Package ‘grid’

October 4, 2016

Version 3.3.1

Priority base

Title The Grid Graphics Package

Author Paul Murrell <paul@stat.auckland.ac.nz>

Maintainer R Core Team <R-core@r-project.org>

Description A rewrite of the graphics layout capabilities, plus some support for interaction.

Imports grDevices, utils

Suggests lattice

R topics documented:

`grid-package` グリッドグラフィックスパッケージ

Description

グラフィックスのレイアウトの書き換えと，幾つかの対話的操作のサポート．

Details

このパッケージは S スタイルのグラフィックスを増補するグラフィックスシステムを含む (**graphics** パッケージを見よ).

更なる情報は以下のビニエット中で得られる：

`grid`	`grid` への入門 (`../doc/grid.pdf`)
`displaylist`	`grid` 中のディスプレイリスト (`../doc/displaylist.pdf`)
`frame`	フレームとパッキング grob (`../doc/frame.pdf`)
`grobs`	`grid` grob を使った作業 (`../doc/grobs.pdf`)
`interactive`	`grid` グラフィックスの編集 (`../doc/interactive.pdf`)
`locndimn`	位置と次元 (`../doc/locndimn.pdf`)
`moveline`	move-to と line-to のデモ (`../doc/moveline.pdf`)
`nonfinite`	`grid` が有限でない値にどのように対応するか (`../doc/nonfinite.pdf`)
`plotexample`	`grid` コードを書く (`../doc/plotexample.pdf`)
`rotated`	回転されたビューポート (`../doc/rotated.pdf`)
`saveload`	持続性のある表現 (`../doc/saveload.pdf`)
`sharing`	複数の grob を同時に修正する (`../doc/sharing.pdf`)
`viewports`	`grid` ビューポートを使った作業 (`../doc/viewports.pdf`)

個々のヘルプページ付きの関数の完全なリストには `library(help="grid")` を使う．

Author(s)

Paul Murrell <paul@stat.auckland.ac.nz>

Maintainer: R Core Team <R-core@r-project.org>

References

Murrell, P. (2005) *R Graphics*. Chapman & Hall/CRC Press.

`absolute.size` グロッブの絶対サイズ

Description

この関数は単位オブジェクトを絶対的単位に変換する．絶対的単位は影響を受けないが，絶対的でない単位は "null" 単位に変換される．

Usage

```
absolute.size(unit)
```

Arguments

`unit`	クラス "unit" のオブジェクト．

Details

絶対的単位は "inches", "cm", そして "lines" のようなものである．絶対的でない単位は "npc" と "native" である．

この関数は `widthDetails` と `heightDetails` メソッド中で使われるようにデザインされている．

Value

クラス "unit" のオブジェクト．

Author(s)

Paul Murrell

See Also

`widthDetails` と `heightDetails` メソッド．

arrow 線に加える矢印を記述する.

Description

どのような矢尻が線に加えられるかの記述を生成する．結果は線を引く関数に渡すことが出来る，例えば grid.lines.

Usage

```
arrow(angle = 30, length = unit(0.25, "inches"),
      ends = "last", type = "open")
```

Arguments

angle	度単位の矢尻の角度(より小さな値はより狭く尖る)．本質的に矢尻の幅を記述する.
length	矢尻の長さを指定する単位 (先端から基礎まで).
ends	"last", "first" または "both" のどれかで，線のどちらの端に矢尻を描くかを指示する.
type	"open" または "closed" のどちらかで矢尻が閉じた三角形であるかどうかを指示する.

Examples

```
arrow()
```

calcStringMetric テキストに対するメトリック情報を計算する

Description

この関数は文字列または表現式ベクトルに対する上部高さ，下部深さそして幅のメトリック情報を返す.

Usage

```
calcStringMetric(text)
```

Arguments

text	文字列または表現式ベクトル.

Value

名前 ascent, descent そして width の三つの数値成分を持つリスト．全ての値はインチ単位.

警告

この関数からのメトリック情報はこの関数が呼ばれた時に有効なフォント設定に基づいている．それはページに描かれる任意のテキストのメトリック情報に必ずしも対応しない．

Author(s)

Paul Murrell

See Also

stringAscent, stringDescent, grobAscent そして grobDescent.

Examples

```
grid.newpage()
grid.segments(.01, .5, .99, .5, gp=gpar(col="grey"))
metrics <- calcStringMetric(letters)
grid.rect(x=1:26/27,
          width=unit(metrics$width, "inches"),
          height=unit(metrics$ascent, "inches"),
          just="bottom",
          gp=gpar(col="red"))
grid.rect(x=1:26/27,
          width=unit(metrics$width, "inches"),
          height=unit(metrics$descent, "inches"),
          just="top",
          gp=gpar(col="red"))
grid.text(letters, x=1:26/27, just="bottom")

test <- function(x) {
    grid.text(x, just="bottom")
    metric <- calcStringMetric(x)
    if (is.character(x)) {
        grid.rect(width=unit(metric$width, "inches"),
                  height=unit(metric$ascent, "inches"),
                  just="bottom",
                  gp=gpar(col=rgb(1,0,0,.5)))
        grid.rect(width=unit(metric$width, "inches"),
                  height=unit(metric$descent, "inches"),
                  just="top",
                  gp=gpar(col=rgb(1,0,0,.5)))
    } else {
        grid.rect(width=unit(metric$width, "inches"),
                  y=unit(.5, "npc") + unit(metric[2], "inches"),
                  height=unit(metric$ascent, "inches"),
                  just="bottom",
                  gp=gpar(col=rgb(1,0,0,.5)))
        grid.rect(width=unit(metric$width, "inches"),
                  height=unit(metric$descent, "inches"),
                  just="bottom",
                  gp=gpar(col=rgb(1,0,0,.5)))
    }
}

tests <- list("t",
```

```
                "test",
                "testy",
                "test\ntwo",
                expression(x),
                expression(y),
                expression(x + y),
                expression(a + b),
                expression(atop(x + y, 2)))

grid.newpage()
nrowcol <- n2mfrow(length(tests))
pushViewport(viewport(layout=grid.layout(nrowcol[1], nrowcol[2]),
                      gp=gpar(cex=5, lwd=.5)))
for (i in 1:length(tests)) {
    col <- (i - 1) %% nrowcol[2] + 1
    row <- (i - 1) %/% nrowcol[2] + 1
    pushViewport(viewport(layout.pos.row=row, layout.pos.col=col))
    test(tests[[i]])
    popViewport()
}
```

---

dataViewport　　データに基づくスケールでビューポートを作る

Description

これは関数に渡された数値に基づく x と/または y 軸スケールを持つビューポートを作るための便利関数である.

Usage

```
dataViewport(xData = NULL, yData = NULL, xscale = NULL,
             yscale = NULL, extension = 0.05, ...)
```

Arguments

xData	データの数値ベクトル.
yData	データの数値ベクトル.
xscale	数値ベクトル(長さ 2).
yscale	数値ベクトル(長さ 2).
extension	数値. もし長さが1より大きければ, 最初の値が xscale の拡大に, 二番目の値が yscale の拡大に用いられる.
...	全ての他の引数は viewport() 関数の呼び出しに渡される.

Details

もし xscale が指定されないと x 中の値が x の範囲に基づいて x スケールを作るのに使われ, extension 中で指定された比率で拡大される. y スケールについても同様.

Value

グリッドビューポートオブジェクト.

Author(s)

Paul Murrell

See Also

viewport と plotViewport.

---

depth　　オブジェクト中のレベルの数を決定する.

Description

ビューポートスタックまたはツリー，ビューポートパス，またはグロッブパス中のレベルの数を決定する，

Usage

```
depth(x, ...)
## S3 method for class 'viewport'
depth(x, ...)
## S3 method for class 'path'
depth(x, ...)
```

Arguments

x	典型的にはビューポート，ビューポートスタック，ビューポートツリー，ビューポートリスト，ビューポートパス，又はグロッブパス，
...	他のメソッドにより使われる引数.

Details

パスの深さはそれらが枝を含まないのでかなり明快である．ビューポートスタックの深さはスタックの成分の深さの和である．ビューポートツリーの深さは親の深さに子供の深さを加えたものである．ビューポートリストの深さはリストの最後の成分の深さである．

Value

整数値.

See Also

viewport, vpPath, gPath.

Examples

```
vp <- viewport()
depth(vp)
depth(vpStack(vp, vp))
depth(vpList(vpStack(vp, vp), vp))
depth(vpPath("vp"))
depth(vpPath("vp1", "vp2"))
```

---

drawDetails　　グリッド描画のカスタム化

Description

この総称的フック関数はグリッドの grob が描かれる度に呼び出される．それらは grob (または gTree)から導かれた新しいクラスの描画をカスタム化する機会を提供する．

Usage

```
drawDetails(x, recording)
preDrawDetails(x)
postDrawDetails(x)
```

Arguments

x	グリッドの grob.
recording	grob をディスプレイリストに加えるかディスプレイリストから再描画するかを指示する論理値.

Details

これらの関数は grob と gTree に対する `grid.draw` メソッドにより呼び出される．

`preDrawDetails` は最初 grob の描画の間に呼び出される．これは任意の追加のビューポートがプッシュされるべき場所である (例えば `grid:::preDrawDetails.frame` を見よ). grob に対する既定動作は `vp` スロット中の任意のビューポートをプッシュすることであり， gTrees に対しては `childrenvp` スロット中の任意のビューポートをまたプッシュしアップすることであり，従って典型的にはここで行われることはないことを注意する．

`drawDetails` が追加の計算とグラフィカルな出力が起こる箇所で次に呼びだされる (例えば `grid:::drawDetails.xaxis` を見よ). gTrees に対する既定動作は `children` スロット中の全ての grob を描画することであり従ってここでは典型的に何も行われないことを注意する．

`postDrawDetails` が最後に呼び出され `preDrawDetails` 中で行われた全てを逆転する (つまりプッシュされた任意のビューポートをポップしアップする；`grid:::postDrawDetails.frame`) を再び例えば見よ). grob に対する既定動作はプッシュされた任意のビューポートをポップすることであり，従ってここでは典型的に何も行われない．

`preDrawDetails` と `postDrawDetails` はまた "grobwidth" と "grobheight" 単位の計算の間にも呼び出されることを注意する．

Value

これらの関数のどれも値を返すことは期待されていない．

Author(s)

Paul Murrell

See Also

grid.draw

---

editDetails　　グリッドの編集をカスタマイズ

Description

この総称的フック関数はグリッド grob が grid.edit や editGrob を使って編集される毎に呼び出される．これは grob (や gTree) から導かれる新しいクラスの編集のカスタム化に対する機会を提供する．

Usage

```
editDetails(x, specs)
```

Arguments

x	グリッドの grob.
specs	名前付き要素のリスト．名前は修正される grob のスロットであり，値はそのスロットに対する新しい値である．

Details

この関数は grid.edit と editGrob により呼び出される．もしスロット中の変更が grob や gTree の子供達に影響を持てば (例えば grid:::editDetails.xaxis を見よ) grob や gTree から導かれたクラスに対するメソッドが書かれるべきである．

スロットは新しい値を持っていることを注意する．

Value

この関数は修正された grob を返さなければならない．

Author(s)

Paul Murrell

See Also

grid.edit

`explode` パスをその成分に*Explode a path into its components.*

Description

パスや grob パスをその成分に分解する.

Usage

```
explode(x)
## S3 method for class 'character'
explode(x)
## S3 method for class 'path'
explode(x)
```

Arguments

`x`	典型的にはビューポートや grob のパスであるが，ゼロやより多くの分離子を含む文字列ベクトルも又与えることができる.

Value

文字列ベクトル.

See Also

`vpPath`, `gPath`.

Examples

```
explode("vp1::vp2")
explode(vpPath("vp1", "vp2"))
```

`gEdit` *Edit* オブジェクトを作り適用する

Description

関数 `gEdit` と `gEditList` は編集操作を表現するオブジェクトを作る (本質的に `editGrob` への引数のリスト).

関数 `applyEdit` と `applyEdits` は一つまたはそれ以上のグラフィカルオブジェクトへの編集操作.

これらの関数は新しい関数とオブジェクトを作る開発者に対して最も有用である.

Usage

```
gEdit(...)
gEditList(...)
applyEdit(x, edit)
applyEdits(x, edits)
```

Arguments

...	editGrob 関数への一つまたはそれ以上の引数 (gEdit に対する)または一つまたはそれ以上の "gEdit" オブジェクト(gEditList に対する).
x	grob (グリッドのグラフィカルオブジェクト).
edit	"gEdit" オブジェクト.
edits	"gEdit" オブジェクトか "gEditList" オブジェクト.

Value

gEdit はクラス "gEdit" のオブジェクトを返す.

gEditList はクラス "gEditList" のオブジェクトを返す.

applyEdit と applyEditList は修正された grob を返す.

Author(s)

Paul Murrell

See Also

grob, editGrob

Examples

```
grid.rect(gp=gpar(col="red"))
# 同じこと，しかしより饒舌
grid.draw(applyEdit(rectGrob(), gEdit(gp=gpar(col="red"))))
```

---

getNames　ディスプレイリスト上の *grob* の名前のリスト

Description

ディスプレイリスト上の全てのトップレベルの grob の名前を含む文字列ベクトルを返す.

Usage

```
getNames()
```

Value

文字列ベクトル.

Author(s)

Paul Murrell

Examples

```
grid.grill()
getNames()
```

gpar　　グリッドのグラフィカルパラメータの処理

Description

gpar() はグラフィカルパラメータ設定のセットを作るために使われるべきである．これはクラス "gpar" のオブジェクトを返す．これは基本的に名前と値の対のリストである．

get.gpar() は現在のグラフィカルパラメータ設定のセットを問い合わせるのに使うことが出来る．

Usage

```
gpar(...)
get.gpar(names = NULL)
```

Arguments

...	幾つかの名前付きの引数.
names	適正なグラフィカルパラメータ名の文字列ベクトル.

Details

全てのグリッドのビューポートと (予め定義された)グラフィカルオブジェクトは gp と呼ばれるスロットを持ち，これは "gpar" オブジェクトを持つ．ビューポートがビューポートスタック上にプッシュされそしてグラフィカルオブジェクトが描画されると，"gpar" 中の設定が強要される．このようにして，グラフィカル出力は gp によりグラフィカルオブジェクトの描画が終了するまで，またはビューポートがビューポートスタックからポップオフされるまで，またはある他のビューポートまたはグラフィカルオブジェクトがプッシュされるか描画されるまで，変更される．

既定のパラメータ設定は ROOT ビューポートにより定義され，これはその設定をグラフィカルデバイスから取る．これらの既定はデバイス間で異なるかもしれない (例えば既定の fill 設定は PDF デバイスと比較した PNG デバイスに対しては異なる).

適正なパラメータ名は次の通りである：

col	線と境界のための色
fill	矩形や多角形等を塗り潰すための色
alpha	透明性のためのアルファチャンネル
lty	線種
lwd	線幅
lex	線幅に対する倍数
lineend	線の端のスタイル(丸，石突，正方形)
linejoin	線の結び目のスタイル(丸，留繋ぎ，はす縁)
linemitre	線の留繋ぎ限界(1より大きな数値)
fontsize	テキストのサイズ(ポイント単位)
cex	fontsize に対する倍数
fontfamily	フォント族
fontface	フォントのフェイス(ボールド，イタリック，...)
lineheight	テキストのサイズの倍数としての一行の高さ
font	フォントのフェイス(fontface に対する別名；後ろ向きの互換性のため)

これらの多くに対するより詳細は基本パッケージ中の対応するグラフィカルパラメータ par に対するヘルプを見よ． (これは少し異なる名前を持つかもしれない，例えば lend, ljoin, lmitre, family．)

色は rgb が返す形式の一つで指定できる，名前(colors を見よ)または現在の palette 中の非負整数添字 (ゼロは透明色とされる)． (負の値は今はエラーになる．)

alpha の設定は個々の色に対するアルファチャンネルに掛けることで結び付けられる (双方のアルファ設定は範囲 0 から 1 に正規化される).

テキストのサイズは fontsize*cex である．線のサイズは fontsize*cex*lineheight.

cex 設定は累積的である；もしビューポートが cex が 0.5 でプッシュされ，別のビューポートが cex が 0.5 でプッシュされると，実効的な cex は 0.25 になる．

alpha と lex の設定もまた累積的である．

fontfamily の変更はある種のデバイスでは無視されるが， PostScript, PDF, X11, Windows そして Quartz ではサポートされる． fontfamily は Hershey フォント族の一つを指定するのに使うことが出来 (例えば HersheySerif)そしてこの指定は全てのデバイス上で尊重される．

fontface の指定は整数や文字列が可能である．もし整数なら，それは R の基本グラフィカル標準に従う：1 = プレーン，2 = ボールド，3 = イタリック，4 = ボールドイタリック．もし文字列なら適正な値は："plain", "bold", "italic", "oblique" そして "bold.italic" である． HersheySerif フォント族の特殊例では "cyrillic", "cyrillic.oblique" そして "EUC" が又利用できる．

全てのパラメータ値は複数の値のベクトルになり得る． (これは常に意味を持つわけではない – 例えば，ビューポートは最初のパラメータ値だけを留意する．)

get.gpar() は全ての現在のグラフィカルパラメータの設定を返す．

Value

クラス "gpar" のオブジェクト．

Author(s)

Paul Murrell

See Also

Hershey.

Examples

```
gp <- get.gpar()
utils::str(gp)
## これは "gpar" オブジェクトを作るだけ：
gpar(col = "red")
gpar(col = "blue", lty = "solid", lwd = 3, fontsize = 16)
get.gpar(c("col", "lty"))
grid.newpage()
vp <- viewport(w = .8, h = .8, gp = gpar(col="blue"))
grid.draw(gTree(children=gList(rectGrob(gp = gpar(col="red")),
                     textGrob(paste("The rect is its own colour (red)",
                                    "but this text is the colour",
                                    "set by the gTree (green)",
                                    sep = "\n"))),
      gp = gpar(col="green"), vp = vp))
```

```
grid.text("This text is the colour set by the viewport (blue)",
          y = 1, just = c("center", "bottom"),
          gp = gpar(fontsize=20), vp = vp)
grid.newpage()
## パラメータに対する多重値の例
pushViewport(viewport())
grid.points(1:10/11, 1:10/11, gp = gpar(col=1:10))
popViewport()
```

---

gPath *Grob* 名を連結する

Description

この関数は grid.edit とその仲間が使うための grob パスを生成するために使われる.

grob パスは入れ子の grob 名のリストである.

Usage

```
gPath(...)
```

Arguments

...	grob 名の文字列ベクトル.

Details

grob 名は gTree 中で同じ親を共有する grobs の間でだけユニークである必要がある.

この関数は grob の親の名前 (そしてその親の名前，云々)を含む grob に対する指定を生成するのに使うことが出来る.

対話的な使用に対しては，パスを直接指定することが可能であるが，そうでなければグリッドの将来のバージョンでパスの分離子が変更される場合に備えこの関数を使うことが強く勧められる.

Value

gPath オブジェクト.

See Also

grob, editGrob, addGrob, removeGrob, getGrob, setGrob

Examples

```
gPath("g1", "g2")
```

Grid　　グリッドグラフィックス

Description

グリッドグラフィックスパッケージに関する一般的情報.

Details

グリッドグラフィックスは標準の R グラフィックスの代替法を提供する．ユーザはグラフィックスデバイス上に任意の矩形領域(ビューポートと呼ばれる)を定義でき各領域に幾つもの座標系を定義できる．描画は任意のビューポート中に利用可能な座標系のどれをも使って指定できる.

グリッドグラフィックスと標準の R グラフィックスは混ぜあわせられない！

(公開されている)グリッドグラフィックス関数のリストを見るには `library(help = grid)` とタイプする.

Author(s)

Paul Murrell

See Also

`viewport`, `grid.layout` そして `unit`.

Examples

```
## 簡単なレイアウトのダイアグラム
grid.show.layout(grid.layout(4,2,
                     heights=unit(rep(1, 4),
                                  c("lines", "lines", "lines", "null")),
                     widths=unit(c(1, 1), "inches")))
## サンプルビューポートのダイアグラム
grid.show.viewport(viewport(x=0.6, y=0.6,
                            w=unit(1, "inches"), h=unit(1, "inches")))
## 派手なプロットの例
grid.multipanel(vp=viewport(0.5, 0.5, 0.8, 0.8))
```

Grid Viewports　　グリッドのビューポートを作る

Description

これらの関数はグラフィックスデバイス上の矩形領域を記述するビューポートを作り，そしてこれらの領域の中に幾つかの座標系を定義する.

Usage

```
viewport(x = unit(0.5, "npc"), y = unit(0.5, "npc"),
         width = unit(1, "npc"), height = unit(1, "npc"),
         default.units = "npc", just = "centre",
         gp = gpar(), clip = "inherit",
         xscale = c(0, 1), yscale = c(0, 1),
         angle = 0,
         layout = NULL,
         layout.pos.row = NULL, layout.pos.col = NULL,
         name = NULL)
vpList(...)
vpStack(...)
vpTree(parent, children)
```

Arguments

x	x 位置を指定する数値ベクトルか単位オブジェクト.
y	y 位置を指定する数値ベクトルか単位オブジェクト.
width	幅を指定する数値ベクトルか単位オブジェクト.
height	高さを指定する数値ベクトルか単位オブジェクト.
default.units	もし x, y, width または height が数値ベクトルとしてだけ与えられた時に使用される既定の単位を指示する文字列.
just	ビューポートのその (x, y) 位置に関する位置揃えを指定する文字列か数値ベクトル. もし二つの値があれば, 最初の値は水平方向の位置揃えで二番目は垂直方向の位置揃えを指示する. 可能な文字列値は : "left", "right", "centre", "center", "bottom", そして "top". 数値の値に対しては, 0 は左揃えで 1 は右揃えを意味する.
gp	クラス gpar のオブジェクトで, 典型的には関数 gpar の呼び出しからの出力. これはグラフィカルパラメータ設定のリストである.
clip	"on", "inherit" または "off" の一つで, このビューポートの拡がりにクリップするか, 親ビューポートからクリップ領域を受け継ぐか, またはクリップを全てオフにするか, を指示する. 後方互換性のために, 論理値 TRUE は "on" にそして FALSE は "inherit" に対応する.
xscale	xスケール上の最小値と最大値を指定する長さ2の数値ベクトル. 限界は同じでないかもしれない.
yscale	yスケール上の最小値と最大値を指定する長さ2の数値ベクトル. 限界は同じでないかもしれない.
angle	ビューポートの回転角を指示する数値ベクトル. 正の値は度単位の回転量を指示し, 正のx軸から半時計回りである.
layout	ビューポートを副領域に分割するグリッドのレイアウトオブジェクト.
layout.pos.row	親のレイアウト中でこのビューポートにより占有される行を与える数値ベクトル.
layout.pos.col	親のレイアウト中でこのビューポートにより占有される列を与える数値ベクトル.
name	ビューポートツリー上にプッシュされた時にビューポートをユニークに特定するのに使われる文字列値.
...	任意の個数のグリッドのビューポートオブジェクト.

parent　　グリッドのビューポートオブジェクト.

children　　vpList オブジェクト.

