useful for labelling purposes.

**Note**

By default, this function does not round or truncate numbers. It simply adds zeroes if n is greater than the available digits in the decimal part of a number. The user can choose to round the numbers first by setting argument round2n = TRUE.

**Author(s)**

Rowan Haigh, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**See Also**

[pad0](#), [GT0](#)

**Examples**

```
local(envir=.PBSmodEnv,expr={
  oldpar = par(no.readonly=TRUE)
  frame()

  #do not show decimals on integers
  addLabel(0.25,0.75,show0(15.2,4))
  addLabel(0.25,0.7,show0(15.1,4))
  addLabel(0.25,0.65,show0(15,4))

  #show decimals on integers
  addLabel(0.25,0.55,show0(15.2,4,TRUE))
  addLabel(0.25,0.5,show0(15.1,4,TRUE))
  addLabel(0.25,0.45,show0(15,4,TRUE))
  par(oldpar)
})
```

---

showAlert                      *Display a Message in an Alert Window*

---

### Description

Display an alert window that contains a specified message and an OK button for dismissing the window.

### Usage

```
showAlert(message, title="Alert", icon="warning")
```

### Arguments

message	message to display in alert window
title	title of alert window
icon	icon to display in alert window; options are "error", "info", "question", or "warning".

### Author(s)

Anisa Egeli, Vancouver Island University, Nanaimo BC

### See Also

[getYes](#)

### Examples

```
## Not run:
local(envir=.PBSmodEnv,expr={
  showAlert("Hello World!")
})

## End(Not run)
```

---

showArgs                      *Display Expected Widget Arguments*

---

### Description

For each widget specified, display its arguments in order with their default values. The display list can be expanded to report each argument on a single line.

### Usage

```
showArgs(widget, width=70, showargs=FALSE)
```

**Arguments**

widget	vector string of widget names; if not specified (default), the function displays information about all widgets in alphabetical order.
width	numeric width used by <code>strwrap</code> to wrap lines of the widget usage section.
showargs	logical; if TRUE, the display also lists each argument on single line after the widget usage section.

**Value**

A text stream to the R console. Invisibly returns the widget usage lines.

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

---

 showHelp

---

*Display HTML Help Pages for Packages in Browser*


---

**Description**

Display the help pages for installed packages that match the supplied pattern in an HTML browser window.

**Usage**

```
showHelp(pattern="methods", ...)
```

**Arguments**

pattern	string pattern to match to package names
...	allows user to specify two additional arguments: remote - character string giving a valid URL for the R_HOME directory on a remote location; update - logical: if TRUE, attempt to update the package index to reflect the currently available packages. (Not attempted if remote is non-NULL.)

**Details**

The specified pattern is matched to R-packages installed on the user's system. The code uses the `utils` function `browseURL` to display the HTML Help Pages using a browser that the system associates with `html` extensions. (See help for `browseURL` for other operating systems.)

**Value**

A list is invisibly returned, comprising:

Apacks	all packages installed on user's system
Spacks	selected packages based on specified pattern
URLs	path and file name of HTML Help Page

Help pages are displayed in a separate browser window.

**Note**

The connection time for browsers (at least in Windows OS) is slow. If the HTML browser program is not already running, multiple matching pages will most likely not be displayed. However, subsequent calls to showHelp should show all matches.

This function will now only work in R ( $\geq 3.2.0$ ) or from SVN revision  $\geq 67548$ . The CRAN gurus now disallow direct calls to `tools::httpdPort`.

**Author(s)**

Rowan Haigh, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**See Also**

[viewCode](#), [showPacks](#)

---

showPacks

*Show Packages Required But Not Installed*

---

**Description**

Show the packages specified by the user and compare these to the installed packages on the user's system. Display packages not installed.

**Usage**

```
showPacks(packs=c("PBSmodelling", "PBSmapping", "PBSdresolve",
  "rgl", "deSolve", "akima", "deldir", "sp", "maptools", "KernSmooth"))
```

**Arguments**

packs            string vector of package names that are compared to installed packages.

**Value**

Invisibly returns a list of Apacks (all packages installed on user's system), Ipacks (packages in packs that are installed), and Mpacks (packages that are missing).

**Author(s)**

Jon T. Schnute, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

---

showRes

*Show Results of Expression Represented by Text*

---

**Description**

Evaluate the supplied expression, reflect it on the command line, and show the results of the evaluation.

**Usage**

```
showRes(x, cr=TRUE, pau=TRUE)
```

**Arguments**

x	an R expression to evaluate
cr	logical: if TRUE, introduce extra carriage returns
pau	logical: if TRUE, pause after expression reflection and execution

**Value**

The results of the expression are return invisibly.

**Author(s)**

Jon T. Schnute, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**Examples**

```
local(envir=.PBSmodEnv, expr={  
  showRes("x=rnorm(100)", pau=FALSE)  
})
```

---

showVignettes	<i>Display Vignettes for Packages</i>
---------------	---------------------------------------

---

**Description**

Create a GUI that displays all vignettes for installed packages. The user can choose to view the source file for building the vignette or the final .pdf file.

**Usage**

```
showVignettes(package)
```

**Arguments**

package            character string specifying package name that exists in the user's R library

**Details**

If the argument package is not specified, the function will look for vignettes in all packages installed on the user's system. The user can choose to view the source file for building the vignette (usually \*.Rnw or \*.Snw files) or the final build from the source code (\*.pdf).

showVignettes uses the **PBSmodelling** function `openFile` to display the .Rnw and .pdf files using programs that the system associates with these extensions. On systems that do not support file extension associations, the function `setPBSext` can temporarily set a command to associate with an extension.

**Author(s)**

Anisa Egeli, Vancouver Island University, Nanaimo BC

**See Also**

[showHelp](#), [openFile](#), [setPBSext](#), [getPBSext](#)

---

sortHistory	<i>Sort an Active or Saved History</i>
-------------	--

---

**Description**

Utility to sort history. When called without any arguments, an interactive GUI is used to pick which history to sort. When called with `hisname`, sort this active history widget. When called with `file` and `outfile`, sort the history located in `file` and save to `outfile`.

**Usage**

```
sortHistory(file="", outfile=file, hisname="")
```

**Arguments**

file	file name of saved history to sort.
outfile	file to save sorted history to.
hisname	name of active history widget and window it is located in, given in the form WINDOW.HISTORY.

