Package 'gridSVG'

October 31, 2025

Title Export 'grid' Graphics as SVG

Version 1.7-7

Description Functions to export graphics drawn with package grid to SVG format. Additional functions provide access to SVG features that are not available in standard R graphics, such as hyperlinks, animation, filters, masks, clipping paths, and gradient and pattern fills.

Imports grDevices, graphics, utils, methods, grid, jsonlite, XML

Suggests lattice

License GPL

NeedsCompilation no

Author Paul Murrell [cre, aut], Simon Potter [aut]

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

Repository CRAN

Date/Publication 2025-10-31 06:10:59 UTC

Contents

animate *Convert animation specifications to SVG elements.*

Description

This function is used to generate <animate> elements based on animation information on a grob. It is generic so new grob classes can write their own methods.

Usage

```
animate(x, dev)
```

Arguments

x	A grob.
dev	A graphics device.

Details

This function is not called directly by the user. It is exposed so that new grob classes can easily write their own methods which call existing methods for standard grobs.

Author(s)

Paul Murrell

animUnit *Generate a set of animation values.*

Description

These functions can be used to generate a set of values for use with grid.animate() to animate some feature of a grob.

Usage

```
animUnit(x, timeid = NULL, id = NULL)
animValue(x, timeid = NULL, id = NULL)
as.animUnit(x, ...)
as.animValue(x, ...)
```

Arguments

x	A set of animation values. Could be a numeric vector, a character vector, a unit vector, a matrix, a list of units.
timeid	A vector that associates each value of x with a time point.
id	A vector that associates each value of x with a different (numeric) identifier.
...	For future use.

Details

A set of animation values is ultimately either a numeric or character vector OR a unit vector. Subsets of the animation values can be defined per time point, or per identifier, or both.

The as functions allow animation values to be specified as matrices or lists, which are converted to formal animation value sets. The grid.animate() function calls these functions so the conversion typically happens automatically.

These functions should only have to be called directly in relatively complex cases where multiple values need to be specified per time point AND per identifier.

Value

An animUnit or animValue object.

Author(s)

Paul Murrell

See Also

grid.animate

Examples

```
require(grid)

animValue(c("visible", "hidden"))
animUnit(unit(1:24, "in"),
         timeid=rep(1:3, each=8),
         id=rep(1:2, 12))
```

`Clipping Paths` *Create the definition of a non-rectangular clipping path.*

Description

A feature of SVG is that elements can be clipped to by more than just a rectangular region. Most graphical elements can be drawn. The purpose of these functions is to define a more sophisticated clipping path that will be applied until the current viewport (or context, see `popContext`) is popped.

Usage

```
clipPath(grob)
registerClipPath(label, clippath)
```

Arguments

`grob`	A grid grob.
`label`	A character identifier that will be used to reference this definition.
`clippath`	A `clipPath` object produced by `clipPath` that defines a clipping path region.

Details

A clipping path will be drawn within the current viewport at the time of definition (if the grob has no `vp` specified).

Most grobs can be used for clipping but there are some limitations on what will actually be used for clipping. In general though, anything that is drawn as the clipping path will have the union of its drawn regions become the new region that the current viewport (or grob) will clip to.

The limitations are as follows:

- Any viewport pushed by the clipping path grob will no longer clip to its contents. However, its clipping region will remain. This means that the clipping region for a pushed viewport will become the union of its contents and the viewport clipping region itself, instead of just the pushed viewport's clipping region.
- When drawing a `textGrob`, only character labels will be used, no `plotmath` expressions will be used.
- No `pointGrobs` are able to be used for clipping.
- Any operations that apply to containers (e.g. gpars, garnishing, animation), will no longer work. Any operations that are not applied to groups are unaffected. This affects in particular viewports, gTrees, and the familiar gridSVG grob grouping that occurs.

Value

None

Author(s)

Simon Potter

See Also

popContext, grid.clipPath, pushClipPath, grid.clip

Coordinate Conversion Functions

Functions for using an imported coordinate system

Description

These functions convert between different units. The conversion occurs within viewports unknown to grid, but imported to R via gridSVGCoords.

Usage

```
viewportConvertX(vpname, x, from, to = "svg")
viewportConvertY(vpname, x, from, to = "svg")
viewportConvertPos(vpname, x, y, from, to = "svg")
viewportConvertWidth(vpname, x, from, to)
viewportConvertHeight(vpname, x, from, to)
viewportConvertDim(vpname, w, h, from, to)
```

Arguments

vpname	The name of the viewport that the unit belongs within.
x, y, w, h	The size of the unit in from units.
from	The type of unit that x is.
to	The unit that x is being converted to.

Details

Although grid has conversion functions available, it only converts units relative to the current viewport. After writing out to SVG, we no longer have actual grid viewports to convert units within.

These functions are designed so that once coordinate information is loaded into gridSVG via gridSVGCoords, we can translate units within each of these viewports. Note: this requires that a gridSVG plot has had viewport information exported.

These functions can be used in much the same way as grid's unit conversion functions, the only difference being that we have a new unit, svg, which represents the size of a unit in SVG pixels.

The viewportConvertPos() and viewportConvertDim() functions are for use with a viewport that has a non-zero rotation (both viewportConvertX() and viewportConvertY() will fail in that situation and viewportConvertWidth() and viewportConvertHeight() will give a not very useful answer).

Value

A numeric vector containing a single value, the value of the new unit, or a list with components `x` and `y` for `viewportConvertPos()`, or a list with components `w` and `h` for `viewportConvertDim()`.

In the case of the `viewportConvertX` and `viewportConvertY` functions, we always return a value that is in terms of SVG pixels.

Author(s)

Simon Potter

`Coordinate System Import/Export`

Importing an external coordinate system

Description

This function is both a getter and a setter function for coordinate information imported from a plot unknown to the current R session.

Usage

```
gridSVGCoords(newcoords = NULL)
```

Arguments

`newcoords`	A named list (names are viewport names) of coordinate information, produced by `grid.export`.

Details

In order to translate between SVG coordinates and the coordinate system that `grid` understands, we first need to import the coordinate information exported from `grid.export`. We can then take the JSON representation of this coordinate information and import it as a named list via `fromJSON`. This can then initialise a coordinate system by passing that named list into `gridSVGCoords`.

We can supply new definitions of a viewport's coordinate system by simply passing in an appropriate list with information for that viewport.

All viewport coordinate system information can be wiped if a single `NA` value is passed in.

Value

If `newcoords` is `NULL`, then we get back a named list representing coordinate system information.

If we pass the named list representing a coordinate system into the function, we get no output. We also get no output if we pass in a single `NA` value.

Author(s)

Simon Potter

fe *Creating a generic filter effect*

Description

This function creates an object that contains all of the basic attributes that each filter effect inherits from. This is not intended to be used directly, instead it is to be used as a convenience function for building up filter effect objects.

Usage

```
fe(...,
   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", result = NULL)
```

Arguments

...	Further attributes to add to the object.
x	A numeric vector or unit object specifying x-location.
y	A numeric vector or unit object specifying y-location.
width	A numeric vector or unit object specifying width.
height	A numeric vector or unit object specifying height.
just	The justification of the pattern relative to its (x, y) location. If there are two values, the first value specifies horizontal justification and the second value specifies vertical justification. Possible string values are: "left", "right", "centre", "center", "bottom", and "top". For numeric values, 0 means left alignment and 1 means right alignment.
hjust	A numeric vector specifying horizontal justification. If specified, overrides the just setting.
vjust	A numeric vector specifying vertical justification. If specified, overrides the just setting.
default.units	A string indicating the default units to use if x, y, width, or height are only given as numeric vectors.
result	A character identifier, naming the result of the filter operation. The result can be used an an input to some filter effects.

Value

A filter.effect object.

Author(s)

Simon Potter

See Also

filterEffect

---

feBlend *Blend two objects together.*

Description

This filter composites two objects together using commonly used imaging software blending modes. It performs a pixel-wise combination of two input images.

Usage

```
feBlend(input1 = NA, input2 = NA,
        mode = c("normal", "multiply", "screen", "darken", "lighten"),
        ...)
```

Arguments

input1	Identifies an input for this filter primtive. See filterInputs.
input2	Identfies a second input for this filter primitive. See filterInputs.
mode	An image blending mode.
...	Further arguments to be passed onto fe.

Details

For more information about this primitive, consult the reference to the SVG specification.

Value

An fe.blend object.

Author(s)

Simon Potter

References

https://www.w3.org/TR/SVG/filters.html#feBlendElement

See Also

filterEffect, fe.

`feColorMatrix` *Apply a matrix transformation on colour values.*

Description

This filter applies a matrix transformation on the RGBA colour and alpha values of every pixel on the input graphics to produce a result with a new set of RGBA colour and alpha values.

Usage

```
feColorMatrix(input = NA,
              type = c("matrix", "saturate",
                       "hueRotate", "luminanceToAlpha"),
              values = NULL, ...)
```

Arguments

`input`	Identifies an input for this filter primtive. See `filterInputs`.
`type`	Indicates the type of matrix operation. The keyword `"matrix"` indicates that a full 5x4 matrix of values will be provided. The other keywords represent convenience shortcuts to allow commonly used color operations to be performed without specifying a complete matrix.
`values`	The contents of `values` depend on what `type` is: • `matrix` A 5x4 matrix of numeric values. • `saturate` A single element numeric vector whose value is between 0 and 1. • `hueRotate` A single element numeric vector whose value represents degrees. • `luminanceToAlpha` Should be left as `NULL` as there are no applicable values.
`...`	Further arguments to be passed onto `fe`.

Details

For more information about this primitive, consult the reference to the SVG specification.

Value

An `fe.color.matrix` object.

Author(s)

Simon Potter

References

https://www.w3.org/TR/SVG/filters.html#feColorMatrixElement

See Also

filterEffect, fe.