Details

ビューポートの位置とサイズはビューポートの親 (グラフィカルデバイスか他のビューポート)により定義される座標系に相対的である. 位置とサイズはそれらを単位オブジェクトで指定することにより非常に柔軟な仕方で指定できる. ビューポートの位置を指定する時, layout.pos.row と layout.pos.col を共に NULL に指定することはビューポートがその親のレイアウトを無視し(その locn により)それ自体の位置とサイズそ指定することを指示する. もし layout.pos.row と layout.pos.col の一方が NULL ならば, これは適当な行/列の全てを専有することを意味する. 例えば layout.pos.row = 1 で layout.pos.col = NULL ならば行 1 の全てを専有することを意味する. layout.pos.row と layout.pos.col の双方に NULL でない値を指定することは適当な行と列の共有部分を専有することを意味する. もし layout.pos.row か layout.pos.col に長さ2のベクトルを指定するとこれは専有する行か列の範囲を指示する. 例えば layout.pos.row = c(1, 3) と layout.pos.col = c(2, 4) は行 1, 2 と 3, そして列 2, 3 と 4 の共通部分中のセルを専有することを意味する.

クリップは直近のビューポートのクリップ設定だけに従う. 例えば, もし viewport1 をクリップしそれから viewport2 をクリップすると, クリップ領域は完全に viewport2 で決まり, viewport1 のサイズと形状は無関係である (勿論 viewport2 がポップされるまで).

もしビューポートが回転されると (それ自体の angle 設定のせいか, それが回転されている別のビューポート中にあるかして) clip フラグは無視される.

ビューポートの名前はユニークである必要はない. プッシュされた時は同じ親を共有するビューポートはユニークな名前を持つ必要があり, これは既存のビューポートと同じ名前のビューポートをプッシュすると既存のビューポートはビューポートツリー中で置き換えられることを意味する. ビューポートの名前は任意の文字列で良いが, グリッドは "ROOT" をトップレベルのビューポートに対する予約語として使う. また downViewport と seekViewport 中のビューポート名を指定する時, ビューポートパスを提供することが可能であり, これは分離記号 (現在 ::) を使って幾つかの名前を連結したものからなる. 従ってこの分離記号をビューポート名として使うことは勧められない.

vpList 中のビューポートは並列にプッシュされる. vpStack 中のビューポートは順番にプッシュされる. vpTree がプッシュされると親が最初にプッシュされ, それから子供が並列でプッシュされる.

Value

クラス viewport の R オブジェクト.

Author(s)

Paul Murrell

See Also

Grid, pushViewport, popViewport, downViewport, seekViewport, upViewport, unit, grid.layout, grid.show.layout.

Examples

```
# ビューポートの例のダイアグラム
grid.show.viewport(viewport(x=0.6, y=0.6,
```

```
                    w=unit(1, "inches"), h=unit(1, "inches")))
# ビューポートのクリップのデモ
clip.demo <- function(i, j, clip1, clip2) {
  pushViewport(viewport(layout.pos.col=i,
                        layout.pos.row=j))
  pushViewport(viewport(width=0.6, height=0.6, clip=clip1))
  grid.rect(gp=gpar(fill="white"))
  grid.circle(r=0.55, gp=gpar(col="red", fill="pink"))
  popViewport()
  pushViewport(viewport(width=0.6, height=0.6, clip=clip2))
  grid.polygon(x=c(0.5, 1.1, 0.6, 1.1, 0.5, -0.1, 0.4, -0.1),
               y=c(0.6, 1.1, 0.5, -0.1, 0.4, -0.1, 0.5, 1.1),
               gp=gpar(col="blue", fill="light blue"))
  popViewport(2)
}

grid.newpage()
grid.rect(gp=gpar(fill="grey"))
pushViewport(viewport(layout=grid.layout(2, 2)))
clip.demo(1, 1, FALSE, FALSE)
clip.demo(1, 2, TRUE, FALSE)
clip.demo(2, 1, FALSE, TRUE)
clip.demo(2, 2, TRUE, TRUE)
popViewport()
# クリップの停止のデモ
grid.newpage()
pushViewport(viewport(w=.5, h=.5, clip="on"))
grid.rect()
grid.circle(r=.6, gp=gpar(lwd=10))
pushViewport(viewport(clip="inherit"))
grid.circle(r=.6, gp=gpar(lwd=5, col="grey"))
pushViewport(viewport(clip="off"))
grid.circle(r=.6)
popViewport(3)
# vpList, vpStack そして vpTree のデモ
grid.newpage()
tree <- vpTree(viewport(w=0.8, h=0.8, name="A"),
               vpList(vpStack(viewport(x=0.1, y=0.1, w=0.5, h=0.5,
                                       just=c("left", "bottom"), name="B"),
                              viewport(x=0.1, y=0.1, w=0.5, h=0.5,
                                       just=c("left", "bottom"), name="C"),
                              viewport(x=0.1, y=0.1, w=0.5, h=0.5,
                                       just=c("left", "bottom"), name="D")),
                      viewport(x=0.5, w=0.4, h=0.9,
                               just="left", name="E")))
pushViewport(tree)
for (i in LETTERS[1:5]) {
  seekViewport(i)
  grid.rect()
  grid.text(current.vpTree(FALSE),
            x=unit(1, "mm"), y=unit(1, "npc") - unit(1, "mm"),
            just=c("left", "top"),
            gp=gpar(fontsize=8))
}
```

grid.add　　グリッドのグラフィカルオブジェクトを加える

Description

gTree や gTree の子孫に grob を加える.

Usage

```
grid.add(gPath, child, strict = FALSE, grep = FALSE,
         global = FALSE, allDevices = FALSE, redraw = TRUE)

addGrob(gTree, child, gPath = NULL, strict = FALSE, grep = FALSE,
        global = FALSE, warn = TRUE)

setChildren(x, children)
```

Arguments

gTree, x	gTree オブジェクト.
gPath	gPath オブジェクト. grid.add に対してはこれはディスプレイリスト上の gTree を指定する. addGrob に対してはこれは指定された gTree の子孫を指定する.
child	grob オブジェクト.
children	gList オブジェクト.
strict	gPath は正確にマッチされるべきかどうかを指示する論理値.
grep	gPath は正規表現として扱われるかどうかを指示するブール値. 値は gPath の要素に渡ってリサイクルされる (たとえば c(TRUE, FALSE) は gPath の全ての奇数番目の要素が正規表現として扱われることを意味する).
global	関数は最初にマッチした gPath だけに作用するか, それともマッチした全てに作用するかを指示するブール値.
warn	指定された gPath を見つけるのに失敗したらエラーを発生するかどうかを指示する論理値.
allDevices	マッチするものを全ての開かれたデバイスで探すか, それとも現在のデバイスだけを探すかを指示するブール値. 現在まだ移植されていない.
redraw	grob を再描画するかどうかを指示する論理値.

Details

addGrob は指定された grob をコピーし, そして修正された grob を返す.

grid.add はディスプレイリスト上の grob を破滅的に変更する. もし redraw が TRUE ならばそれは変更を反映するために全てを再描画する.

setChildren は一つの gTree の全ての子供を一度に設定する (addGrob を繰り返し呼び出す代わりに) ための基本関数である.

Value

addGrob は grob オブジェクトを返す； grid.add は NULL を返す.

Author(s)

Paul Murrell

See Also

grob, getGrob, addGrob, removeGrob.

---

grid.bezier　　ベジエ曲線を描く

Description

これらの関数はベジエ曲線を作り描く (四つの制御点に関して描かれる曲線).

Usage

```
grid.bezier(...)
bezierGrob(x = c(0, 0.5, 1, 0.5), y = c(0.5, 1, 0.5, 0),
           id = NULL, id.lengths = NULL,
           default.units = "npc", arrow = NULL,
           name = NULL, gp = gpar(), vp = NULL)
```

Arguments

x	スプライン制御点の x 位置を指定する数値ベクトルまたは単位オブジェクト.
y	スプライン制御点の y 位置を指定する数値ベクトルまたは単位オブジェクト.
id	x と y 中の位置を複数のベジエ曲線に分離するのに使われる数値ベクトル. 同じ id を持つ全ての位置は同じベジエ曲線に属する.
id.lengths	x と y 中の位置を複数のベジエ曲線に分離するのに使われる数値ベクトル. 個別のベジエ曲線を構成する位置の引き続くブロックを指定する.
default.units	もし x や y が数値ベクトルとしてだけ与えられた時使われる既定の単位を指示する文字列.
arrow	ベジエ曲線のどちらかの端点に置かれる arrow 関数が生成するような矢尻を記述するリスト.
name	文字列識別子.
gp	クラス gpar のオブジェクトで，典型的には関数 gpar の呼び出しからの出力. これは基本的にグラフィカルパラメータ設定のリストである.
vp	グリッドのビューポートオブジェクト(または NULL).
...	bezierGrob に渡される引数.

Details

どちらの関数も beziergrob (ベジエ曲線を記述するグラフィカルオブジェクト)を作るが grid.bezier だけがベジエ曲線を描く.

ベジエ曲線は四つの制御点に関して描かれる直線である.

x と y に対する欠損値は許されない (つまり制御点が欠損することは不正である).

曲線は現在 X-スプラインに基づく近似を使い描かれる.

Value

grob オブジェクト.

See Also

Grid, viewport, arrow.

grid.xspline.

Examples

```
x <- c(0.2, 0.2, 0.4, 0.4)
y <- c(0.2, 0.4, 0.4, 0.2)

grid.newpage()
grid.bezier(x, y)
grid.bezier(c(x, x + .4), c(y + .4, y + .4),
            id=rep(1:2, each=4))
grid.segments(.4, .6, .6, .6)
grid.bezier(x, y,
            gp=gpar(lwd=3, fill="black"),
            arrow=arrow(type="closed"),
            vp=viewport(x=.9))
```

---

grid.cap　　ラスター画像のキャプチャ

Description

グラフィックスデバイスの現在の内容をラスター(ビットマップ)画像としてキャプチャする.

Usage

```
grid.cap()
```

Details

この関数はスクリーン上のグラフィックスデバイスに対してだけ実装されている.

Value

R の色名の行列，もし利用できなければ NULL.

Author(s)

Paul Murrell

See Also

grid.raster

それがサポートされているかどうかを見るには dev.capabilities.

Examples

```
dev.new(width=0.5, height=0.5)
grid.rect()
grid.text("hi")
cap <- grid.cap()
dev.off()

if(!is.null(cap))
    grid.raster(cap, width=0.5, height=0.5, interpolate=FALSE)
```

---

grid.circle 円を描く

Description

円を作り描画する関数.

Usage

```
grid.circle(x=0.5, y=0.5, r=0.5, default.units="npc", name=NULL,
            gp=gpar(), draw=TRUE, vp=NULL)
circleGrob(x=0.5, y=0.5, r=0.5, default.units="npc", name=NULL,
            gp=gpar(), vp=NULL)
```

Arguments

x	x 位置を指定する数値ベクトルか単位オブジェクト.
y	y 位置を指定する数値ベクトルか単位オブジェクト.
r	半径を指定する数値ベクトルか単位オブジェクト.
default.units	もし x, y, width または height が数値としてだけ与えられた時使われる既定の単位を指示する文字列.
name	文字識別子.
gp	クラス gpar のオブジェクトで，典型的には関数 gpar の呼び出しの出力. これは基本的にはグラフィックスパラメータの設定のリストである.
draw	グラフィックス出力を生成するかどうかを指示する論理値.
vp	グリッドのビューポートオブジェクト(または NULL).

Details

どちらの関数も円の grob (円を記述するグラフィカルオブジェクト)を作るが，grid.circle() だけが円を描く (そしてもし draw が TRUE の時だけ).

半径は任意の単位で与えることが出来る；もし単位が相対的(例えば "npc" や "native")ならば半径はそれが幅か高さとして解釈されるのに応じて異なる．そうした場合これらの二つの値の小さいほうが結果になる．この効果を見るには grid.circle() とタイプしウィンドのサイズを調整する.

非常に小さな半径に対して何が起こるかはデバイス依存である：円は見え無くなるかもしれないし固定した最小値で表示されるかもしれない：半径ゼロの円はプロットされない.

Value

円の grob．grid.circle() は値を不可視で返す.

警告

半径に対する負の値は黙って絶対値に変換される.

Author(s)

Paul Murrell

See Also

Grid, viewport

---

grid.clip　　クリップ領域を設定する

Description

これらの関数は現在の座標系を変更せずに現在のビューポート内にクリップ領域を設定する.

Usage

```
grid.clip(...)
clipGrob(x = unit(0.5, "npc"), y = unit(0.5, "npc"),
         width = unit(1, "npc"), height = unit(1, "npc"),
         just = "centre", hjust = NULL, vjust = NULL,
         default.units = "npc", name = NULL, vp = NULL)
```

Arguments

x	x 位置を指定する数値ベクトルか単位オブジェクト.
y	y 位置を指定する数値ベクトルか単位オブジェクト.
width	幅を指定する数値ベクトルか単位オブジェクト.
height	高さを指定する数値ベクトルか単位オブジェクト.

just	クリップ矩形のその (x, y) 位置に関する位置揃え．もし二つの値があれば，最初の値は水平位置揃え，二番目の値は垂直位置揃えを指定する．可能な文字値は："left", "right", "centre", "center", "bottom", そして "top"．数値に対しては 0 は左揃えで 1 は右揃えを意味する．
hjust	水平位置揃えを指定する数値ベクトル．もし指定されると just 設定を上書きする．
vjust	垂直位置揃えを指定する数値ベクトル．もし指定されると just 設定を上書きする．
default.units	もし x, y, width または height が数値ベクトルとしてだけ与えられている時使用される既定単位を指示する文字列．
name	文字列識別子．
vp	グリッドのビューポートオブジェクト(または NULL)．
...	clipGrob に渡される引数．

Details

どちらの関数もクリップ矩形 (クリップ矩形を記述するグラフィカルオブジェクト)を作るが， grid.clip だけがクリップを実行する．

ビューポートのプッシュとポップは 常にクリップ grob によって設定されたクリップ領域を上書きし，ビューポートがクリップ領域を明示的に実行するかどうかに依存しない．

Value

clipGrob はクリップ grob を返す．

Author(s)

Paul Murrell

See Also

Grid, viewport

Examples

```
# ビューポート全体に渡って描画するがクリップする
grid.clip(x = 0.3, width = 0.1)
grid.lines(gp=gpar(col="green", lwd=5))
# ビューポート全体に渡って描画するがクリップする(異なった場所に)
grid.clip(x = 0.7, width = 0.1)
grid.lines(gp=gpar(col="red", lwd=5))
# ビューポートが新しいクリップ領域を設定する
pushViewport(viewport(width=0.5, height=0.5, clip=TRUE))
grid.lines(gp=gpar(col="grey", lwd=3))
# 元々のビューポートに戻る；
# クリップ領域を以前の grid.clip() から得る
# (以前のビューポートクリップ領域からではない)
popViewport()
grid.lines(gp=gpar(col="black"))
```

grid.convert 異なったグリッド座標系間の変換

Description

これらの関数は単位オブジェクトを取り，それを異なった座標系での同値な単位オブジェクトに変換する．

Usage

```
convertX(x, unitTo, valueOnly = FALSE)
convertY(x, unitTo, valueOnly = FALSE)
convertWidth(x, unitTo, valueOnly = FALSE)
convertHeight(x, unitTo, valueOnly = FALSE)
convertUnit(x, unitTo,
            axisFrom = "x", typeFrom = "location",
            axisTo = axisFrom, typeTo = typeFrom,
            valueOnly = FALSE)
```

Arguments

x	単位オブジェクト．
unitTo	単位をそこに変換する座標系．適正な座標系については unit 関数を見よ．
axisFrom	単位オブジェクトが x 方向の値か y 方向の値を表すかを指示する "x" か "y"．
typeFrom	単位オブジェクトが位置か長さかを指示する "location" か "dimension"．
axisTo	axisFrom と同じであるが，作られるべき単位オブジェクトに適用される．
typeTo	typeFrom であるが，作られるべき単位オブジェクトに適用される．
valueOnly	論理値，もし TRUE ならば関数は単位オブジェクトを返さず，変換された数値だけを返す．

Details

convertUnit 関数は一般的な変換を許す．他の四つの関数は単に最も普通のケースに対するそのフロントエンドである．

変換は現在のビューポート内で行われる．

現在全ての適正な座標系に変換することは不可能である (例えば "strwidth" や "grobwidth")．これら全てが不可能かどうかは確かではないが，この段階では単に不可能のように見える．

グリッドの通常の用法では，これらの関数は必要になるべきではない．もしユーザ座標系ではなく位置や次元をインチで表現したければ， unit(0.134, "native") の様なものではなく単に unit(1, "inches") の様なものを行うべきである．

しかしながらある場合には，ユーザが単位値に基づいて計算を実行することが必要になり，そしてこの関数が必要になる．そうした場合，下の警告に留意して欲しい．

Value

指定された座標系での単位オブジェクト (valueOnly が TRUE でない限り，その場合は返り値は数値である).

警告

変換は現在のデバイスのサイズに対してのみ適正である．もしデバイスのサイズが変更されると少なくともある種の変換は不正になる．例えば，単位オブジェクトを次のように作る： oneinch <- convertUnit(unit(1, "inches"), "native". そしてデバイスのサイズを変更すると，1インチ単位の単位オブジェクトは最早物理的な1インチには対応しない.

Author(s)

Paul Murrell

See Also

unit

Examples

```
## 同語反復
convertX(unit(1, "inches"), "inches")
## 物理的単位
convertX(unit(2.54, "cm"), "inches")
convertX(unit(25.4, "mm"), "inches")
convertX(unit(72.27, "points"), "inches")
convertX(unit(1/12*72.27, "picas"), "inches")
convertX(unit(72, "bigpts"), "inches")
convertX(unit(1157/1238*72.27, "dida"), "inches")
convertX(unit(1/12*1157/1238*72.27, "cicero"), "inches")
convertX(unit(65536*72.27, "scaledpts"), "inches")
convertX(unit(1/2.54, "inches"), "cm")
convertX(unit(1/25.4, "inches"), "mm")
convertX(unit(1/72.27, "inches"), "points")
convertX(unit(1/(1/12*72.27), "inches"), "picas")
convertX(unit(1/72, "inches"), "bigpts")
convertX(unit(1/(1157/1238*72.27), "inches"), "dida")
convertX(unit(1/(1/12*1157/1238*72.27), "inches"), "cicero")
convertX(unit(1/(65536*72.27), "inches"), "scaledpts")

pushViewport(viewport(width=unit(1, "inches"),
                      height=unit(2, "inches"),
                      xscale=c(0, 1),
                      yscale=c(1, 3)))
  ## 位置と次元
  convertY(unit(2, "native"), "inches")
  convertHeight(unit(2, "native"), "inches")
  ## "x" から "y"  へ(変換は "inches" による)
  convertUnit(unit(1, "native"), "native",
              axisFrom="x", axisTo="y")
  ## 幾つかの値を一度に変換
  convertX(unit(c(0.5, 2.54), c("npc", "cm")),
               c("inches", "native"))
popViewport()
```

```
## 複雑な単位を変換
convertX(unit(1, "strwidth", "Hello"), "native")
```

---

grid.copy　　グリッドのグラフィカルオブジェクトのコピーを作る

Description

この関数は冗長であり将来のバージョンでは消えるであろう.

Usage

```
grid.copy(grob)
```

Arguments

grob	grob オブジェクト.

Value

grob オブジェクトのコピー.

Author(s)

Paul Murrell

See Also

grid.grob.

---

grid.curve　　位置間に曲線を描く

Description

これらの関数はある位置から別の位置への曲線を作り描く.

