

# Package ‘PBSmodelling’

January 2, 2012

**Version** 2.63.229

**Date** 2011-09-09

**Title** GUI Tools Made Easy: Interact with Models, Explore Data, Give Dynamic Presentations

**Author** Jon T. Schnute <Jon.Schnute@dfo-mpo.gc.ca>, Alex Couture-Beil  
<aLex@mofo.ca>, Rowan Haigh <rowan.haigh@dfo-mpo.gc.ca>, and  
A.R. (Rob) Kronlund <Allen.Kronlund@dfo-mpo.gc.ca>

**Maintainer** Jon Schnute <Jon.Schnute@dfo-mpo.gc.ca>

**Depends** R (>= 2.7.0), methods

**Suggests** PBSmapping, PBSddesolve, deSolve, KernSmooth, XML

**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. The package contains examples that illustrate models built with 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.

**License** GPL (>= 2)

**Repository** CRAN

**Date/Publication** 2011-09-10 06:17:02

## R topics documented:

addArrows . . . . .	4
addLabel . . . . .	5
addLegend . . . . .	6

calcFib . . . . .	6
calcGM . . . . .	7
calcMin . . . . .	8
CCA.qbr . . . . .	10
chooseWinVal . . . . .	11
cleanProj . . . . .	13
cleanWD . . . . .	14
clearAll . . . . .	15
clearPBSext . . . . .	15
clearRcon . . . . .	16
clearWinVal . . . . .	17
clipVector . . . . .	17
closeWin . . . . .	18
compileC . . . . .	19
compileDescription . . . . .	20
convSlashes . . . . .	20
createVector . . . . .	21
createWin . . . . .	22
declareGUIoptions . . . . .	24
doAction . . . . .	25
drawBars . . . . .	26
evalCall . . . . .	26
expandGraph . . . . .	27
exportHistory . . . . .	28
findPat . . . . .	29
findPrefix . . . . .	29
findProgram . . . . .	30
focusWin . . . . .	31
genMatrix . . . . .	32
getChoice . . . . .	33
getGUIoptions . . . . .	34
getOptions . . . . .	35
getOptionsFileName . . . . .	35
getOptionsPrefix . . . . .	36
getPBSext . . . . .	37
getPBSoptions . . . . .	38
getWinAct . . . . .	38
getWinFun . . . . .	39
getWinVal . . . . .	39
getYes . . . . .	40
GT0 . . . . .	41
importHistory . . . . .	42
initHistory . . . . .	43
isWhat . . . . .	45
loadC . . . . .	46
loadOptions . . . . .	47
loadOptionsGUI . . . . .	48
openExamples . . . . .	48

openFile . . . . .	49
openPackageFile . . . . .	51
openProjFiles . . . . .	52
openUG . . . . .	53
packList . . . . .	54
pad0 . . . . .	55
parseWinFile . . . . .	56
pause . . . . .	57
PBSmodelling . . . . .	57
PBSoptions-class . . . . .	58
pickCol . . . . .	59
plotACF . . . . .	60
plotAsp . . . . .	61
plotBubbles . . . . .	62
plotCsum . . . . .	63
plotDens . . . . .	64
plotFriedEggs . . . . .	65
plotTrace . . . . .	66
presentTalk . . . . .	67
promptOpenFile . . . . .	68
promptSaveFile . . . . .	69
promptWriteOptions . . . . .	70
readList . . . . .	71
readPBSoptions . . . . .	71
resetGraph . . . . .	72
restorePar . . . . .	73
runDemos . . . . .	74
runExample . . . . .	74
runExamples . . . . .	75
scalePar . . . . .	76
selectDir . . . . .	77
selectFile . . . . .	78
setFileOption . . . . .	79
setGUIoptions . . . . .	80
setPathOption . . . . .	81
setPBSext . . . . .	82
setPBSoptions . . . . .	82
setwdGUI . . . . .	83
setWidgetColor . . . . .	83
setWidgetState . . . . .	85
setWinAct . . . . .	86
setWinVal . . . . .	87
show0 . . . . .	88
showAlert . . . . .	89
showArgs . . . . .	90
showHelp . . . . .	90
showPacks . . . . .	91
showRes . . . . .	92

showVignettes . . . . .	93
sortHistory . . . . .	93
talk-class . . . . .	94
testAlpha . . . . .	96
testCol . . . . .	97
testLty . . . . .	98
testLwd . . . . .	99
testPch . . . . .	99
testWidgets . . . . .	100
unpackList . . . . .	103
updateGUI . . . . .	104
vbdata . . . . .	105
vbpars . . . . .	105
view . . . . .	106
viewCode . . . . .	107
writeList . . . . .	108
writePBSoptions . . . . .	109

**Index** **110**

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 paramaters for the function arrows.

**Details**

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

**Author(s)**

Jon Schnute, Pacific Biological Station, Nanaimo BC

**See Also**

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

**Examples**

```
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")
```

---

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 Schnute, Pacific Biological Station, Nanaimo BC

**See Also**

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

**Examples**

```
resetGraph()
addLabel(0.75,seq(from=0.9,to=0.1,by=-0.10),c('a','b','c'), col="#0033AA")
```

---

`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**

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

**Author(s)**

Jon Schnute, Pacific Biological Station, Nanaimo BC

**See Also**

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

**Examples**

```
resetGraph(); n <- sample(1:length(colors()),15); clr <- colors()[n]
addLegend(.2,1,fill=clr,leg=clr,cex=1.5)
```

---

`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**

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

**Value**

Vector of the last len Fibonacci numbers calculated.