**Details**

After selecting a history to sort (either from given arguments, or interactive GUI) the R data editor window will be displayed. The editor will have one column named `"new"` which will have numbers 1,2,3,...,n. This represents the current ordering of the history. You may change the numbers around to define a new order. The list is sorted by reassigning the index in row *i* as index *i*.

For example, if the history had three items 1,2,3. Reordering this to 3,2,1 will reverse the order; changing the list to 1,2,1,1 will remove entry 3 and create two duplicates of entry 1.

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[importHistory](#), [initHistory](#)

---

talk-class

*S4: Present Talk Classes*

---

**Description**

The function `presentTalk` is a tool that facilitates lectures and workshops in R. It allows the presenter to show code snippets alongside their execution, making use of R's graphical capabilities.

For `presentTalk` to work, six S4 class objects are created:

talk	root element that constitutes a talk;
section	branch element that defines a section within a talk;
text	leaf element that specifies text to be printed on the R console;
file	leaf element that specifies files to be opened by the OS;
code	leaf element that specifies R code to be executed;
break	leaf element that specifies where to allow a break in the talk.

The leaf elements, also termed *primitive* elements, occur in isolation and cannot contain other elements. Therefore, only two levels of nesting are supported: sections within a talk and primitives within a section.

See Appendix B in the **PBSmodelling** User's Guide for more information.

## Details

This function uses a convenience function called `xmlGetAttr` (from the package **XML**) that retrieves the value of a named attribute in an XML node.

The function `presentTalk` translates the XML code into a list structure called `.presentTalk` below the global object `.PBSmod`. The GUI is represented as a list structure called `presentwin` under `.PBSmod`, as for all GUI objects in **PBSmodelling**.

## Slots Available

### talk

<code>name</code>	character	string giving the name of the talk (required)
<code>sections</code>	list	list of sections within the talk
<code>files</code>	list	list of files within the talk

### section

<code>name</code>	character	string giving the name of the section (required)
<code>items</code>	list	list of the four primitive (leaf-element) S4 classes
<code>button</code>	logical	should GUI have a button that selects section?
<code>col</code>	integer	column in lower section of GUI to place button
<code>section_id</code>	integer	specify if section does not immediately follow a talk

### text

<code>text</code>	character	text to display on the R console
<code>"break"</code>	logical	break the presentation after displaying the text specified?

### file

<code>name</code>	character	string giving the name in the GUI for a group of files to open (required)
<code>filename</code>	character	individual file names associated with the group name in the GUI
<code>"break"</code>	logical	break the presentation after opening the group of files?
<code>button</code>	logical	should GUI add a button that opens this group of files?
<code>col</code>	integer	column in lower section of GUI to place button

### code

<code>show</code>	logical	show the code snippet in the R console?
<code>print</code>	logical	print the results of running the R code?
<code>code</code>	character	the actual chunk of R code
<code>"break"</code>	character	string describing where to introduce breaks in the code segment
<code>eval</code>	logical	evaluate the R code?

### break

<code>.xData</code>	NULL	allows a break in the talk for user interaction on the R console.
---------------------	------	---

## Creating S4 Objects

Objects can be created by calls of the form:

```
new("talk", name=name)
new("section",
  name      = node$attributes["name"],
  button    = as.logical(xmlGetAttr(node, "button", FALSE)),
```

```

    col      = as.integer(xmlGetAttr(node,"col",2))
  new("text",
    text     = xmlValue(node),
    "break"  = as.logical(xmlGetAttr(node,"break",TRUE))
  new("file",
    name     = xmlGetAttr(node,"name",""),
    "break"  = as.logical(xmlGetAttr(node,"break",TRUE)),
    filename = xmlValue(node),
    button   = as.logical(xmlGetAttr(node,"button",FALSE)),
    col      = as.integer(xmlGetAttr(node,"col",3))
  new("code",
    show     = as.logical(xmlGetAttr(node,"show",TRUE)),
    print    = as.logical(xmlGetAttr(node,"print",TRUE)),
    code     = xmlValue(node),
    "break"  = tolower(xmlGetAttr(node,"break","print")))
  new("break")

```

**Author(s)**

**Alex Couture-Beil**, Software Engineer  
 Earthly Technologies, Victoria BC

Maintainer: **Rowan Haigh**, Program Head – Offshore Rockfish  
 Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC  
*locus opus*: remote office, Vancouver BC  
 Last modified Rd: 2023-10-18

**See Also**

[presentTalk](#) for presenting a talk in R.  
[xmlGetAttr](#) for retrieving the value of a named attribute in an XML node.  
[setClass](#) for creating a class definition.  
[PBSoptions-class](#) for a complicated S4 class implementation.

---

 testAlpha

*Test Various Alpha Transparency Values*


---

**Description**

Display how the alpha transparency for `rgb()` varies.

**Usage**

```
testAlpha(alpha=seq(0,1,len=25), fg="blue", bg="yellow",
  border="black", grid=FALSE, ...)
```

**Arguments**

alpha	numeric – vector of alpha transparency values values from 0 to 1.
fg	character – foreground colour of the top shape that varies in transparency.
bg	character – background colour (remains constant) of the underlying shape.
border	character – border colour (which also changes in transparency) of the foreground polygon.
grid	logical – if TRUE, lay a grey grid on the background colour.
...	dots – additional graphical arguments to send to the the plotting functions.

**Value**

Invisibly returns the compound RGB matrix for fg, alpha, bg, and border.

**Author(s)**

**Jon T. Schnute**, Research Scientist Emeritus  
Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

Maintainer: **Rowan Haigh**, Program Head – Offshore Rockfish  
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC  
*locus opus*: remote office, Vancouver BC  
Last modified Rd: 2023-10-18

**See Also**

In package **PBSmodelling**:  
[testCol](#), [testLty](#), [testLwd](#)  
In package **PBSstools**:  
[testPch](#)

---

testCol

*Display Colours Available Using a Set of Strings*

---

**Description**

Display colours as round patches in a plot. Useful for programming purposes. Colours can be specified in any of 3 different ways:

- (i) by colour name,
- (ii) by hexadecimal colour code created by `rgb()`, or
- (iii) by calling one of the colour palettes.

**Usage**

```
testCol(cnam=colors()[sample(length(colors()),15)])
```

**Arguments**

cnam                    character – vector of colour names to display. Defaults to 15 random names from the color palette to use as patterns.