Usage

```
grid.curve(...)
curveGrob(x1, y1, x2, y2, default.units = "npc",
          curvature = 1, angle = 90, ncp = 1, shape = 0.5,
          square = TRUE, squareShape = 1,
          inflect = FALSE, arrow = NULL, open = TRUE,
          debug = FALSE,
          name = NULL, gp = gpar(), vp = NULL)
arcCurvature(theta)
```

Arguments

x1	開始点の x 位置を指定する数値ベクトルか単位オブジェクト.
y1	開始点の y 位置を指定する数値ベクトルか単位オブジェクト.
x2	終了点の x 位置を指定する数値ベクトルか単位オブジェクト.
y2	終了点の y 位置を指定する数値ベクトルか単位オブジェクト.
default.units	もし x1, y1, x2 または y2 が数値としてだけ与えられた時に使われる既定の単位.
curvature	曲率量を与える数値．負の値は左回り曲線を，正の値は右回り曲線を，ゼロは直線を与える.
angle	0 と 180 の間の数値で，曲線の制御点を歪める量を与える．歪み 90 未満の値は曲線を開始点の方に歪め，歪みが 90 より大きな値は曲線を終了点の方に歪める.
ncp	曲線を描くのに使われる制御点の数．制御点が大きほどより滑らかな曲線になる.
shape	-1 と 1 の間の数値ベクトルで，その制御点に関する曲線の形状を制御する．より詳細は grid.xspline を見よ.
square	曲線に対する制御点が市街ブロック風に作るか鋭角にかを制御する論理値． ncp が 1 で angle が 90 ならば，これは典型的に TRUE で，さもなければこれは恐らく FALSE に設定されるべきである(下の例を見よ).
squareShape	square が TRUE ならば挿入される任意の追加制御点に関する曲線の挙動を制御する shape 値.
inflect	曲線を半分にカットし逆転するかどうかを指定する論理値(下の例を見よ).
arrow	arrow 関数が作るような，曲線のどちらかの端に置かれる矢尻を記述するリスト.
open	曲線を閉じるかどうかを指示する論理値 (開始点と終了点を結ぶ).
debug	デバッグ情報を描くべきかどうかを指示する論理値.
name	文字識別子.
gp	クラス gpar のオブジェクトで，典型的には関数 gpar への呼び出しからの出力．これは基本的にはグラフィカルパラメータ設定のリストである.
vp	グリッドのビューポートオブジェクト(または NULL).
...	curveGrob に渡される引数.
theta	角度(度単位).

Details

どちらの関数も曲線 grob (曲線を記述するグラフィカルオブジェクト) を作るが grid.curve だけが曲線を描く.

arcCurvature 関数は角度 theta に対応する弧上の制御点が生成されるように curvature を計算するのに使うことが出来る．これは典型的に希望する弧に対応する曲線を生成するように大きな ncp と共に使われる.

Value

grob オブジェクト.

See Also

Grid, viewport, grid.xspline, arrow

Examples

```
curveTest <- function(i, j, ...) {
  pushViewport(viewport(layout.pos.col=j, layout.pos.row=i))
  do.call("grid.curve", c(list(x1=.25, y1=.25, x2=.75, y2=.75), list(...)))
  grid.text(sub("list\\((.*)\\)", "\\1",
                deparse(substitute(list(...)))),
            y=unit(1, "npc"))
  popViewport()
}
# grid.newpage()
pushViewport(plotViewport(c(0, 0, 1, 0),
                          layout=grid.layout(2, 1, heights=c(2, 1))))
pushViewport(viewport(layout.pos.row=1,
                      layout=grid.layout(3, 3, respect=TRUE)))
curveTest(1, 1)
curveTest(1, 2, inflect=TRUE)
curveTest(1, 3, angle=135)
curveTest(2, 1, arrow=arrow())
curveTest(2, 2, ncp=8)
curveTest(2, 3, shape=0)
curveTest(3, 1, curvature=-1)
curveTest(3, 2, square=FALSE)
curveTest(3, 3, debug=TRUE)
popViewport()
pushViewport(viewport(layout.pos.row=2,
                      layout=grid.layout(3, 3)))
curveTest(1, 1)
curveTest(1, 2, inflect=TRUE)
curveTest(1, 3, angle=135)
curveTest(2, 1, arrow=arrow())
curveTest(2, 2, ncp=8)
curveTest(2, 3, shape=0)
curveTest(3, 1, curvature=-1)
curveTest(3, 2, square=FALSE)
curveTest(3, 3, debug=TRUE)
popViewport(2)
```

---

grid.delay 計算と *grob* の生成のカプセル化

Description

計算と計算結果に依存する grob の生成を共に含み，場面が再描画されると(例えばデバイスのサイズが変更されたり編集されたりして)計算と grob の生成が再実行されるような表現式を評価する．

エキスパートだけが使用することを意図している．

Usage

```
delayGrob(expr, list, name=NULL, gp=NULL, vp=NULL)
grid.delay(expr, list, name=NULL, gp=NULL, vp=NULL)
```

Arguments

expr	モード expression のオブジェクト，または call，または未評価の表現式.
list	その中で expr が評価される環境を定義するリスト.
name	文字列識別子.
gp	クラス gpar のオブジェクトで，典型的には関数 gpar の呼び出しからの出力．これは基本的にはグラフィカルパラメータの設定である.
vp	グリッドのビューポートオブジェクト(または NULL).

Details

特別なクラス "delayedgrob" の grob が生成される (そして grid.delay の場合は描かれる)．このクラスに対する makeContent メソッドはリストを評価環境として表現式を (そしてその環境を親とするグリッドの名前空間として)評価する.

expr 引数はその結果として grob を返すべきである.

これらの関数は grid.record() と recordGrob() 関数の類似物である；違いはこれらの関数は makeContent() フックに基づいている一方，それらは drawDetails() フックに基づいていることである.

Note

この関数は関数 recordGraphics の代わりに使われなければ ならない； recordGraphics の使用に関する恐ろしい警告の全てがまたここで責任を持って適用される.

Author(s)

Paul Murrell

See Also

recordGraphics

Examples

```
grid.delay({
              w <- convertWidth(unit(1, "inches"), "npc")
              rectGrob(width=w)
            },
            list())
```

grid.display.list　グリッドのディスプレイリストを制御する

Description

グリッドのディスプレイリストをオン・オフする

Usage

```
grid.display.list(on=TRUE)
engine.display.list(on=TRUE)
```

Arguments

on	グリッドのディスプレイリストをオン・オフすべきかを指示する論理値.

Details

描画とビューポート設定の操作は (既定では)グリッドのディスプレイリスト中に記録される．これは編集操作に続いて再描画を行うことを許す.

このディスプレイリストは非常に大きくなる可能性があり，ある場合にはそれをオフにすることが有用になり得る.

全てのグラフィックス出力も又 R のグラフィックスエンジンのメインのディスプレイリストに記録される(既定で)．これはデバイスのサイズ変更に続く再描画をサポートしデバイス間のコピーを可能にする.

このディスプレイをオフにすることはデバイスのサイズ変更とコピーに対してグリッドがそれ自体のディスプレイリストから再描画することを意味する．これはグラフィックスエンジンのディスプレイリストを使うよりも遅くなる.

Value

無い.

警告

ディスプレイリストをオンにすることはディスプレイリストの消去を惹き起こす!

ディスプレイリストとグラフィックスエンジンリストを共にをオフにすると再描画が一切行われない結果になる.

Author(s)

Paul Murrell

grid.DLapply　　グリッドのディスプレイリストの修正

Description

現在のディスプレイリストの各要素に対して関数を呼び出す.

Usage

```
grid.DLapply(FUN, ...)
```

Arguments

FUN	関数：この関数の最初の引数はディスプレイリストの各要素に渡される.
...	FUN に渡される追加引数.

Details

この関数は極めて危険である (グリッドのディスプレイリストに対して).

二つの形式的な努力がディスプレイリストに関する完全なゴミで終わることを避けるために行われる.

1. ディスプレイリストは全ての新しい要素が生成されてからのみ置き換えられる (従って生成の途中のエラーは中途のディスプレイリストには反映しない).
2. 全ての新しい要素は NULL かそれらが置き換えようとしている要素のクラスを継承しなければならない.

Value

これらの関数の副作用は普通グリッドのディスプレイリストを変更することである.

See Also

Grid.

Examples

```
grid.newpage()
grid.rect(width=.4, height=.4, x=.25, y=.75, gp=gpar(fill="black"), name="r1")
grid.rect(width=.4, height=.4, x=.5, y=.5, gp=gpar(fill="grey"), name="r2")
grid.rect(width=.4, height=.4, x=.75, y=.25, gp=gpar(fill="white"), name="r3")
grid.DLapply(function(x) { if (is.grob(x)) x$gp <- gpar(); x })
grid.refresh()
```

grid.draw グリッドの *grob* を描画する

Description

グラフィカルオブジェクトからグラフィカルな出力を作る.

Usage

```
grid.draw(x, recording=TRUE)
```

Arguments

x	クラス "grob" のオブジェクトか NULL.
recording	描画操作をグリッドのディスプレイリストに記録すべきかどうかを指示する論理値.

Details

これは grob と gTree オブジェクトに対するメソッドを持つ総称的関数関数である.

grob と gTree メソッドは自動的に任意のビューポートを vp スロットにプッシュし，そして自動的に gp 中の任意の gpar 設定を適用する．加えて gTree メソッドは childrenvp スロット中の任意のビューポートをプッシュしアップし，そして children スロット中の任意の grob に対して grid.draw を自動的に呼び出す.

grob と gTree に対するメソッドは総称的なフック関数 preDrawDetails, drawDetails そして postDrawDetails を呼び出し，grob や gTree から導かれたクラスが追加のビューポートのプッシュ/ポップの実行や grobs と gTrees に対する既定動作を超える追加出力を生成することを許す.

Value

無い.

Author(s)

Paul Murrell

See Also

grob.

Examples

```
grid.newpage()
## グラフィカルオブジェクトを作るが，しかし描画しない
l <- linesGrob()
## それを描画する
grid.draw(l)
```

grid.edit　　グリッドのグラフィカルオブジェクトの記述を編集

Description

grob のスロットの一つの値を変更しそれを再描画する.

Usage

```
grid.edit(gPath, ..., strict = FALSE, grep = FALSE,
          global = FALSE, allDevices = FALSE, redraw = TRUE)

grid.gedit(..., grep = TRUE, global = TRUE)

editGrob(grob, gPath = NULL, ..., strict = FALSE, grep = FALSE,
         global = FALSE, warn = TRUE)
```

Arguments

grob	grob オブジェクト.
...	新しいスロット値を指定するゼロかそれ以上の名前付き引数.
gPath	gPath オブジェクト. grid.edit に対してはこれはディスプレイリスト上の grob を指定する. editGrob に対してはこれは指定された grob の子孫を指定する.
strict	gPath は正確にマッチされるべきかどうかを指示するブール値.
grep	gPath が正規表現として扱われるべきかどうかを指示するブール値. 値は gPath の要素を跨がりリサイクルされる (例えば c(TRUE, FALSE) は gPath の全ての奇数番目の要素が正規表現として扱われるべきであることを意味する).
global	関数が gPath の最初のマッチに影響すべきか, 全てのマッチに影響すべきかを指示するブール値.
warn	指定された gPath を見つけるのに失敗したらエラーを発生するかどうかを指示する論理値.
allDevices	全ての開かれたデバイスがマッチに対して検索されるか, または現在のデバイスだけかを指示するブール値. まだ実装されていない.
redraw	grob を再描画するかどうかを指示する論理値.

Details

editGrob は指定された grob をコピーし修正された grob を返す.

grid.edit はディスプレイリスト上の grob を破壊的に修正する. もし redraw が TRUE ならばそれから変更を反映するために全てを再描画する.

どちらの関数も grob がカスタム化された行動を実行することを許すために editDetails を呼び出し, 修正された grob が依然首尾一貫しているかどうかをチェックするために validDetails を呼び出す.

grid.gedit (global に対する g)は異なった既定値に対する grid.edit の単なる便利なラッパである.

Value

editGrob は grob オブジェクトを返す； grid.edit は NULL を返す.

Author(s)

Paul Murrell

See Also

grob, getGrob, addGrob, removeGrob.

Examples

```
grid.newpage()
grid.xaxis(name = "xa", vp = viewport(width=.5, height=.5))
grid.edit("xa", gp = gpar(col="red"))
# チックが無い(at が NULL)なので動作しない
try(grid.edit(gPath("xa", "ticks"), gp = gpar(col="green")))
grid.edit("xa", at = 1:4/5)
# 今や動作すべきである
try(grid.edit(gPath("xa", "ticks"), gp = gpar(col="green")))
```

---

grid.force *grob* をその成分に強制する

Description

ある種の grob は描画時に描画する内容を生成する；この関数はそうした grob をそれらの描画時の内容に置き換える.

Usage

```
grid.force(x, ...)
## Default S3 method:
grid.force(x, redraw = FALSE, ...)
## S3 method for class 'gPath'
grid.force(x, strict = FALSE, grep = FALSE, global = FALSE,
                           redraw = FALSE, ...)
## S3 method for class 'grob'
grid.force(x, draw = FALSE, ...)
forceGrob(x)
grid.revert(x, ...)
## S3 method for class 'gPath'
grid.revert(x, strict = FALSE, grep = FALSE, global = FALSE,
                            redraw = FALSE, ...)
## S3 method for class 'grob'
grid.revert(x, draw = FALSE, ...)
```

Arguments

x	既定のメソッドに対しては x は指定されるべきではない．さもなければ x は grob か gPath であるべきである．もし x が文字列ならば，それは gPath と仮定される．
strict	path は正確にマッチされるべきかどうかを指示するブール値.
grep	path は正規表現として扱われるべきか.
global	関数が path の最初のマッチに影響するか，それとも全てのマッチに影響するかを指示する論理値.
draw	強制された後で grob を描画すべきかどうかを指示する論理値.
redraw	強制操作の後で **grid** シーンを再描画するかどうかを指示する論理値.
...	メソッドが使う追加引数.

Details

ある種の grob は描画時までは実際に描かれる内容の生成を待つ (at や NULL で grid.xaxis() により生成される軸は，それがどのようなチックマークを描くかを決定可能になる前にそのようなビューポートに描き込もうとしているのかを見なければならないので，良い例になる).

そうした grob の内容(例えばチックマーク)は普通 grid.ls() には不可視であるか grid.edit() ではアクセスできない.

grid.force() 関数は grob を描画時内容で 置き換える．例えば軸は実際に描かれる軸チックマークを表現する線とテキストを持つバニラ gTree で置き換えられる．これはチックマークを grid.ls() に可視で grid.edit() でアクセス出来るようにする.

forceGrob() 関数は grid.force() に対する内部的実働関数であり，普通ユーザから直接呼び出されることはない．これは移出されるので必要なら grob クラスをカスタム化するメソッドを書くことが出来る.

grid.revert() 関数は grid.force() の効果を逆転し，強制された内容をオリジナルのものに置き換える.

警告

明示的な grob 手順の強制は grob が 現在の描画コンテキスト中で grob が描かれたかのような結果を作る．異なった描画コンテキスト中で結果を描画することは意味がないかもしれない.

注意

これらの関数は描画時に makeContext() と makeContent() メソッドを使ってそれらの内容を生成する grob に対してだけ効果を持つ (描画時に preDrawDetails() と drawDetails() メソッドを使ってそれらの内容を生成する grob に対してでは無い).

Author(s)

Paul Murrell

Examples

```
grid.newpage()
pushViewport(viewport(width=.5, height=.5))
# xaxis を描く
grid.xaxis(name="xax")
grid.ls()
# xaxis を強制する
grid.force()
grid.ls()
# xaxis を戻す
grid.revert()
grid.ls()
# yaxis を描き強制する
grid.force(yaxisGrob(), draw=TRUE)
grid.ls()
# yaxis を元に戻す
grid.revert()
grid.ls()
# xaxis だけを強制する
grid.force("xax")
grid.ls()
# 全てを強制する
grid.force()
grid.ls()
# xaxis だけを元に戻す
grid.revert("xax")
grid.ls()
```

---

grid.frame　パック化オブジェクトに対するフレームを作る

Description

これらの関数は， grid.pack, grid.place, packGrob そして placeGrob とともにグラフィカルなイメージを構成するための GUI 構成子風のインタフェイスの一部分である．アイデアはこの関数を使いフレームを作り，それからオブジェクトをフレーム中にパックしたり置いたりするために grid.pack や何やらを使うということである．

Usage

```
grid.frame(layout=NULL, name=NULL, gp=gpar(), vp=NULL, draw=TRUE)
frameGrob(layout=NULL, name=NULL, gp=gpar(), vp=NULL)
```

Arguments

layout	グリッドのレイアウトまたは NULL．これはフレームを幾つかの行と列を初期幅と高さで初期化するのに使うことが出来る．
name	文字列識別子．
vp	クラス viewport のオブジェクトか NULL．
gp	クラス gpar のオブジェクト；典型的には関数 gpar への呼び出しからの出力．
draw	フレームを描画するべきか．

Details

どちらの関数もフレーム grob (フレームを記述するグラフィカルオブジェクト)を作るが，grid.frame() だけがフレームを描画する (そして draw が TRUE の時だけ)．実際には何も描かれないが，フレームをディスプレイリスト上に置かれ，それはオブジェクトがフレーム中にパックされるにつれ出力が動的に更新されることを意味する．デバッグに対しても有用かもしれない．

Value

フレーム grob．grid.frame() は値を不可視で返す．

Author(s)

Paul Murrell

See Also

grid.pack

Examples

```
grid.newpage()
grid.frame(name="gf", draw=TRUE)
grid.pack("gf", rectGrob(gp=gpar(fill="grey")), width=unit(1, "null"))
grid.pack("gf", textGrob("hi there"), side="right")
```

---

grid.function　関数を表す曲線を描く

Description

関数を表す曲線

Usage

```
grid.function(...)
functionGrob(f, n = 101, range = "x", units = "native",
             name = NULL, gp=gpar(), vp = NULL)

grid.abline(intercept, slope, ...)
```

Arguments

f	単一の引数を取り名前が x と y の数値成分を持つリストを返す関数．
n	関数 f への入力として生成される数の値．
range	"x", "y" または数値．‘詳細’節を見よ．
units	関数により生成される x と y 値に対して使われる単位を指定する文字列．
intercept	数値．
slope	数値．

...	grid.function() に渡される引数.
name	文字列識別子.
gp	クラス gpar のオブジェクトで，典型的には関数 gpar への呼び出しからの出力．これは基本的にはグラフィカルパラメータ設定のリスト.
vp	グリッドのビューポートオブジェクト(または NULL).

Details

値 n が生成され関数 f へ渡され，一連の線が結果の値 x と y を通って引かれる.

n 個の値の生成は range の値に依存する．既定のケースでは dim は "x" で，これは x 値のセットが現在の x 次元中のビューポートのスケールの範囲をカバーするように生成される．もし dim が "y" なら値は代わりに現在の y スケールから使われる．もし range が数値ベクトルならば，値はその範囲から生成される.

grid.abline() は intercept と slope でパラメータ化された直線に対する簡単なフロントエンドである.

Value

functiongrob grob.

Author(s)

Paul Murrell

See Also

Grid, viewport

Examples

```
# abline
# 注意：スクリーン上の ROOT ビューポート，(0, 0) はトップ左
#       そして "native" はピクセル単位!
grid.function(function(x) list(x=x, y=0 + 1*x))
# 既定の正規化された "native" 座標のより"通常"のビューポート
grid.newpage()
pushViewport(viewport())
grid.function(function(x) list(x=x, y=0 + 1*x))
# すこし簡単
grid.newpage()
pushViewport(viewport())
grid.abline()
# サイン曲線
grid.newpage()
pushViewport(viewport(xscale=c(0, 2*pi), yscale=c(-1, 1)))
grid.function(function(x) list(x=x, y=sin(x)))
# 制約付きサイン曲線
grid.newpage()
pushViewport(viewport(xscale=c(0, 2*pi), yscale=c(-1, 1)))
grid.function(function(x) list(x=x, y=sin(x)),
              range=0:1)
# 逆サイン曲線
grid.newpage()
pushViewport(viewport(xscale=c(-1, 1), yscale=c(0, 2*pi)))
```

```
grid.function(function(y) list(x=sin(y), y=y),
              range="y")
# パラメトリックな曲線
grid.newpage()
pushViewport(viewport(xscale=c(-1, 1), yscale=c(-1, 1)))
grid.function(function(t) list(x=cos(t), y=sin(t)),
              range=c(0, 9*pi/5))
# 物理的な abline
grid.newpage()
grid.function(function(x) list(x=x, y=0 + 1*x),
              units="in")
```

---

grid.get　　グリッドのグラフィカルオブジェクトを得る

Description

grob やその子孫を検索する.

Usage

```
grid.get(gPath, strict = FALSE, grep = FALSE, global = FALSE,
         allDevices = FALSE)

grid.gget(..., grep = TRUE, global = TRUE)

getGrob(gTree, gPath, strict = FALSE, grep = FALSE, global = FALSE)
```

Arguments

gTree	gTree オブジェクト.
gPath	gPath オブジェクト. grid.get に対してはこれはディスプレイリスト上の grob を指定する. getGrob に対してはこれは指定された gTree の子孫を指定する.
strict	gPath は正確にマッチされるべきかどうかを指示するブール値.
grep	gPath は正規表現として扱われるべきかどうかを指示するブール値. 値は gPath の要素を跨ってリサイクルされる (例えば c(TRUE, FALSE) は gPath の全ての奇数番目の要素が正規表現として扱われることを意味する).
global	関数は gPath の最初のマッチにだけに影響するか, それとも全てのマッチが影響されるべきかを指示するブール値.
allDevices	全ての開かれたデバイスに対してマッチを探索すべきかどうか, それとも単に現在のデバイスだけか指示するブール値. まだ実装されていない.
...	grid.get に渡される引数.

Details

grid.gget (global の g)は異なった既定値持つ grid.get の便利なラッパである.

Value

grob オブジェクト.

Author(s)

Paul Murrell

See Also

grob, getGrob, addGrob, removeGrob.

Examples

```
grid.xaxis(name="xa")
grid.get("xa")
grid.get(gPath("xa", "ticks"))

grid.draw(gTree(name="gt", children=gList(xaxisGrob(name="axis"))))
grid.get(gPath("gt", "axis", "ticks"))
```

---

grid.grab 現在のグリッドの出力を捕捉する

Description

現在のグリッドのディスプレイリストまたはユーザ固有のコードから生成されたシーンから gTree オブジェクトを作る,

Usage

```
grid.grab(warn = 2, wrap = FALSE, ...)
grid.grabExpr(expr, warn = 2, wrap = FALSE, ...)
```

Arguments

expr	評価されるべき表現式. 典型的にはグリッドの描画関数へのある呼び出し.
warn	発行される警告の量を指定する整数. 0 は警告なしを意味し, 1 は grab が忠実にオリジナルのシーンを表現していないことが確実な場合に警告することを意味する. 2 は grab が忠実にオリジナルのシーンを表現していない可能性がある場合に警告することを意味する.
wrap	出力がどのように捕捉されるかを指示する論理値. もし TRUE ならば, ディスプレイリスト上の各非 grob 要素はそれを grob 中にラップすることで捕捉される.
...	gTree に渡される引数で, 例えば生成される gTree の名前またはクラス, もしくは双方.