**Author(s)**

Jon Schnute, Pacific Biological Station, Nanaimo BC

---

calcGM	<i>Calculate the Geometric Mean, Allowing for Zeroes</i>
--------	--

---

**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**

x	vector of numbers
offset	value to add to all components, including zeroes
exzero	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, Nanaimo BC

**Examples**

```
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 <code>"L-BFGS-B"</code> there are six levels of tracing. Default is 0.
<code>maxit</code>	The maximum number of iterations. Default is 1000.
<code>reltol</code>	Relative convergence tolerance. The algorithm stops if it is unable to reduce the value by a factor of <code>reltol*(abs(val)+reltol)</code> at a step. Default is <code>1e-8</code> .
<code>steptol</code>	A positive scalar providing the minimum allowable relative step length. Default is <code>1e-6</code> .
<code>temp</code>	Temperature controlling the <code>"SANN"</code> method. It is the starting temperature for the cooling schedule. Default is 10.
<code>repN</code>	Reports the parameter and objective function values on the R-console every <code>repN</code> evaluations. Default is 0 for no reporting.
<code>...</code>	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 Schnute, Pacific Biological Station, Nanaimo BC

**See Also**

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

In the stats package: `nlm`, `nlminb`, and `optim`.

**Examples**

```
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)); };
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);
```

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 “syrs” and “cyrs” 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-sci.pac.dfo-mpo.gc.ca/sa-mfpd/statsamp/StatSamp\\_GFBio.htm](http://www-sci.pac.dfo-mpo.gc.ca/sa-mfpd/statsamp/StatSamp_GFBio.htm)

**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

```
# Plot age proportions (blue bubbles = survey data, red = commercial)
data(CCA.qbr); 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,pow=.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)
```

---

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

choice	vector of strings from which to choose
varname	variable name to which choice is assigned in the target GUI
winname	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, Nanaimo BC

**See Also**

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

**Examples**

```
## Not run:
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() {
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) {
```

```

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=F);
  yfld <- getChoice(paste("Pick y-field from",fnam),flds,gui=F)
  plot(file[,xfld],file[,yfld],xlab=xfld,ylab=yfld,
       pch=16,cex=1.2,col="red");
  again <- getChoice("Plot another pair?",gui=F) }
}
require(PBSmodelling)
createWin(wlist,astext=T); 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:
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, Nanaimo BC

**Examples**

```
## Not run:
cleanWD(c("* .bak", "* .tmp", "junk*"))

## End(Not run)
```

---

clearAll *Remove all R Objects From the Global Environment*

---

**Description**

Generic function to clear all objects from .RData in R

**Usage**

```
clearAll(hidden=TRUE, verbose=TRUE, PBSSave=TRUE)
```

**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.

**Author(s)**

Jon Schnute, Pacific Biological Station, 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, Malaspina University-College, Nanaimo BC

**See Also**

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

---

clearRcon

*Clear the R Console Window*

---

**Description**

Clear the R console window using a Visual Basic shell script.

**Usage**

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

**Arguments**

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

**Details**

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

Similarly, `focusRcon()` gives the focus to the R console window by creating a Visual Basic shell script called `focusRgui.vba` in R's temporary working directory, then executes it using the shell command.

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, Nanaimo BC

**See Also**

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

---

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, Malaspina University-College, 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**

vec	vector object to clip
clip	value or string specifying repeated values to clip from ends
end	end to clip clip 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, Nanaimo BC

**See Also**

[createVector](#)

**Examples**

```
x=c(0,0,0,0,1,1,1,1,0,0)
clipVector(x,0)
```

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

```
x=c("red","tide","red","red")
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 <i>window description file(s)</i> on the line(s) defining WINDOW widgets. If name is omitted, all active windows will be closed.
------	---

**Author(s)**

Alex Couture-Beil, Malaspina University-College, 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:  
compileC("myFile.c", lib="myLib", options="myObj.o")  
  
## 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, Malaspina University-College, 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, 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")
```

**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.

**Author(s)**

Alex Couture-Beil, Malaspina University-College, Nanaimo BC

**See Also**

[createWin](#)

## Examples

```
## Not run:
#user defined function which is called on new data
drawLiss <- function() {
  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),
  vectorLabels=c("x cycles", "y cycles", "y phase", "points"),
  func="drawLiss");

## End(Not run)
```

---

createWin

*Create a GUI Window*

---

## Description

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

## Usage

```
createWin( fname, astext=FALSE, env=parent.frame() )
```

## 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, Malaspina University-College, 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
drawLiss <- function() {
  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=pptype); invisible(NULL); };
createWin(system.file("testWidgets/LissWin.txt",package="PBSmodelling"));

## End(Not run)

#####
# Environment example:

#function in global
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() #hello Jon

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

---

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:
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 must contain the backtick character `` wherever there is to be an internal double quote "" character. For example,

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

**Value**

Invisibly returns the string expression act.

**Note**

The action is

**Author(s)**

Rowan Haigh, Pacific Biological Station, Nanaimo BC

**Examples**

```
createWin("button text=\"list directory files\" func=doAction action=print(ls())", astart=TRUE)
```

---

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

---

**Description**

Draw a linear barplot on the current plot.

**Usage**

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

**Arguments**

x	x-coordinates
y	y-coordinates
width	bar width, computed if missing
base	y-value of the base of each bar
...	further graphical parameters (see par) may also be supplied as arguments

**Author(s)**

Jon Schnute, Pacific Biological Station, Nanaimo BC

**Examples**

```
plot(0:10,0:10,type="n")  
drawBars(x=1:9,y=9:1,col="deepskyblue4",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 supercede 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)).

**Author(s)**

Rowan Haigh, Pacific Biological Station, Nanaimo BC

**See Also**

[doAction](#)

---

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 Schnute, Pacific Biological Station, Nanaimo BC

**See Also**

[resetGraph](#)

**Examples**

```
resetGraph(); 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")
```

---

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, Malaspina University-College, Nanaimo BC

**See Also**

[importHistory](#), [initHistory](#), [promptSaveFile](#)

---

findPat	<i>Search a Character Vector to Find Multiple Patterns</i>
---------	--

---

**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, Nanaimo BC

**Examples**

```
#find all strings with a vowel, or that start with a number
findPat(c("[aeoiy]", "^[0-9]"), c("hello", "WRLD", "11b"))
```

---

findPrefix	<i>Find a Prefix Based on Names of Existing Files</i>
------------	---

---

**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**

<code>suffix</code>	character vector of suffixes
<code>prefix</code>	character vector of prefixes
<code>path</code>	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**