---

feComponentTransfer *Perform Colour Component-wise Remapping.*

Description

This filter primitive performs component-wise remapping of data by taking a colour transfer function, and applying that to the set of RGBA colour components.

It allows operations like brightness adjustment, contrast adjustment, colour balance or thresholding.

The calculations are performed on non-premultiplied colour values. If the input graphics consists of premultiplied colour values, those values are automatically converted into non-premultiplied colour values for this operation. (Note that the undoing and redoing of the premultiplication can be avoided if alpha transfer function is the identity transform and all alpha values on the source graphic are set to 1.)

Usage

```
feComponentTransfer(input = NA, transfers = NULL, ...)
addComponentFunction(ct, channel = c("R", "G", "B", "A"), func)
transferFunction(type = c("identity", "table", "discrete",
                          "linear", "gamma"),
                 tableValues = numeric(),
                 slope = 1, intercept = 0,
                 amplitude = 1, exponent = 1, offset = 0)
```

Arguments

input	Identifies an input for this filter primtive. See filterInputs.
transfers	A named list of transfer.function objects (produced by transferFunction). The name for each element of the list should be one of R, G, B or A.
...	Further arguments to be passed onto fe.
ct	An fe.component.transfer object, produced by feComponentTransfer.
channel	The colour channel that func will be applied to.
func	A transfer.function object, produced by transferFunction.
type	Indicates the type of component transfer function. The type of function determines the applicability of the other arguments.
tableValues	When type is "table", this is a list of values which define the lookup table.
slope	When type is "linear", the slope of the linear function.
intercept	When type is "linear", the intercept of the linear function.
amplitude	When type is "gamma", the amplitude of the gamma function.
exponent	When type is "gamma", the exponent of the gamma function.
offset	When type is "gamma", the offset of the gamma function.

Details

For more information about this primitive, consult the references to the SVG specification.

Value

For `feComponentTransfer`, an `fe.component.transfer` object.

For `addComponentFunction`, none.

For `transferFunction`, a `transfer.function` object.

Author(s)

Simon Potter

References

https://www.w3.org/TR/SVG/filters.html#feComponentTransferElement, https://www.w3.org/TR/SVG/filters.html#feFuncRElement

See Also

`filterEffect`, `fe`.

---

feComposite *Combine images using Porter-Duff operations.*

Description

This filter performs the combination of the two input images pixel-wise in image space using one of the Porter-Duff compositing operations.

The arithmetic operation is useful for combining the output from the `feDiffuseLighting` and `feSpecularLighting` filter effects with texture data. It is also useful for implementing dissolve.

Usage

```
feComposite(input1 = NA, input2 = NA,
            operator = c("over", "in", "out", "atop",
                         "xor", "arithmetic"),
            k1 = 0, k2 = 0, k3 = 0, k4 = 0, ...)
```

Arguments

`input1`	Identifies an input for this filter primtive. See `filterInputs`.
`input2`	Identfies a second input for this filter primitive. See `filterInputs`.
`operator`	The compositing operation that is to be performed. All of the `operator` types except `"arithmetic"` match the corresponding operation as described in the referenced Porter-Duff text. The arithmetic operator is described in the referenced SVG specification.

k1	A numeric value. Only applicable if operator is "arithmetic".
k2	A numeric value. Only applicable if operator is "arithmetic".
k3	A numeric value. Only applicable if operator is "arithmetic".
k4	A numeric value. Only applicable if operator is "arithmetic".
...	Further arguments to be passed onto fe.

Details

For more information about this primitive, consult the reference to the SVG specification.

Value

An fe.composite object.

Author(s)

Simon Potter

References

https://www.w3.org/TR/SVG/filters.html#feCompositeElement

Compositing Digital Images, T. Porter and T. Duff. SIGGRAPH '84 Conference Proceedings, Association for Computing Machinery, Volume 18, Number 3, July 1984.

See Also

filterEffect, fe.

---

feConvolveMatrix *Apply a matrix convolution filter effect.*

Description

A convolution combines pixels in the input image with neighbouring pixels to produce a resulting image. A wide variety of imaging operations can be achieved through convolutions, including blurring, edge detection, sharpening, embossing and beveling.

Usage

```
feConvolveMatrix(input = NA, order = 3,
                 kernelMatrix = matrix(),
                 divisor = 1, bias = 0,
                 targetX = 1, targetY = 1,
                 edgeMode = c("duplicate", "wrap", "none"),
                 kernelUnitLength = NA, preserveAlpha = FALSE,
                 ...)
```

Arguments

`input`	Identifies an input for this filter primtive. See `filterInputs`.
`order`	A numeric vector of length 1 or 2. Indicates the number of cells in each dimension for `kernelMatrix`. The values provided must be integers greater than zero. The first number (orderX), indicates the number of columns in the matrix. The second number (orderY), indicates the number of rows in the matrix. If this is a vector of length one then the number of rows is assumed to be same as the number of columns specified.
`kernelMatrix`	The kernel matrix for the convolution. The number of entries must correspond with the values given by `order`.
`divisor`	After applying the `kernelMatrix` to the input image to yield a number, that number is divided by `divisor` to yield the final destination colour value. A divisor that is the sum of all the matrix values tends to have an evening effect on the overall colour intensity of the result. It is an error to specify a divisor of zero. The default value is the sum of all values in kernelMatrix, with the exception that if the sum is zero, then the divisor is set to 1.
`bias`	After applying the `kernelMatrix` to the input image to yield a number and applying the `divisor`, the `bias` attribute is added to each component. One application of `bias` is when it is desirable to have 0.5 gray value be the zero response of the filter. The bias property shifts the range of the filter. This allows representation of values that would otherwise be clamped to 0 or 1.
`targetX`	Determines the positioning in X of the convolution matrix relative to a given target pixel in the input image. The leftmost column of the matrix is column number zero. The value must be such that: $0 <= targetX < orderX$. By default, the convolution matrix is centered in X over each pixel of the input image (i.e., $targetX = \lfloor orderX/2 \rfloor$.
`targetY`	Determines the positioning in Y of the convolution matrix relative to a given target pixel in the input image. The topmost row of the matrix is row number zero. The value must be such that: $0 <= targetY < orderY$. By default, the convolution matrix is centered in Y over each pixel of the input image (i.e., $targetY = \lfloor orderY/2 \rfloor$.
`edgeMode`	Determines how to extend the input image as necessary with colour values so that the matrix operations can be applied when the kernel is positioned at or near the edge of the input image. • `"duplicate"` indicates that the input image is extended along each of its borders as necessary by duplicating the colour values at the given edge of the input image. • `"wrap"` indicates that the input image is extended by taking the colour values from the opposite edge of the image. • `"none"` indicates that the input image is extended with pixel values of zero for R, G, B and A.
`kernelUnitLength`	The first number is the `dx` value. The second number is the `dy` value. If the `dy` value is not specified, it defaults to the same value as `dx`. Indicates the intended distance in current filter units (i.e., units as determined by the value of the filter

effect container's primitiveUnits) between successive columns and rows, respectively, in the kernelMatrix. By specifying value(s) for kernelUnitLength, the kernel becomes defined in a scalable, abstract coordinate system. If kernelUnitLength is not specified, the default value is one pixel in the offscreen bitmap, which is a pixel-based coordinate system, and thus potentially not scalable.

preserveAlpha — A value of FALSE indicates that the convolution will apply to all channels, including the alpha channel.

A value of TRUE indicates that the convolution will only apply to the colour channels. In this case, the filter will temporarily unpremultiply the colour component values, apply the kernel, and then re-premultiply at the end.

... — Further arguments to be passed onto fe.

Details

For more information about this primitive, consult the reference to the SVG specification.

Value

An fe.convolve.matrix object.

Author(s)

Simon Potter

References

https://www.w3.org/TR/SVG/filters.html#feConvolveMatrixElement

See Also

filterEffect, fe.

---

feDiffuseLighting *Light an image using the alpha channel as a bump map.*

Description

This filter primitive lights an image using the alpha channel as a bump map. The resulting image is an RGBA opaque image based on the light colour with alpha = 1 everywhere. The lighting calculation follows the standard diffuse component of the Phong lighting model. The resulting image depends on the light colour, light position and surface geometry of the input bump map.

Usage

```
feDiffuseLighting(input = NA,
                  surfaceScale = 1, diffuseConstant = 1,
                  kernelUnitLength = NA, col = "white",
                  lightSource = NULL, ...)
```

Arguments

input	Identifies an input for this filter primtive. See filterInputs.
surfaceScale	Scale applied to the input alpha surface.
diffuseConstant	kd in the Phong lighting model. Must be non-negative.
kernelUnitLength	The first number is the dx value. The second number is the dy value. If the dy value is not specified, it defaults to the same value as dx. Indicates the intended distance in current filter units (i.e., units as determined by the value of parent filter container's primitiveUnits) for dx and dy, respectively, in the surface normal calculation formulas. By specifying value(s) for kernelUnitLength, the kernel becomes defined in a scalable, abstract coordinate system. If kernelUnitLength is not specified, the dx and dy values should represent very small deltas relative to a given (x,y) position, which might be implemented in some cases as one pixel in the intermediate image offscreen bitmap, which is a pixel-based coordinate system, and thus potentially not scalable.
col	The colour to apply to the light from lightSource.
lightSource	A light source object, produced by one of feDistantLight, fePointLight, or feSpotLight.
...	Further arguments to be passed onto fe.

Details

For more information about this primitive, consult the reference to the SVG specification.

Value

An fe.diffuse.lighting object.

Author(s)

Simon Potter

References

https://www.w3.org/TR/SVG/filters.html#feDiffuseLightingElement

See Also

filterEffect fe, feDistantLight, fePointLight, feSpotLight.

feDisplacementMap *Displace pixel values from a filter input.*

Description

This filter primitive uses the pixels values from the image from input2 to spatially displace the image from input1.