Details

グリッドの出力を gTree として捕捉する四種類の方法がある．

出力を捕捉する二つの関数がある：既存の描画を捕捉するには `grid.grab` を使い表現式からの出力(何も描くこと無く)を捕捉するには `grid.grabExpr` を使う．

これらの関数の各々に対して，出力は二つの方法で捕捉できる．一つの方法はより賢明であろうとしディスプレイリスト上の全てのビューポート (ポップされたものを含む)を含む childrenvp スロットを持つ gTree を作り，そしてディスプレイリスト上の全ての grob は新しい gTree の子供になる；各子供はそれらが適当なビューポート中で描画されるように vp スロットに vpPath を持つ．換言すれば gTree はディスプレイリスト上の全ての要素を含むが，少々変更された型式を持つ．

他の方法 `wrap=TRUE` は grob をディスプレイリスト上の全ての要素に対して grob を生成する(そしてこれら全てを gTree の子供とする)．

最初のアプローチはよりコンパクトでエレガントな gTree を作り，これは作業に関してより柔軟であるが，全ての可能なグリッド出力を忠実に複製する保証はない．二番目のアプローチはより直截的であり，そして作業しにくいが，オリジナルの出力を常に忠実に複製すべきである．

Value

gTree オブジェクト．

See Also

gTree

Examples

```
pushViewport(viewport(w=.5, h=.5))
grid.rect()
grid.points(stats::runif(10), stats::runif(10))
popViewport()
grab <- grid.grab()
grid.newpage()
grid.draw(grab)
```

---

grid.grep *Search for grobs*

Description

gPath を与えディスプレイリスト上か与えられた grob 内で全てのマッチする grob を見つける．

Usage

```
grid.grep(path, x = NULL, grobs = TRUE, viewports = FALSE,
          strict = FALSE, grep = FALSE, global = FALSE,
          no.match = character())
```

Arguments

path	gPath.
x	grob か NULL．もし NULL なら，ディスプレイリストが探索される．
grobs	grob を探索すべきかを指示する論理値.
viewports	ビューポートを探索すべきかを指示する論理値.
strict	path は正確にマッチされるべきかどうかを指示するブール値.
grep	path を正規表現として扱うべきか.
global	関数は path の最初のマッチだけに影響するか，それともマッチした全てが影響されるかを指示するブール値.
no.match	マッチするものが見つからなければ返される値.

Value

gPath か，もし global が TRUE ならば gPaths のリスト．マッチするものが無ければ no.match が返される．

See Also

grid.ls()

Examples

```
# "grandparent" と呼ばれる gTree,
# childrenvp vpStack (vp1 中に vp2)を持つ "parent" と呼ばれる子 gTree,
# そして vp vpPath (vp2 の下)を持つ子 grob
sampleGTree <- gTree(name="grandparent",
                     children=gList(gTree(name="parent",
                          children=gList(grob(name="child", vp="vp1::vp2")),
                          childrenvp=vpStack(viewport(name="vp1"),
                                             viewport(name="vp2")))))
# grob を探す
grid.grep("parent", sampleGTree)
grid.grep("parent", sampleGTree, strict=TRUE)
grid.grep("grandparent", sampleGTree, strict=TRUE)
grid.grep("grandparent::parent", sampleGTree)
grid.grep("parent::child", sampleGTree)
grid.grep("[a-z]", sampleGTree, grep=TRUE)
grid.grep("[a-z]", sampleGTree, grep=TRUE, global=TRUE)
# ビューポートを探す
grid.grep("vp1", sampleGTree, viewports=TRUE)
grid.grep("vp2", sampleGTree, viewports=TRUE)
grid.grep("vp", sampleGTree, viewports=TRUE, grep=TRUE)
grid.grep("vp2", sampleGTree, viewports=TRUE, strict=TRUE)
grid.grep("vp1::vp2", sampleGTree, viewports=TRUE)
# 双方を探す
grid.grep("[a-z]", sampleGTree, viewports=TRUE, grep=TRUE, global=TRUE)
```

grid.grill 網目を描く

Description

この関数はグリッドのビューポート内に網目を描く.

Usage

```
grid.grill(h = unit(seq(0.25, 0.75, 0.25), "npc"),
           v = unit(seq(0.25, 0.75, 0.25), "npc"),
           default.units = "npc", gp=gpar(col = "grey"), vp = NULL)
```

Arguments

h	垂直の網目線の水平位置を指示する数値ベクトルまたは単位オブジェクト.
v	水平の網目線の垂直位置を指示する数値ベクトルまたは単位オブジェクト.
default.units	もし h または v が数値ベクトルとしてだけ与えられた時に使われる既定単位を指示する文字列.
gp	クラス gpar のオブジェクトで，典型的には関数 gpar への呼び出しからの出力．これは基本的にはグラフィカルパラメータ設定のリスト.
vp	グリッドのビューポートオブジェクト.

Value

無い.

Author(s)

Paul Murrell

See Also

Grid, viewport.

grid.grob 別名 *"Grob"* として知られるグリッドのグラフィカルオブジェクトを作る

Description

グリッドのグラフィカルオブジェクト(省略形 “grob”)を作る.

grob() と gTree() は基本的な構成子であり， grobTree() と gList() は新しい grob を作るために幾つかの grob を引数に取る.

Usage

```
## Grob Creation:

grob (..., name = NULL, gp = NULL, vp = NULL, cl = NULL)
gTree(..., name = NULL, gp = NULL, vp = NULL, children = NULL,
      childrenvp = NULL, cl = NULL)
grobTree(..., name = NULL, gp = NULL, vp = NULL,
         childrenvp = NULL, cl = NULL)
gList(...)

## Grob Properties:
childNames(gTree)
is.grob(x)
```

Arguments

...	grob と gTree に対しては，グラフィカルオブジェクトの重要な特性を記述する名前付きのスロット．gList と grobTree に対しては一連の grob オブジェクト．
name	grob に対する文字列識別子．ディスプレイリストと別の grob の子供またはどちらかにある grob を見つけるために使われる．
children	"gList" オブジェクト．
childrenvp	viewport オブジェクト(または NULL)．
gp	gpar オブジェクト，典型的には関数 gpar への呼び出しからの出力．基本的にはグラフィカルパラメータ設定のリスト．
vp	viewport オブジェクト(または NULL)．
cl	新しいクラスに対するクラス属性を与える文字列．
gTree	"gTree" オブジェクト．
x	R オブジェクト．

Details

これらの関数は基本的な "grob", "gTree" または "gList" オブジェクト，またはこれらの一つから導かれた新しいクラスを作るのに使うことが出来る．

グリッドのグラフィカルオブジェクト(“grob”)はグラフィカルな事項の記述である．これらの基本的なクラスはグラフィカルオブジェクトの検証，描画そして修正に対する既定の動作を提供する． grob() と gTree() は共に返されるオブジェクトが内部的に一貫性を持つことをチェックするために関数 validDetails を呼び出す．

"gTree" は他の grob を子供として持つことが出来る；gTree が描画される時，それはその子供全てを描画する．その子供を描画する前に， gTree はその childrenvp スロットをプッシュし，そしてそれから vpPath を使い子供が childrenvp 内のそれらの位置を特定できるように逆に巡回する(upViewport を呼び出す)．

grob の名前は一般にユニークである必要はないが，全ての gTree の全ての子供は異なった名前を持たなければならない． grob 名は任意の文字列で良いが， grob 名に gPath 分離記号 (現在 ::)を使うことは勧められない．

関数 childNames は gTree の子供である grob の名前を返す．

全てのグリッドのプリミティブ (grid.lines, grid.rect, ...) そしてある種の高水準のグリッド成分 (例えば grid.xaxis と grid.yaxis) はこれらのクラスから導かれる．

grobTree は gTree の唯一の成分が grob の時 (従って全ての名前無しの引数は gTree の子供になる) の gTree の単なる便利なラッパである.

関数 grid.grob は廃止された.

Value

クラス "grob" の R オブジェクト， **gr**aphical **ob**ject.

Author(s)

Paul Murrell

See Also

grid.draw, grid.edit, grid.get.

---

grid.layout　　グリッドのレイアウトを作る

Description

この関数はグリッドのレイアウトを返し，これは矩形領域の分割を記述する.

Usage

```
grid.layout(nrow = 1, ncol = 1,
        widths = unit(rep_len(1, ncol), "null"),
        heights = unit(rep_len(1, nrow), "null"),
        default.units = "null", respect = FALSE,
        just="centre")
```

Arguments

nrow	レイアウト中の行数を指定する整数.
ncol	レイアウト中の列数を指定する整数.
widths	レイアウト中の列の幅を記述する数値ベクトル化単位オブジェクト.
heights	レイアウト中の行の幅を記述する数値ベクトル化単位オブジェクト.
default.units	もし widths か heights が数値ベクトルとしてだけ与えられるならば使われる既定の単位を指示する文字列. .
respect	論理値か数値行列. もし論理値なら，これは行の高さと列の幅が互いを尊重するかどうかを指示する. もし行列ならば，ゼロでない値は対応する行と列が尊重すべきことを指示する(下の例を見よ).
just	文字列または数値ベクトルでもしそれがその親ビューポートと同じサイズでなければレイアウトをどのように位置揃えするかを指示する文字列か数値ベクトル. もし二つの値があれば，最初の値は水平方向の位置揃え，二番目の値は垂直方向の位置揃えを指定する. 可能な文字列値は： "left", "right", "centre", "center", "bottom", そして "top". 数値に対しては， 0 は右揃えを，そして 1 は右揃えを意味する. この文脈では，例えば "left" は親のビューポートの左隅の最左翼のレイアウト列の左隅で位置揃えすることを意味する.

Details

レイアウトの widths と heights に対して与えられた単位オブジェクトはレイアウトに対してだけ意味を持つ特殊な units を使うかもしれない．これは "null" 単位で，どれだけの利用可能な幅/高さの相対的な割合を列/行が占めるかを指示する．レイアウト中の相対的な幅と高さのより優れた記述に対しては参考文献を見よ．

Value

グリッドのレイアウトオブジェクト．

警告

この関数は基本の R のグラフィックス関数 layout と混同されるべきではない．特に layout をグリッドのグラフィックスと一緒に使わない． layout に対するドキュメントはある種の有用な情報を提供するかもしれず，そしてこの関数は比較可能な状況では同一の挙動をすべきである． grid.layout 関数は行の高さと列の幅に対するより広い範囲の単位を指定する能力を 加えており，入れ子のレイアウトを許す(viewport を見よ)．

Author(s)

Paul Murrell

References

Murrell, P. R. (1999), Layouts: A Mechanism for Arranging Plots on a Page, *Journal of Computational and Graphical Statistics*, **8**, 121–134.

See Also

Grid, grid.show.layout, viewport, layout

Examples

```
## 様々なレイアウト(あるものは少々途中で曲がっている ...)
layout.torture()
## レイアウトの位置揃えのデモ
grid.newpage()
testlay <- function(just="centre") {
  pushViewport(viewport(layout=grid.layout(1, 1, widths=unit(1, "inches"),
                        heights=unit(0.25, "npc"),
                        just=just)))
  pushViewport(viewport(layout.pos.col=1, layout.pos.row=1))
  grid.rect()
  grid.text(paste(just, collapse="-"))
  popViewport(2)
}
testlay()
testlay(c("left", "top"))
testlay(c("right", "top"))
testlay(c("right", "bottom"))
testlay(c("left", "bottom"))
testlay(c("left"))
testlay(c("right"))
testlay(c("bottom"))
testlay(c("top"))
```

grid.lines　　グリッドのビューポート中に線を描く

Description

これらの関数はグリッドのビューポート中に線を描く.

Usage

```
grid.lines(x = unit(c(0, 1), "npc"),
          y = unit(c(0, 1), "npc"),
          default.units = "npc",
          arrow = NULL, name = NULL,
          gp=gpar(), draw = TRUE, vp = NULL)
linesGrob(x = unit(c(0, 1), "npc"),
          y = unit(c(0, 1), "npc"),
          default.units = "npc",
          arrow = NULL, name = NULL,
          gp=gpar(), vp = NULL)
grid.polyline(...)
polylineGrob(x = unit(c(0, 1), "npc"),
             y = unit(c(0, 1), "npc"),
             id=NULL, id.lengths=NULL,
             default.units = "npc",
             arrow = NULL, name = NULL,
             gp=gpar(), vp = NULL)
```

Arguments

x	x 値を指定する数値ベクトルか単位オブジェクト.
y	y 値を指定する数値ベクトルか単位オブジェクト.
default.units	もし x か y が数値ベクトルとしてだけ与えられる時に使われる既定の単位を指定する文字列.
arrow	arrow 関数が作成するような線のどちらかの端に置かれる矢尻を記述するリスト.
name	文字列の識別子.
gp	クラス gpar のオブジェクトで，典型的には関数 gpar への呼び出しからの出力．これは基本的にはグラフィカルパラメータ設定のリスト.
draw	グラフィックス出力を生成するかどうかを指示する論理値.
vp	グリッドのビューポートオブジェクト(または NULL).
id	x と y 中の位置を複数の線に分離するために使われれる数値ベクトル. 同じ id を持つ全ての位置は同じ線に属する.
id.lengths	x と y 中の位置を複数の線に分離するために使われれる数値ベクトル. 別個の線を構成する引き続く位置のブロックを指定する.
...	polylineGrob へ渡される引数.

Details

最初の二つの関数は線の grob (線を記述するグラフィカルオブジェクト)を作り，grid.lines は線を描く(もし draw が TRUE なら).

次の二つの関数は複数線の grob を作り描く，これは丁度線の grob のようであるが，複数の異なった線を描くことが出来る.

Value

線の grob か複数線の grob. grid.lines は線の grob を不可視で返す.

Author(s)

Paul Murrell

See Also

Grid, viewport, arrow

Examples

```
grid.lines()
# id を使う(注意：位置は引き続くブロック中にない)
grid.newpage()
grid.polyline(x=c((0:4)/10, rep(.5, 5), (10:6)/10, rep(.5, 5)),
              y=c(rep(.5, 5), (10:6/10), rep(.5, 5), (0:4)/10),
              id=rep(1:5, 4),
              gp=gpar(col=1:5, lwd=3))
# id.lengths を使う
grid.newpage()
grid.polyline(x=outer(c(0, .5, 1, .5), 5:1/5),
              y=outer(c(.5, 1, .5, 0), 5:1/5),
              id.lengths=rep(4, 5),
              gp=gpar(col=1:5, lwd=3))
```

---

grid.locator マウスのクリックを捕捉する

Description

ユーザが現在のグラフィックスデバイス内でマウスを一回クリックし現在のビューポート内の指定座標でのマウスクリック位置を返すことを許す.

Usage

```
grid.locator(unit = "native")
```

Arguments

unit	マウスのクリック位置を返す座標系．適正な座標系については unit 関数を見よ.

Details

この関数は形式的であり (graphics パッケージの関数 locator と同様に) 従ってコマンドラインとグラフィックス描画はユーザが現在のデバイス内でマウスをクリックするまでブロックされる.

Value

現在のビューポート内の指定座標でのマウスがクリックされた位置を表現する単位オブジェクト.

もしユーザがマウスボタン 1 をクリックしなければ，関数は(不可視で) NULL を返す.

Author(s)

Paul Murrell

See Also

パッケージ **graphics** 中の viewport, unit, locator，そして応用についてはパッケージ **lattice** 中の trellis.focus と panel.identify.

Examples

```
if (interactive()) {
  ## より洗練された unit as.character メソッドを書く必要
  unittrim <- function(unit) {
    sub("^([0-9]+|[0-9]+[.][0-9])[0-9]*", "\\1", as.character(unit))
  }
  do.click <- function(unit) {
    click.locn <- grid.locator(unit)
    grid.segments(unit.c(click.locn$x, unit(0, "npc")),
                  unit.c(unit(0, "npc"), click.locn$y),
                  click.locn$x, click.locn$y,
                  gp=gpar(lty="dashed", col="grey"))
    grid.points(click.locn$x, click.locn$y, pch=16, size=unit(1, "mm"))
    clickx <- unittrim(click.locn$x)
    clicky <- unittrim(click.locn$y)
    grid.text(paste0("(", clickx, ", ", clicky, ")"),
              click.locn$x + unit(2, "mm"), click.locn$y,
              just="left")
  }
  do.click("inches")
  pushViewport(viewport(width=0.5, height=0.5,
                        xscale=c(0, 100), yscale=c(0, 10)))
  grid.rect()
  grid.xaxis()
  grid.yaxis()
  do.click("native")
  popViewport()
}
```

grid.ls *grob* かビューポートの名前をリストする

Description

grob やビューポートの名前のりストを返す.

これは grob (gTrees を含む)とビューポート(vpTree を含む)に対するメソッドを含む総称的関数である.

Usage

```
grid.ls(x=NULL, grobs=TRUE, viewports=FALSE, fullNames=FALSE,
        recursive=TRUE, print=TRUE, flatten=TRUE, ...)

nestedListing(x, gindent="  ", vpindent=gindent)
pathListing(x, gvpSep=" | ", gAlign=TRUE)
grobPathListing(x, ...)
```

Arguments

x	grob かビューポートか NULL. もし NULL なら，現在のグリッドディスプレイがリストされる. プリント関数に対しては，これは grid.ls への呼び出しの結果であるべきである.
grobs	grob をリストするかどうかを示す論理値.
viewports	ビューポートをリストするかどうかを示す論理値.
fullNames	オブジェクト名をオブジェクトのタイプに関する情報で装飾するかどうかを指示する論理値.
recursive	再帰的な構造がそれらの子供もリストすべきかどうかを指示する論理値.
print	リスティングかリスティングをプリントする関数をプリントするかどうかを指示する論理値.
flatten	リスティングをフラット化すべきかどうかを指示する論理値. さもなければより複雑な階層的オブジェクトが作られる.
gindent	grob に対する出力中の入れ子を表示するために使われるインデント量.
vpindent	ビューポートに対する出力中の入れ子を表示するために使われるインデント量.
gvpSep	ビューポートのパスを grob のパスから分離するのに使われる文字列.
gAlign	全ての grob パスの左端を揃えるかどうかを指示する論理値.
...	print 関数に渡される引数.

Details

もし引数 x が NULL ならば，グリッドのディスプレイリストの現在の内容がリストされる(ビューポートと grob の双方)．換言すれば，現在のシーンを表現する全てのオブジェクトがリストされる．

換言すれば x は grob かビューポートで無ければならない．

この関数の既定動作は現在のシーン中の grob に関する情報をプリントすることである．シーン中のビューポートに関する情報を加えることも又可能である．既定ではリスティングは再帰的であり， gTree の全ての子供と全ての入れ子のビューポートが報告される．

情報の書式は print 引数で制御することが出来，整形を実行する関数を与えることが出来る． nestedListing 関数は grob かビューポート毎に一行を作り，入れ子を表示するためにインデントが使われる． pathListing 関数は grob かビューポート毎に一行を作り， grob かビューポートのパスと grob のパスが入れ子を示すために使われる．grobPathListing は grob に対する行だけを示す簡単な派生物である．ユーザは新しい関数を定義できる．

Value

この関数の結果は "gridFlatListing" オブジェクト (もし flatten が TRUE なら)か "gridListing" オブジェクトである．

前者は単純(フラット)なベクトルのリストである．これは例えば grob やビューポート名のリストを使ったプログラム，またはリスティングのための新しいディスプレイ関数を書くのに便利である．

後者はより複雑な階層的オブジェクト(リストのリスト)であるが，より詳細な情報は含まないのでより高度なカスタム化のために有用かもしれない．

Author(s)

Paul Murrell

See Also

grob viewport

Examples

```
# "parent" と呼ばれる gTree と childrenvp vpTree (vp1 中に vp2)
# そして "child" と呼ばれる grob で vp vpPath (vp2 の下方)を持つ
sampleGTree <- gTree(name="parent",
                     children=gList(grob(name="child", vp="vp1::vp2")),
                     childrenvp=vpTree(parent=viewport(name="vp1"),
                                       children=vpList(viewport(name="vp2"))))
grid.ls(sampleGTree)
# viewports も示す
grid.ls(sampleGTree, view=TRUE)
# ビューポートだけを示す
grid.ls(sampleGTree, view=TRUE, grob=FALSE)
# 別の表示
# 入れ子のリスティング，カスタムインデント
grid.ls(sampleGTree, view=TRUE, print=nestedListing, gindent="--")
# パスのリスティング
grid.ls(sampleGTree, view=TRUE, print=pathListing)
# パスのリスティング，grob の整列なし
```

```
grid.ls(sampleGTree, view=TRUE, print=pathListing, gAlign=FALSE)
# grob パスのリスティング
grid.ls(sampleGTree, view=TRUE, print=grobPathListing)
# パスのリスティング, grob だけ
grid.ls(sampleGTree, print=pathListing)
# パスのリスティング, ビューポートだけ
grid.ls(sampleGTree, view=TRUE, grob=FALSE, print=pathListing)
# 生のフラットなリスティング
str(grid.ls(sampleGTree, view=TRUE, print=FALSE))
```

---

grid.move.to　　指定位置に移動するか描画する

Description

グリッドは現在位置の概念を持つ．これらの関数はその位置を設定する．

Usage

```
grid.move.to(x = 0, y = 0, default.units = "npc", name = NULL,
             draw = TRUE, vp = NULL)

moveToGrob(x = 0, y = 0, default.units = "npc", name = NULL,
           vp = NULL)

grid.line.to(x = 1, y = 1, default.units = "npc",
             arrow = NULL, name = NULL,
             gp = gpar(), draw = TRUE, vp = NULL)

lineToGrob(x = 1, y = 1, default.units = "npc", arrow = NULL,
           name = NULL, gp = gpar(), vp = NULL)
```

Arguments

x	x 値を指定する数値ベクトルか単位オブジェクト．
y	y 値を指定する数値ベクトルか単位オブジェクト．
default.units	x か y が数値ベクトルとしてだけ与えられた時使う既定の単位を指定する文字列．
arrow	線のどちらかの端に置かれる矢印を記述するリストで， arrow 関数が作るようなもの．
name	文字列識別子．
draw	グラフィックス出力を作成するかどうかを指示する論理値．
gp	クラス gpar のオブジェクトで，典型的には関数 gpar の呼び出しからの出力．これは基本的にはグラフィカルパラメータ設定のリストである．
vp	グリッドのビューポートオブジェクト(または NULL)．

Details

どちらの関数も (move-to/line-to を記述するグラフィカルオブジェクト)を作るが，grid.move.to/line.to() だけが move.to/line.to を描画する (そして draw が TRUE の時だけ).