```
#TODO give better examples from the library's example dir
findPrefix( c(".txt", ".r") )

#or use R's dir for similar functionality
dir(pattern="txt$")
dir(pattern="^[a-h]")
```

---

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**

<code>name</code>	name of a program to locate
<code>includename</code>	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

**See Also**[Sys.which](#)**Examples**

```
findProgram( "gcc" )
findProgram( "notepad" )
findProgram( "R", 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**

<code>winName</code>	name of window to focus
<code>winVal</code>	if TRUE, associate <code>winName</code> with the default window for <code>setWinVal</code> and <code>getWinVal</code>

**Author(s)**

Alex Couture-Beil, Malaspina University-College, Nanaimo BC

**Examples**

```
## Not run:
focus <- function() {
  winName <- getWinVal()$select;
  focusWin(winName);
  cat("calling focusWin(\"", winName, "\")\n", sep="");
  cat("getWinVal()$myvar = ", getWinVal()$myvar, "\n\n", sep=""); };

#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 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 Schnute, Pacific Biological Station, Nanaimo BC

### See Also

[plotBubbles](#)

### Examples

```
plotBubbles(genMatrix(20,6))
```

---

getChoice	<i>Choose One String Item from a List of Choices</i>
-----------	--

---

**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 getChoice 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, getChoice 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, Nanaimo BC

**See Also**

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

**Examples**

```
## Not run:
#-- Example 1
getChoice(c("Fame", "Fortune", "Health", "Beauty", "Lunch"),
  "What do you want?", qcolor="red", gui=F)

#-- Example 2
getChoice(c("Homer Simpson", "Wilberforce Humphries", "Miss Marple"),
  "Who's your idol?", horiz=F, radio=T, gui=F)

## 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:
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, ...)
```

**Arguments**

<code>option.object</code>	options class object used for storing package options
<code>...</code>	any number of user options to set, where the named argument is the option key and the value is the option value
<code>key</code>	name of option to retrieve – if missing all options are returned

**Value**

`getOptions`: returns the value of the option specified by key.  
If key is missing, a list of all options.

**Author(s)**

Alex Couture-Beil

**See Also**

See [PBSoptions-class](#) for more details and an example using **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**

option.object	options class object used for storing package options
name	new name for default file name

**Value**

getOptionsFileName: the default file name

**See Also**

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

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

---

getOptionsPrefix	<i>Get and Set GUI Prefix of Options Class</i>
------------------	--

---

**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

**See Also**

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

**Examples**

```
.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 )
```

---

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

---

**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` 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, Malaspina University-College, Nanaimo BC

**See Also**

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

---

getPBSoptions                      *Retrieve A User Option*

---

**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, Malaspina University-College, Nanaimo BC

**See Also**

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

---

getWinAct                          *Retrieve the Last Window Action*

---

**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, Malaspina University-College, 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, Malaspina University-College, 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; and scope="G" creates global variables(pos=1).
asvector	return a vector instead of a list. WARNING: 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/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, Malaspina University-College, 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:
#default settings
if(getYes("Print the number 1?"))
  print(1)

## 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 Schnute, Pacific Biological Station, Nanaimo BC

**See Also**

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

**Examples**

```
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()
```

---

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, Malaspina University-College, Nanaimo BC

**See Also**

[exporthistory](#), [inithistory](#), [promptOpenFile](#)

---

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, Malaspina University-College, Nanaimo BC

## See Also

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

## Examples

```
## Not run:
# 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 astext=TRUE. P.S. Watch out for special characters which
# must be "escaped" twice (first for R, then PBSmodelling).

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);
}

initHistory("myHistory", "myHistoryIndex", "myHistorySize")
createWin(winDesc, astext=TRUE)

## 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 Schnute, Pacific Biological Station, Nanaimo BC

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 .c and .r 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.
<b>Ratio</b>	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            *Load and Save Options Values to and from a GUI*

---

### 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.

---

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](#), [openProjFiles](#), [openPackageFile](#)

## Examples

```
## Not run:
# Copies example1.c and example2.r from the examples directory in
# myPackage to the working directory, and opens these files
openExamples("myPackage", "example1", c(".r", ".c"))

## End(Not run)
```

---

openFile

*Open a File with an Associated Program*

---

## Description

Open a file using the program associated with its extension defined by the Windows shell. Non-windows users, or users wishing to override the default application, can specify a program association using setPBSext.

## Usage

```
openFile(fname, package=NULL)
```

**Arguments**

fname	name(s) of file(s) to open.
package	(optional) open files relative to this package

**Value**

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

**Warning**

Windows only: `openFile( ".." )` does not work as expected, but `openFile( "../.." )` only opens the first parent directory.

**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, Malaspina University-College, Nanaimo BC

**See Also**

[getPBSext](#), [setPBSext](#), [clearPBSext](#), [writePBSoptions](#)

**Examples**

```
## Not run:

#use openFile directly:
openFile( "doc/PBSmodelling-UG.pdf", package="PBSmodelling" )

#via doAction in a window description file:
createWin( "button text=help func=doAction action=\"openFile( 'doc/PBSmodelling-UG.pdf', package='PBSmodelling' )" )

#####
# Set up firefox to open .html files (only applicable if firefox is NOT the default web browser
setPBSext("html", "'c:/Program Files/Mozilla Firefox/firefox.exe" file://%f')
openFile("foo.html")

## End(Not run)
```

---

openPackageFile	<i>Open a File from a Package Subdirectory</i>
-----------------	--

---

### Description

Open a file from a package in the R library, given the package name and the file path relative to the package root directory.

### Usage

```
openPackageFile(package, filepath)
```

### Arguments

package	name of the package
filepath	path to file from the package's root directory

### Details

The `openFile` function is used to open the file, using associations set by `setPBSext`.

To use this function in a *window description file*, the `package` and `filepath` arguments must be specified as the action of the widget that calls `openPackageFile`. Furthermore, `package` and `filepath` must be separated by commas (e.g., `action=myPackage,/doc/help.pdf`).