**Author(s)**

**Rowan Haigh**, Program Head – Offshore Rockfish  
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC  
*locus opus*: remote office, Vancouver BC  
Last modified Rd: 2023-10-18

**See Also**

In package **PBSmodelling**:

[pickCol](#), [testAlpha](#)

In package **PBSstools**:

[testPch](#)

In package **grDevices**:

[palettes](#)

**Examples**

```
local(envir=.PBSmodEnv,expr={
  # mix and match patterns
  testCol(c("sky", "fire", "sea", "wood"))
})
local(envir=.PBSmodEnv,expr={
  # display transparencies of blue
  testCol(rgb(0,0,1,seq(0.05,1,0.05)))
})
local(envir=.PBSmodEnv,expr={
  # display colours of the rainbow
  testCol(rainbow(64,end=0.75))
})
local(envir=.PBSmodEnv,expr={
  # display basic palette colours
  testCol(1:length(palette()))
})
local(envir=.PBSmodEnv,expr={
  # mix colour types
  testCol(c("#9e7ad3", "purple", 6))
})
```

---

testLty

*Display Line Types Available*


---

**Description**

Display line types available.

**Usage**

```
testLty(newframe=TRUE, n=1:18, ...)
```

**Arguments**

`newframe`      logical – if TRUE, create a new blank frame, otherwise overlay current frame.  
`n`                numeric – vector of line type numbers.  
`...`            dots – additional arguments for function lines.

**Note**

Quick representation of line types for reference purposes.

**Author(s)**

**Rowan Haigh**, Program Head – Offshore Rockfish  
 Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC  
*locus opus*: remote office, Vancouver BC  
 Last modified Rd: 2023-10-18

**See Also**

In package **PBSmodelling**:  
[testLwd](#), [testCol](#)  
 In package **PBSstools**:  
[testPch](#)

---

 testLwd

---

*Display Line Widths*


---

**Description**

Display line widths. User can specify particular ranges for `lwd`. Colours can also be specified and are internally repeated as necessary.

**Usage**

```
testLwd(lwd=1:20, col=c("black", "blue"), newframe=TRUE)
```

**Arguments**

`lwd`            line widths to display. Ranges can be specified.  
`col`            colours to use for lines. Patterns are repeated if `length(lwd) > length(col)`  
`.`  
`newframe`      if TRUE, create a new blank frame, otherwise overlay current frame.

**Author(s)**

Rowan Haigh, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**Examples**

```
local(envir=.PBSmodEnv,expr={
  testLwd(3:15,col=c("salmon","aquamarine","gold"))
})
```

---

testWidgets

*Display Sample GUIs and their Source Code*


---

**Description**

Display an interactive GUI to demonstrate the available widgets in PBS Modelling. A text window displays the *window description file* source code. The user can modify this sample code and recreate the test GUI by pressing the button below.

The *Window Description Files* can be found in the R directory  
`.../library/PBSmodelling/testWidgets`.

**Usage**

```
testWidgets()
```

**Details**

Following are the widgets and default values supported by PBS Modelling. For detailed descriptions, see Appendix A in 'PBSModelling-UG.pdf' located in the R directory `.../library/PBSmodelling/doc`.

```
button text="Calculate" font="" fg="black" bg="" disablefg=NULL
  width=0 name=NULL function="" action="button" sticky=""
  padx=0 pady=0
```

```
check name mode="logical" checked=FALSE text="" font="" fg="black"
  bg="" disablefg=NULL function="" action="check" edit=TRUE
  sticky="" padx=0 pady=0
```

```
data nrow ncol names modes="numeric" rowlabels="" collabels=""
  rownames="X" colnames="Y" font="" fg="black" bg="" entryfont=""
  entryfg="black" entrybg="white" noeditfg="black" noeditbg="gray"
  values="" byrow=TRUE function="" enter=TRUE action="data"
  edit=TRUE width=6 borderwidth=0 sticky="" padx=0 pady=0
```

```
droplist name values=NULL choices=NULL labels=NULL selected=1
  add=FALSE font="" fg="black" bg="white" function="" enter=TRUE
  action="droplist" edit=TRUE mode="character" width=20
  sticky="" padx=0 pady=0
```

```

entry name value="" width=20 label=NULL font="" fg="" bg=""
    entryfont="" entryfg="black" entrybg="white" noeditfg="black"
    noeditbg="gray" edit=TRUE password=FALSE function="" enter=TRUE
    action="entry" mode="numeric" sticky="" padx=0 pady=0

grid nrow=1 ncol=1 toptitle="" sidetitle="" topfont="" sidefont=""
    topfg=NULL sidefg=NULL fg="black" topbg=NULL sidebg=NULL bg=""
    byrow=TRUE borderwidth=1 relief="flat" sticky="" padx=0 pady=0

history name="default" function="" import="" fg="black" bg=""
    entryfg="black" entrybg="white" text=NULL textsize=0 sticky=""
    padx=0 pady=0

image file=NULL varname=NULL subsample=NULL sticky="" padx=0 pady=0

include file=NULL name=NULL

label text="" name="" mode="character" font="" fg="black" bg=""
    sticky="" justify="left" anchor="center" wraplength=0 width=0
    padx=0 pady=0

matrix nrow ncol names rowlabels="" collabels="" rownames=""
    colnames="" font="" fg="black" bg="" entryfont="" entryfg="black"
    entrybg="white" noeditfg="black" noeditbg="gray" values=""
    byrow=TRUE function="" enter=TRUE action="matrix" edit=TRUE
    mode="numeric" width=6 borderwidth=0 sticky="" padx=0 pady=0

menu nitems=1 label font="" fg="" bg=""

menuitem label font="" fg="" bg="" function action="menuitem"

notebook tabs name=NULL selected=1 tabpos="top" font="" fg=NULL
    bg=NULL width=0 height=0 homogeneous=FALSE arcradius=2
    tabbevelsize=0 function=NULL action="notebook" sticky="we"
    padx=0 pady=0

null bg="" padx=0 pady=0

object name rowshow=0 font="" fg="black" bg="" entryfont=""
    entryfg="black" entrybg="white" noeditfg="black" noeditbg="gray"
    vertical=FALSE collabels=TRUE rowlabels=TRUE function=""
    enter=TRUE action="data" edit=TRUE width=6 borderwidth=0
    sticky="" padx=0 pady=0

progressbar name value=0 maximum=100 style="normal" width=NULL
    height=NULL vertical=FALSE fg=NULL bg=NULL relief="sunken"
    borderwidth=2 sticky="" padx=0 pady=0