Usage

```
feDisplacementMap(input1 = NA, input2 = NA,
                  scale = 0,
                  xChannelSelector = c("A", "R", "G", "B"),
                  yChannelSelector = c("A", "R", "G", "B"),
                  ...)
```

Arguments

input1	Identifies an input for this filter primtive. See filterInputs.
input2	Identfies a second input for this filter primitive. See filterInputs.
scale	Displacement scale factor. The amount is expressed in the coordinate system established by attribute primitiveUnits on the parent filter container. When the value of scale is 0, this operation has no effect on the source image.
xChannelSelector	Indicates which channel from input2 to use to displace the pixels in input along the x-axis.
yChannelSelector	Indicates which channel from input2 to use to displace the pixels in input along the y-axis.
...	Further arguments to be passed onto fe.

Details

For more information about this primitive, consult the reference to the SVG specification.

Value

An fe.displacement.map object.

Author(s)

Simon Potter

References

https://www.w3.org/TR/SVG/filters.html#feDisplacementMapElement

See Also

filterEffect, fe.

---

feDistantLight *Create a Distant Light Source*

Description

This filter primitive defines a distant light source that can be used within a lighting filter primitive: feDiffuseLighting or feSpecularLighting.

Usage

```
feDistantLight(azimuth = 0, elevation = 0, ...)
```

Arguments

azimuth	Direction angle for the light source on the x-y plane (clockwise), in degrees from the x axis.
elevation	Direction angle for the light source from the x-y plane towards the z axis, in degrees. Note the positive z-axis points towards the viewer of the content.
...	Further arguments to be passed onto fe.

Details

For more information about this primitive, consult the reference to the SVG specification.

Value

An fe.distant.light object.

Author(s)

Simon Potter

References

https://www.w3.org/TR/SVG/filters.html#feDistantLightElement

See Also

filterEffect, fe, feDiffuseLighting, feSpecularLighting.

feFlood *Create and fill a rectangular region.*

Description

This filter primitive creates a rectangle filled with a specified colour. The rectangle is as large as the filter primitive subregion established by the x, y, width and height attributes passed onto fe via ....

Usage

```
feFlood(col = "black", ...)
```

Arguments

col	A colour that will be used to fill the filter region.
...	Further arguments to be passed onto fe.

Details

For more information about this primitive, consult the reference to the SVG specification.

Value

An fe.flood object.

Author(s)

Simon Potter

References

https://www.w3.org/TR/SVG/filters.html#feFloodElement

See Also

filterEffect, fe.

feGaussianBlur *Apply a Gaussian blur to an image.*

Description

This filter effect primitive performs a Gaussian blur on the input image.

Usage

```
feGaussianBlur(input = NA, sd = 0, ...)
```

Arguments

input	Identifies an input for this filter primtive. See filterInputs.
sd	The value of sd can be a numeric vector with either one or two elements. If two numbers are provided, the first number represents a standard deviation value along the x-axis of the current coordinate system and the second value represents a standard deviation in Y. If one number is provided, then that value is used for both X and Y.
...	Further arguments to be passed onto fe.

Details

For more information about this primitive, consult the reference to the SVG specification.

Value

An fe.gaussian.blur object.

Author(s)

Simon Potter

References

https://www.w3.org/TR/SVG/filters.html#feGaussianBlurElement

See Also

filterEffect, fe.

feImage *Draw a referred image.*

Description

This filter effect primitive refers to a graphic external to this filter container, which is loaded or rendered into an RGBA raster and becomes the result of the filter effect primitive.

Usage

```
feImage(preserveAspectRatio = "xMidYMid meet", href = "", ...)
```

Arguments

preserveAspectRatio	See references for appropriate values and behaviour.
href	A URL reference to a stand-alone image resource such as a JPEG, PNG or SVG file. e.g. http://example.com/img.jpg
...	Further arguments to be passed onto fe.

Details

For more information about this primitive, consult the reference to the SVG specification.

Value

An fe.image object.

Author(s)

Simon Potter

References

https://www.w3.org/TR/SVG/filters.html#feImageElement https://www.w3.org/TR/SVG/coords.html#PreserveAspectRatioAttribute

See Also

filterEffect, fe.

feMerge *Composite image layers together.*

Description

This filter primitive composites input image layers on top of each other using the "over" operator with "input1" (corresponding to the first child merge node) on the bottom and the last specified input, "inputN" (corresponding to the last child merge node), on top.

Usage

```
feMerge(mergeNodes = NULL, ...)
addMergeNode(fe, mergeNode, after = NA)
feMergeNode(input = NA)
```

Arguments

mergeNodes	A list of fe.merge.node objects, produced by feMergeNode.
...	Further arguments to be passed onto fe.
fe	An fe.merge object, created by feMerge.
mergeNode	An fe.merge.node object, created by feMerge.
after	The position to add mergeNode to in the list of fe's children. When NA, appends to the end of the list of children.
input	Identifies an input for this filter primtive. See filterInputs.

Details

If you wish to add more merge nodes after an fe.merge object has been created, use addMergeNode to add merge nodes to the filter primitive.

For more information about the feMerge primitive, consult the reference to the SVG specification.

Value

For feMerge, an fe.merge object.

For addMergeNode, an fe.merge object.

For feMergeNode, an fe.merge.node object.

Author(s)

Simon Potter

References

https://www.w3.org/TR/SVG/filters.html#feMergeElement

See Also

filterEffect, fe.

---

feMorphology *"Fatten" or "thin" artwork.*

Description

This filter primitive performs "fattening" or "thinning" of artwork. It is particularly useful for fattening or thinning an alpha channel.

Usage

```
feMorphology(input = NA, operator = c("erode", "dilate"),
             radius = unit(0, "npc"), default.units = "npc", ...)
```

Arguments

input	Identifies an input for this filter primtive. See filterInputs.
operator	A keyword indicating whether to erode (i.e., thin) or dilate (fatten) the source graphic, input.
radius	The radius (or radii) for the operation. If two values are provided, the first value represents a x-radius and the second value represents a y-radius. If one radius is provided, then that value is used for both xand y.
default.units	A string indicating the default units to use if radius is only given as a numeric vector.
...	Further arguments to be passed onto fe.

Details

For more information about this primitive, consult the reference to the SVG specification.

Value

An fe.morphology object.

Author(s)

Simon Potter

References

https://www.w3.org/TR/SVG/filters.html#feMorphologyElement

See Also

filterEffect, fe.

`feOffset` *Offset an input image relative to its current position.*

Description

This filter primitive offsets the input image relative to its current position in the image space by the specified vector.

This is important for effects like drop shadows.

Usage

```
feOffset(input = NA,
         dx = unit(0, "npc"), dy = unit(0, "npc"),
         default.units = "npc", ...)
```

Arguments

`input`	Identifies an input for this filter primtive. See `filterInputs`.
`dx`	The amount to offset `input` by along the x-axis.
`dy`	The amount to offset `input` by along the y-axis.
`default.units`	A string indicating the default units to use if `dx` or `dy` are only given as numeric vectors.
`...`	Further arguments to be passed onto `fe`.

Details

For more information about this primitive, consult the reference to the SVG specification.

Value

An `fe.offset` object.

Author(s)

Simon Potter

References

https://www.w3.org/TR/SVG/filters.html#feOffsetElement

See Also

`filterEffect`, `fe`.

fePointLight *Create a Point Light Source*

Description

This filter primitive defines a point light source that can be used within a lighting filter primitive: feDiffuseLighting or feSpecularLighting.

Usage

```
fePointLight(z = unit(0, "npc"), default.units = "npc", zdim = "x", ...)
```

Arguments

z	A numeric vector or unit object specifying z-location.
default.units	A string indicating the default units to use if z if given as a numeric vector.
zdim	Either "x" or "y". Determines the dimension to which z will be located relative to. This is necessary because R graphics has no concept of a z-dimension.
...	Further arguments to be passed onto fe.

Details

For more information about this primitive, consult the reference to the SVG specification.

Value

An fe.point.light object.

Author(s)

Simon Potter

References

http://www.w3.org/TR/SVG/filters.html#fePointLightElement

See Also

filterEffect, fe, feDiffuseLighting, feSpecularLighting.

`feSpecularLighting` *Light an image using the alpha channel as a bump map.*

Description

This filter primitive lights a source graphic using the alpha channel as a bump map. The resulting image is an RGBA image based on the light colour. The lighting calculation follows the standard specular component of the Phong lighting model. The resulting image depends on the light colour, light position and surface geometry of the input bump map. The result of the lighting calculation is added. The filter primitive assumes that the viewer is at infinity in the z direction (i.e., the unit vector in the eye direction is (0,0,1) everywhere).

This filter primitive produces an image which contains the specular reflection part of the lighting calculation. Such a map is intended to be combined with a texture using the add term of the arithmetic method in `feComposite`. Multiple light sources can be simulated by adding several of these light maps before applying it to the texture image.

Usage

```
feSpecularLighting(input = NA,
                   surfaceScale = 1, specularConstant = 1,
                   specularExponent = 1, kernelUnitLength = NA,
                   col = "white", lightSource = NULL, ...)
```

Arguments

`input`	Identifies an input for this filter primtive. See `filterInputs`.
`surfaceScale`	Scale applied to the input alpha surface.
`specularConstant`	kd in the Phong lighting model. Must be non-negative.
`specularExponent`	Numeric exponent for specular term, larger is more "shiny". Range [1,128].
`kernelUnitLength`	The first number is the dx value. The second number is the dy value. If the dy value is not specified, it defaults to the same value as dx. Indicates the intended distance in current filter units (i.e., units as determined by the value of parent filter container's `primitiveUnits`) for dx and dy, respectively, in the surface normal calculation formulas. By specifying value(s) for `kernelUnitLength`, the kernel becomes defined in a scalable, abstract coordinate system. If `kernelUnitLength` is not specified, the dx and dy values should represent very small deltas relative to a given (x,y) position, which might be implemented in some cases as one pixel in the intermediate image offscreen bitmap, which is a pixel-based coordinate system, and thus potentially not scalable.
`col`	The colour to apply to the light from `lightSource`.
`lightSource`	A light source object, produced by one of `feDistantLight`, `fePointLight`, or `feSpotLight`.
`...`	Further arguments to be passed onto `fe`.