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

### Note

\*\*\* This function is deprecated and will be removed in a future version. Use [openFile](#) or [doAction](#) with `openFile`. \*\*\*

### Author(s)

Anisa Egeli, Vancouver Island University, Nanaimo BC

### See Also

[openFile](#), [setPBSext](#), [openProjFiles](#), [openExamples](#)

### Examples

```
## Not run:
#use openFile directly:
openFile( "doc/PBSmodelling-UG.pdf", package="PBSmodelling" )

#via doAction in a window description file:
createWin( "button text=help func=doAction action=\"openFile( 'doc/PBSmodelling-UG.pdf', package='PBSmodelling' )"
```

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

---

openProjFiles	<i>Open Files with a Common Prefix</i>
---------------	--

---

### Description

Open one or more files from the working directory, given one file prefix and one or more file suffixes.

### Usage

```
openProjFiles(prefix, suffix, package=NULL, warn=NULL, alert=TRUE)
```

### Arguments

prefix	a single prefix to prepend to each suffix
suffix	a character vector of suffixes to append to the prefix
package	name of the package that contains templates, or NULL to not use templates
warn	if specified, use to temporarily override the current R warn option during this function's activity; if NULL, the current warning settings are used.
alert	if TRUE, an alert message is shown should any files fail to be opened; if FALSE, no alert is displayed.

### Details

The suffixes may contain wildcards ("\*" to match 0 or more characters, "?" to match any single character).

For any file that does not exist in the working directory, a template can optionally be copied from a directory named `templates` in the specified package. The templates in this directory should have the prefix `template`, followed by the suffix to match when `openProjFiles` is called (e.g., `template.c` to match the suffix `.c`). After being copied to the working directory, the new file is renamed to use the specified prefix.

To use this function in a *window description file*, the `package` and `suffix` arguments must be specified as the action of the widget that calls `openProjFiles`. Furthermore, `package` and each `suffix` must be separated by commas. For example, `action=myPackage, .r, .c` will try to open a `.r` and `.c` file in the working directory, copying templates from the `template` directory for the package **myPackage**, if the files didn't already exist. To disable templates, leave `package` unspecified but keep the leading comma (e.g., `action=, .r, .c`). When the function is called from a widget in this fashion, the prefix is taken from the value of a widget named `prefix`.

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

**Note**

\*\*\* This function is deprecated and will be removed in a future version. Use [openFile](#) or [doAction](#) with `openFile`. \*\*\*

\*\*\* Warning: this function is buggy; see examples for an alternative. \*\*\*

**Author(s)**

Anisa Egeli, Vancouver Island University, Nanaimo BC

**See Also**

[openFile](#), [setPBSext](#), [openExamples](#), [openPackageFile](#)

**Examples**

```
## Not run:
openProjFiles("foo", c(".r", ".c"), package="myPackage")

#use openFile directly:
openFile( dir(pattern="^prefix.*\\.suffix$") )

#example with multiple suffixes (prefix=foo, suffix=.r, .c)
openFile( dir(pattern="^foo.*\\.(r|c)$") )

## End(Not run)
```

---

openUG

*Open Package User Guide*

---

**Description**

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

**Usage**

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

**Arguments**

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

**Details**

We assume that the name of the User's Guide follows 'PBS' conventions. This means a user's guide in 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, Pacific Biological station, Nanaimo BC

**See Also**

[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,  
         lenv=parent.frame(), tenv=.GlobalEnv)
```

**Arguments**

stuff	string vector of object names
target	target list object
value	an optional explicit value to assign to stuff
lenv	local environment where objects are located
tenv	target environment where target list is 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 lenv 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

**Author(s)**

Rowan Haigh, Pacific Biological Station, Nanaimo BC

**See Also**

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

**Examples**

```
fn = function() {  
  alpha=rnorm(10)  
  beta=letters  
  gamma=mean  
  delta=longley  
  packList(c("alpha", "beta", "gamma", "delta")) }  
fn(); print(PBSlist)
```

---

pad0

*Pad Numbers with Leading Zeroes*

---

**Description**

Convert numbers to integers then text, and pad them with leading zeroes.

**Usage**

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

**Arguments**

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

**Value**

A character vector representing x with leading zeroes.

**Author(s)**

Rowan Haigh, Pacific Biological Station, Nanaimo BC

**Examples**

```
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, Malaspina University-College, Nanaimo BC

**See Also**

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

**Examples**

```
## Not run:  
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 Schnute, Pacific Biological Station, 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, odesolve, 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()`.

---

PBSoptions-class      *Project Options Class*

---

## 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 – use the access functions instead.

**Author(s)**

Alex Couture-Beil

**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**

```
#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=optionsillyhatday text=\"silly hat day\"",
  "entry name=optionphotos mode=character label=\"photos directory\"",
  "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 ))
```

---

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**

`returnValue` 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, Malaspina University-College, Nanaimo BC

**See Also**

[testCol](#)

**Examples**

```
## Not run:  
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**

`file` data frame, matrix, or vector of numeric values.  
`lags` maximum number of lags to use in the ACF calculation.  
`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 `plotACF` in conjunction with `samplesHistory("*", beg=0, plot=FALSE)` rather than `samplesAutoC` which calls `plotAutoC`.