```

```

radio name value text="" font="" fg="black" bg="" function=""
  action="radio" edit=TRUE mode="numeric" selected=FALSE
  sticky="" padx=0 pady=0

slide name from=0 to=100 value=NA showvalue=FALSE
  orientation="horizontal" font="" fg="black" bg="" function=""
  action="slide" sticky="" padx=0 pady=0

slideplus name from=0 to=1 by=0.01 value=NA font="" fg="black"
  bg="" entryfont="" entryfg="black" entrybg="white" function=""
  enter=FALSE action="slideplus" sticky="" padx=0 pady=0

spinbox name from to by=1 value=NA label="" font="" fg="black"
  bg="" entryfont="" entryfg="black" entrybg="white" function=""
  enter=TRUE edit=TRUE action="droplist" width=20 sticky=""
  padx=0 pady=0

table name rowshow=0 font="" fg="black" bg="white" rowlabels=""
  collabels="" function="" action="table" edit=TRUE width=10
  sticky="" padx=0 pady=0

text name height=8 width=30 edit=FALSE scrollbar=TRUE fg="black"
  bg="white" mode="character" font="" value="" borderwidth=1
  relief="sunken" sticky="" padx=0 pady=0

vector names length=0 labels="" values="" vecnames="" font=""
  fg="black" bg="" entryfont="" entryfg="black" entrybg="white"
  noeditfg="black" noeditbg="gray" vertical=FALSE function=""
  enter=TRUE action="vector" edit=TRUE mode="numeric" width=6
  borderwidth=0 sticky="" padx=0 pady=0

window name="window" title="" vertical=TRUE bg="#D4D0C8"
  fg="#000000" onclose="" remove=FALSE

```

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[createWin](#), [showArgs](#)

## Description

Get/print objects from or put objects into a temporary work environment called `.PBSmodEnv`. These objects include `.PBSmod`, which controls the GUI system.

## Usage

```
tget(x, penv=NULL, tenv=.PBSmodEnv)
tcall(x, penv=NULL, tenv=.PBSmodEnv)
tprint(x, penv=NULL, tenv=.PBSmodEnv)
tput(x, penv=NULL, tenv=.PBSmodEnv)
```

## Arguments

<code>x</code>	character object – name (with or without quotes) of an object to retrieve or store in the temporary environment.
<code>penv</code>	environment – parent environment, defaults to <code>parent.frame()</code> called from within the function.
<code>tenv</code>	environment – temporary working environment, defaults to <code>.PBSmodEnv</code> .

## Details

These accessor functions were developed as a response to the CRAN repository policy statement: “Packages should not modify the global environment (user’s workspace).”

There are also wrapper functions called `.win.tget`, `.win.tcall`, and `.win.tprint` that can be used in *window description files* to launch functions or print objects from the `.PBSmodEnv` workspace. The wrapper uses `getWinAct` to get the function (or object) name that a user specifies in the *action* argument of a widget command.

## Value

Objects are retrieved from or sent to the temporary working environment to/from the place where the function(s) are called. Additionally, `tcall` invisibly returns the object without transferring, which is useful when the object is a function that the user may wish to call, for example, `tcall(myfunc)()`.

## Note

Additional wrapper functions to access functions in `.PBSmodEnv` are named with the prefix `.win`.

## Author(s)

Rowan Haigh, Program Head – Offshore Rockfish  
 Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC  
*locus opus*: remote office, Vancouver BC  
 Last modified Rd: 2023-10-25

## References

[CRAN Repository Policy](#)

**Examples**

```
## Not run:
local(envir=.PBSmodEnv,expr={
  str(tcall(.PBSmod),2)
})

## End(Not run)
```

---

`unpackList`*Unpack List Elements into Variables*

---

**Description**

Make local or global variables (depending on the scope specified) from the named components of a list.

**Usage**

```
unpackList(x, scope="L")
```

**Arguments**

<code>x</code>	named list to unpack.
<code>scope</code>	If "L", create variables local to the parent frame that called the function. If "P", create variables in the temporary package workspace called <code>.PBSmodEnv</code> . If "G", create global variables.

**Value**

A character vector of unpacked variable names.

**Author(s)**

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

**See Also**

[packList](#), [readList](#), [writeList](#)

**Examples**

```
local(envir=.PBSmodEnv,expr={
  x <- list(a=21,b=23);
  unpackList(x);
  print(a);
})
```

---

`updateGUI`*Update Active GUI With Local Values*

---

**Description**

Update the currently active GUI with values from R's memory at the specified location.

**Usage**

```
updateGUI(scope = "L")
```

**Arguments**

`scope` either "L" for the parent frame, "P" for the temporary work environment `.PBSmodEnv`, "G" for the global environment, or an explicit R environment.

**Details**

If the characteristics of the local R objects do not match those of the GUI objects, the update will fail.

**Value**

Invisibly returns a Boolean vector that specifies whether the objects in the local R environment match items in the active GUI.

**Author(s)**

Rob Kronlund, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**See Also**

[getWinVal](#), [setWinVal](#)

**Examples**

```
## Not run:
local(envir=.PBSmodEnv,expr={
  #law of free food: http://www.phdcomics.com/comics.php?f=1223
  createWin( c(
    "vector names=\"foodquality hunger cost\" values=\"0.6 0.8 0.1\" width=10",
    "entry name=taste edit=F label=taste:" ), astext=TRUE )
  getWinVal( scope="P" )
  taste <- foodquality * hunger / cost
  updateGUI()
})

## End(Not run)
```

---

vbdata	<i>Data: Lengths-at-Age for von Bertalanffy Curve</i>
--------	---

---

**Description**

Lengths-at-age for freshwater mussels (*Anodonta kennerlyi*).

**Usage**

```
data(vbdata)
```

**Format**

A data frame with 16 rows and 2 columns `c("age", "len")`.

**Details**

Data for demonstration of the von Bertalanffy model used in the `calcMin` example.

**Source**

Fisheries and Oceans Canada - Mittertreiner and Schnute (1985)

**References**

Mittertreiner, A. and Schnute, J. (1985) Simplex: a manual and software package for easy nonlinear parameter estimation and interpretation in fishery research. *Canadian Technical Report of Fisheries and Aquatic Sciences* **1384**, xi + 90 pp.

---

vbpars	<i>Data: Initial Parameters for a von Bertalanffy Curve</i>
--------	---

---

**Description**

Starting parameter values for `Linf`, `K`, and `t0` for von Bertalanffy minimization using length-at-age data (`vbdata`) for freshwater mussels (*Anodonta kennerlyi*).