Details

For more information about this primitive, consult the reference to the SVG specification.

Value

An `fe.specular.lighting` object.

Author(s)

Simon Potter

References

https://www.w3.org/TR/SVG/filters.html#feSpecularLightingElement

See Also

`filterEffect` `fe`, `feDistantLight`, `fePointLight`, `feSpotLight`.

`feSpotLight` *Create a Spot Light Source*

Description

This filter primitive defines a spot light source that can be used within a lighting filter primitive: `feDiffuseLighting` or `feSpecularLighting`.

Usage

```
feSpotLight(x = unit(0, "npc"), y = unit(0, "npc"), z = unit(0, "npc"),
            pointsAtX = unit(1, "npc"), pointsAtY = unit(1, "npc"),
            pointsAtZ = unit(0, "npc"), zdim = "x",
            default.units = "npc", specularExponent = 1,
            limitingConeAngle = NA, ...)
```

Arguments

`x`	A numeric vector or unit object specifying the x-location of the light source.
`y`	A numeric vector or unit object specifying the y-location of the light source.
`z`	A numeric vector or unit object specifying the z-location of the light source.
`pointsAtX`	A numeric vector or unit object specifying the x-location that the light points at.
`pointsAtY`	A numeric vector or unit object specifying the y-location that the light points at.
`pointsAtZ`	A numeric vector or unit object specifying the z-location that the light points at.
`zdim`	Either `"x"` or `"y"`. Determines the dimension to which `z` and `pointsAtZ` will be located relative to. This is necessary because R graphics has no concept of a z-dimension.

default.units A string indicating the default units to use if x, y, z, pointsAtX, pointsAtY, pointsAtZ are only given as numeric vectors.

specularExponent
Exponent value controlling the focus for the light source.

limitingConeAngle
If NA, no limiting cone is applied, otherwise a limiting cone which restricts the region where the light is projected. No light is projected outside the cone. limitingConeAngle represents the angle in degrees between the spot light axis (i.e. the axis between the light source and the point to which it is pointing at) and the spot light cone.

... Further arguments to be passed onto fe.

Details

For more information about this primitive, consult the reference to the SVG specification.

Value

An fe.spot.light object.

Author(s)

Simon Potter

References

https://www.w3.org/TR/SVG/filters.html#feSpotLightElement

See Also

filterEffect, fe, feDiffuseLighting, feSpecularLighting.

---

feTile *Fill a rectangle with a tiled pattern of an input image.*

Description

This filter primitive fills a target rectangle with a repeated, tiled pattern of an input image. The target rectangle is as large as the filter primitive subregion established by the x, y, width and height arguments that are passed onto fe by feTile.

Usage

```
feTile(input = NA, ...)
```

Arguments

input	Identifies an input for this filter primtive. See filterInputs.
...	Further arguments to be passed onto fe.

Details

For more information about this primitive, consult the reference to the SVG specification.

Value

An fe.tile object.

Author(s)

Simon Potter

References

https://www.w3.org/TR/SVG/filters.html#feTileElement

See Also

filterEffect, fe.

---

feTurbulence *Create an image using the Perlin turbulence function.*

Description

This filter primitive creates an image using the Perlin turbulence function. It allows the synthesis of artificial textures like clouds or marble.

Usage

```
feTurbulence(baseFrequency = 0, numOctaves = 1,
             seed = 1, stitchTiles = FALSE,
             type = c("turbulence", "fractalNoise"), ...)
```

Arguments

baseFrequency	The base frequency (frequencies) parameter(s) for the noise function. If a two element numeric vector is provided, the first number represents a base frequency in the X direction and the second value represents a base frequency in the Y direction. If one number is provided, then that value is used for both X and Y.
numOctaves	The numOctaves parameter for the noise function.
seed	The starting number for the pseudo random number generator.

stitchTiles — If `stitchTiles` is `FALSE`, no attempt it made to achieve smooth transitions at the border of tiles which contain a turbulence function. Sometimes the result will show clear discontinuities at the tile borders.

If `stitchTiles` is `TRUE`, then the user agent will automatically adjust `baseFrequency`-x and `baseFrequency`-y values such that the `feTurbulence`'s width and height (i.e., the width and height of the current subregion) contains an integral number of the Perlin tile width and height for the first octave. The `baseFrequency` will be adjusted up or down depending on which way has the smallest relative (not absolute) change as follows: Given the frequency, calculate $lowFreq = \lfloor width * frequency \rfloor / width$ and $hiFreq = \lceil width * frequency \rceil / width$. If $frequency/lowFreq < hiFreq/frequency$ then use `lowFreq`, else use `hiFreq`. While generating turbulence values, generate lattice vectors as normal for Perlin Noise, except for those lattice points that lie on the right or bottom edges of the active area (the size of the resulting tile). In those cases, copy the lattice vector from the opposite edge of the active area.

type — Indicates whether the filter primitive should perform a noise or turbulence function.

... — Further arguments to be passed onto `fe`.

Details

For more information about this primitive, consult the reference to the SVG specification.

Value

An `fe.turbulence` object.

Author(s)

Simon Potter

References

https://www.w3.org/TR/SVG/filters.html#feTurbulenceElement

See Also

`filterEffect`, `fe`.

Filter Inputs *Identifies input for a filter effect primitive.*

Description

How to use and identify inputs for filter effect primitives.

Filter Inputs

The value chosen for a filter effect primitive can be either one of six keywords or can be a string which matches a previous `result` attribute value within the same filter effect container. If no value is provided and this is the first filter effect primitive, then the input will be `SourceGraphic`. If no value is provided and this is a subsequent filter effect primitive, then this filter effect primitive will use the result from the previous filter primitive as its input.

If the value for `result` appears multiple times within a given filter container, then a reference to that result will use the closest preceding filter primitive with the given value for the `result` results. Forward references to results are an error and will not draw.

Definitions for the seven possible options:

- `SourceGraphic` This keyword represents the appearance of grobs before they are being filtered. For raster effects filter primitives, the grobs will be rasterized into an initially clear RGBA raster in image space. Pixels left untouched by the original graphic will be left clear. The image is specified to be rendered in linear RGBA pixels. The alpha channel of this image captures any anti-aliasing specified by SVG. (Since the raster is linear, the alpha channel of this image will represent the exact percent coverage of each pixel.)
- `SourceAlpha` This keyword represents the appearance of grobs before they are being filtered. `SourceAlpha` has all of the same rules as `SourceGraphic` except that only the alpha channel is used. The input image is an RGBA image consisting of implicitly black color values for the RGB channels, but whose alpha channel is the same as `SourceGraphic`. If this option is used, then some implementations might need to rasterize the graphics elements in order to extract the alpha channel.
- `BackgroundImage` This keyword represents an image snapshot of the canvas under the filter region at the time that the referring grob is being filtered.
- `BackgroundAlpha` Same as `BackgroundImage` except only the alpha channel is used.
- `FillPaint` This keyword represents the value of the `fill` property on the grob being filtered. The `FillPaint` image has conceptually infinite extent. Frequently this image is opaque everywhere, but it might not be if the "paint" itself has alpha, as in the case of a gradient or pattern which itself includes transparent or semi-transparent parts.
- `StrokePaint` This keyword represents the value of the `col` property on the grob being filtered. The `StrokePaint` image has conceptually infinite extent. Frequently this image is opaque everywhere, but it might not be if the "paint" itself has alpha, as in the case of a gradient or pattern which itself includes transparent or semi-transparent parts.
- The result of any filter effect operation. This is the name that has been given to the `result` argument of a filter primitive.

Author(s)

Simon Potter

References

https://www.w3.org/TR/SVG/filters.html#FilterPrimitiveInAttribute

`filterEffect` *Creating Filter Effects*

Description

Create objects which describe filter effects. These objects can be used to add filter effect primitives. They can be used to apply a filter effect to grobs and also to define a filter effect so that it may be used multiple times.

Usage

```
filterEffect(feList = NULL, filterUnits = c("coords", "bbox"),
             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",
             primitiveUnits = c("coords", "bbox"))
addFilterEffect(filter, filterEffect, after = NA)
```

Arguments

`feList`	A list of filter effect primitives. For example a list containing a gaussian blur primitive created by `feGaussianBlur`.
`filterUnits`	If `"bbox"`, the filter effect itself is positioned relative to the bounding box of the referring grob. All units attempt to be converted to equivalent "npc" coordinates as a reulst. If `"coords"`, uses grid coordinates to determine positioning.
`x`	A numeric vector or unit object specifying x-location.
`y`	A numeric vector or unit object specifying y-location.
`width`	A numeric vector or unit object specifying width.
`height`	A numeric vector or unit object specifying height.
`just`	The justification of the pattern relative to its (x, y) location. If there are two values, the first value specifies horizontal justification and the second value specifies vertical justification. Possible string values are: `"left"`, `"right"`, `"centre"`, `"center"`, `"bottom"`, and `"top"`. For numeric values, 0 means left alignment and 1 means right alignment.
`hjust`	A numeric vector specifying horizontal justification. If specified, overrides the `just` setting.
`vjust`	A numeric vector specifying vertical justification. If specified, overrides the `just` setting.
`default.units`	A string indicating the default units to use if `x`, `y`, `width`, or `height` are only given as numeric vectors.

primitiveUnits	If "bbox", all filter effect primitives will be positioned relative to the bounding box of the filter effect region (determined by x, y, width, height and filterUnits). All units attempt to be converted to equivalent "npc" coordinates. If "coords", uses grid coordinates to determine positioning.
filter	A filter effect container object, as created by filterEffect.
filterEffect	A filter effect primitive object.
after	Numeric. Determines where amongst the children of filter that filterEffect should be added. NA indicates that filterEffect should be appended to the end of the list of children.