**Author(s)**

Rowan Haigh, Pacific Biological Station, Nanaimo BC

**Examples**

```
resetGraph(); plotACF(trees,lwd=2,lags=30);
```

---

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**

x	vector of x-coordinate points in the plot.
y	vector of y-coordinate points in the plot.
asp	y/x aspect ratio.
...	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, Malaspina University-College, Nanaimo BC

**Examples**

```
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)")
```

---

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=0.1, prettyaxis=FALSE, ...)
```

### Arguments

<i>z</i>	input matrix, array (2 dimensions) or data frame.
<i>xval</i>	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>	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>	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>	size (inches) of the largest bubble.
<i>lwd</i>	line width for drawing circles.
<i>clr</i>	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>	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.
...	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 Schnute, Pacific Biological Station, Nanaimo BC

**See Also**

[genMatrix](#)

**Examples**

```
plotBubbles(round(genMatrix(40,20),0),clrs=c("green","grey","red"));

data(CCA.qbr)
plotBubbles(CCA.qbr,cpro=TRUE,powr=.5,dnam=TRUE,size=.15,
  ylim=c(0,70),xlab="Year",ylab="Quillback Rockfish Age")
```

---

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**

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

**Author(s)**

Rowan Haigh, Pacific Biological Station, Nanaimo BC

**Examples**

```
x <- rgamma(n=1000, shape=2)
plotCsum(x)
```

---

**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, clrs=c("blue", "red", "green", "magenta", "navy"), ...)
```

**Arguments**

file	data frame, matrix, or vector of numeric values.
clrs	vector of colours. Patterns are repeated if the number of fields exceed the length of clrs.
...	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, Nanaimo BC

**Examples**

```
z <- data.frame(y1=rnorm(50, sd=2), y2=rnorm(50, sd=1), y3=rnorm(50, sd=.5))
plotDens(z, lwd=3)
```

---

`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**

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

**See Also**

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

`KernSmooth::bkde2D`, `grDevices::contourLines`, `graphics::contour`

**Examples**

```
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")} )
```

---

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) exceeded 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, Nanaimo BC

**Examples**

```
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)
```

---

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                    talk xml file name

**Details**

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

**See Also**

See [PBStalk-class](#) for more details on **PBSmodelling**'s talk presentation classes.

**Examples**

```
wd <- getwd() #save current directory
talk_dir <- system.file("examples", package = "PBSmodelling" )
setwd( talk_dir )
presentTalk( "swisstalk.xml" )

#restore working directory once talk GUI is closed
setwd( wd )
```

---

promptOpenFile                    *Deprecated: Display Dialogue: Open File*

---

**Description**

Deprecated: use [selectFile](#) instead.

**Usage**

```
promptOpenFile(initialfile="", filetype=list(c("*","All Files")),
               open=TRUE)
```

**Arguments**

initialfile	file name of the text file containing the list.
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:".
open	logical: if TRUE display <b>Open</b> prompt, if FALSE display <b>Save As</b> prompt.

**Value**

The file name and path of the file selected by the user.

**Note**

\*\*\* This function is deprecated and will be removed in a future version. Use [selectFile](#). \*\*\*

**Author(s)**

Alex Couture-Beil, Malaspina University-College, Nanaimo BC

**See Also**

[selectFile](#)

---

promptSaveFile	<i>Deprecated: Display Dialogue: Save File</i>
----------------	--

---

### Description

Deprecated: use [selectFile](#) instead.

### Usage

```
promptSaveFile(initialfile="", filetype=list(c("*", "All Files")),
               save=TRUE)
```

### Arguments

initialfile	file name of the text file containing the list.
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:".
save	logical: if TRUE display <b>Save As</b> prompt, if FALSE display <b>Open</b> prompt.

### Value

The file name and path of the file selected by the user.

### Note

\*\*\* This function is deprecated and will be removed in a future version. Use [selectFile](#). \*\*\*

### Author(s)

Alex Couture-Beil, Malaspina University-College, Nanaimo BC

### See Also

[selectFile](#)

---

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:  
promptWriteOptions() #uses default filename PBSOptions.txt  
  
## End(Not run)
```

---

readList	<i>Read a List from a File in PBS Modelling Format</i>
----------	--

---

**Description**

Read in a list previously saved to a file by `writeList`. At present, only two formats are supported - R's native format used by the `dput` function or an ad hoc 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.

**Author(s)**

Alex Couture-Beil, Malaspina University-College, Nanaimo BC

**See Also**

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

---

readPBSoptions	<i>Read PBS Options from an External File</i>
----------------	---

---

**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 Schnute, Pacific Biological Station, 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 Schnute, Pacific Biological Station, Nanaimo BC

**See Also**

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

**Examples**

```
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 co-incide 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, Malaspina University-College, 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;  
odesolve/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, 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;

odesolve/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)**

Alex Couture-Beil, Malaspina University-College, 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 Schnute, Pacific Biological Station, Nanaimo BC

**See Also**

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

**Examples**

```
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 the default directory chooser prompt provided by the Operating System.

**Usage**

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

**Arguments**

initialdir	initially selected directory
mustexist	if logical value is TRUE, only a existing directory can be selected
title	title for the prompt window
usewidget	store the selected directory in the named entry widget

**Value**

The directory path selected by the user

**Author(s)**

Alex Couture-Beil

**See Also**

[selectFile](#)

**Examples**

```
## Not run:
dir(selectDir(title="select a directory to list contents of"))

#integration with widget via doAction
createWin( c( "entry foo mode=character",
  "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
defaulttextension	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

**See Also**

[selectDir](#)

**Examples**

```
## Not run:
# Open a filename, and return it line by line in a vector
scan(promptOpenFile(),what=character(),sep="\n")

# Illustrates how to set filetype.
promptOpenFile("initial_file.txt", filetype=list(c(".txt", "text files"),
          c(".r", "R files"), c("*", "All Files")))

#integration with widget via doAction
createWin( c( "entry foo mode=character",
  "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:  
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:  
setGUIOptions("editor")  
  
## End(Not run)
```

---

setPathOption	<i>Set a PBS Path Option Interactively</i>
---------------	--

---

**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:  
setPathOption("myPath")  
  
## End(Not run)
```

---

setPBSext	<i>Set a Command Associated with a File Name Extension</i>
-----------	--

---

**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	string specifying the extension suffix.
cmd	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, Malaspina University-College, Nanaimo BC

**See Also**

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

---

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 <code>sublist=TRUE</code> .

**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, Malaspina University-College, Nanaimo BC

**See Also**

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

---

 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**

```
createWin( "button text=\"Change working directory\" func=setwdGUI", astart=TRUE )
```

---

 setWidgetColor

*Update Widget Color*

---

**Description**

Update the foreground and background colors of a widget

**Usage**

```
setWidgetColor( name, radioValue, winName = .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 and Allen R. Kronlund

**Examples**

```
createWin("label \"hello world\" name=hello", astext=TRUE)
setWidgetColor( "hello", bg="lightgreen", fg="purple" )

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" )
```

---

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 behavior 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 and Allen R. Kronlund

### Examples

```
## Not run:
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

*Add a Window Action to the Saved Action Vector*

---

### 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, Malaspina University-College, 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, Malaspina University-College, Nanaimo BC

**See Also**

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

**Examples**

```
## Not run:
winDesc <- c("vector length=3 name=vec",
            "matrix nrow=2 ncol=2 name=mat",
            "slideplus name=foo");
createWin(winDesc, astart=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**

```
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))
```

---

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:
showAlert("Hello World!")