**Usage**

```
data(vbpars)
```

**Format**

A matrix with 3 rows and 3 columns `c("Linf", "K", "t0")`. Each row contains the starting values, minima, and maxima, respectively, for the three parameters.

**Details**

Data for demonstration of the von Bertalanffy model used in the `calcMin` example.

**References**

Mittertreiner, A. and Schnute, J. (1985) Simplex: a manual and software package for easy nonlinear parameter estimation and interpretation in fishery research. *Canadian Technical Report of Fisheries and Aquatic Sciences* **1384**, xi + 90 pp.

---

view	<i>View First/Last/Random n Elements/Rows of an Object</i>
------	--

---

**Description**

View the first or last or random n elements or rows of an object. Components of lists will be subset using iterative calls to `view`.

**Usage**

```
view(obj, n=5, last=FALSE, random=FALSE, print.console=TRUE, ...)
```

**Arguments**

<code>obj</code>	object – an R object to view.
<code>n</code>	numeric – first (default)/last/random n elements/rows of <code>obj</code> to view.
<code>last</code>	logical – if TRUE, last n elements/rows of <code>obj</code> are displayed.
<code>random</code>	logical – if TRUE, n random elements/rows (without replacement) of <code>obj</code> are displayed.
<code>print.console</code>	logical – if TRUE, print the results to the console (default). The results are also returned invisibly should the user wish to assign the output to an object.
<code>...</code>	dots – additional arguments (e.g., <code>replace=TRUE</code> if specifying <code>random=TRUE</code> ).

**Value**

Invisibly returns the results of the call to `view`.

**Note**

If `random=TRUE`, random sampling will take place before the last operator is applied.

**Author(s)**

**Rowan Haigh**, Program Head – Offshore Rockfish  
Pacific Biological Station (PBS), Fisheries & Oceans Canada (DFO), Nanaimo BC  
*locus opus*: remote office, Vancouver BC  
Last modified Rd: 2019-03-12

**See Also**

In package **PBSmodelling**:  
[lisp](#), [showArgs](#), [testCol](#), [viewCode](#)  
 In package **utils**:  
[head](#), [tail](#)

viewCode

*View Package R Code***Description**

View the R code of all functions in a specified package installed on the user's system.

**Usage**

```
viewCode(pkg="PBSmodelling", funs, output=4, ...)
```

**Arguments**

pkg	string name of a package installed on the user's computer.
funs	string vector of explicit function names from pkg to view.
output	numeric value: 1 = function names only, 2 = function names with brief description, 3 = functions and their arguments, and 4 = function R-code (default).
...	allows user to specify two additional arguments for output=2: remote - character string giving a valid URL for the R_HOME directory on a remote location; update - logical: if TRUE, attempt to update the package index to reflect the currently available packages. (Not attempted if remote is non-NULL.)

Also, if user specifies pat=TRUE, then funs, if specified, are treated like patterns.

**Details**

If funs is not specified, then all functions, including hidden (dot) functions are displayed.  
 If the package has a namespace, functions there are also displayed.

**Value**

Invisibly returns source code of all functions in the specified package. The function invokes `openFile` to display the results.

**Note**

Output type 2 (function name with brief description) will now only work in R ( $\geq 3.2.0$ ) or from SVN revision  $\geq 67548$ . The CRAN gurus now disallow direct calls to `tools:::httpdPort`.

**Author(s)**

Rowan Haigh, Pacific Biological Station, Fisheries and Oceans Canada, Nanaimo BC

**See Also**

[showHelp](#), [view](#)

---

writeList

*Write a List to a File in PBS Modelling Format*

---

**Description**

Write an ASCII text representation in either "D" format or "P" format. The "D" format makes use of dput and dget and produces an R representation of the list. The "P" format represents a simple list in an easy-to-read, ad hoc PBSmodelling format.

**Usage**

```
writeList(x, fname, format="D", comments="")
```

**Arguments**

x	R list object to write to an ASCII text file.
fname	file name of the text file to create.
format	format of the file to create: "D" or "P".
comments	vector of character strings to use as initial-line comments in the file.

**Details**

**The D Format:** The "D" format is equivalent to using R's base functions dput and dget, which support all R objects.

**The P Format:**

The "P" format only supports a list that may contain lists, vectors, matrices, arrays, and data frames. Scalars are treated like vectors. It writes each list element using the following conventions:

1. \$ denotes the start of a list element, and the element's name follows this character; for nested lists, \$ separates each nesting level;
2. \$\$, on the next line, denotes a line used to describe the element's structure, which includes object type, mode(s), names (if vector), rownames (if matrix or data), and colnames (if matrix or data); and
3. subsequent lines contain data (one line for a vector and multiple lines for a matrix or other data).

If a list's elements are unnamed, have the name NA, or have the empty string as a name, this function generates names ("P" format only). If two list elements share the same name, the list will export correctly, but it will import incorrectly.

Arrays with three or more dimensions have `dim` and `dimnames` arguments. `dim` describes the dimension of the data (a vector as returned by `dim(some_array)`) and `dimnames` is a vector of length `sum(dim(some_array)+1)` and is constructed as follows:

foreach dimension `d`, first append the name of the dimension `d` and then append all labels within that dimension

Multiple rows of data for matrices or data frames must have equal numbers of entries (separated by whitespace).

Note that array data are written the same format as they are displayed in the R console:

```
nrow=dim()[1], ncol=dim()[2]
```

repeated by scrolling through successively higher dimensions, increasing the index from left to right within each dimension. The flattened table will have `dim()[2]` columns.

For complete details, see "PBSmodelling-UG.pdf" at the location described when loading this package.

### Value

String containing the file name.

### Author(s)

Alex Couture-Beil, Vancouver Island University, Nanaimo BC

### See Also

[packList](#), [readList](#), [unpackList](#)

### Examples

```
## Not run:
local(envir=.PBSmodEnv,expr={
  cwd = getwd(); setwd(tempdir())
  test <- list(a=10,b=euro,c=view(WorldPhones),d=view(USArrests))
  writeList(test,"test.txt",format="P",
    comments=" Scalar, Vector, Matrix, Data Frame")
  openFile("test.txt")
  setwd(cwd)
})
local(envir=.PBSmodEnv,expr={
  cwd = getwd(); setwd(tempdir())
  ##Example of dimnames for Arrays
  dimnames(Titanic)
  writeList( list( Titanic ), format="P")
  setwd(cwd)
})

## End(Not run)
```

---

writePBSoptions	<i>Write PBS Options to an External File</i>
-----------------	--

---

**Description**

Save options that were set using setPBSoptions, setPBSext, or interfaces such as loadC. These options can be reloaded using readPBSoptions.