Details

This is primarily a container object to hold filter effect primitives.

Value

A filter object.

Author(s)

Simon Potter

References

https://www.w3.org/TR/SVG/filters.html#FilterElement

See Also

Any of the filter effect primitives (named fe*), e.g. feGaussianBlur.

---

garnish *Convert animation specifications to SVG elements.*

Description

This function is used to generate a list of SVG attributes based on information on a grob. It is generic so new grob classes can write their own methods.

Usage

```
garnish(x, ...)
```

Arguments

x	A grob.
...	For future use.

Details

This function is not called directly by the user. It is exposed so that new grob classes can easily write their own methods which call existing methods for standard grobs.

Author(s)

Paul Murrell

`getSVGFonts` *Manage SVG fonts*

Description

These functions control the SVG font stacks that are used when exporting text to SVG.

Usage

```
getSVGFonts()
setSVGFonts(fontStacks)
```

Arguments

`fontStacks`	A list of font stacks (typically the modified result from `getSVGFonts()`).

Details

`getSVGFonts()` returns a list of three font stacks called `serif`, `sans`, and `mono`. The user can modify the values in each stack and then reset the stacks by calling `setSVGFonts()` (a default value will always be forced at the end of each font stack).

Value

A list (for `getSVGFonts()`).

Author(s)

Simon Potter

Gradient Fills *Create a definition of a gradient fill.*

Description

A feature of SVG is that elements can be filled with a gradient that is defined somewhere in the document. The purpose of the `registerGradientFill` function is to create a definition of a gradient fill so that it can be referred to by grobs drawn by gridSVG.

Usage

```
registerGradientFill(label, gradient)
```

Arguments

`label`	A character identifier for a gradient fill.
`gradient`	A gradient object filled with gradient stops. See `linearGradient` and `radialGradient`.

Value

None.

Author(s)

Simon Potter

See Also

`linearGradient`, `radialGradient`, `grid.gradientFill`

Gradient Objects *Create Linear and Radial Gradients*

Description

Create objects which describe linear and radial gradients. These objects can later be used to apply a gradient fill to grobs, and also to define a gradient so that it may be reused multiple times.

Usage

```
linearGradient(col = c("black", "white"),
               stops = seq(0, 1, length.out = length(col)),
               gradientUnits = c("bbox", "coords"),
               x0 = unit(0, "npc"), x1 = unit(1, "npc"),
               y0 = unit(0, "npc"), y1 = unit(1, "npc"),
               default.units = "npc",
               spreadMethod = c("pad", "reflect", "repeat"))
radialGradient(col = c("black", "white"),
               stops = seq(0, 1, length.out = length(col)),
               gradientUnits = c("bbox", "coords"),
               x = unit(0.5, "npc"), y = unit(0.5, "npc"),
               r = unit(0.5, "npc"),
               fx = unit(0.5, "npc"), fy = unit(0.5, "npc"),
               default.units = "npc",
               spreadMethod = c("pad", "reflect", "repeat"))
```

Arguments

`col`	A vector of colours used for gradient stops.
`stops`	A numeric vector of offsets (typically between 0 and 1) to place the the colours (`col`) at.
`gradientUnits`	If `"bbox"`, the gradient is positioned relative to the bounding box of the referring grob. All units attempt to be converted to equivalent "npc" coordinates as a result. If `"coords"`, uses grid coordinates to determine positioning.
`x0`	Numeric or unit object indicating the starting x-location of the linear gradient.
`x1`	Numeric or unit object indicating the stopping x-location of the linear gradient.
`y0`	Numeric or unit object indicating the starting y-location of the linear gradient.
`y1`	Numeric or unit object indicating the stopping y-location of the linear gradient.
`x`	Numeric or unit object indicating the x-location of the radial gradient.
`y`	Numeric or unit object indicating the y-location of the radial gradient.
`r`	A numeric vector or unit object specifying the radius of the radial gradient.
`fx`	A numeric vector or unit object specifying an x-location. Determines the x-location of the focal point of the radial gradient.
`fy`	A numeric vector or unit object specifying an y-location. Determines the y-location of the focal point of the radial gradient.
`default.units`	A string indicating the default units to use if `x`, `y`, `r`, `fx` or `fy` are only given as numeric vectors.
`spreadMethod`	A character vector determining when happens when a gradient begins or ends within its bounds. See details.

Details

When defining gradient stops via `col` and `stops`, the order is important. Gradient stops which are defined earlier are drawn first, with later stops being drawn over the top.

For `spreadMethod` the possible values are:

pad Use the terminal colors of the gradient to fill the remainder of the target region.

reflect Reflect the gradient pattern start-to-end, end-to-start, start-to-end, etc. continuously until the target region is filled.

repeat Repeat the gradient pattern start-to-end, start-to-end, start-to-end, etc. continuously until the target region is filled.

Value

A `gradient` object.

Author(s)

Simon Potter

`grid.animate` *Animate a grid grob*

Description

Creates an animated.grob object. Useful in conjunction with `grid.export`, to produce an SVG document with animated graphical elements.

Usage

```
animateGrob(grob, ...,
            duration=1,
            rep=FALSE, revert=FALSE,
            begin=0, interpolate="linear", group=FALSE)
grid.animate(path, ..., group=FALSE, redraw = FALSE,
             strict=FALSE, grep=FALSE, global=FALSE)
```

Arguments

`grob`	A grob to add animation to.
`path`	A grob path specifying a drawn grob.
`...`	Arguments of the grob to animate.
`duration`	The duration in seconds of the animation.
`rep`	The number of times the animation should repeat. `FALSE` means once, `TRUE` means indefinitely.

revert	What should happen when (if) the animation ends; TRUE means revert to the first animated value, FALSE means finish on the last animated value.
begin	When the animation should begin (seconds).
interpolate	A character value describing how animation values are interpreted. One of linear or discrete.
group	A logical indicating whether the animation values should be applied to the overall group element in SVG or to individual SVG elements.
redraw	A logical value to indicate whether to redraw the grob.
strict	A boolean indicating whether the path must be matched exactly.
grep	Whether the path should be treated as a regular expression.
global	A boolean indicating whether the function should affect just the first match of the path, or whether all matches should be affected.

Value

An animated.grob object.

Author(s)

Paul Murrell

See Also

grid.export

---

grid.clipPath *Apply a clipping path to a grid grob.*

Description

Creates a pathClipped.grob object which is a normal grid grob, with a clipping path applied to it. Used in conjunction with registerClipPath, to produce an SVG document containing graphical elements with masked content.

Usage

```
grid.clipPath(path, clippath = NULL, label = NULL,
              group = TRUE, redraw = FALSE,
              strict = FALSE, grep = FALSE, global = FALSE)
clipPathGrob(x, clippath = NULL, label = NULL, group = TRUE)
```

Arguments

x	A grob to clip.
path	A grob path specifying a drawn grob.
clippath	A grob defining a clipping region.
label	A label that is associated with a definition of a clipping path. This is the label used to make a clipping path definition with registerClipPath.
group	A logical vector that indicates whether the opacity mask should be applied to the overall parent group for the relevant SVG element, or to individual SVG elements.
redraw	A logical value to indicate whether to redraw the grob.
strict	A boolean indicating whether the path must be matched exactly.
grep	Whether the path should be treated as a regular expression.
global	A boolean indicating whether the function should affect just the first match of the path, or whether all matches should be affected.

Details

If label is specified, uses a clipping path that has been supplied to registerClipPath. If clippath is specified it will be used as the clipping path applied to each grob. If both are specified, it will attempt to define the clipping path with the given label, as well as applying the clipping path to the appropriate grobs.

Value

A pathClipped.grob object (for clipPathGrob).

Author(s)

Simon Potter

See Also

registerClipPath, pushClipPath.

---

grid.comment *Create a grid grob representing a comment*

Description

Creates a comment.grob object which is a grid nullGrob, with a comment attached. Useful in conjunction with grid.export, to produce an SVG document with comments inserted at the point where the grob is "drawn".

Usage

```
grid.comment(comment, name = NULL, vp = NULL)
commentGrob(comment, name = NULL, vp = NULL)
```

Arguments

comment	A character vector used to write out a comment. If this has a length greater than one, each element is assumed to be a line.
name	A character identifier.
vp	The viewport to which the grob belongs.

Value

A comment.grob object.

Author(s)

Simon Potter

See Also

grid.export

---

grid.element *Create a grid grob representing an SVG element*

Description

Creates a element.grob object which is a grid gTree, representing an SVG element. Useful in conjunction with grid.export, to produce an SVG document with elements inserted at particular points. The element (and its children) are inserted at the point where the grob is "drawn". Text can be inserted in a similar manner with grid.textNode.

Usage

```
grid.element(el, name = NULL, attrs = NULL,
             namespace = NULL, namespaceDefinitions = NULL,
             children = NULL, vp = NULL,
             childrenvp = NULL, asis = FALSE)
elementGrob(el, name = NULL, attrs = NULL,
            namespace = NULL, namespaceDefinitions = NULL,
            children = NULL, vp = NULL,
            childrenvp = NULL, asis = FALSE)
grid.textNode(text, name = NULL, vp = NULL)
textNodeGrob(text, name = NULL, vp = NULL)
```

Arguments

el	The name of the SVG element to create, e.g. "rect".
text	A single element character vector of text directly into insert into the SVG image.
name	A character identifier.
attrs	A list, where the names are SVG attribute names, and values are the values given to the SVG attributes.
namespace	A character vector specifying the namespace for this new element.
namespaceDefinitions	A character vector or a list with each element being a string. These give the URIs identifying the namespaces uniquely. The elements should have names which are used as prefixes. A default namespace has "" as the name. The values here are used only for defining new namespaces and not for determining the namespace to use for this particular element.
children	A gList object containing children of this element (if any).
vp	A viewport object to draw within.
childrenvp	A viewport object to use for the children of the element grob.
asis	If TRUE, SVG id attributes will be generated from the name with no modification so that we can easily refer to the generated elements.