## End(Not run)
```

---

showArgs	<i>Display Expected Widget Arguments</i>
----------	--

---

**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, Malaspina University-College, Nanaimo BC

---

showHelp	<i>Display HTML Help Pages for Packages in Browser</i>
----------	--

---

**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:

<code>apacks</code>	all packages installed on user's system
<code>spacks</code>	selected packages based on specified pattern
<code>URLs</code>	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.

**Author(s)**

Rowan Haigh, Pacific Biological Station, 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","PBSddesolve",  
"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 Schnute, Pacific Biological Station, 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 Schnute, Pacific Biological Station, Nanaimo BC

**Examples**

```
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, Malaspina University-College, Nanaimo BC

**See Also**

[importHistory](#), [initHistory](#)

---

talk-class

*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

name	character	string giving the name of the talk (required)
sections	list	list of sections within the talk
files	list	list of files within the talk

### section

name	character	string giving the name of the section (required)
items	list	list of the four primitive (leaf-element) S4 classes
button	logical	should GUI have a button that selects section?
col	integer	column in lower section of GUI to place button
section_id	integer	specify if section does not immediately follow a talk

### text

text	character	text to display on the R console
"break"	logical	break the presentation after displaying the text specified?

### file

name	character	string giving the name in the GUI for a group of files to open (required)
filename	character	individual file names associated with the group name in the GUI
"break"	logical	break the presentation after opening the group of files?
button	logical	should GUI add a button that opens this group of files?
col	integer	column in lower section of GUI to place button

### code

show	logical	show the code snippet in the R console?
print	logical	print the results of running the R code?
code	character	the actual chunk of R code
"break"	character	string describing where to introduce breaks in the code segment
eval	logical	evaluate the R code?

### break

.xData	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

**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	foreground colour of the top shape that varies in transparency.
bg	background colour (remains constant) of the underlying shape.
border	border colour (which also changes in transparency) of the foreground polygon.

grid                    logical: if TRUE, lay a grey grid on the background colour.  
 ...                    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 Schnute, Pacific Biological Station, Nanaimo BC

**See Also**

[testCol](#), [testPch](#), [testLty](#), [testLwd](#)

---

 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                    vector of colour names to display. Defaults to 15 random names from the color palette to use as patterns.

**Author(s)**

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

**See Also**

[pickCol](#), [testAlpha](#), [testPch](#), [palettes](#)

**Examples**

```
# mix and match patterns
testCol(c("sky", "fire", "sea", "wood"))

# display transparencies of blue
testCol(rgb(0,0,1,seq(0.05,1,0.05)))

# display colours of the rainbow
testCol(rainbow(64,end=0.75))

# display basic palette colours
testCol(1:length(palette()))

# 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**

<code>newframe</code>	if TRUE, create a new blank frame, otherwise overlay current frame.
<code>n</code>	vector of line type numbers.
<code>...</code>	additional arguments for function lines.

**Note**

Quick representation of line types for reference purposes.

**Author(s)**

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

**See Also**

[testLwd](#), [testPch](#), [testCol](#)

---

testLwd	<i>Display Line Widths</i>
---------	----------------------------

---

**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 <code>length(lwd) &gt; length(col)</code> .
newframe	if TRUE, create a new blank frame, otherwise overlay current frame.

**Author(s)**

Rowan Haigh, Pacific Biological Station, Nanaimo BC

**Examples**

```
testLwd(3:15, col=c("salmon", "aquamarine", "gold"))
```

---

testPch	<i>Display Plotting Symbols or Octal Strings</i>
---------	--

---

**Description**

Display plotting symbols or octal strings. User can specify particular ranges (increasing continuous integer) for pch.

**Usage**

```
testPch(pch=1:100, ncol=10, grid=TRUE, newframe=TRUE, octal=FALSE, ...)
```

**Arguments**

pch	symbol codes or octal string numbers.
ncol	number of columns in display (can only be 2, 5, or 10). Most sensibly this is set to 10.
grid	logical: if TRUE, grid lines are plotted for visual aid.
newframe	logical: if TRUE reset the graph, otherwise overlay on top of the current graph.
octal	logical: if TRUE, show octal strings (backslash characters) used in text statements (e.g., "30\272C" = 30°C).
...	additional arguments for functions points or text.

**Author(s)**

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

**See Also**

[testLty](#), [testLwd](#), [addLabel](#)

**Examples**

```
testPch(123:255)
testPch(1:25,ncol=5)
testPch(41:277,octal=TRUE)
```

---

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, Malaspina University-College, Nanaimo BC

**See Also**

[createWin](#), [showArgs](#)

---

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**

x	named list to unpack.
scope	If "L", create variables local to the parent frame that called the function. If "G", create global variables.

**Value**

A character vector of unpacked variable names.

**Author(s)**

Alex Couture-Beil, Malaspina University-College, Nanaimo BC

**See Also**

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

**Examples**

```
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, "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, Nanaimo BC

**See Also**

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

**Examples**

```
#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="L" )
taste <- foodquality * hunger / cost
updateGUI()
```

---

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

---

*View First/Last/Random n Elements/Rows of an Object*


---

**Description**

View the first or last or random n elements or rows of an object. Components of lists will be subset also.