**Usage**

```
writePBSoptions(fname="PBSoptions.txt")
```

**Arguments**

fname	file name or full path of file to which the options will be saved.
-------	--

**Note**

Options with names starting with "." will not be saved.

**Author(s)**

Anisa Egeli, Vancouver Island University, Nanaimo BC

**See Also**

[readPBSoptions](#), [setPBSoptions](#), [setPBSext](#), [promptWriteOptions](#)

# Index

- \* **M01\_widgetDefs.r**
  - dot-widget.defs, 44
- \* **M02\_guiFuns.r**
  - dot-convert.funs, 28
  - dot-create.funs, 29
  - dot-gui.funs, 31
- \* **M03\_plotFuns.r**
  - plotBubbles, 85
  - testAlpha, 121
  - testCol, 122
  - testLty, 123
- \* **M04\_listFuns.r**
  - dot-list.funs, 36
  - packList, 75
- \* **M05\_projectFuns.r**
  - dot-project.funs, 38
- \* **M06\_supportFuns.r**
  - dot-c.support.funs, 27
  - dot-convert.funs, 28
  - dot-support.funs, 41
  - dot-win.funs, 45
  - getPBSext, 59
  - openFile, 73
  - openUG, 74
  - selectDir, 100
  - setPBSext, 105
  - tget, 127
  - view, 132
- \* **M08\_historyFuns.r**
  - dot-history.funs, 35
- \* **M09\_options\_manager.r**
  - getOptions, 56
- \* **M10\_presentTalk.r**
  - dot-talk.funs, 42
  - talk-class, 119
- \* **arith**
  - calcFib, 7
  - calcGM, 7
- \* **array**
  - genMatrix, 53
- \* **character**
  - convSlashes, 21
  - doAction, 26
  - evalCall, 46
  - showArgs, 114
  - showPacks, 116
  - viewCode, 133
- \* **classes**
  - PBSoptions-class, 80
  - talk-class, 119
- \* **color**
  - lucent, 71
  - pickCol, 82
  - testAlpha, 121
  - testCol, 122
  - testLty, 123
  - testLwd, 124
- \* **connection**
  - readPBSoptions, 94
- \* **datasets**
  - CCA.qbr, 10
  - dot-widget.defs, 44
  - vbdata, 131
  - vbpars, 131
- \* **data**
  - clipVector, 18
- \* **device**
  - chooseWinVal, 12
  - clearRcon, 17
  - expandGraph, 48
  - getChoice, 54
  - resetGraph, 95
  - showHelp, 115
- \* **environment**
  - dot-PBSmodEnv, 38
  - lisp, 68
  - tget, 127
- \* **file**

- findPrefix, 50
- findProgram, 51
- openExamples, 72
- openFile, 73
- packList, 75
- readList, 93
- selectDir, 100
- selectFile, 101
- unpackList, 129
- writeList, 134
- \* **graphs**
  - plotACF, 83
  - plotDens, 87
  - plotTrace, 90
- \* **hplot**
  - drawBars, 46
  - plotAsp, 84
  - plotBubbles, 85
  - plotCsum, 86
  - plotFriedEggs, 88
  - plotSidebars, 89
- \* **interface**
  - compileC, 20
  - dot-win.funs, 45
  - loadC, 69
- \* **iplot**
  - addArrows, 4
  - addLabel, 5
  - addLegend, 6
- \* **list**
  - exportHistory, 49
  - importHistory, 64
  - packList, 75
  - parseWinFile, 78
  - readList, 93
  - sortHistory, 118
  - unpackList, 129
  - writeList, 134
- \* **manip**
  - dot-c.support.funs, 27
  - dot-gui.funs, 31
  - dot-project.funs, 38
  - dot-support.funs, 41
  - tget, 127
- \* **methods**
  - clearAll, 16
  - clearPBSext, 16
  - clearWinVal, 18
  - focusWin, 52
  - getOptions, 56
  - getOptionsFileName, 57
  - getOptionsPrefix, 58
  - getPBSext, 59
  - getPBSoptions, 60
  - getWinAct, 60
  - getWinFun, 61
  - getWinVal, 61
  - loadOptions, 70
  - loadOptionsGUI, 71
  - PBSoptions-class, 80
  - setPBSext, 105
  - setPBSoptions, 106
  - setWidgetColor, 108
  - setWidgetState, 109
  - setWinAct, 111
  - setWinVal, 111
  - updateGUI, 130
- \* **minimization**
  - calcMin, 8
- \* **nonlinear**
  - calcMin, 8
- \* **optimize**
  - calcMin, 8
  - GT0, 63
  - restorePar, 96
  - scalePar, 99
- \* **package**
  - openUG, 74
  - PBSmodelling, 79
  - showPacks, 116
  - viewCode, 133
- \* **print**
  - pad0, 77
  - show0, 112
  - view, 132
- \* **programming**
  - compileC, 20
  - dot-convert.funs, 28
  - dot-create.funs, 29
  - dot-history.funs, 35
  - dot-list.funs, 36
  - dot-talk.funs, 42
  - evalCall, 46
  - loadC, 69
- \* **supportFuns**
  - openUG, 74