Value

An element.grob object. For grid.textNode a textnode.grob object.

Author(s)

Simon Potter

See Also

grid.export

---

grid.export *Generate SVG output from a grid graphic*

Description

Produces an SVG version of the current grid page.

Usage

```
grid.export(name = "Rplots.svg",
            exportCoords = c("none", "inline", "file"),
            exportMappings = c("none", "inline", "file"),
            exportJS = c("none", "inline", "file"),
            res = NULL,
            prefix = "",
            addClasses = FALSE,
            indent = TRUE,
            htmlWrapper = FALSE,
            usePaths = c("vpPaths", "gPaths", "none", "both"),
            uniqueNames = TRUE,
            annotate = TRUE,
            progress = FALSE,
            compression = 0,
            strict = TRUE,
            rootAttrs = NULL,
            xmldecl = xmlDecl())
```

Arguments

`name`	The name of the SVG file to produce. If this parameter is `NULL` or `""`, a list containing the SVG document, coordinate information, and JavaScript utility functions are returned.
`exportCoords`	If this parameter is not `none` a coordinates file is exported. If this parameter is `file`, the coordinates information is written to a file, while `inline` will include the contents within the SVG document.
`exportMappings`	If this parameter is not `none` a mapping file is exported. If this parameter is `file`, the mapping information is written to a file, while `inline` will include the contents within the SVG document.
`exportJS`	If this parameter is not `none` a JavaScript file is written out. This contains useful functions for manipulating gridSVG plots in the browser, including unit conversion functions. If this parameter is `file`, the JavaScript file is written to a file, while `inline` will include the contents within the SVG document.
`res`	The device resolution to print at (in DPI). If `NULL`, this is automatically calculated to be the resolution of the current device. Typically the PDF device would be used, and this uses a resolution of 72, i.e. 72 DPI.
`prefix`	A prefix to apply to all generated SVG ID attributes. Useful for ensuring unique IDs when many SVG images exist within the same HTML document. If a valid prefix has been given, the root `<svg>` element will be given an ID attribute with the prefix as its value.
`addClasses`	If `TRUE`, adds an SVG class attribute to all grobs and viewports which holds the value of the class of the grob or viewport. If the class attribute already exists

(via grid.garnish or grid.element), the resulting SVG class attribute will be the union of the existing class attribute and the grob/viewport classes.

`indent` Determines whether the resulting SVG document will be exported with indentation present.

Indentation makes the document more readable, but when `indent` is set to `FALSE`, parsing the SVG in JavaScript is easier because there are no empty text nodes.

`htmlWrapper` If `TRUE`, saves a wrapping HTML file. This file contains a snippet of HTML which links to the exported SVG file.

`usePaths` If this parameter is set to `vpPaths`, then when writing out viewports gridSVG will set the SVG element ID to the current `vpPath` instead of the current viewport name.

If this parameter is set to `gPaths`, gridSVG will set the names of grobs to be the current `gPath` instead of the current grob name.

When `none`, viewports and grobs will not incorporate paths.

When `both`, viewports and grobs will both use paths.

`uniqueNames` If `TRUE`, gridSVG will make an attempt to produce unique grob names. Unique `id` attributes are required for valid SVG. It is highly recommended that mapping information is used when this parameter is `TRUE`.

`annotate` If `TRUE`, an SVG metadata element will be introduced directly below the root `<svg>` element. This element contains XML that describes the information that gridSVG used to draw the image (mostly arguments to `grid.export`). This output may be useful for debugging purposes.

`progress` If `TRUE`, messages will be displayed in the console that show how quickly gridSVG is progressing when exporting an SVG image. This is particularly useful when there are large images being exported so we have a reasonable estimate of how long exporting will take.

`compression` An integer between 0 and 9 indicating the level of (gzip) compression applied to the SVG image when it is saved to a file. Higher values of `compression` indicate smaller file sizes at the expense of increased computation.

`strict` A logical indicating whether checks should be made that all attributes added to SVG elements are valid. If this is `TRUE` and invalid attributes are detected, those attributes are removed, with a warning.

`rootAttrs` A named character vector containing attributes for the top-level `<svg>` element.

`xmldecl` This parameter sets the XML declaraction that will be applied to the SVG document.

By default this parameter simply declares that the document is XML version 1.0, along with the character encoding that was used to export the SVG document.

If `xmldecl` is `NULL`, then no XML declaration is printed. This may be useful when you want only the SVG document and nothing more.

Details

The `uniqueNames` parameter is set to `TRUE` by default in order to ensure that each SVG element ID is unique. This is a requirement of XML (which SVG is based on). This differs from `usePaths`

because usePaths can still generate names that are not unique (there are several ways for this to happen). uniqueNames modifies grob and viewport names with a numeric suffix to ensure uniqueness. When FALSE, only grob names will be kept unmodified because modifying viewport names would affect coordinate information.

Occasionally the XML package can report warnings, despite valid SVG being produced. If spurious warnings are being produced, set options(gridSVGWarnings = FALSE) to ignore them.

See the files in the directory gridSVG/tests for examples of things that can be done. See the file gridSVG/doc/overview.tex for limitations.

Value

When name has a valid filename the side effect is to produce an SVG file of the specified name.

Optionally a JavaScript file containing coordinate transformation information is also exported.

Optionally a JavaScript file containing name mapping information is also exported.

Optionally a JavaScript file containing utility JavaScript functions is also exported.

When name has a filename with zero characters, a named list is returned with four elements. svg is the SVG root node (and all its children, see the XML package for more information on how to use this. coords contains the list of coordinate information for exported viewports. mappings is a list containing information on how names have been modified during the exporting process. utils is a character vector containing JavaScript code to manipulate gridSVG plots in the browser.

This list is always returned but when a valid filename is given, it is returned invisibly.

Author(s)

Paul Murrell

See Also

grid.hyperlink, grid.animate, grid.garnish

---

grid.filter *Associate a filter effect with a grid grob.*

Description

Creates a filtered.grob object which is a normal grid grob, with a filter effect applied to it Used in conjunction with registerFilter, to produce an SVG document containing graphical elements with filter effects.

Usage

```
grid.filter(path, filter = NULL, label = NULL,
            group = TRUE, redraw = FALSE,
            strict = FALSE, grep = FALSE, global = FALSE)
filterGrob(x, filter = NULL, label = NULL, group = TRUE)
```

Arguments

x	A grob to filter.
path	A grob path specifying a drawn grob.
filter	A filter object, provided by the filterEffect function. Provides the definition of a filter effect that will be applied to x or path.
label	A label that is associated with a definition of a filter effect. This is the label used to create a filter effect definition with registerFilter.
group	A logical vector that indicates whether the filter effect should be applied to the overall parent group for the relevant SVG element, or to individual SVG elements.
redraw	A logical value to indicate whether to redraw the grob.
strict	A boolean indicating whether the path must be matched exactly.
grep	Whether the path should be treated as a regular expression.
global	A boolean indicating whether the function should affect just the first match of the path, or whether all matches should be affected.

Details

If label is specified, uses a filter effect that has been supplied to registerFilter. If filter is specified it will be used as the filter effect applied to each grob. If both are specified, it will attempt to define the filter effect with the given label, as well as applying the filter effect to the appropriate grobs.

Value

A filtered.grob object (for filterGrob).

Author(s)

Simon Potter

See Also

registerFilter, filterEffect.

---

grid.garnish *Associate arbitrary SVG attributes with a grid grob*

Description

Creates an svg.grob object which is a normal grid grob, with SVG attributes attached. Useful in conjunction with grid.export, to produce an SVG document with attributes that have no corresponding concept in grid graphics.

Usage

```
garnishGrob(x, ..., group=TRUE)
grid.garnish(path, ..., group=TRUE, redraw=FALSE,
             strict = FALSE, grep=FALSE, global=FALSE)
```

Arguments

x	A grob.
path	A grob path specifying a drawn grob.
...	Arbitrary SVG attribute settings.
group	A logical indicating whether the SVG attributes should be attached to the overall parent group for the relevant SVG element, or to individual SVG elements.
redraw	A logical value to indicate whether to redraw the grob.
strict	A boolean indicating whether the path must be matched exactly.
grep	Whether the path should be treated as a regular expression.
global	A boolean indicating whether the function should affect just the first match of the path, or whether all matches should be affected.

Details

The SVG attribute settings can be vectors (in the case of garnishing individual SVG elements) or even named vectors (if you want precise control over which attribute value is apportioned to which individual SVG element).

Value

A garnished.grob object.

Author(s)

Paul Murrell

See Also

grid.export

---

grid.gradientFill *Associate a gradient fill with a grid grob*

Description

Creates a gradientFilled.grob object which is a normal grid grob, with a gradient fill used in place of a regular fill. Used in conjunction with registerGradientFill, to produce an SVG document containing graphical elements with gradient fills.