Value

move.to grob か line.to grob. grid.move.to/line.to() は値を不可視で返す.

Author(s)

Paul Murrell

See Also

Grid, viewport, arrow

Examples

```
grid.newpage()
grid.move.to(0.5, 0.5)
grid.line.to(1, 1)
grid.line.to(0.5, 0)
pushViewport(viewport(x=0, y=0, w=0.25, h=0.25, just=c("left", "bottom")))
grid.rect()
grid.grill()
grid.line.to(0.5, 0.5)
popViewport()
```

---

grid.newpage　グリッドデバイス上の新しいページに移動する

Description

この関数は現在のデバイスを消したり新しいページに移動する.

Usage

```
grid.newpage(recording = TRUE)
```

Arguments

recording	新ページ操作をグリッドのディスプレイリスト上に保存すべきかを指示する論理値.

Details

新しいページは塗りつぶし色で塗られ (gpar("fill")), それはしばしば透明である. キャンバス色のデバイス(スクリーン上のデバイス X11, windows そして quartz)に対しては，ページは最初キャンバス色でそれから背景色で塗られる.

"before.grid.newpage" と "grid.newpage" と呼ばれる二つのフックがある (setHook を見よ). 後者は新しいページの注釈のテストコード中で使われる. フック関数は引数無しで呼び出される. (もし値が文字列なら，get は **grid** の名前空間の中からそれに対して呼び出される.)

Value

無い.

Author(s)

Paul Murrell

See Also

Grid

---

grid.null　　ナルグラフィカルオブジェクト

Description

これらの関数は NULL グラフィカルオブジェクトを作る. これは幅と高さがゼロで何も描かない. これは他の描画の置き場所または不可視の参照点として使うことが出来る.

Usage

```
nullGrob(x = unit(0.5, "npc"), y = unit(0.5, "npc"),
         default.units = "npc",
         name = NULL, vp = NULL)
grid.null(...)
```

Arguments

x	x 位置を指定する数値ベクトルまたは単位オブジェクト.
y	yx 位置を指定する数値ベクトルまたは単位オブジェクト.
default.units	もし x, y, width または height が数値ベクトルとしてだけ与えられた時使われる既定単位を指定する文字列.
name	文字列識別子.
vp	グリッドのビューポート(または NULL).
...	nullGrob() に渡される引数.

Value

ナル grob.

Author(s)

Paul Murrell

See Also

Grid, viewport

Examples

```
grid.newpage()
grid.null(name="ref")
grid.rect(height=grobHeight("ref"))
grid.segments(0, 0, grobX("ref", 0), grobY("ref", 0))
```

---

grid.pack　フレーム中のオブジェクトをパックする

Description

これらの関数は， grid.frame と frameGrob と共に，グラフィカルイメージを構成する GUI 構成子風のインタフェイスの一部である．アイデアは grid.frame または frameGrob を使いフレームを作り，それからこれらの関数を使いオブジェクトをフレーム中にパックするということである．

Usage

```
grid.pack(gPath, grob, redraw = TRUE, side = NULL,
          row = NULL, row.before = NULL, row.after = NULL,
          col = NULL, col.before = NULL, col.after = NULL,
          width = NULL, height = NULL,
          force.width = FALSE, force.height = FALSE, border = NULL,
          dynamic = FALSE)

packGrob(frame, grob, side = NULL,
          row = NULL, row.before = NULL, row.after = NULL,
          col = NULL, col.before = NULL, col.after = NULL,
          width = NULL, height = NULL,
          force.width = FALSE, force.height = FALSE, border = NULL,
          dynamic = FALSE)
```

Arguments

gPath	gPath オブジェクトで，ディスプレイリスト上のフレームを指定する．
frame	クラス frame のオブジェクトで，典型的には grid.frame への呼び出しからの出力．
grob	クラス grob のオブジェクト．パックされるオブジェクト．
redraw	出力は更新されるべきかどうかを指示するブール値．
side	"left", "top", "right", "bottom" の一つでどちら側でオブジェクトをパックするかを指示する．
row	どの行にオブジェクトを加えるか． 1 とフレーム中の現在の行数プラス1の間でなければならず，オブジェクトが全ての行を占めれば NULL．
row.before	この行の丁度一つ前の新しい行にオブジェクトを加える．
row.after	この行の丁度一つ後の新しい行にオブジェクトを加える．
col	どの列にオブジェクトを加えるか． 1 とフレーム中の現在の列数プラス1の間でなければならず，オブジェクトが全ての列を占めれば NULL．
col.before	この列の丁度一つ前の新しい列にオブジェクトを加える．

col.after	この列の丁度一つ後の新しい列にオブジェクトを加える.
width	オブジェクトが加えられる列の幅を指定する (幅をオブジェクトから取るのを許す代わりに).
height	オブジェクトが加えられる行の高さを指定する (高さをオブジェクトから取るのを許す代わりに).
force.width	grob がその中にパックされる行の幅は grid.pack への呼び出し中で指定された幅か，幅と既に存在する幅の最大値のどちらかであるべきことを指示する論理値.
force.height	grob がその中にパックされる列の高さは grid.pack への呼び出し中で指定された高さ，その高さと既に存在する高さの最大値のどちらかであるべきことを指示する論理値.
border	オブジェクトの周囲を指示する長さ4の unit オブジェクト.
dynamic	もし幅/高さがパックされている grob から取られていれば，このブール値フラグは grobの幅/高さの単位が grob を直接参照するか，それとも grob への gPath を使うか，を指定する．後者の場合，grob への変更は幅/高さの再計算を惹き起こす.

Details

packGrob は与えられたフレーム grob を修正し修正された grob を返す.

grid.pack はディスプレイリスト上のフレーム grob を破壊的に変更する (そしてもし redraw が TRUE ならばディスプレイリストを再描画する).

これらは非常に柔軟な関数(であるはず)である．既にフレーム中にあるオブジェクトに相対的に新しいオブジェクトをどこに付け加えるかを指定する多くの異なった方法がある．この関数は指定が自己矛盾していないかをチェックする.

オブジェクトに付け加えられる行/列の幅/高さは width/height が指定されない限りオブジェクト自体から取られることを注意する.

Value

packGrob はフレーム grob を返すが， grid.pack は NULL を返す．.

Author(s)

Paul Murrell

See Also

grid.frame, grid.place, grid.edit そして gPath.

---

grid.path　　パスを描く

Description

これらの関数はパスを作り描く．最後の点は自動的に最初の点と結ばれる.

Usage

```
pathGrob(x, y,
         id=NULL, id.lengths=NULL,
         rule="winding",
         default.units="npc",
         name=NULL, gp=gpar(), vp=NULL)
grid.path(...)
```

Arguments

x	x 位置を指定する数値ベクトルまたは単位オブジェクト.
y	y 位置を指定する数値ベクトルまたは単位オブジェクト.
id	x と y 中の位置を副パスに分離するために使われる数値ベクトル. 同じ id を持つ全ての位置は同じ副パスに属する.
id.lengths	x と y 中の位置を副パスに分離するために使われる数値ベクトル. 個別の副パスを構成する位置の連続ブロックを指定する.
rule	塗りつぶし規則を指定する文字列： "winding" または "evenodd" のどちらか.
default.units	もし x か y が数値ベクトルとしてだけ与えられた時に使われる既定単位を指示する文字列.
name	文字列識別子.
gp	クラス gpar のオブジェクトで，典型的には関数 gpar への呼び出しからの出力. これは基本的にはグラフィカルパラメータ設定のリストである.
vp	グリッドのビューポートオブジェクト(または NULL).
...	pathGrob() に渡される引数.

Details

どちらの関数もパス grob (パスを表すグラフィカルオブジェクト)を作るが， grid.path だけがパスを描く(そして draw が TRUE の時だけ).

パスは多角形に似ているが，前者は穴を含むことが出来，塗りつぶし規則で解釈される；これらはパスの境界がそれをそれぞれ奇数もしくはゼロで無い回数囲めば領域を振りつぶす.

全てのデバイスがこの関数をサポートするわけではない：例えば xfig と pictex はしない.

Value

grob オブジェクト.

Author(s)

Paul Murrell

See Also

Grid, viewport

Examples

```
pathSample <- function(x, y, rule, gp = gpar()) {
    if (is.na(rule))
        grid.path(x, y, id = rep(1:2, each = 4), gp = gp)
    else
        grid.path(x, y, id = rep(1:2, each = 4), rule = rule, gp = gp)
    if (!is.na(rule))
        grid.text(paste("Rule:", rule), y = 0, just = "bottom")
}

pathTriplet <- function(x, y, title) {
    pushViewport(viewport(height = 0.9, layout = grid.layout(1, 3),
                          gp = gpar(cex = .7)))
    grid.rect(y = 1, height = unit(1, "char"), just = "top",
              gp = gpar(col = NA, fill = "grey"))
    grid.text(title, y = 1, just = "top")
    pushViewport(viewport(layout.pos.col = 1))
    pathSample(x, y, rule = "winding",
               gp = gpar(fill = "grey"))
    popViewport()
    pushViewport(viewport(layout.pos.col = 2))
    pathSample(x, y, rule = "evenodd",
               gp = gpar(fill = "grey"))
    popViewport()
    pushViewport(viewport(layout.pos.col = 3))
    pathSample(x, y, rule = NA)
    popViewport()
    popViewport()
}

pathTest <- function() {
    grid.newpage()
    pushViewport(viewport(layout = grid.layout(5, 1)))
    pushViewport(viewport(layout.pos.row = 1))
    pathTriplet(c(.1, .1, .9, .9, .2, .2, .8, .8),
                c(.1, .9, .9, .1, .2, .8, .8, .2),
                "Nested rectangles, both clockwise")
    popViewport()
    pushViewport(viewport(layout.pos.row = 2))
    pathTriplet(c(.1, .1, .9, .9, .2, .8, .8, .2),
                c(.1, .9, .9, .1, .2, .2, .8, .8),
                "Nested rectangles, outer clockwise, inner anti-clockwise")
    popViewport()
    pushViewport(viewport(layout.pos.row = 3))
    pathTriplet(c(.1, .1, .4, .4, .6, .9, .9, .6),
                c(.1, .4, .4, .1, .6, .6, .9, .9),
                "Disjoint rectangles")
    popViewport()
    pushViewport(viewport(layout.pos.row = 4))
    pathTriplet(c(.1, .1, .6, .6, .4, .4, .9, .9),
                c(.1, .6, .6, .1, .4, .9, .9, .4),
                "Overlapping rectangles, both clockwise")
    popViewport()
    pushViewport(viewport(layout.pos.row = 5))
    pathTriplet(c(.1, .1, .6, .6, .4, .9, .9, .4),
                c(.1, .6, .6, .1, .4, .4, .9, .9),
```

```
                "Overlapping rectangles, one clockwise, other anti-clockwise")
    popViewport()
    popViewport()
}

pathTest()
```

---

grid.place　　オブジェクトをフレーム内に置く

Description

これらの関数は grid.pack() と packGrob の簡単な(そしてより速い) 代替物を与える．これらはオブジェクトを既存のフレームレイアウトの行と列の中に置くために使うことが出来る．これらは新しい行と列を付け加える機能を提供せず，行と列の高さと幅に影響しない．

Usage

```
grid.place(gPath, grob, row = 1, col = 1, redraw = TRUE)
placeGrob(frame, grob, row = NULL, col = NULL)
```

Arguments

gPath	gPath オブジェクトで，ディスプレイリスト上のフレームを指定する．
frame	クラス class frame のオブジェクトで，典型的には grid.frame への呼び出しからの出力．
grob	クラス grob のオブジェクト．置かれるオブジェクト．
row	どの行にオブジェクト加えるか． 1 と現在のフレーム中の行数の間でなければならない．
col	どの列にオブジェクト加えるか． 1 と現在のフレーム中の列数の間でなければならない．
redraw	出力が更新されるべきかどうかを指示するブール値．

Details

placeGrob は与えられたフレーム grob を修正し，そして修正されたフレーム grob を返す．

grid.place はディスプレイリスト上のフレーム grob を破壊的に修正する (そしてもし redraw が TRUE ならディスプレイリストを再描画する)．

Value

placeGrob はフレーム grob を返すが， grid.place は NULL を返す．

Author(s)

Paul Murrell

See Also

grid.frame, grid.pack, grid.edit, and gPath.

grid.plot.and.legend 簡単なプロット凡例のデモ

Description

この関数はグリッドを使い始めからどのように基本的なプロットと凡例を描くことが出来るかの簡単なデモに対するラッパである.

Usage

```
grid.plot.and.legend()
```

Author(s)

Paul Murrell

Examples

```
grid.plot.and.legend()
```

grid.points データシンボルを描く

Description

これらの関数はデータシンボルを作り描く.

Usage

```
grid.points(x = stats::runif(10),
            y = stats::runif(10),
            pch = 1, size = unit(1, "char"),
            default.units = "native", name = NULL,
            gp = gpar(), draw = TRUE, vp = NULL)
pointsGrob(x = stats::runif(10),
           y = stats::runif(10),
           pch = 1, size = unit(1, "char"),
           default.units = "native", name = NULL,
           gp = gpar(), vp = NULL)
```

Arguments

x	x 値を指定する数値ベクトルか単位オブジェクト.
y	y 値を指定する数値ベクトルか単位オブジェクト.
pch	どのような種類のプロットシンボルを使うかを指示する数値または文字列ベクトル. これらの値の解釈については points を見よ. そして下の fill に注意.
size	プロットシンボルのサイズを指定する単位オブジェクト.

default.units もし x か y が数値ベクトルとしてだけ与えられた時に使う既定の単位を指示する文字列.

name 文字列識別子.

gp クラス gpar の R オブジェクトで，典型的には関数 gpar への呼び出しからの出力．これは基本的にはグラフィカルパラメータ設定のリストである；fill (パッケージ **graphics** の points 中における bg では無く) は "塗りつぶし"に使われる，つまりシンボルの背景を pch = 21:25 で色を塗る.

draw グラフィックス出力を作るべきかどうかを指示する論理値.

vp グリッドのビューポートオブジェクト(または NULL).

Details

どちらの関数も点の grob (点を記述するグラフィカルオブジェクト) を作るが，grid.points だけが点を描く (そして draw が TRUE の時だけ).

Value

点の grob. grid.points は値を不可視で返す.

Author(s)

Paul Murrell

See Also

Grid, viewport

---

grid.polygon 多角形を描く

Description

これらの関数は多角形を作り描く．最後の点は最初の点と自動的に結ばれる.

Usage

```
grid.polygon(x=c(0, 0.5, 1, 0.5), y=c(0.5, 1, 0.5, 0),
             id=NULL, id.lengths=NULL,
             default.units="npc", name=NULL,
             gp=gpar(), draw=TRUE, vp=NULL)
polygonGrob(x=c(0, 0.5, 1, 0.5), y=c(0.5, 1, 0.5, 0),
             id=NULL, id.lengths=NULL,
             default.units="npc", name=NULL,
             gp=gpar(), vp=NULL)
```

Arguments

x	x 位置を指定する数値ベクトルか単位オブジェクト.
y	y 位置を指定する数値ベクトルか単位オブジェクト.
id	x と y 中の位置を多角形に分離するために使われる数値ベクトル. 同じ id の位置は同じ多角形に属する.
id.lengths	x と y 中の位置を多角形に分離するために使われる数値ベクトル. 別個の多角形を構成する位置の連続するブロックを指定する. id の位置は同じ多角形に属する.
default.units	もし x, y, width そして height が数値ベクトルとしてのみ与えられている時使われる既定の単位を指示する文字列.
name	文字列識別子.
gp	クラス gpar のオブジェクトで, 典型的には関数 gpar への呼び出しからの出力. これは基本的にはグラフィカルパラメータ設定のリストである.
draw	グラフィックス出力を生成するかどうかを指示する論理値.
vp	グリッドのビューポートオブジェクト(または NULL).

Details

どちらの関数も多角形 grob (多角形を記述するグラフィカルオブジェクト)を作るが, grid.polygon だけが多角形を描く (そして draw が TRUE の時だけ).

Value

grob オブジェクト.

Author(s)

Paul Murrell

See Also

Grid, viewport

Examples

```
grid.polygon()
# id を使う：(注意：位置は引き続くブロック中にない)
grid.newpage()
grid.polygon(x=c((0:4)/10, rep(.5, 5), (10:6)/10, rep(.5, 5)),
             y=c(rep(.5, 5), (10:6/10), rep(.5, 5), (0:4)/10),
             id=rep(1:5, 4),
             gp=gpar(fill=1:5))
# id.lengths を使う
grid.newpage()
grid.polygon(x=outer(c(0, .5, 1, .5), 5:1/5),
             y=outer(c(.5, 1, .5, 0), 5:1/5),
             id.lengths=rep(4, 5),
             gp=gpar(fill=1:5))
```

grid.pretty 分点の思慮あるセットを生成する

Description

与えられた範囲内で分点の見栄えの良いセットを生成する.

Usage

```
grid.pretty(range)
```

Arguments

range	数値ベクトル

Value

分点の数値ベクトル.

Author(s)

Paul Murrell

grid.raster ラスターオブジェクトを作る

Description

与えられた位置，サイズそして方角のラスターオブジェクト(ビットマップイメージ) を作る.

Usage

```
grid.raster(image,
            x = unit(0.5, "npc"), y = unit(0.5, "npc"),
            width = NULL, height = NULL,
            just = "centre", hjust = NULL, vjust = NULL,
            interpolate = TRUE, default.units = "npc",
            name = NULL, gp = gpar(), vp = NULL)

rasterGrob(image,
           x = unit(0.5, "npc"), y = unit(0.5, "npc"),
           width = NULL, height = NULL,
           just = "centre", hjust = NULL, vjust = NULL,
           interpolate = TRUE, default.units = "npc",
           name = NULL, gp = gpar(), vp = NULL)
```

Arguments

image	ラスターオブジェクトに強制変換される任意の R オブジェクト.
x	x 位置を指定する数値ベクトルか単位オブジェクト.
y	y 位置を指定する数値ベクトルか単位オブジェクト.
width	幅を指定する数値ベクトルか単位オブジェクト.
height	高さを指定する数値ベクトルか単位オブジェクト.
just	矩形のその (x, y) 位置に対する位置揃え. もし二つの値があると, 最初の値は水平方向の位置揃えで二番目の値は垂直方向の位置揃えを指定する. 可能な文字列値は: "left", "right", "centre", "center", "bottom", そして "top". 数値の値に対しては 0 は左揃え, 1 は右揃えを意味する.
hjust	水平方向の位置揃えを指定する数値ベクトル. もし指定されると just の設定を上書きする.
vjust	垂直方向の位置揃えを指定する数値ベクトル. もし指定されると just の設定を上書きする.
default.units	もし x, y, width または height が数値ベクトルとしてだけ与えられた時に使われる既定の単位を指示する文字列.
name	文字列識別子.
gp	クラス gpar のオブジェクトで, 典型的には関数 gpar への呼び出しからの出力. これは基本的にグラフィカルパラメータ設定のリストである.
vp	グリッドのビューポートオブジェクト(または NULL).
interpolate	イメージを線形に補間するかどうかを指示する論理値 (別法は最近接近傍補間を使うことであるがよりゴツゴツした結果を与える).

Details

width も height もどちらも指定される必要がなく, その場合画像のアスペクト比は保存される. もし width と height の双方が指定されると, 画像は歪められる可能性が高い.

全てのグラフィックスデバイスがラスターイメージを作る機能を持つわけではなく, ある種のそれは回転イメージを作ることが出来ないかもしれない (つまり, もしラスターオブジェクトが回転されたビューポート中で描かれた場合). rasterImage の下のコメントも見よ.

alpha を含む gp 中の全てのグラフィカルパラメータ設定は無視される.

Value

rastergrob grob.

Author(s)

Paul Murrell

See Also

as.raster.

サポートされているかどうかを見るには dev.capabilities.

Examples

```
redGradient <- matrix(hcl(0, 80, seq(50, 80, 10)),
                      nrow=4, ncol=5)
# 補間
grid.newpage()
grid.raster(redGradient)
# むらのある
grid.newpage()
grid.raster(redGradient, interpolate=FALSE)
# むらがあり伸びた
grid.newpage()
grid.raster(redGradient, interpolate=FALSE, height=unit(1, "npc"))
# 同じラスターを何度も描く
grid.newpage()
grid.raster(0, x=1:3/4, y=1:3/4, w=.1, interp=FALSE)
```

---

grid.record 計算と描画をカプセル化する

Description

計算と計算に依存する描画を含む表現式を評価し，シーンが再描画される時(例えばデバイスのサイズ変更や編集) に計算と描画が再実行されるようにする.

エキスパートの使用だけを意図している.