**\* utilities**

- chooseWinVal, 12
- cleanProj, 14
- cleanWD, 15
- clipVector, 18
- closeWin, 19
- compileDescription, 21
- createVector, 22
- createWin, 23
- doAction, 26
- findPat, 50
- getChoice, 54
- initHistory, 65
- isWhat, 67
- pause, 79
- runDemos, 97
- runExample, 97
- runExamples, 98
- showArgs, 114
- showHelp, 115
- showRes, 117
- showVignettes, 118
- testCol, 122
- testLty, 123
- testLwd, 124
- testWidgets, 125
- .PBSdimnameHelper (dot-gui.funs), 31
- .PBSmodEnv (dot-PBSmodEnv), 38
- .addslashes (dot-support.funs), 41
- .adjustAllColours (dot-gui.funs), 31
- .autoConvertMode (dot-gui.funs), 31
- .buildgrid (dot-gui.funs), 31
- .catError (dot-gui.funs), 31
- .catError2 (dot-gui.funs), 31
- .check.object.exists (dot-gui.funs), 31
- .cleanLoadC (dot-c.support.funs), 27
- .convertMatrixListToDataFrame  
(dot-convert.funs), 28
- .convertMatrixListToMatrix  
(dot-convert.funs), 28
- .convertMode (dot-convert.funs), 28
- .convertOldGridToNewGrid  
(dot-convert.funs), 28
- .convertParamStrToList  
(dot-convert.funs), 28
- .convertParamStrToVector  
(dot-convert.funs), 28
- .convertVecToArray (dot-convert.funs),  
28
- .createTkFont (dot-create.funs), 29
- .createWidget (dot-create.funs), 29
- .createWidget.data, 29
- .createWidget.history, 29, 31
- .createWidget.history  
(dot-history.funs), 35
- .createWidget.matrix, 29
- .dClose (dot-support.funs), 41
- .dUpdateDesc (dot-support.funs), 41
- .do.gui.call (dot-gui.funs), 31
- .doClean (dot-project.funs), 38
- .doCleanWD (dot-project.funs), 38
- .extractData (dot-gui.funs), 31
- .extractFuns (dot-gui.funs), 31
- .extractVar, 29
- .extractVar (dot-gui.funs), 31
- .fibC (dot-support.funs), 41
- .fibCall (dot-support.funs), 41
- .fibClosedForm (dot-support.funs), 41
- .fibR (dot-support.funs), 41
- .findSquare (dot-support.funs), 41
- .forceMode (dot-support.funs), 41
- .getArrayPts (dot-support.funs), 41
- .getButton (dot-talk.funs), 42
- .getButtons (dot-talk.funs), 42
- .getHiddenEnv (dot-project.funs), 38
- .getHome (dot-project.funs), 38
- .getIndexForSection (dot-talk.funs), 42
- .getMatrixListSize (dot-gui.funs), 31
- .getMenus (dot-talk.funs), 42
- .getParamFromStr, 29
- .getParamFromStr (dot-gui.funs), 31
- .getPrefix (dot-project.funs), 38
- .getSectionNames (dot-talk.funs), 42
- .getSimilarColour (dot-gui.funs), 31
- .getTalkIndexes (dot-talk.funs), 42
- .getValueForWidgetSetup (dot-gui.funs),  
31
- .guiCompileC (dot-c.support.funs), 27
- .guiDyn (dot-c.support.funs), 27
- .guiSource (dot-c.support.funs), 27
- .inCollection (dot-gui.funs), 31
- .initPBSoptions (dot-support.funs), 41
- .libName (dot-c.support.funs), 27
- .loadCRunComparison  
(dot-c.support.funs), 27
- .makeCleanVec (dot-project.funs), 38

- .map.add (dot-gui.funs), 31
- .map.get (dot-gui.funs), 31
- .map.getAll (dot-gui.funs), 31
- .map.init (dot-gui.funs), 31
- .map.set (dot-gui.funs), 31
- .mapArrayToVec (dot-support.funs), 41
- .matrixHelp (dot-gui.funs), 31
- .maxActionSize (dot-widget.defs), 44
- .mergelists (dot-list.funs), 36
- .mergelists2 (dot-project.funs), 38
- .mergeVectors (dot-project.funs), 38
- .nextSlide (dot-talk.funs), 42
- .optionsNotUpdated (dot-project.funs), 38
- .pFormatDefs, 37
- .pFormatDefs (dot-widget.defs), 44
- .packWidgetsIntoGrid (dot-gui.funs), 31
- .parseTalkFile (dot-talk.funs), 42
- .parsegrid (dot-gui.funs), 31
- .parsemenu (dot-gui.funs), 31
- .presentTalkOpenFile (dot-talk.funs), 42
- .prevSlide (dot-talk.funs), 42
- .readList.C (dot-list.funs), 36
- .readList.P (dot-list.funs), 36
- .readList.P.convertData, 29
- .regex.complex (dot-widget.defs), 44
- .regex.logical (dot-widget.defs), 44
- .regex.numeric (dot-widget.defs), 44
- .removeFromList (dot-project.funs), 38
- .searchCollection (dot-gui.funs), 31
- .sectiondrop (dot-talk.funs), 42
- .selectCleanBoxes (dot-project.funs), 38
- .setMatrixElement (dot-gui.funs), 31
- .setOption (dot-project.funs), 38
- .setWinValHelper (dot-gui.funs), 31
- .setsection (dot-talk.funs), 42
- .showLog (dot-project.funs), 38
- .showOptions (getOptions), 56
- .slidedrop (dot-talk.funs), 42
- .sortHelper (dot-history.funs), 35
- .sortHelperActive (dot-history.funs), 35
- .sortHelperFile (dot-history.funs), 35
- .sortWidget (dot-history.funs), 35
- .startSlide (dot-talk.funs), 42
- .stopWidget (dot-gui.funs), 31
- .stripComments (dot-gui.funs), 31
- .stripExt (dot-project.funs), 38
- .stripSlashes (dot-gui.funs), 31
- .stripSlashesVec (dot-gui.funs), 31
- .superobject.redraw (dot-gui.funs), 31
- .superobject.saveValues, 29
- .superobject.saveValues (dot-gui.funs), 31
- .table.getvalue (dot-gui.funs), 31
- .table.setvalue (dot-gui.funs), 31
- .tagDefs, 44
- .tagDefs (dot-widget.defs), 44
- .tagOptionChecks, 44
- .tagOptionChecks (dot-widget.defs), 44
- .tclArrayToVector (dot-support.funs), 41
- .trimWhiteSpace (dot-gui.funs), 31
- .tryOpen (dot-project.funs), 38
- .updateFile (dot-history.funs), 35
- .updateHistory (dot-history.funs), 35
- .updateHistoryButtons (dot-history.funs), 35
- .updateSlide (dot-talk.funs), 42
- .validateWindowDescList (dot-gui.funs), 31
- .validateWindowDescWidgets (dot-gui.funs), 31
- .viewPkgDemo (dot-support.funs), 41
- .viewPkgVignette (dot-support.funs), 41
- .widgetDefs (dot-widget.defs), 44
- .win.chFile (dot-win.funs), 45
- .win.chTest (dot-win.funs), 45
- .win.closeALL (dot-win.funs), 45
- .win.closeChoice (dot-win.funs), 45
- .win.closeSDE (dot-win.funs), 45
- .win.makeChoice (dot-win.funs), 45
- .win.restoreCWD (dot-win.funs), 45
- .win.runExHelperQuit (dot-win.funs), 45
- .win.tcall (dot-win.funs), 45
- .win.tget (dot-win.funs), 45
- .win.tprint (dot-win.funs), 45
- .writeList.P (dot-list.funs), 36
- addArrows, 4, 6
- addHistory (initHistory), 65
- addLabel, 5, 5, 6
- addLegend, 5, 6, 6
- backHistory (initHistory), 65
- break-class (talk-class), 119
- calcFib, 7, 42
- calcGM, 7