**Usage**

```
view(obj, n=5, last=FALSE, random=FALSE, ...)
```

**Arguments**

obj	object to view.
n	first (default)/last/random n elements/rows of obj to view.
last	logical: if TRUE, last n elements/rows of obj are displayed.
random	logical: if TRUE, n random elements/rows (without replacement) of obj are displayed.
...	additional arguments (e.g., replace=T if specifying random=T).

**Author(s)**

Rowan Haigh, Pacific Biological Station, Nanaimo BC

---

`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**

<code>pkg</code>	string name of a package installed on the user's computer.
<code>funs</code>	string vector of explicit function names from <code>pkg</code> to view.
<code>output</code>	numeric value: 1 = function names only, 2 = function names with brief description, 3 = functions and their arguments, and 4 = function R-code (default).
<code>...</code>	allows user to specify two additional arguments for <code>output=2</code> : <code>remote</code> - character string giving a valid URL for the <code>R_HOME</code> directory on a remote location; <code>update</code> - logical: if TRUE, attempt to update the package index to reflect the currently available packages. (Not attempted if <code>remote</code> 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.

**Author(s)**

Rowan Haigh, Pacific Biological Station, 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

<code>x</code>	R list object to write to an ASCII text file.
<code>fname</code>	file name of the text file containing the list.
<code>format</code>	format of the file to create: "D" or "P".
<code>comments</code>	vector of character strings to use as initial-line comments in the file.

### Details

The "D" format is equivalent to using R's base functions `dput` and `dget`, which support all R objects.

The "P" format only supports named lists of vectors, matrices, arrays, and data frames. Scalars are treated like vectors. Nested lists are not supported.

The "P" format writes each named element in a list using the following conventions: (i) `\$` followed by the name of the data object to denote the start of that object's description; (ii) `\$\$` on the next line to describe the object's structure - object type, mode(s), names (if vector), rownames (if matrix or data), and colnames (if matrix or data); and (iii) subsequent lines of data (one line for vector, multiple lines for matrix or data).

Arrays with three or more dimensions have `dim` and `dimnames` arguments. `Dim` is 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 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).

Using "P" formatting, array data are written the same way that 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" in the R directory `.../library/PBSmodelling/doc`.

**Author(s)**

Alex Couture-Beil, Malaspina University-College, Nanaimo BC

**See Also**

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

**Examples**

```
## Not run:
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")

## End(Not run)

##Example of dimnames for Arrays
dimnames(Titanic)
writeList( list( Titanic ), format="P")
```

---

writePBSOptions

*Write PBS Options to an External File*

---

**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

- \*Topic **arith**
  - calcFib, 6
  - calcGM, 7
- \*Topic **array**
  - genMatrix, 32
- \*Topic **character**
  - convSlashes, 20
  - doAction, 25
  - evalCall, 26
  - showArgs, 90
  - showPacks, 91
  - viewCode, 107
- \*Topic **classes**
  - PBSoptions-class, 58
  - talk-class, 94
- \*Topic **color**
  - pickCol, 59
  - testAlpha, 96
  - testCol, 97
  - testLty, 98
  - testLwd, 99
  - testPch, 99
- \*Topic **datasets**
  - CCA.qbr, 10
  - vbdata, 105
  - vbpars, 105
- \*Topic **data**
  - clipVector, 17
- \*Topic **device**
  - chooseWinVal, 11
  - clearRcon, 16
  - expandGraph, 27
  - getChoice, 33
  - resetGraph, 72
  - showHelp, 90
- \*Topic **documentation**
  - openUG, 53
- \*Topic **file**
  - findPrefix, 29
  - findProgram, 30
  - openExamples, 48
  - openFile, 49
  - openPackageFile, 51
  - openProjFiles, 52
  - packList, 54
  - promptOpenFile, 68
  - promptSaveFile, 69
  - readList, 71
  - selectDir, 77
  - selectFile, 78
  - unpackList, 103
  - writeList, 108
- \*Topic **graphs**
  - plotACF, 60
  - plotDens, 64
  - plotTrace, 66
- \*Topic **hplot**
  - drawBars, 26
  - plotAsp, 61
  - plotBubbles, 62
  - plotCsum, 63
  - plotFriedEggs, 65
- \*Topic **interface**
  - compileC, 19
  - loadC, 46
- \*Topic **iplot**
  - addArrows, 4
  - addLabel, 5
  - addLegend, 6
- \*Topic **list**
  - exportHistory, 28
  - importHistory, 42
  - packList, 54
  - parseWinFile, 56
  - readList, 71
  - sortHistory, 93
  - unpackList, 103
  - writeList, 108

- \*Topic **methods**
  - clearAll, 15
  - clearPBSext, 15
  - clearWinVal, 17
  - focusWin, 31
  - getOptions, 35
  - getOptionsFileName, 35
  - getOptionsPrefix, 36
  - getPBSext, 37
  - getPBSoptions, 38
  - getWinAct, 38
  - getWinFun, 39
  - getWinVal, 39
  - loadOptions, 47
  - loadOptionsGUI, 48
  - PBSoptions-class, 58
  - setPBSext, 82
  - setPBSoptions, 82
  - setWidgetColor, 83
  - setWidgetState, 85
  - setWinAct, 86
  - setWinVal, 87
  - updateGUI, 104
- \*Topic **nonlinear**
  - calcMin, 8
- \*Topic **optimize**
  - calcMin, 8
  - GT0, 41
  - restorePar, 73
  - scalePar, 76
- \*Topic **package**
  - openUG, 53
  - PBSmodelling, 57
  - showPacks, 91
  - viewCode, 107
- \*Topic **print**
  - pad0, 55
  - show0, 88
  - view, 106
- \*Topic **programming**
  - compileC, 19
  - evalCall, 26
  - loadC, 46
- \*Topic **utilities**
  - chooseWinVal, 11
  - cleanProj, 13
  - cleanWD, 14
  - clipVector, 17
  - closeWin, 18
  - compileDescription, 20
  - createVector, 21
  - createWin, 22
  - doAction, 25
  - findPat, 29
  - getChoice, 33
  - initHistory, 43
  - isWhat, 45
  - pause, 57
  - runDemos, 74
  - runExample, 74
  - runExamples, 75
  - showArgs, 90
  - showHelp, 90
  - showRes, 92
  - showVignettes, 93
  - testCol, 97
  - testLty, 98
  - testLwd, 99
  - testPch, 99
  - testWidgets, 100
- addArrows, 4, 5, 6
- addHistory (initHistory), 43
- addLabel, 5, 5, 6, 100
- addLegend, 5, 6
- backHistory (initHistory), 43
- break-class (talk-class), 94
- calcFib, 6
- calcGM, 7
- calcMin, 8, 9, 42, 73, 77, 105, 106
- CCA.qbr, 10
- chooseWinVal, 11, 34, 41
- cleanProj, 13
- cleanWD, 14, 16
- clearAll, 15
- clearHistory (initHistory), 43
- clearPBSext, 15, 16, 37, 50, 82
- clearRcon, 16
- clearWinVal, 16, 17, 40
- clipVector, 17
- closeWin, 18, 23
- code-class (talk-class), 94
- compileC, 19, 47
- compileDescription, 20, 23, 56
- convSlashes, 20