Usage

```
recordGrob(expr, list, name=NULL, gp=NULL, vp=NULL)
grid.record(expr, list, name=NULL, gp=NULL, vp=NULL)
```

Arguments

expr	モード expression または call または未評価の表現式.
list	その中で expr が評価される環境を定義するリスト.
name	文字列識別子.
gp	クラス gpar のオブジェクトで，典型的には関数 gpar への呼び出しからの出力．これは基本的にはグラフィカルパラメータ設定のリストである.
vp	グリッドのビューポートオブジェクト(または NULL).

Details

特別なクラス "recordedGrob" の grob が作られる (そして grid.record の場合は描画される)．このクラスに対する drawDetails メソッドはリストを評価環境として表現式を評価する (そしてグリッドの名前空間がその環境の親になる).

Note

この関数は関数 recordGraphics の代わりに使われるべきである； recordGraphics の使用に関するものすごい警告は確実にここでも適用される.

Author(s)

Paul Murrell

See Also

recordGraphics

Examples

```
grid.record({
              w <- convertWidth(unit(1, "inches"), "npc")
              grid.rect(width=w)
            },
            list())
```

---

grid.rect 矩形を描く

Description

これらの関数は矩形を作り描画する.

Usage

```
grid.rect(x = unit(0.5, "npc"), y = unit(0.5, "npc"),
          width = unit(1, "npc"), height = unit(1, "npc"),
          just = "centre", hjust = NULL, vjust = NULL,
          default.units = "npc", name = NULL,
          gp=gpar(), draw = TRUE, vp = NULL)
rectGrob(x = unit(0.5, "npc"), y = unit(0.5, "npc"),
         width = unit(1, "npc"), height = unit(1, "npc"),
         just = "centre", hjust = NULL, vjust = NULL,
         default.units = "npc", name = NULL,
         gp=gpar(), vp = NULL)
```

Arguments

x	x 位置を指定する数値ベクトルか単位オブジェクト.
y	y 位置を指定する数値ベクトルか単位オブジェクト.
width	幅を指定する数値ベクトルか単位オブジェクト.
height	高さを指定する数値ベクトルか単位オブジェクト.
just	矩形のその (x, y) 位置に対する位置揃え. もし二つの値があると, 最初の値は水平方向の位置揃えで二番目の値は垂直方向の位置揃えを指定する. 可能な文字列値は : "left", "right", "centre", "center", "bottom", そして "top". 数値の値に対しては 0 は左揃え, 1 は右揃えを意味する.
hjust	水平方向の位置揃えを指定する数値ベクトル. もし指定されると just の設定を上書きする.
vjust	垂直方向の位置揃えを指定する数値ベクトル. もし指定されると just の設定を上書きする.

default.units	もし x, y, width または height が数値ベクトルとしてだけ与えられた時に使われる既定の単位を指示する文字列.
name	文字列識別子.
gp	クラス gpar のオブジェクトで，典型的には関数 gpar への呼び出しからの出力．これは基本的にグラフィカルパラメータ設定のリストである.
draw	グラフィカルな出力を作るべきかどうかを指示する論理値.
vp	グリッドのビューポートオブジェクト(または NULL).

Details

どちらの関数も矩形 grob (矩形を表現するグラフィカルオブジェクト)を返すが，grid.rect は矩形を描画する (そして draw が TRUE の時だけ).

Value

矩形 grob．grid.rect は値を不可視で返す.

Author(s)

Paul Murrell

See Also

Grid, viewport

---

grid.refresh 現在のグリッドのシーンをリフレッシュする

Description

現在のグリッドのディスプレイリストを再生する.

Usage

```
grid.refresh()
```

Author(s)

Paul Murrell

grid.remove　　グリッドのグラフィカルオブジェクトを取り除く

Description

gTree または gTree の子孫を grob から取り除く.

Usage

```
grid.remove(gPath, warn = TRUE, strict  =  FALSE, grep = FALSE,
            global = FALSE, allDevices = FALSE, redraw = TRUE)

grid.gremove(..., grep = TRUE, global = TRUE)

removeGrob(gTree, gPath, strict = FALSE, grep = FALSE,
           global = FALSE, warn = TRUE)
```

Arguments

gTree	gTree オブジェクト.
gPath	gPath オブジェクト. grid.remove に対してはこれはディスプレイリスト上の gTree を指定する. removeGrob に対してはこれは指定された gTree の子孫を意味する.
strict	gPath が正確にマッチされるべきかどうかを指示するブール値.
grep	gPath が正規表現として扱われるべきかどうかを指示するブール値. 値は gPath の要素を跨ってリサイクルされる. (例えば c(TRUE, FALSE) は gPath の奇数番目の要素だけが正規表現として扱われる).
global	関数が gPath の最初のマッチに影響するか, または全てのマッチに影響すべきかを指示するブール値.
allDevices	全ての開かれたデバイスにマッチするものを探すか, または単に現在のデバイスを探すかを指示するブール値. まだ実装されていない.
warn	指定された grob が見つからなければエラーを発生するかを指示する論理値.
redraw	grob を再描画するかどうかを指示する. 論理値.
...	grid.get に渡される引数.

Details

removeGrob は指定された grob をコピーしそして修正された grob を返す.

grid.remove はディスプレイリスト上の grob を破壊的に修正する. もし redraw が TRUE ならばそれから変更を反映するために全てを再描画する.

grid.gremove (global の g)は異なった既定値を持つ grid.remove に対する便利なラッパである.

Value

removeGrob は grob オブジェクトを返す ; grid.remove は NULL を返す.

Author(s)

Paul Murrell

See Also

grob, getGrob, removeGrob, removeGrob.

---

grid.reorder *gTree の子供を並べ替える*

Description

gTree の子供達が描画される順序を変更する.

Usage

```
grid.reorder(gPath, order, back=TRUE, grep=FALSE, redraw=TRUE)
reorderGrob(x, order, back=TRUE)
```

Arguments

gPath	現在のシーン中の gTree を指定する gPath オブジェクト.
x	修正される gTree オブジェクト.
order	gTree の子供達に対する新しい描画順序を指定する文字列ベクトルか数値ベクトル. gTree の全ての子供達を参照していないかもしれない(詳細節を見よ).
back	gTree の全ての子供達を参照していない時(詳細節を見よ)に何が起きるかを制御する.
grep	gPath は正規表現として扱うべきか?
redraw	修正されたシーンは再描画されるべきか?

Details

最も単純な場合, order は gTree の全ての子供に対する新しい順序を指定する. 子供達は名前でも既存の数値順序でも指定することが出来る.

もし the order が gTree の全ての子供を指定していなければ, 既定では order で指定された子供達は最初に描画され, それから全ての残りの子供達が描画される. もし back=FALSE ならば order 中に指定されない子供達が最初に描画され, 次に指定された子供達が続く. これは後送りや前送りの再描画の指定を容易にする. order 引数は常に後が先の順序である.

ディスプレイリストは grob とビューポートの混合物であるので (従ってどのような再順序化が意味を持つのかは明らかでなく描かれないシーンは終了するほうがよっぽど簡単である) グリッドのディスプレイリスト (現在のシーン中のトップレベルの grob) を並べかえることは不可能である. もしグリッドのディスプレイリストを並べ替えたければ, grid.grab() で gTree を作りそれからその gTree を並べ替え(そして再描画す)る.

Value

grid.reorder() は現在のシーンを修正するという副作用のために呼び出される. reorderGrob() は修正された gTree を返す.

警告

この関数は描かれない gTree を返すかもしれない．例えば，二つの子供 A と B (この順序)を持ちそして子供 B の幅が子供 A の幅に依存する (例えばテキストの断片のまわりのボックス)．もしこれが grid.reorder() で起きれば，修正は実行されない．もしこれが reorderGrob() で起きれば単にオリジナルの順序を回復することが可能であるべきである．

Author(s)

Paul Murrell

Examples

```
# 二つの子供 "red-rect" と "blue-rect" (この順序)を持つ gTree
gt <- gTree(children=gList(
                rectGrob(gp=gpar(col=NA, fill="red"),
                         width=.8, height=.2, name="red-rect"),
                rectGrob(gp=gpar(col=NA, fill="blue"),
                         width=.2, height=.8, name="blue-rect")),
            name="gt")
grid.newpage()
grid.draw(gt)
# 全ての順序を数値として指定(blue-rect, red-rect)
grid.reorder("gt", 2:1)
# 全ての順序を文字列で指定する
grid.reorder("gt", c("red-rect", "blue-rect"))
# 後ろにしたいものだけを文字列で指定
grid.reorder("gt", "blue-rect")
# 前にしたいものだけを文字列で指定
grid.reorder("gt", "blue-rect", back=FALSE)
```

---

grid.segments 線分を描く

Description

これらの関数は線分を作り描く．

Usage

```
grid.segments(x0 = unit(0, "npc"), y0 = unit(0, "npc"),
              x1 = unit(1, "npc"), y1 = unit(1, "npc"),
              default.units = "npc",
              arrow = NULL,
              name = NULL, gp = gpar(), draw = TRUE, vp = NULL)
segmentsGrob(x0 = unit(0, "npc"), y0 = unit(0, "npc"),
              x1 = unit(1, "npc"), y1 = unit(1, "npc"),
              default.units = "npc",
              arrow = NULL, name = NULL, gp = gpar(), vp = NULL)
```

Arguments

x0	線分の開始x位置を指示する数値.
y0	線分の開始y位置を指示する数値.
x1	線分の停止x位置を指示する数値.
y1	線分の停止y位置を指示する数値.
default.units	文字列.
arrow	線分の両端に置かれる矢印を記述するリストで, arrow 関数が作り出すようなもの.
name	文字列識別子.
gp	クラス gpar のオブジェクト.
draw	グラフィックスの出力を生成すべきかどうかを指示する論理値.
vp	グリッドのビューポートオブジェクト(または NULL).

Details

どちらの関数も線分 grob (線分を記述するグラフィカルオブジェクト)を作るが, grid.segments だけが線分を描く (そして draw が TRUE の時だけ).

Value

線分の grob. grid.segments は不可視で値を返す.

Author(s)

Paul Murrell

See Also

Grid, viewport, arrow

---

grid.set　　グリッドのグラフィカルオブジェクトを設定する

Description

grob か grob の子孫を置き換える.

Usage

```
grid.set(gPath, newGrob, strict = FALSE, grep = FALSE,
         redraw = TRUE)

setGrob(gTree, gPath, newGrob, strict = FALSE, grep = FALSE)
```

Arguments

gTree	gTree オブジェクト.
gPath	gPath オブジェクト. grid.set に対してはこれはディスプレイリスト上の grob を指定する. setGrob に対してはこれは指定された gTree の子孫を指定する.
newGrob	grob オブジェクト.
strict	gTree は正確にマッチされるべきかどうかを指示するブール値.
grep	gPath を正規表現として扱うべきかどうかを指示するブール値. 値は gPath の要素に跨ってリサイクルされる (例えば c(TRUE, FALSE) は gPath の全ての奇数番目の要素が正規表現として扱われるべきことを扱われる).
redraw	grob を再描画するかどうかを指示する論理値.

Details

setGrob は指定された grob をコピーし修正された grob を返す.

grid.set はディスプレイリスト上の grob を破壊的に置き換える. もし redraw が TRUE ならば変更を反映するために全てを再描画する.

これらの関数は普通ユーザが呼び出すべきではない.

Value

setGrob は grob オブジェクトを返す ; grid.set は NULL を返す.

Author(s)

Paul Murrell

See Also

grid.grob.

---

grid.show.layout グリッドのレイアウトのダイアグラムを描く

Description

この関数はグリッドのレイアウトのダイアグラムを描くためにグリッドグラフィックスを使う.

Usage

```
grid.show.layout(l, newpage=TRUE, vp.ex = 0.8, bg = "light grey",
             cell.border = "blue", cell.fill = "light blue",
             cell.label = TRUE, label.col = "blue",
             unit.col = "red", vp = NULL)
```

Arguments

l	グリッドのレイアウトオブジェクト.
newpage	ダイアグラムを描く前に新しいページに移動するかどうかを指示する論理値.
vp.ex	正数，典型的には $(0, 1]$ 中でレイアウトのスケールを指示する.
bg	背景に使われる色.
cell.border	レイアウト中のセルの境界を描くのに使われる色.
cell.fill	レイアウト中のセルを塗りつぶすのに使われる色.
cell.label	レイアウトのセルにラベルを付けるかどうか指示する論理値.
label.col	レイアウトのセルのラベルに対して使われる色.
unit.col	列/行の幅/高さのラベルに使われる色.
vp	グリッドのビューポートオブジェクト(または NULL).

Details

注釈に対する余白を提供するビューポートが vp 中に作られ，そしてその新しいビューポート中にレイアウトが描かれる．余白は明るい灰色で塗りつぶされ，新しいビューポートは白で塗りつぶされ黒い境界で枠が付けられ，レイアウト領域は明るい青で塗り潰され青色で枠が付く．ダイアグラムは赤いテキストを使いレイアウトの列と行の幅と高さ(単位を含む)で注釈される． (全ての色は既定色で関数引数でカスタム化出来る．)

Value

無し.

Author(s)

Paul Murrell

See Also

Grid, viewport, grid.layout

Examples

```
## 簡単なレイアウトのダイアグラム
grid.show.layout(grid.layout(4,2,
                     heights=unit(rep(1, 4),
                                  c("lines", "lines", "lines", "null")),
                     widths=unit(c(1, 1), "inches")))
```

grid.show.viewport　グリッドのビューポートのダイアグラムを描く

Description

この関数はグリッドのビューポートのダイアグラムを描くためにグリッドのグラフィックスを使う.

Usage

```
grid.show.viewport(v, parent.layout = NULL, newpage = TRUE,
                   vp.ex = 0.8, border.fill="light grey",
                   vp.col="blue", vp.fill="light blue",
                   scale.col="red",
                   vp = NULL)
```

Arguments

v	グリッドのビューポートオブジェクト.
parent.layout	グリッドのレイアウトオブジェクト．もしこれが NULL でなく v 中で与えられたビューポートがレイアウトに関して指定された位置を持てば，ダイアグラムはレイアウトと v がレイアウト中でどのセルを占めるかを示す.
newpage	ダイアグラムを描く前に新しいページに移るかどうかを指示する論理値.
vp.ex	正数，典型的には $(0, 1]$ 中にあり，レイアウトのスケールを示す.
border.fill	境界余白を塗りつぶす色.
vp.col	ビューポート領域の境界に対する色.
vp.fill	ビューポート領域を塗りつぶす色.
scale.col	ビューポートの軸を描く色.
vp	グリッドのビューポートオブジェクト(または NULL).

Details

注釈のための余白を与えるビューポートが vp 中に作られ，ダイアグラムが新しいビューポート中に描かれる．既定では余白は明るい灰色で塗りつぶされ，新しいビューポートは白で塗り潰され黒い境界で囲まれ，そしてビューポート領域は明るい青で塗りつぶされ青い境界で囲まれる．ダイアグラムはビューポートの幅と高さ (単位を含む)，ビューポートの (x, y) 位置，そしてビューポートの x と y スケール，赤い線とテキストを使い描かれる.

Value

無し.

Author(s)

Paul Murrell

See Also

Grid, viewport

Examples

```
## サンプルのビューポートのダイアグラム
grid.show.viewport(viewport(x=0.6, y=0.6,
                            w=unit(1, "inches"), h=unit(1, "inches")))
grid.show.viewport(viewport(layout.pos.row=2, layout.pos.col=2:3),
                   grid.layout(3, 4))
```

---

grid.text　　テキストを描く

Description

これらの関数はテキストと plotmath 表現式を作り描く.

Usage

```
grid.text(label, x = unit(0.5, "npc"), y = unit(0.5, "npc"),
          just = "centre", hjust = NULL, vjust = NULL, rot = 0,
          check.overlap = FALSE, default.units = "npc",
          name = NULL, gp = gpar(), draw = TRUE, vp = NULL)

textGrob(label, x = unit(0.5, "npc"), y = unit(0.5, "npc"),
          just = "centre", hjust = NULL, vjust = NULL, rot = 0,
          check.overlap = FALSE, default.units = "npc",
          name = NULL, gp = gpar(), vp = NULL)
```

Arguments

label	文字列または expression ベクトル. 他のオブジェクトは as.graphicsAnnot により強制変換される.
x	x 値を指定する数値ベクトルまたは単位オブジェクト.
y	y 値を指定する数値ベクトルまたは単位オブジェクト.
just	その (x, y) 位置に関するテキストの位置揃え. もし二つの値があれば, 最初の値は水平方向の位置揃え, そして二番目の値は垂直方向の位置揃えを指定する. 可能な文字列値は : "left", "right", "centre", "center", "bottom", そして "top". 数値の値に対しては, 0 は左揃え 1 は右揃えを意味する.
hjust	水平方向の位置揃えを指定する数値ベクトル. もし指定されると just 設定を上書きする.
vjust	垂直方向の位置揃えを指定する数値ベクトル. もし指定されると just 設定を上書きする.
rot	テキストを回転する角度.
check.overlap	重複するテキストをチェックし取り除くかどうかを指示する論理値.
default.units	もし x か y が数値ベクトルとしてだけ与えられた時使われる既定の単位を指定する文字列.

name	文字列識別子.
gp	クラス gpar のオブジェクトで，典型的には関数 gpar への呼び出しからの出力である．これは基本的にはグラフィカルパラメータ設定のリストである.
draw	グラフィックス出力が生成されるべきかどうかを指示する論理値.
vp	グリッドのビューポートオブジェクト(または NULL).

Details

どちらの関数もテキスト grob (テキストを記述するグラフィカルオブジェクト)を作るが，grid.text だけがテキストを描く (そして draw が TRUE の時だけ).

もし label 引数が表現式ならば，出力はグラフィックステキストに対するように数式注釈として整形される.

Value

テキスト grob. grid.text は値を不可視で返す.

Author(s)

Paul Murrell

See Also

Grid, viewport

Examples

```
grid.newpage()
x <- stats::runif(20)
y <- stats::runif(20)
rot <- stats::runif(20, 0, 360)
grid.text("SOMETHING NICE AND BIG", x=x, y=y, rot=rot,
          gp=gpar(fontsize=20, col="grey"))
grid.text("SOMETHING NICE AND BIG", x=x, y=y, rot=rot,
          gp=gpar(fontsize=20), check=TRUE)
grid.newpage()
draw.text <- function(just, i, j) {
  grid.text("ABCD", x=x[j], y=y[i], just=just)
  grid.text(deparse(substitute(just)), x=x[j], y=y[i] + unit(2, "lines"),
            gp=gpar(col="grey", fontsize=8))
}
x <- unit(1:4/5, "npc")
y <- unit(1:4/5, "npc")
grid.grill(h=y, v=x, gp=gpar(col="grey"))
draw.text(c("bottom"), 1, 1)
draw.text(c("left", "bottom"), 2, 1)
draw.text(c("right", "bottom"), 3, 1)
draw.text(c("centre", "bottom"), 4, 1)
draw.text(c("centre"), 1, 2)
draw.text(c("left", "centre"), 2, 2)
draw.text(c("right", "centre"), 3, 2)
draw.text(c("centre", "centre"), 4, 2)
draw.text(c("top"), 1, 3)
draw.text(c("left", "top"), 2, 3)
```

```
draw.text(c("right", "top"), 3, 3)
draw.text(c("centre", "top"), 4, 3)
draw.text(c(), 1, 4)
draw.text(c("left"), 2, 4)
draw.text(c("right"), 3, 4)
draw.text(c("centre"), 4, 4)
```

---

grid.xaxis *Draw an X-Axis*

Description

これらの関数は xaxis を作り描く.

Usage

```
grid.xaxis(at = NULL, label = TRUE, main = TRUE,
           edits = NULL, name = NULL,
           gp = gpar(), draw = TRUE, vp = NULL)

xaxisGrob(at = NULL, label = TRUE, main = TRUE,
          edits = NULL, name = NULL,
          gp = gpar(), vp = NULL)
```

Arguments

at	チックマークのx値位置の数値ベクトル.
label	チックマーク上にラベルを描くかどうかを指示する論理値，または使用するラベルを指定する表現式か文字列ベクトル．もし論理値でなければ at 引数と同じ長さでなければならない.
main	x軸をビューポートの底(TRUE)か上(FALSE)に描くかを指示する論理値.
edits	at が NULL である限り. 軸が最初に作られ再描画の過程で (軸の子供に)適用される編集操作を含む gEdit または gEditList.
name	文字列識別子.
gp	クラス gpar のオブジェクトで，典型的には関数 gpar への呼び出しの出力．これは典型的にはグラフィカルパラメータ設定のリストである.
draw	グラフィックス出力を生成するかどうかを指示する論理値.
vp	グリッドのビューポートオブジェクト(または NULL).

Details

どちらの関数も xaxis (xaxis を表現するグラフィカルオブジェクト)を作るが，grid.xaxis だけがx軸を描く (そして draw が TRUE の時だけ).

Value

x軸 grob. grid.xaxis は値を不可視で返す.

子供

もし x軸 grob の at スロットが NULL で無ければ x軸は次の子供を持つ：

major チックマークの基礎にある直線を表現する.

ticks チックマークを表現する.

labels チックラベルを表現する.

もし at スロットが NULL ならば子供は無くチックは現在のビューポートのスケールに基づいて描かれる.

Author(s)

Paul Murrell

See Also

Grid, viewport, grid.yaxis

---

grid.xspline *X*スプリンを描く

Description

これらの関数は制御点に関して描かれるxスプライン曲線を作り描く.