- calcMin, 8, 9, 64, 96, 100, 131, 132
- CCA.qbr, 10
- chooseWinVal, 12, 55, 63
- cleanProj, 14, 40
- cleanWD, 15, 17, 40
- clearAll, 16
- clearHistory (initHistory), 65
- clearPBSext, 16, 17, 59, 74, 105
- clearRcon, 17
- clearWinVal, 17, 18, 62
- clipVector, 18
- closeWin, 19, 24
- code-class (talk-class), 119
- col2rgb, 72
- compileC, 20, 28, 70
- compileDescription, 21, 24, 78
- convSlashes, 21
- createVector, 19, 22, 24
- createWin, 19, 21, 22, 23, 26, 78, 112, 127
  
- declareGUIoptions, 25, 56, 104
- doAction, 26, 47
- dot-c.support.funs, 27
- dot-convert.funs, 28
- dot-create.funs, 29
- dot-gui.funs, 31
- dot-history.funs, 35
- dot-list.funs, 36
- dot-PBSmodEnv, 38
- dot-project.funs, 38
- dot-support.funs, 41
- dot-talk.funs, 42
- dot-widget.defs, 44
- dot-win.funs, 45
- drawBars, 46
  
- environment, 24
- evalCall, 26, 46, 90
- expandGraph, 48
- exportHistory, 49, 64, 66
  
- file-class (talk-class), 119
- findPat, 50
- findPrefix, 40, 50
- findProgram, 51
- findSuffix (findPrefix), 50
- firstHistory (initHistory), 65
- focusRgui (clearRcon), 17
- focusWin, 52
  
- forwHistory (initHistory), 65
  
- genMatrix, 53, 86
- getChoice, 12, 54, 63
- getGUIoptions, 25, 40, 55, 95, 104
- getOptions, 56, 81
- getOptionsFileName, 57, 81
- getOptionsPrefix, 58, 81
- getPBSext, 16, 59, 60, 74, 105, 118
- getPBSoptions, 60, 95, 107
- getWinAct, 26, 45, 60
- getWinFun, 61
- getWinVal, 12, 18, 24, 45, 55, 61, 112, 130
- getYes, 62, 114
- glob2rx, 68
- GT0, 9, 63, 77, 96, 100, 113
  
- head, 133
  
- importHistory, 49, 64, 66, 119
- initHistory, 24, 49, 64, 65, 119
- isWhat, 67
  
- jumpHistory (initHistory), 65
  
- lastHistory (initHistory), 65
- lisp, 37, 38, 68, 133
- loadC, 20, 28, 69
- loadOptions, 57, 70, 81
- loadOptionsGUI, 71, 81
- ls, 68
- lucent, 71
  
- openExamples, 72
- openFile, 16, 59, 73, 73, 75, 95, 105, 118
- openUG, 74
  
- packList, 37, 38, 75, 94, 129, 135
- pad0, 77, 113
- palettes, 123
- parseWinFile, 21, 24, 62, 78
- pause, 79
- PBSmodelling, 79
- PBSmodelling-package (PBSmodelling), 79
- PBSoptions-class, 80
- pickCol, 72, 82, 123
- plotACF, 83
- plotAsp, 47, 84
- plotBubbles, 54, 85, 89, 90
- plotCsum, 86

- plotDens, 87
- plotFriedEggs, 88, 90
- plotSidebars, 89
- plotTrace, 90
- presentTalk, 44, 45, 91, 121
- print, PBSOptions-method
  - (PBSOptions-class), 80
- promptWriteOptions, 25, 56, 92, 136
- readList, 37, 45, 76, 93, 129, 135
- readPBSOptions, 42, 56, 60, 93, 94, 107, 136
- regular expression, 68
- resetGraph, 49, 95
- restorePar, 9, 64, 96, 100
- rgb, 72
- rmHistory (initHistory), 65
- runDemos, 42, 97, 98, 99
- runExample, 97, 99
- runExamples, 42, 45, 97, 98, 98
- saveOptions, 57
- saveOptions (loadOptions), 70
- saveOptionsAs (loadOptions), 70
- saveOptionsGUI (loadOptionsGUI), 71
- scalePar, 9, 64, 89, 96, 99
- section-class (talk-class), 119
- selectDir, 100, 102
- selectFile, 101, 101
- setClass, 121
- setFileOption, 102, 104
- setGUIoptions, 25, 40, 56, 101, 103
- setOptions (getOptions), 56
- setOptionsFileName
  - (getOptionsFileName), 57
- setOptionsPrefix (getOptionsPrefix), 58
- setPathOption, 103, 104
- setPBSext, 16, 59, 74, 105, 118, 136
- setPBSoptions, 93, 103, 104, 106, 136
- setwdGUI, 107
- setWidgetColor, 108
- setWidgetState, 109
- setWinAct, 111
- setWinVal, 12, 24, 55, 62, 111, 130
- show, PBSOptions-method
  - (PBSOptions-class), 80
- show0, 77, 112
- showAlert, 63, 114
- showArgs, 31, 36, 45, 114, 127, 133
- showHelp, 75, 115, 118, 134
- showPacks, 116, 116
- showRes, 117
- showVignettes, 42, 118
- sortHistory, 118
- Sys.which, 52
- talk-class, 119
- tcall (tget), 127
- testAlpha, 121, 123
- testCol, 72, 82, 122, 122, 124, 133
- testLty, 122, 123
- testLwd, 122, 124, 124
- testWidgets, 31, 36, 45, 125
- text-class (talk-class), 119
- tget, 38, 45, 68, 76, 127
- tprint (tget), 127
- tput (tget), 127
- unpackList, 37, 76, 94, 129, 135
- updateGUI, 130
- vbdata, 131, 131
- vbpars, 131
- view, 132, 134
- viewCode, 75, 116, 133, 133
- widgets (testWidgets), 125
- writeList, 37, 76, 94, 129, 134
- writePBSOptions, 74, 93, 95, 107, 136
- xmlGetAttr, 121