Usage

```
grid.gradientFill(path, gradient = NULL, label = NULL,
                  alpha = 1, group = TRUE, redraw = FALSE,
                  strict = FALSE, grep = FALSE, global = FALSE)
gradientFillGrob(x, gradient = NULL, label = NULL,
                 alpha = 1, group = TRUE)
```

Arguments

x	A grob to add a pattern fill to.
path	A grob path specifying a drawn grob.
gradient	A gradient object, provided by the linearGradient and radialGradient functions. Provides the definition of a gradient fill that will be applied to x or path.
label	A label that is associated with a definition of a gradient fill. This is the label used to create a gradient fill definition with registerGradientFill.
alpha	The alpha channel for transparency. A value between 0 and 1.
group	A logical vector that indicates whether the gradient fill should be applied to the overall parent group for the relevant SVG element, or to individual SVG elements.
redraw	A logical value to indicate whether to redraw the grob.
strict	A boolean indicating whether the path must be matched exactly.
grep	Whether the path should be treated as a regular expression.
global	A boolean indicating whether the function should affect just the first match of the path, or whether all matches should be affected.

Details

If label is specified, uses a gradient that has been supplied to registerGradientFill. If gradient is specified it will be used as the gradient fill applied to each grob. If both are specified, it will attempt to define the gradient with the given label, as well as applying a gradient fill to the appropriate grobs.

Value

A gradientFilled.grob object (for gradientFillGrob).

Author(s)

Simon Potter

See Also

linearGradient, radialGradient, registerGradientFill

grid.hyperlink *Associate a hyperlink with a grid grob*

Description

Creates a linked.grob object which is a normal grid grob, with a hyperlink attached. Useful in conjunction with `grid.export`, to produce an SVG document with hyperlinked graphical elements.

Usage

```
grid.hyperlink(path, href, show=NULL, group=TRUE, redraw=FALSE,
               strict=FALSE, grep=FALSE, global=FALSE)
hyperlinkGrob(x, href, show=NULL, group=TRUE)
```

Arguments

`x`	A grob to add a hyperlink to.
`path`	A grob path specifying a drawn grob.
`href`	A valid Xlink URI. Can be a vector of several links (see `group` argument below).
`show`	A character vector specifying how the link should be opened. `NULL` and `""` will avoid adding an attribute. The most common cases are to use `"new"` to open a link in a new window/tab, or `"replace"` to open the link in the current window/tab.
`group`	A logical indicating whether the hyperlinks should be attached to the overall parent group for the relevant SVG element, or to individual SVG elements.
`redraw`	A logical value to indicate whether to redraw the grob.
`strict`	A boolean indicating whether the `path` must be matched exactly.
`grep`	Whether the `path` should be treated as a regular expression.
`global`	A boolean indicating whether the function should affect just the first match of the `path`, or whether all matches should be affected.

Value

A linked.grob object.

Author(s)

Paul Murrell

See Also

`grid.export`

grid.mask *Apply an opacity mask to a grid grob.*

Description

Creates a masked.grob object which is a normal grid grob, with an opacity mask applied to it. Used in conjunction with registerMask, to produce an SVG document containing graphical elements with masked content.

Usage

```
grid.mask(path, mask = NULL, label = NULL, group = TRUE, redraw = FALSE,
          strict = FALSE, grep = FALSE, global = FALSE)
maskGrob(x, mask = NULL, label = NULL, group = TRUE)
```

Arguments

x	A grob to mask.
path	A grob path specifying a drawn grob.
mask	A mask object, provided by the mask function. Provides the definition of an opacity mask that will be applied to x or path.
label	A label that is associated with a definition of an opacity mask. This is the label used to create an opacity mask definition with registerMask.
group	A logical vector that indicates whether the opacity mask should be applied to the overall parent group for the relevant SVG element, or to individual SVG elements.
redraw	A logical value to indicate whether to redraw the grob.
strict	A boolean indicating whether the path must be matched exactly.
grep	Whether the path should be treated as a regular expression.
global	A boolean indicating whether the function should affect just the first match of the path, or whether all matches should be affected.

Details

If label is specified, uses a mask that has been supplied to registerMask. If mask is specified it will be used as the opacity mask applied to each grob. If both are specified, it will attempt to define the opacity mask with the given label, as well as applying the mask to the appropriate grobs.

Value

A masked.grob object (for maskGrob).

Author(s)

Simon Potter

See Also

registerMask, mask, pushMask.

---

grid.patternFill *Associate a pattern fill with a grid grob*

Description

Creates a patternFilled.grob object which is a normal grid grob, with a pattern fill used in place of a regular fill. Used in conjunction with registerPatternFill, to produce an SVG document containing graphical elements with pattern fills.

Usage

```
grid.patternFill(path, pattern = NULL, label = NULL,
                 alpha = 1, group = TRUE, redraw = FALSE,
                 strict = FALSE, grep = FALSE, global = FALSE)
patternFillGrob(x, pattern = NULL, label = NULL,
                alpha = 1, group = TRUE)
```

Arguments

x	A grob to add a pattern fill to.
pattern	A pattern object, provided by the pattern function. Provides the definition of a pattern fill that will be applied to x or path.
label	A label that is associated with a definition of a pattern fill. This is the label used to create a pattern fill definition with registerPatternFill.
path	A grob path specifying a drawn grob.
alpha	The alpha channel for transparency. A value between 0 and 1.
group	A logical vector that indicates whether the pattern fill should be applied to the overall parent group for the relevant SVG element, or to individual SVG elements.
redraw	A logical value to indicate whether to redraw the grob.
strict	A boolean indicating whether the path must be matched exactly.
grep	Whether the path should be treated as a regular expression.
global	A boolean indicating whether the function should affect just the first match of the path, or whether all matches should be affected.

Details

If label is specified, uses a pattern that has been supplied to registerPatternFill. If pattern is specified it will be used as the fill pattern applied to each grob. If both are specified, it will attempt to define the pattern with the given label, as well as applying a pattern fill to the appropriate grobs.

Value

A patternFilled.grob object (for `patternFillGrob`).

Author(s)

Simon Potter

See Also

`registerPatternFill`

`grid.script` *Create a grid grob containing an SVG script*

Description

Creates a script object which is a normal grid grob containing an SVG script. Useful in conjunction with `grid.export`, to produce an SVG document with script elements.

Usage

```
scriptGrob(script=NULL, filename=NULL, type="application/ecmascript",
           inline=FALSE, name=NULL)
grid.script(...)
```

Arguments

`script`	A character value specifying script code.
`filename`	The name of a file that contains script code.
`type`	The type of the script code.
`inline`	A logical specifying whether the script code from the file should be included inline or just referenced.
`name`	A character value giving a name for the grob.
`...`	Arguments to be passed into `scriptGrob`.

Value

A script.grob object.

Author(s)

Paul Murrell

See Also

`grid.export`

gridsvg *gridSVG Graphics Device*

Description

Provides a convenient and familiar graphics device interface for the gridSVG package.

Usage

```
gridsvg(name = "Rplots.svg",
        exportCoords = c("none", "inline", "file"),
        exportMappings = c("none", "inline", "file"),
        exportJS = c("none", "inline", "file"),
        res = NULL,
        prefix = "",
        addClasses = FALSE,
        indent = TRUE,
        htmlWrapper = FALSE,
        usePaths = c("vpPaths", "gPaths", "none", "both"),
        uniqueNames = TRUE,
        annotate = TRUE,
        progress = FALSE,
        compression = 0,
        strict = TRUE,
        rootAttrs = NULL,
        xmldecl = xmlDecl(), ...)
dev.off(which = dev.cur())
```

Arguments

name, exportCoords, exportMappings, exportJS, res, prefix, addClasses, indent, htmlWrapper, usePaths, uniqueNames, annotate, progress, compression, strict, rootAttrs, xmldecl	These parameters are passed onto grid.export.
...	Further parameters that are passed onto a NULL pdf graphics device. Useful parameters include width and height.
which	An integer specifying a device number.

Details

These functions provide a more familiar and perhaps convenient interface to gridSVG than grid.export. It uses a PDF device as drawing occurs, but when the device needs to be written out (via dev.off) then it will save an SVG image instead.

When a grid display list is not in use, or any device other than the gridsvg device is used, the behaviour of dev.off is the same as dev.off from the grDevices package.

Value

gridsvg returns nothing.

dev.off will return in the same manner as grid.export. A list is always returned, but invisibly when an invalid filename is given.

Author(s)

Simon Potter

See Also

pdf and grid.export.

---

gridSVG.newpage *Move to a New Page on a gridSVG Device*

Description

This function erases the current device or moves to a new page. In addition, it clears any definitions of referenced content defined by gridSVG.

Usage

```
gridSVG.newpage(wipeRefs = TRUE, recording = TRUE)
```

Arguments

wipeRefs	A logical value that determines whether referenced content should be deleted.
recording	A logical value to indicate whether the new-page operation should be saved onto the Grid display list.

Details

When creating a gridSVG image, it is possible to create referenced content. An example is pattern fills. This function should be used in order to remove the definitions of referenced content.

Value

None.

Author(s)

Simon Potter

grobToDev *Convert a grob to device calls*

Description

This function is used to make calls to a device to draw a grob. It is generic so new grob classes can write their own methods.

Usage

```
grobToDev(x, dev)
```

Arguments

x	A grob.
dev	A graphics device.

Details

This function is not called directly by the user. It is exposed so that new grob classes can easily write their own methods which call existing methods for standard grobs.

The difference between this function and `primToDev()` is that this one takes care of setting up coordinate systems based on the grid viewports so that SVG output is positioned correctly, then it calls `primToDev()` to produce the actual SVG elements.

Author(s)

Paul Murrell

Import Coordinate JS *Importing JavaScript coordinate information.*

Description

This function reads in a JavaScript file and transforms it into JSON text. This text is then transformed into a list that can be used in conjunction with `gridSVGCoords`.

Usage

```
readCoordsJS(filename)
```

Arguments

filename	A character vector that represents a file name. This file should be a JavaScript file containing coordinate information produced by `grid.export`.

Details

In order to use the `fromJSON` function to parse JSON text, the JavaScript file produced by `grid.export` needs to be transformed. It needs to transform from being an assignment of an object literal to simply the object literal itself.

This function performs that task by producing a valid JSON string ready for parsing by `fromJSON`. It then returns the parsed list.