Usage

```
grid.xspline(...)
xsplineGrob(x = c(0, 0.5, 1, 0.5), y = c(0.5, 1, 0.5, 0),
            id = NULL, id.lengths = NULL,
            default.units = "npc",
            shape = 0, open = TRUE, arrow = NULL, repEnds = TRUE,
            name = NULL, gp = gpar(), vp = NULL)
```

Arguments

x	スプラインの制御点のx位置を指定する数値ベクトルまたは単位オブジェクト.
y	スプラインの制御点のy位置を指定する数値ベクトルまたは単位オブジェクト.
id	x と y 中の位置を複数の x スプラインに分離するために使われる数値ベクトル. 同じ id を持つ全ての位置は同じxスプラインに属する.
id.lengths	x と y 中の位置を複数の x スプラインに分離するために使われる数値ベクトル. 別個のxスプラインを構成する位置の連続するブロックを指定する.
default.units	もし x か y が数値ベクトルとしてだけ与えられた時に使われる既定の単位を指定する文字列.
shape	-1 と 1 の間の値の数値ベクトルで, 制御点に関するスプラインの形状を制御する.

open	スプラインが線であるか閉じた形状であるかを指定する論理値.
arrow	xスプラインの各々の端に置かれる矢尻を記述するリストで， arrow 関数により作られるようなもの.
repEnds	最初と最後の制御点が曲線を描く時複製されるべきかどうかを指示する論理値 (下の詳細節を見よ).
name	文字列識別子.
gp	クラス gpar のオブジェクトで，典型的には関数 gpar への呼び出しからの出力．これは基本的にはグラフィカルパラメータの設定のリストである.
vp	グリッドのビューポートオブジェクト(または NULL).
...	xsplineGrob に渡される引数.

Details

どちらの関数もxスプライン grob (xスプラインを記述するグラフィカルオブジェクト) を作るが grid.xspline だけがxスプラインを描く.

xスプラインは制御点に関して描かれる線である．線は制御点を通ったり(補間)，制御点に近づく(近似)かもしれない；挙動は各制御点に対する形状パラメータで決定される.

もし形状パラメータがゼロより大きければ，スプラインは制御点を近似する (そして形状が1ならば3次Bスプラインに非常に近い)．もし形状パラメータが0ならば，スプラインは制御点で鋭角の隅を作る.

開いたxスプラインに対しては，開始と終了の制御点は形状 0 を持たなければならない(そしてゼロでない値は警告なしにゼロに変換される).

開いたxスプラインに対しては，既定では開始と終了制御点は実際には曲線が描かれる前に複製される．曲線(補間または近似)は四つの制御点のセットの二番目と三番目の間に描かれるので，この既定動作は結果の曲線が指定された最初の制御点から始まり最後の制御点で終わる．既定の動作は repEnds 引数でオフに出来，その場合描かれる曲線は二番目の制御点から始まり(近似的に)そして最初と最後から二番目の制御点で終わる(近似的に).

repEnds 引数は閉じたxスプラインに対しては無視される.

x と y に対する欠損値は許されない (つまりそれは欠損した制御点に対しては適正ではない).

閉じたxスプラインに対しては，曲線は自動的に制御点の最終点と初期点間で描かれる.

Value

grob オブジェクト.

References

Blanc, C. and Schlick, C. (1995), "X-splines : A Spline Model Designed for the End User", in *Proceedings of SIGGRAPH 95*, pp. 377–386. http://dept-info.labri.fr/~schlick/DOC/sig1.html

See Also

Grid, viewport, arrow.

xspline.

Examples

```
x <- c(0.25, 0.25, 0.75, 0.75)
y <- c(0.25, 0.75, 0.75, 0.25)

xsplineTest <- function(s, i, j, open) {
  pushViewport(viewport(layout.pos.col=j, layout.pos.row=i))
  grid.points(x, y, default.units="npc", pch=16, size=unit(2, "mm"))
  grid.xspline(x, y, shape=s, open=open, gp=gpar(fill="grey"))
  grid.text(s, gp=gpar(col="grey"),
            x=unit(x, "npc") + unit(c(-1, -1, 1, 1), "mm"),
            y=unit(y, "npc") + unit(c(-1, 1, 1, -1), "mm"),
            hjust=c(1, 1, 0, 0),
            vjust=c(1, 0, 0, 1))
  popViewport()
}

pushViewport(viewport(width=.5, x=0, just="left",
                      layout=grid.layout(3, 3, respect=TRUE)))
pushViewport(viewport(layout.pos.row=1))
grid.text("Open Splines", y=1, just="bottom")
popViewport()
xsplineTest(c(0, -1, -1, 0), 1, 1, TRUE)
xsplineTest(c(0, -1,  0, 0), 1, 2, TRUE)
xsplineTest(c(0, -1,  1, 0), 1, 3, TRUE)
xsplineTest(c(0,  0, -1, 0), 2, 1, TRUE)
xsplineTest(c(0,  0,  0, 0), 2, 2, TRUE)
xsplineTest(c(0,  0,  1, 0), 2, 3, TRUE)
xsplineTest(c(0,  1, -1, 0), 3, 1, TRUE)
xsplineTest(c(0,  1,  0, 0), 3, 2, TRUE)
xsplineTest(c(0,  1,  1, 0), 3, 3, TRUE)
popViewport()
pushViewport(viewport(width=.5, x=1, just="right",
                      layout=grid.layout(3, 3, respect=TRUE)))
pushViewport(viewport(layout.pos.row=1))
grid.text("Closed Splines", y=1, just="bottom")
popViewport()
xsplineTest(c(-1, -1, -1, -1), 1, 1, FALSE)
xsplineTest(c(-1, -1,  0, -1), 1, 2, FALSE)
xsplineTest(c(-1, -1,  1, -1), 1, 3, FALSE)
xsplineTest(c( 0,  0, -1,  0), 2, 1, FALSE)
xsplineTest(c( 0,  0,  0,  0), 2, 2, FALSE)
xsplineTest(c( 0,  0,  1,  0), 2, 3, FALSE)
xsplineTest(c( 1,  1, -1,  1), 3, 1, FALSE)
xsplineTest(c( 1,  1,  0,  1), 3, 2, FALSE)
xsplineTest(c( 1,  1,  1,  1), 3, 3, FALSE)
popViewport()
```

`grid.yaxis` *y*軸を描く

Description

これらの関数は y 軸を作り描く.

Usage

```
grid.yaxis(at = NULL, label = TRUE, main = TRUE,
           edits = NULL, name = NULL,
           gp = gpar(), draw = TRUE, vp = NULL)

yaxisGrob(at = NULL, label = TRUE, main = TRUE,
          edits = NULL, name = NULL,
          gp = gpar(), vp = NULL)
```

Arguments

at	チックマークのy値位置の数値ベクトル.
label	チックマーク上にラベルを描くかどうかを指示する論理値，または使用するラベルを指定する表現式か文字列ベクトル．もし論理値でなければ at 引数と同じ長さでなければならない.
main	x軸をビューポートの底(TRUE)か上(FALSE)に描くかを指示する論理値.
edits	at が NULL である限り. 軸が最初に作られ再描画の過程で (軸の子供に)適用される編集操作を含む gEdit または gEditList.
name	文字列識別子.
gp	クラス gpar のオブジェクトで，典型的には関数 gpar への呼び出しの出力．これは典型的にはグラフィカルパラメータ設定のリストである.
draw	グラフィックス出力を生成するかどうかを指示する論理値.
vp	グリッドのビューポートオブジェクト(または NULL).

Details

どちらの関数も yaxis grob (y 軸を表現するグラフィカルオブジェクト)を作るが，grid.yaxis だけがy軸を描く (そして draw が TRUE の時だけ).

Value

yaxis grob. grid.yaxis は値を不可視で返す.

子供

もしy軸 grob の at スロットが NULL で無ければ y軸は次の子供を持つ：

major チックマークの基礎にある直線を表現する.

ticks チックマークを表現する.

labels チックラベルを表現する.

もし at スロットが NULL ならば子供は無くチックは現在のビューポートのスケールに基づいて描かれる.

Author(s)

Paul Murrell

See Also

Grid, viewport, grid.xaxis

grobName — *grob* に対する名前を作る

Description

この関数は grob に対する(セッション内で)ユニークな名前を grob のクラスに基づいて作る.

Usage

```
grobName(grob = NULL, prefix = "GRID")
```

Arguments

grob	grob オブジェクトか NULL.
prefix	名前の接頭辞部分.

Value

型式 prefix.class(grob).index の文字列

Author(s)

Paul Murrell

grobWidth — *grob* の幅を記述する単位を作る

Description

これらの関数は grob の幅か高さを記述する単位オブジェクトを作る. これは総称的関数である.

Usage

```
grobWidth(x)
grobHeight(x)
grobAscent(x)
grobDescent(x)
```

Arguments

x	grob オブジェクト.

Value

単位オブジェクト.

Author(s)

Paul Murrell

See Also

unit と stringWidth

---

grobX *grob* の境界位置を記述する単位を作る

Description

これらの関数は grob の境界のある点を記述する単位オブジェクトを作る．これらは総称的関数である．

Usage

```
grobX(x, theta)
grobY(x, theta)
```

Arguments

x	grob または gList または gTree または gPath.
theta	位置が grob 境界のどこにあるかを指示する角度． "east", "north", "west" または "south" のどれかで，それぞれ角度 0, 90, 180 そして 270 に対応する．

Details

角度は反時計回りで，ゼロは原点が形状の端点間の中心に位置し3時の方向を指している．

もし grob が単一の形状を記述していれば，境界値は形状の正確な辺に対応する．

もし grob が複数の形状を記述すれば，境界値は grob で記述される全ての形状(複数の矩形，円，xスプライン，またはテキスト) のまわりのバウンディングボックスの辺か，grob で記述される全ての形状 (複数の多角形，点，線，複数の線，そして線分)のまわりの凸包に対応する．

凸包はデータシンボルの位置に基づいておりデータシンボルの拡がりを考慮しないため，点の grob は現在特殊なケースである．

任意の矢尻の拡がりは現在考慮されない．

Value

単位オブジェクト．

Author(s)

Paul Murrell

See Also

unit と grobWidth

legendGrob 凡例 *grob* を作る

Description

凡例 grob を作る(進行中の)

Usage

```
legendGrob(labels, nrow, ncol, byrow = FALSE,
           do.lines = has.lty || has.lwd, lines.first = TRUE,
           hgap = unit(1, "lines"), vgap = unit(1, "lines"),
           default.units = "lines", pch, gp = gpar(), vp = NULL)

grid.legend(..., draw=TRUE)
```

Arguments

labels	凡例ラベル(表現式)
nrow, ncol	整数；それぞれ凡例の“レイアウト”行と列の数． nrow はオプションで典型的にはラベルと ncol の数から計算される．
byrow	凡例の行が最初に埋められるかどうかを指示する論理値．
do.lines	凡例線を描くかどうかを指示する論理値．
lines.first	凡例線が最初にそしてそれから単純な“下側”の凡例シンボルを描く．
hgap	凡例の項目間の水平スペース
vgap	凡例の項目間の垂直スペース
default.units	既定の単位，unit を見よ．
pch	数値もしくは文字の凡例のシンボルで pointsGrob() に渡される；数値コードの解釈については points も見よ．
gp	クラス gpar の R オブジェクトで，典型的には関数 gpar への呼び出しからの出力で，基本的にはグラフィカルパラメータ設定のリスト．
vp	グリッドの viewport オブジェクト(または NULL)．
...	grid.legend() に対して：上の全ての引数は legendGrob() に渡される．
draw	グラフィックス出力が生成されるべきかどうかを指示する論理値．

Value

どちらの関数も凡例 grob (プロット凡例を記述するグラフィカルオブジェクト)を作るが，grid.legend だけがそれを描く(draw が TRUE の時だけ)．

See Also

Grid, viewport; pointsGrob, linesGrob.

grid.plot.and.legend は簡単な例を含む．

Examples

```
## データ：
n <- 10
x <- stats::runif(n) ; y1 <- stats::runif(n) ; y2 <- stats::runif(n)
## grob を作る：
plot <- gTree(children=gList(rectGrob(),
                pointsGrob(x, y1, pch=21, gp=gpar(col=2, fill="gray")),
                pointsGrob(x, y2, pch=22, gp=gpar(col=3, fill="gray")),
                xaxisGrob(),
                yaxisGrob()))
legd <- legendGrob(c("Girls", "Boys", "Other"), pch=21:23,
                   gp=gpar(col = 2:4, fill = "gray"))
gg <- packGrob(packGrob(frameGrob(), plot),
               legd, height=unit(1,"null"), side="right")

## 新しいデバイスページにそれを描く：
grid.newpage()
pushViewport(viewport(width=0.8, height=0.8))
grid.draw(gg)
```

---

makeContent　　グリッドの *grob* をカスタム化する

Description

これらの総称的なフック関数はグリッドの grob が描かれる度に呼び出される．それらは文脈の描画と grob (または gTree)から導かれた新しいクラスの内容の描画のカスタム化に対する機会を提供する．

Usage

```
makeContext(x)
makeContent(x)
```

Arguments

x	グリッドの grob.

Details

これらの関数は grob と gTree に対する `grid.draw` メソッドにより呼び出される．

`makeContext` は最初 grob の描画中に呼び出される．この関数は `x` の `vp` スロット(そして/または `x` が gTree なら `childrenvp` スロット)を修正するのに使われるべきである．この関数は修正された `x` を返す べきである．grob に対する既定の挙動は `vp` スロット中の任意のビューポートをプッシュし，そして gTree に対しては `childrenvp` スロット中の任意のビューポートをプッシュしアップすることであるので，この関数は grob または gTree に対する文脈の描画をカスタム化するのに使われることを注意する．

`makeContent` が次に任意の追加の計算が行われグラフィカルな内容が生成されるべき箇所で呼び出される (例えば `grid::makeContent.xaxis` を見よ)．この関数は gTree の `children` を修正するために使われるべきである．この関数は修正された `x` を返すべきである． gTree に対する既定の挙動は `children` スロット中の全ての grob を描画することで

あり，従ってこの関数は gTree 中の内容の描画をカスタム化することに使われる．簡単な grob に対する内容の描画をカスタム化することも可能であるが，より注意が必要になる；例えばこの関数はこの場合 drawDetails() メソッドを持つ標準的なグリッドのプリミティブ関数を返すべきである．

makeContent() が返す x は makeContext() により行われる任意の変更を 含むように，これらの関数は効果に於いて 累積的であるべきである．

makeContext はまた "grobwidth" と "grobheight" 単位の計算中でも呼び出されることを注意する．

Value

どちらの関数も grob か gTree (x の修正版)を返すことを期待されている．

Author(s)

Paul Murrell

See Also

grid.draw

---

plotViewport 標準的なプロットレイアウトでビューポートを作る

Description

これは標準の S 風のプロットレイアウトを持つビューポートを作るための便利関数である – つまりテキストの行数で与えられる余白で囲まれた中心化プロット領域.

Usage

```
plotViewport(margins=c(5.1, 4.1, 4.1, 2.1), ...)
```

Arguments

margins	基本グラフィックス中の par(mar) と同様に解釈される数値ベクトル.
...	全ての他の引数は viewport() 関数の呼び出しに渡される.

Value

グリッドのビューポートオブジェクト.

Author(s)

Paul Murrell

See Also

viewport と dataViewport.

Querying the Viewport Tree
現在のグリッドのビューポイント(ツリー)を得る

Description

current.viewport() はグリッドが描き込みを行おうとしているビューポートを返す.

current.parent は現在のビューポートの親を返す.

current.vpTree はグリッドのビューポートツリーの全体を返す.

current.vpPath は現在のビューポートへのビューポートパスを返す.

current.transform は現在のビューポートに対する変換行列を返す.

current.rotation は現在のビューポートに対する (全部の)回転を返す.

Usage

```
current.viewport()
current.parent(n=1)
current.vpTree(all=TRUE)
current.vpPath()
current.transform()
```

Arguments

n	世代数.
all	全てのビューポートのツリーを返すべきかかどうかを指示する論理値.

Details

現在のビューポートの祖父母(それ以上)を current.parent() の n 引数を使って得ることが出来る.

ルートビューポートの親は NULL である. ルートビューポートの祖父母を要求することはエラーである.

もし all が FALSE ならば current.vpTree は現在のビューポートの副ツリーだけを返す.

Value

current.viewport または current.vpTree からのグリッドビューポートオブジェクト.

current.transform は 4x4 の変換行列を返す.

もし現在のビューポートが ROOT ビューポートならば, ビューポートのパス.

Author(s)

Paul Murrell

See Also

viewport

Examples

```
grid.newpage()
pushViewport(viewport(width=0.8, height=0.8, name="A"))
pushViewport(viewport(x=0.1, width=0.3, height=0.6,
  just="left", name="B"))
upViewport(1)
pushViewport(viewport(x=0.5, width=0.4, height=0.8,
  just="left", name="C"))
pushViewport(viewport(width=0.8, height=0.8, name="D"))
current.vpPath()
upViewport(1)
current.vpPath()
current.vpTree()
current.viewport()
current.vpTree(all=FALSE)
popViewport(0)
```

---

resolveRasterSize　ラスター *grob* のサイズを決定するユーティリティ関数

Description

ラスター grob の幅と高さの双方もしくは一方が明示的に与えられない時幅と高さを決定する.

結果はラスターイメージのアスペクト比と物理的な描画文脈のアスペクト比の双方に依存するので，結果はこの関数がその中で呼び出される描画文脈に対してだけ正当である.

Usage

```
resolveRasterSize(x)
```

Arguments

x	ラスター grob

Details

ラスター grob は幅と高さもしくはどちらかを持つ NULL で指定でき，これはラスターが描かれるサイズは描画時に決定されることを意味する.

Value

明示的な幅と高さを持つラスター grob.

See Also

grid.raster

Examples

```
# 正方形のラスター
rg <- rasterGrob(matrix(0))
# ページ全体を塗りつぶす(ページが正方形の時)
grid.newpage()
resolveRasterSize(rg)$height
grid.draw(rg)
# 高い幅が狭い領域に強制
grid.newpage()
pushViewport(viewport(width=.1))
resolveRasterSize(rg)$height
grid.draw(rg)
```

---

roundrect 丸い隅の矩形を描く

Description

丸い隅の矩形を一つ描く.

Usage

```
roundrectGrob(x=0.5, y=0.5, width=1, height=1,
              default.units="npc",
              r=unit(0.1, "snpc"),
              just="centre",
              name=NULL, gp=NULL, vp=NULL)
grid.roundrect(...)
```

Arguments

x, y, width, height	矩形の位置とサイズ.
default.units	もし x, y, width そして height が数値ベクトルとしてだけ与えられた時に使われる既定の単位を指示する文字列.
r	丸い隅の半径.
just	矩形のその位置に対する位置揃え.
name	grob を識別する名前.
gp	grob に適用されるグラフィカルパラメータ.
vp	ビューポートオブジェクトまたは NULL.
...	roundrectGrob() に渡される引数.

Details

現在この関数は丸い隅の矩形を一つだけ描くことが出来る.

Examples

```
grid.roundrect(width=.5, height=.5, name="rr")
theta <- seq(0, 360, length=50)
for (i in 1:50)
    grid.circle(x=grobX("rr", theta[i]),
                y=grobY("rr", theta[i]),
                r=unit(1, "mm"),
                gp=gpar(fill="black"))
```

---

showGrob　　グリッドの *grob* のラベル

Description

(既定では)現在のグリッドのシーンにシーン中の各 grob の名前を示すラベル付きのグラフィカルディスプレイを作成する．シーン中の特定の grob にだけラベルを付けることも出来る．

Usage

```
showGrob(x = NULL,
         gPath = NULL, strict = FALSE, grep = FALSE,
         recurse = TRUE, depth = NULL,
         labelfun = grobLabel, ...)
```

Arguments

x	もし NULL なら現在のグリッドのシーンにラベルが付く．さもなければ描画されそれからラベルが付く grob (または gTree).
gPath	現在のシーンの一部分またはラベルを付ける grob を指定するパス.
strict	gPath が厳密かどうかを指定する論理値.
grep	gPath が正規表現かどうかを指示する論理値.
recurse	gTree の子供もラベルを付けるか?
depth	指定された深さ(ベクトルかもしれない)の grob だけを表示する.
labelfun	各 grob からラベルを生成するために使われる関数.
...	生成されたラベルの細かい詳細を制御する labelfun に渡される引数.

Details

どんなラベルもグリッドのディスプレイリスト上には記録され無いので，オリジナルのシーンは grid.refresh の呼び出しで再生出来る．

See Also

grob と gTree

Examples

```
    grid.newpage()
    gt <- gTree(childrenvp=vpStack(
                  viewport(x=0, width=.5, just="left", name="vp"),
                  viewport(y=.5, height=.5, just="bottom", name="vp2")),
                children=gList(rectGrob(vp="vp::vp2", name="child")),
                name="parent")
    grid.draw(gt)
    showGrob()
    showGrob(gPath="child")
    showGrob(recurse=FALSE)
    showGrob(depth=1)
    showGrob(depth=2)
    showGrob(depth=1:2)
    showGrob(gt)
    showGrob(gt, gPath="child")
    showGrob(just="left", gp=gpar(col="red", cex=.5), rot=45)
    showGrob(labelfun=function(grob, ...) {
        x <- grobX(grob, "west")
        y <- grobY(grob, "north")
        gTree(children=gList(rectGrob(x=x, y=y,
                width=stringWidth(grob$name) + unit(2, "mm"),
                height=stringHeight(grob$name) + unit(2, "mm"),
                gp=gpar(col=NA, fill=rgb(1, 0, 0, .5)),
                just=c("left", "top")),
                textGrob(grob$name,
                         x=x + unit(1, "mm"), y=y - unit(1, "mm"),
                         just=c("left", "top"))))
    })

## Not run:
    # 高水準のパッケージからの例

    library(lattice)
    # 最初の例の後に Ctrl-c
    example(histogram)
    showGrob()
    showGrob(gPath="plot_01.ylab")

    library(ggplot2)
    # 最初の例の後に Ctrl-c
    example(qplot)
    showGrob()
    showGrob(recurse=FALSE)
    showGrob(gPath="panel-3-3")
    showGrob(gPath="axis.title", grep=TRUE)
    showGrob(depth=2)

## End(Not run)
```

---

showViewport　　グリッドのビューポートを表示する.