- createVector, 18, 21, 23
- createWin, 18, 20, 21, 22, 56, 88, 103
- declareGUIOptions, 24, 34, 81
- doAction, 25, 27, 51, 53
- drawBars, 26
- environment, 23
- evalCall, 26
- expandGraph, 27
- exporthHistory, 28, 42, 44
- file-class (talk-class), 94
- findPat, 29
- findPrefix, 29
- findProgram, 30
- findSuffix (findPrefix), 29
- firstHistory (initHistory), 43
- focusRcon (clearRcon), 16
- focusWin, 31
- forwHistory (initHistory), 43
- genMatrix, 32, 63
- getChoice, 12, 33, 41
- getGUIOptions, 24, 34, 72, 81
- getOptions, 35, 59
- getOptionsFileName, 35, 59
- getOptionsPrefix, 36, 59
- getPBSext, 16, 37, 38, 50, 82, 93
- getPBSoptions, 38, 72, 83
- getWinAct, 38
- getWinFun, 39
- getWinVal, 12, 17, 23, 34, 39, 88, 104
- getYes, 40, 89
- GT0, 9, 41, 73, 77, 89
- importHistory, 28, 42, 44, 94
- initHistory, 23, 28, 42, 43, 94
- isWhat, 45
- jumpHistory (initHistory), 43
- lastHistory (initHistory), 43
- loadC, 19, 46
- loadOptions, 36, 47, 59
- loadOptionsGUI, 48, 59
- openExamples, 48, 51, 53
- openFile, 16, 37, 49, 49, 51, 53, 54, 72, 82, 93
- openPackageFile, 49, 51, 53
- openProjFiles, 49, 51, 52
- openUG, 53
- packList, 54, 71, 103, 109
- pad0, 55, 89
- palettes, 97
- parseWinFile, 20, 23, 40, 56
- pause, 57
- PBSmodelling, 57
- PBSmodelling-package (PBSmodelling), 57
- PBSoptions-class, 35, 36, 47, 48, 96
- PBSoptions-class, 58
- PBSstalk-class, 67
- pickCol, 59, 97
- plotACF, 60
- plotAsp, 61
- plotBubbles, 32, 62, 66
- plotCsum, 63
- plotDens, 64
- plotFriedEggs, 65
- plotTrace, 66
- presentTalk, 67, 96
- print, PBSoptions-method  
(PBSoptions-class), 58
- promptOpenFile, 42, 68
- promptSaveFile, 28, 69
- promptWriteOptions, 24, 34, 70, 109
- readList, 54, 71, 103, 109
- readPBSoptions, 34, 38, 70, 71, 83, 109
- resetGraph, 28, 72
- restorePar, 9, 42, 73, 77
- rmHistory (initHistory), 43
- runDemos, 74, 75, 76
- runExample, 74, 76
- runExamples, 74, 75, 75
- saveOptions, 36
- saveOptions (loadOptions), 47
- saveOptionsAs (loadOptions), 47
- saveOptionsGUI (loadOptionsGUI), 48
- scalePar, 9, 42, 66, 73, 76
- section-class (talk-class), 94
- selectDir, 77, 79
- selectFile, 68, 69, 77, 78
- setClass, 96
- setFileOption, 79, 81
- setGUIOptions, 24, 34, 80
- setOptions (getOptions), 35

setOptionsFileName  
    (getOptionsFileName), 35  
setOptionsPrefix (getOptionsPrefix), 36  
setPathOption, 80, 81  
setPBSext, 16, 37, 50, 51, 53, 82, 93, 109  
setPBSoptions, 70, 80, 81, 82, 109  
setwdGUI, 83  
setWidgetColor, 83  
setWidgetState, 85  
setWinAct, 86  
setWinVal, 12, 23, 34, 40, 87, 104  
show, PBSoptions-method  
    (PBSoptions-class), 58  
show0, 88  
showAlert, 41, 89  
showArgs, 90, 103  
showHelp, 54, 90, 93, 107  
showPacks, 91, 91  
showRes, 92  
showVignettes, 93  
sortHistory, 93  
Sys.which, 31  
  
talk-class, 94  
testAlpha, 96, 97  
testCol, 60, 97, 97, 98  
testLty, 97, 98, 100  
testLwd, 97, 98, 99, 100  
testPch, 97, 98, 99  
testWidgets, 100  
text-class (talk-class), 94  
  
unpackList, 54, 71, 103, 109  
updateGUI, 104  
  
vbdata, 105, 105  
vbpars, 105  
view, 106, 107  
viewCode, 54, 91, 107  
  
widgets (testWidgets), 100  
writeList, 54, 71, 103, 108  
writePBSoptions, 50, 70, 72, 83, 109  
  
xmlGetAttr, 96