Description

現在のグリッドのビューポート(既定)のグラフィカルな表示を作る．特定のビューポートだけを表示することも出来る．各ビューポートは矩形で描かれビューポートの名前のラベルが付く．

Usage

```
showViewport(vp = NULL, recurse = TRUE, depth = NULL,
             newpage = FALSE, leaves = FALSE,
             col = rgb(0, 0, 1, 0.2), fill = rgb(0, 0, 1, 0.1),
             label = TRUE, nrow = 3, ncol = nrow)
```

Arguments

vp	もし NULL なら現在のビューポートのツリーが表示される．さもなければビューポート(または vpList または vpStack または vpTree)，またはどのビューポートを表示するかを指定する vpPath.
recurse	指定されたビューポートの子供達もまた表示するか?
depth	指定された深さ(深さのベクトルかもしれない)のビューポートだけを表示する．
newpage	表示のために新しいページを開始するか? さもなければビューポートは現在のプロットの上に表示される．
leaves	小さなディスプレイの行列を作り，葉のビューポートはそれ自体のディスプレイ中にある．
col	各ビューポートに対する矩形の境界と各ビューポートに対するラベルを描くのに使われる色．もしベクトルならば，最初の色はトップレベルのビューポートに使われ，二番目の色はその子供達に使われ，三番目の色は彼らの子供達に使われる，云々．
fill	各ビューポートを塗りつぶすのに使われる色． col 毎のベクトルかもしれない．
label	ビューポートはラベル(ビューポートの名前の)を付けるか?
nrow, ncol	leaves が TRUE の時の行と列の数．さもなければ無視される．

See Also

viewport と grid.show.viewport

Examples

```
showViewport(viewport(width=.5, height=.5, name="vp"))

grid.newpage()
pushViewport(viewport(width=.5, height=.5, name="vp"))
upViewport()
showViewport(vpPath("vp"))

showViewport(vpStack(viewport(width=.5, height=.5, name="vp1"),
                     viewport(width=.5, height=.5, name="vp2")),
             newpage=TRUE)

showViewport(vpStack(viewport(width=.5, height=.5, name="vp1"),
```

```
        viewport(width=.5, height=.5, name="vp2")),
fill=rgb(1:0, 0:1, 0, .1),
newpage=TRUE)
```

---

stringWidth　　文字列や数学表現の幅と高さを記述する単位を作る

Description

これらの関数は文字列の幅と高さを記述する単位オブジェクトを作る.

Usage

```
stringWidth(string)
stringHeight(string)
stringAscent(string)
stringDescent(string)
```

Arguments

string	文字列ベクトルまたは言語オブジェクト (‘plotmath’ の呼び出しに対して使われるような).

Value

unit オブジェクト.

Author(s)

Paul Murrell

See Also

unit と grobWidth

計算の背後にある印刷術の概念のより詳細についてはパッケージ **graphics** 中の strwidth.

---

unit　　単位オブジェクトを作る関数

Description

この関数は単位値のベクトルである単位オブジェクトを作る. 単位値は典型的には単に関連する単位を備えた単一の数値である.

Usage

```
unit(x, units, data=NULL)
```

Arguments

x	数値ベクトル.
units	対応する数値に対する単位を指定する文字列ベクトル.
data	この引数は特別な unit タイプに対する追加の情報を提供するのに使われる.

Details

単位オブジェクトはユーザが異なった多数の座標系中の位置や次元を指定することを可能にする．全ての描画はビューポートに相対的になされ， units はそのビューポート中でどの座標系を使うかを指定する.

可能な units (座標系)は：

"npc" 正規化された親座標(既定値)．ビューポートの原点は (0, 0) でビューポートは幅と高さを持つ．例えば (0.5, 0.5) はビューポートの中心である.

"cm" センチメートル.

"inches" インチ． 1 in = 2.54 cm.

"mm" ミリメートル． 10 mm = 1 cm.

"points" ポイント． 72.27 pt = 1 in.

"picas" パイカ． 1 pc = 12 pt.

"bigpts" ビッグポイント． 72 bp = 1 in.

"dida" ジーダ． 1157 dd = 1238 pt.

"cicero" キケロ. 1 cc = 12 dd.

"scaledpts" スケールドポイント． 65536 sp = 1 pt.

"lines" テキストの行数．位置と次元はビューポートの既定のテキストのサイズの倍数で与えられる (ビューポートの fontsize と lineheight で指定されるような).

"char" ビューポートの名目上のフォントの高さの倍数 (ビューポートの fontsize で指定されるような).

"native" 位置と次元はビューポートの xscale と yscale に関して相対的である.

"snpc" 正方形に正規化された親座標．水平と垂直方向の位置/次元に対して同じ回答を与えることを除けば正規化親座標と同じ．これは npc 幅と npc 高さの小さい方を使う．これはビューポートの比率であるようなものを作るのに役立つが，正方形でなければならない (または固定されたアスペクト比を持つ).

"strwidth" data 引数中で指定された文字列の幅の倍数．フォントのサイズはビューポートのポイントサイズで決定される.

"strheight" data 引数中で指定された文字列の高さの倍数．フォントのサイズはビューポートのポイントサイズで決定される.

"grobwidth" data 引数中で指定された grob の幅の倍数.

"grobheight" data 引数中で指定された grob の高さの倍数.

殆どの通常の単位に対する幾つかの変種がまた許される．例えば， "inches" と "centimetre" の代わりに "in" か "inch"， または "cm" の代わりに "centimeter" を使うことが出来る.

特殊な units 値の "null" がまた許されるが，グリッドのレイアウトの列の幅や行の高さを指定する時にだけ意味を持つ.

unit.length() が1より大きければ data 引数はリストでなければならない．例えば

```
unit(rep(1, 3), c("npc", "strwidth", "inches"),
data = list(NULL, "my string", NULL)).
```

単位オブジェクトを通常の仕方による部分操作と部分的付値をすることが可能である(例を見よ)が，単位オブジェクトを連結するには特別な関数 unit.c が用意されている．

単位オブジェクトに対するある算術と要約操作が定義されている．特に単位オブジェクトの和と差 (例えば unit(1, "npc") - unit(1, "inches")), 単位オブジェクトのリストの最小値と最大値を指定することが可能である(例えば min(unit(0.5, "npc"), unit(1, "inches"))).

Value

クラス "unit" のオブジェクト．

警告

幾つかの単位オブジェクトを連結するための特別な関数 unit.c がある．

c 関数は正しい回答を与えない．

単位 "mylines", "mychar", "mystrwidth", "mystrheight" が使われてきた．これらは依然受け入れられるが， "lines", "char", "strwidth", "strheight" と正確に同じ動作をする．

Author(s)

Paul Murrell

See Also

unit.c

Examples

```
unit(1, "npc")
unit(1:3/4, "npc")
unit(1:3/4, "npc") + unit(1, "inches")
min(unit(0.5, "npc"), unit(1, "inches"))
unit.c(unit(0.5, "npc"), unit(2, "inches") + unit(1:3/4, "npc"),
       unit(1, "strwidth", "hi there"))
x <- unit(1:5, "npc")
x[2:4]
x[2:4] <- unit(1, "mm")
x
```

---

unit.c 単位オブジェクトを結合する

Description

この関数は引数で指定される単位オブジェクトを結合して新しい単位オブジェクトを作る．

Usage

```
unit.c(...)
```

Arguments

...	任意個数の単位オブジェクト.

Value

クラス unit のオブジェクト.

Author(s)

Paul Murrell

See Also

unit.

---

unit.length 単位オブジェクトの長さ

Description

単位オブジェクトの長さは単位オブジェクト中の単位値の数と定義される.
この関数は総称的な length 関数に対する単位メソッドを優先するため廃止予定である.

Usage

```
unit.length(unit)
```

Arguments

unit	単位オブジェクト.

Value

整数値.

Author(s)

Paul Murrell

See Also

unit

Examples

```
length(unit(1:3, "npc"))
length(unit(1:3, "npc") + unit(1, "inches"))
length(max(unit(1:3, "npc") + unit(1, "inches")))
length(max(unit(1:3, "npc") + unit(1, "strwidth", "a"))*4)
length(unit(1:3, "npc") + unit(1, "strwidth", "a")*4)
```

unit.pmin 単位の並列的な最小値と最大値

Description

引数の i 番目の値の i 番目の値が最少(最大)である単位オブジェクトを返す.

Usage

```
unit.pmin(...)
unit.pmax(...)
```

Arguments

...	ひとつまたはそれ以上の単位オブジェクト.

Details

結果の長さは引数の長さの最大値である；短い引数は通常のようにリサイクルされる.

Value

単位オブジェクト.

Author(s)

Paul Murrell

Examples

```
max(unit(1:3, "cm"), unit(0.5, "npc"))
unit.pmax(unit(1:3, "cm"), unit(0.5, "npc"))
```

unit.rep 単位オブジェクトの要素を複製する

Description

単位を times と length.out 中で与えられた値に従い複製する.

この関数は総称的な rep 関数を優先して廃止予定である.

Usage

```
unit.rep(x, ...)
```

Arguments

x	クラス "unit" のオブジェクト.
...	times と length.out のような rep に渡される引数.

Value

クラス "unit" のオブジェクト.

Author(s)

Paul Murrell

See Also

rep

Examples

```
rep(unit(1:3, "npc"), 3)
rep(unit(1:3, "npc"), 1:3)
rep(unit(1:3, "npc") + unit(1, "inches"), 3)
rep(max(unit(1:3, "npc") + unit(1, "inches")), 3)
rep(max(unit(1:3, "npc") + unit(1, "strwidth", "a"))*4, 3)
rep(unit(1:3, "npc") + unit(1, "strwidth", "a")*4, 3)
```

---

valid.just 位置揃えを検証する

Description

位置揃えの指定が適正かを決定し，文字と数値の組み合わせを単一の位置揃え値に解消するためのユーティリティ関数.

Usage

```
valid.just(just)
resolveHJust(just, hjust)
resolveVJust(just, vjust)
```

Arguments

just	"left" のような文字列値か， 0のような数値としての位置揃え指定.
hjust	数値の水平位置揃え指定
vjust	数値の垂直位置揃え指定

Details

これらの関数は新しい grob クラスを書く時 validDetails メソッド中で有用かもしれない.

Value

位置揃えの数値表現 (例えば "left" は0, "right" は1，等 ...). もし位置揃えが適正でなければエラーを与える.

Author(s)

Paul Murrell

validDetails グリッドの *grob* の検証のカスタム化

Description

この総称的なフック関数はグリッドの grob が grob, gTree, grid.edit または editGrob を使い作られたり編集される度に呼び出される．これは grob (または gTree)から導かれた新しいクラスの検証をカスタム化するための機会を提供する．

Usage

```
validDetails(x)
```

Arguments

x	グリッドの grob.

Details

この関数は grob, gTree, grid.edit そして editGrob により呼び出される． grob または gTree から導かれたクラスに対して新しいクラスのスロットの値を検証するメソッドが書かれるべきである (例えば grid:::validDetails.axis を見よ).

grob と gTree に対する標準的なスロットは自動的に検証される (例えば grob に対する vp, gp スロットと加えて children，そして gTree に対する childrenvp スロット) ので，新しいクラスに固有のスロットだけを扱う必要があることを注意する．

Value

この関数は検証済みの grob を返さなければならない．

Author(s)

Paul Murrell

See Also

grid.edit

vpPath ビューポートの名前を連結する

Description

この関数は downViewport や seekViewport 中で使うためのビューポートのパスを作るのに使うことが出来る．

ビューポートパスは入れ子になったビューポート名のリストである．

Usage

```
vpPath(...)
```

Arguments

...	ビューポートの名前である文字列ベクトル.

Details

ビューポート名はビューポートツリー中で同じ親を共有するビューポート間でだけユニークである必要がある.

この関数はビューポートの親の名前(そしてその親の名前等々) を含むビューポートに対する指定を生成するのに使うことが出来る.

対話的な使用に対しては，パスを直接指定することが可能であるが，さもなければグリッドの将来のバージョンでパス分離記号が変更される場合に備えこの関数を使うことを強く勧める.

Value

vpPath オブジェクト.

See Also

viewport, pushViewport, popViewport, downViewport, seekViewport, upViewport

Examples

```
vpPath("vp1", "vp2")
```

---

widthDetails　　グリッドの *grob* の幅と高さ

Description

この総称的関数はグリッドの grob のサイズを決定するのに使われる.

Usage

```
widthDetails(x)
heightDetails(x)
ascentDetails(x)
descentDetails(x)
```

Arguments

x	グリッドの grob.

Details

これらの関数は "grobwidth" と "grobheight" 単位の計算中に呼び出される． grob のサイズが決定できるような箇所では grob や gTree から導かれたクラスに対するメソッドが書かれるべきである (例えば grid:::widthDetails.frame を見よ).

grob の上部高さは既定では grob の高さであり， grob の下部高さは既定ではラベルが単一の文字列値か表現式の場合を除いてはゼロである.

Value

単位オブジェクト.

Author(s)

Paul Murrell

See Also

absolute.size.

Working with Viewports
グリッドのビューポートツリーの保守と移動

Description

グリッドは入れ子の描画文脈であるビューポートのツリーを維持する.

これらの関数はビューポートの追加と削除そしてツリー内のビューポートを移動する手段を提供する.

Usage

```
pushViewport(..., recording=TRUE)
popViewport(n = 1, recording=TRUE)
downViewport(name, strict=FALSE, recording=TRUE)
seekViewport(name, recording=TRUE)
upViewport(n = 1, recording=TRUE)
```

Arguments

...	一つもしくはそれ以上のクラス "viewport" のオブジェクト.
n	幾つのビューポートをポップするかまたは上昇移動するかを指定する整数値. 特殊な値 0 はルートのすぐ上のビューポートをポップしたりまたはそれへ移動することを指示する.
name	ツリー中のビューポートを特定する文字列値.
strict	vpPath が正確にマッチされるべきかどうかを指示するブール値.
recording	ビューポート捜査がグリッドのディスプレイリスト上に記録されるべきかどうかを指示する論理値.

Details

viewport() 関数で作られるオブジェクトは単に描画内容の記述である. ビューポートオブジェクトはそれが何らかの描画効果を持つ前にビューポートツリーにプッシュされる必要がある.

ビューポートツリーは常に(システムにより作られる)単一のルートビューポートを持ち, これはデバイス全体 (と既定のグラフィカルパラメータ設定)に対応する. ビューポートはツリーに pushViewport() を用いて加えることが出来, popViewport() を用いてツリーから削除できる.

常に唯一つの現在のビューポートがあり，これはビューポートツリー中の現在の位置である．全ての描画とビューポート捜査は現在のビューポートに対して相対的である．ビューポートがプッシュされるとそれが現在のビューポートになる．ビューポートがポップされると親のビューポートが現在のビューポートになる．現在のビューポートをビューポートツリーから削除することなしに，現在のビューポートの親に移動するには upViewport を使う．ビューポートツリーを更に下ってあるビューポートに移動するには downViewport を使い，ツリー内のどこかにあるビューポートに移動するには seekViewport を使う．

もしビューポートがプッシュされツリー内の同じレベルのビューポートと同じ name を持つと，それはツリー内の既存のビューポートを置き換える．

Value

downViewport はそれが下降したビューポートの数を返す．

これは depth <- downViewport() そして upViewport(depth) の様な操作で出発点に戻るために役に立つ．

Author(s)

Paul Murrell

See Also

viewport と vpPath.

Examples

```
# 同じビューポートを数回プッシュする
grid.newpage()
vp <- viewport(width=0.5, height=0.5)
pushViewport(vp)
grid.rect(gp=gpar(col="blue"))
grid.text("Quarter of the device",
  y=unit(1, "npc") - unit(1, "lines"), gp=gpar(col="blue"))
pushViewport(vp)
grid.rect(gp=gpar(col="red"))
grid.text("Quarter of the parent viewport",
  y=unit(1, "npc") - unit(1, "lines"), gp=gpar(col="red"))
popViewport(2)
# 幾つかのビューポートをプッシュしそれからそれらの間を移動する
grid.newpage()
grid.rect(gp=gpar(col="grey"))
grid.text("Top-level viewport",
  y=unit(1, "npc") - unit(1, "lines"), gp=gpar(col="grey"))
if (interactive()) Sys.sleep(1.0)
pushViewport(viewport(width=0.8, height=0.7, name="A"))
grid.rect(gp=gpar(col="blue"))
grid.text("1. Push Viewport A",
  y=unit(1, "npc") - unit(1, "lines"), gp=gpar(col="blue"))
if (interactive()) Sys.sleep(1.0)
pushViewport(viewport(x=0.1, width=0.3, height=0.6,
  just="left", name="B"))
grid.rect(gp=gpar(col="red"))
grid.text("2. Push Viewport B (in A)",
  y=unit(1, "npc") - unit(1, "lines"), gp=gpar(col="red"))
```

```
if (interactive()) Sys.sleep(1.0)
upViewport(1)
grid.text("3. Up from B to A",
  y=unit(1, "npc") - unit(2, "lines"), gp=gpar(col="blue"))
if (interactive()) Sys.sleep(1.0)
pushViewport(viewport(x=0.5, width=0.4, height=0.8,
  just="left", name="C"))
grid.rect(gp=gpar(col="green"))
grid.text("4. Push Viewport C (in A)",
  y=unit(1, "npc") - unit(1, "lines"), gp=gpar(col="green"))
if (interactive()) Sys.sleep(1.0)
pushViewport(viewport(width=0.8, height=0.6, name="D"))
grid.rect()
grid.text("5. Push Viewport D (in C)",
  y=unit(1, "npc") - unit(1, "lines"))
if (interactive()) Sys.sleep(1.0)
upViewport(0)
grid.text("6. Up from D to top-level",
  y=unit(1, "npc") - unit(2, "lines"), gp=gpar(col="grey"))
if (interactive()) Sys.sleep(1.0)
downViewport("D")
grid.text("7. Down from top-level to D",
  y=unit(1, "npc") - unit(2, "lines"))
if (interactive()) Sys.sleep(1.0)
seekViewport("B")
grid.text("8. Seek from D to B",
  y=unit(1, "npc") - unit(2, "lines"), gp=gpar(col="red"))
pushViewport(viewport(width=0.9, height=0.5, name="A"))
grid.rect()
grid.text("9. Push Viewport A (in B)",
  y=unit(1, "npc") - unit(1, "lines"))
if (interactive()) Sys.sleep(1.0)
seekViewport("A")
grid.text("10. Seek from B to A (in ROOT)",
  y=unit(1, "npc") - unit(3, "lines"), gp=gpar(col="blue"))
if (interactive()) Sys.sleep(1.0)
seekViewport(vpPath("B", "A"))
grid.text("11. Seek from\nA (in ROOT)\nto A (in B)")
popViewport(0)
```

---

xDetails　　グリッドの *grob* の境界

Description

これらの総称的関数はグリッドの grob の境界上の位置を決めるために使われる.

Usage

```
xDetails(x, theta)
yDetails(x, theta)
```

Arguments

x	グリッドの grob.
theta	度単位の数値角度で， 3時方向から反時計回りで測られる． または 以下の文字列の一つ： "north", "east", "west", "south".

Details

グリッドの grob の境界は grob の中心から角度 theta の線を引きそれを grob の凸包と交わらせることで決定される (プリミティブな grob に対しては，中心は x 方向と y 方向の最小値と最大値の真ん中として決定される).

これらの関数は grobX と grobY 関数で作られる "grobx" と "groby" 単位の計算で呼び出される． grob の境界が決定できるような箇所では grob や gTree から導かれたクラスに対するメソッドが書かれるべきである.

Value

単位オブジェクト.

Author(s)

Paul Murrell

See Also

grobX, grobY.

---

xsplinePoints — *X*スプライン*(*またはベジエ曲線*)*を描くのに使われる点を返す.

Description

Xスプライン(または Bezier 曲線)を描く代わりに，書くのに使われる点を返す．この関数はXスプラインに対する一連の線分を描くのに使われる一連の点を返す．これは例えば曲線のクリップの様な，Xスプライン曲線の事後処理に便利である.

Usage

```
xsplinePoints(x)
bezierPoints(x)
```

Arguments

x	xsplineGrob() 関数により作られるような Xsplinegrob (または bezierGrob() 関数が作るような beziergrob).

Details

この関数が返す点はこの関数が呼び出された時の内容の強制的な描画に対してだけ意味がある.

Value

幾つXスプラインを描くかに依存する．もし唯一つなら，共に単位オブジェクト(インチ単位) である x と y という名前の二つの成分を持つリスト．もし幾つかのXスプラインが書かれるならばこの関数の結果はリストのリストである．

Author(s)

Paul Murrell

See Also

xsplineGrob と bezierGrob

Examples

```
grid.newpage()
xsg <- xsplineGrob(c(.1, .1, .9, .9), c(.1, .9, .9, .1), shape=1)
grid.draw(xsg)
trace <- xsplinePoints(xsg)
grid.circle(trace$x, trace$y, default.units="inches", r=unit(.5, "mm"))

grid.newpage()
vp <- viewport(width=.5)
xg <- xsplineGrob(x=c(0, .2, .4, .2, .5, .7, .9, .7),
                  y=c(.5, 1, .5, 0, .5, 1, .5, 0),
                  id=rep(1:2, each=4),
                  shape=1,
                  vp=vp)
grid.draw(xg)
trace <- xsplinePoints(xg)
pushViewport(vp)
invisible(lapply(trace, function(t) grid.lines(t$x, t$y, gp=gpar(col="red"))))
popViewport()

grid.newpage()
bg <- bezierGrob(c(.2, .2, .8, .8), c(.2, .8, .8, .2))
grid.draw(bg)
trace <- bezierPoints(bg)
grid.circle(trace$x, trace$y, default.units="inches", r=unit(.5, "mm"))
```

Index