Value

A list of coordinate information.

Author(s)

Simon Potter

`Import Mappings JS` *Importing JavaScript mapping information.*

Description

This function reads in a JavaScript file and transforms it into JSON text. This text is then transformed into a list that can be used in conjunction with `gridSVGMappings`.

Usage

```
readMappingsJS(filename)
```

Arguments

`filename`	A character vector that represents a file name. This file should be a JavaScript file containing mapping information produced by `grid.export`.

Details

In order to use the `fromJSON` function to parse JSON text, the JavaScript file produced by `grid.export` needs to be transformed. It needs to transform from being an assignment of an object literal to simply the object literal itself.

This function performs that task by producing a valid JSON string ready for parsing by `fromJSON`. It then returns the parsed list.

Value

A list of mapping information.

Author(s)

Simon Potter

listSVGDefinitions *List All Reference Definitions*

Description

Returns a listing of the labels given to reference definitions.

Usage

```
listSVGDefinitions(print = TRUE)
```

Arguments

print	If TRUE, prints the listing of reference definitions.

Details

When definitions of referenced content are stored in gridSVG via any of the register* functions (e.g. registerPatternFill), we can use this function to show us all of the labels given when content is registered.

Value

A data frame, returned invisibly.

Author(s)

Simon Potter

Mapping Names to IDs *Mapping Viewport, Grob and Reference Names to SVG IDs*

Description

This function is both a getter and a setter function for mapping information imported from a plot unknown to the current R session.

Usage

```
gridSVGMappings(newmappings = NULL)
```

Arguments

newmappings	A named list mapping information, produced by grid.export.

Details

In order to generate unique names for SVG IDs, gridSVG output will not produce the same names as are visible on the grid display list. This function will store and return mapping information. This is information on how names have been translated from their original grob/viewport names to their SVG IDs.

Mapping information is stored as a list with 4 components, viewport mapping information, grob mapping information, reference mapping information and the ID separator used at the time of exporting.

Viewport, grob, and reference mapping information is stored as the name of the object, paired with a vector of suffixes associated with these names. When combined with the ID separator, we can construct the SVG IDs that have been applied, given each name. Use `getSVGMappings` to do this.

Value

If `newmappings` is `NULL`, then we get back a named list representing name mapping information.

If we pass the named list representing mapping information into the function, we get no output.

Author(s)

Simon Potter

`Opacity Masks` *Create the definition of an opacity mask.*

Description

A feature of SVG is that elements can have an opacity mask applied to it. An opacity mask is an image that, for various levels of opacity, makes the object that is being masked inherit the same levels of opacity. The purpose of these functions is to define an opacity mask that will be applied until the current viewport (or context, see `popContext`) is popped. Alternatively it can also be applied to grobs.

Usage

```
mask(grob,
     x = unit(0.5, "npc"), y = unit(0.5, "npc"),
     width = unit(1, "npc"), height = unit(1, "npc"),
     default.units = "npc",
     just = "centre", hjust = NULL, vjust = NULL)
registerMask(label, mask = NULL, ...)
```

Arguments

grob	A grob or gTree that will be drawn as the opacity mask.
x	A numeric vector or unit object specifying x-location.
y	A numeric vector or unit object specifying y-location.
width	A numeric vector or unit object specifying width.
height	A numeric vector or unit object specifying height.
default.units	A string indicating the default units to use if x, y, width, or height are only given as numeric vectors.
just	The justification of the pattern relative to its (x, y) location. If there are two values, the first value specifies horizontal justification and the second value specifies vertical justification. Possible string values are: "left", "right", "centre", "center", "bottom", and "top". For numeric values, 0 means left alignment and 1 means right alignment.
hjust	A numeric vector specifying horizontal justification. If specified, overrides the just setting.
vjust	A numeric vector specifying vertical justification. If specified, overrides the just setting.
label	A character identifier that will be used to reference this definition.
mask	A mask object that defines the mask.
...	Arguments to be given to mask.

Details

When registering the mask, the rectangular region that the mask applies to will become fixed.

When referring to an opacity mask, the masked content will be opaque at the same coordinates that the mask is opaque. The same applies when there is any level of transparency, as any transparency in the mask will also apply in the same corresponding region of the masked object.

The mask's opacity is defined as being the level of luminance present in the mask. This means anything black is fully transparent, while anything white is completely opaque. The background is assumed to be black (i.e. fully transparent). The alpha value in a mask will still be used, but its effect is combined with the computed opacity from the luminance of the mask.

By using an opacity mask it is possible to have a grob with non-uniform opacity. In other words, rather than specifying an opacity via gpar's alpha parameter, which is uniform across the grob, we can define varying opacities on a grob via an opacity mask.

The x, y, width, height parameters determine the location and dimensions of the area to apply the mask to. This means we can apply a mask to any rectangular region, relative to the viewport in which it is defined (via registerMask).

Value

For mask, a mask object.

Author(s)

Simon Potter

See Also

`grid.mask`, `pushMask`, `popContext`.

`Pattern Fills` *Create a definition of a fill pattern.*

Description

A feature of SVG is that elements can be filled with a pattern that is defined somewhere in the document. The purpose of these functions is to create the definition of a fill pattern so that it can be referred to by grobs drawn by gridSVG.

Usage

```
pattern(grob,
        x = unit(0, "npc"), y = unit(0, "npc"),
        width = unit(0.1, "npc"), height = unit(0.1, "npc"),
        default.units = "npc",
        just = "centre", hjust = NULL, vjust = NULL,
        dev.width = 7, dev.height = 7)
registerPatternFill(label, pattern = NULL, ...)
registerPatternFillRef(label, refLabel, pattern = NULL, ...)
```

Arguments

`label`	A character identifier for the definition.
`refLabel`	A character identifier referring to an existing pattern definition that has been created by `registerPatternFill`.
`pattern`	A `pattern` object created by `pattern`.
`grob`	A grid grob or tree of grobs.
`x`	A numeric vector or unit object specifying x-location.
`y`	A numeric vector or unit object specifying y-location.
`width`	A numeric vector or unit object specifying width.
`height`	A numeric vector or unit object specifying height.
`just`	The justification of the pattern relative to its (x, y) location. If there are two values, the first value specifies horizontal justification and the second value specifies vertical justification. Possible string values are: `"left"`, `"right"`, `"centre"`, `"center"`, `"bottom"`, and `"top"`. For numeric values, 0 means left alignment and 1 means right alignment.
`hjust`	A numeric vector specifying horizontal justification. If specified, overrides the `just` setting.
`vjust`	A numeric vector specifying vertical justification. If specified, overrides the `just` setting.

default.units A string indicating the default units to use if x, y, width, or height are only given as numeric vectors.

dev.width, dev.height
The width and height of the fill pattern's graphics region in inches. The default values are 7.

... Arguments to be be passed onto pattern.

Details

The pattern fill is drawn off-screen on a new device. The size of this device is determined by dev.width and dev.height. The grob and vp that have been given are then drawn within this device. This is relevant for determining what the pattern definition looks like.

The previous arguments do not determine the size of the pattern as it is being used (i.e. how big each "tile" is). This is set by the x, y, width, height arguments. The values of these arguments are relative to the current viewport as this function is being called. From then on, the definition of the location and size of the pattern are fixed.

In summary, the pattern function defines what a pattern looks like, along with how big each tile is (and its position).

To avoid repetition of pattern definitions, use registerPatternFillRef to reuse an existing pattern definition (referred to by refLabel). This means that a pattern "tile" can now be reused, repositioned and rescaled without having to describe how it needs to be drawn.

In general use, first create a pattern object, then either give a label to the definition (for grobs to use), or alternatively simply pass on the pattern object to grid.patternFill.

Value

A pattern object for pattern, none otherwise.

Author(s)

Simon Potter

See Also

grid.patternFill

---

popContext *Leaving A Modified Viewport Context*

Description

A modified viewport context is where the appearance of grobs is no longer determined solely by the grob itself and the viewport into which they're drawn. This can occur when applying clipping paths and opacity masks, which modify the appearance of anything drawn after they have been applied. This function should be used when attempting to stop the effect of a modified viewport context (e.g. to stop clipping to paths).

Usage

```
popContext(n = 1)
```

Arguments

n	The number of contexts to pop. A warning will be given when n is greater than the number that has been applied.

Details

Popping a context can produce a warning. In this case it is recommended that the context "pushing" and "popping" be revised to have matching pairs of pushes and pops.

Value

None.

Author(s)

Simon Potter

See Also

grid.clipPath and grid.mask

---

primToDev *Convert a grob to device calls*

Description

This function is used to make calls to a device to draw a grob. It is generic so new grob classes can write their own methods.

Usage

```
primToDev(x, dev)
```

Arguments

x	A grob.
dev	A graphics device.

Details

This function is not called directly by the user. It is exposed so that new grob classes can easily write their own methods which call existing methods for standard grobs.

Author(s)

Paul Murrell

`pushClipPath` *Apply a clipping context to the current viewport.*

Description

This function is intended to be used similarly to `grid.clip`. The only difference is that a non-rectangular clipping region can be applied.

Usage

```
pushClipPath(clippath = NULL, label = NULL, name = NULL, draw = TRUE)
popClipPath()
```

Arguments

`clippath`	A graphics object, used as the definition of a clipping path.
`label`	A label for a defined reference.
`name`	A character identifier for the grob applying the clipping context.
`draw`	A logical value indicating whether graphics output should be produced.

Details

If `label` is specified, uses a clipping path that has been supplied to `registerClipPath`. If `clippath` is specified it will be used as the new clipping context for the current viewport. If both are specified, it will attempt to define the clipping path with the given label, as well as adding the clipping path as a clipping context for the current viewport.

`popClipPath` is an alias for `popContext`

Value

A `pushClipPath` grob. The value is returned invisibly.

Author(s)

Simon Potter

See Also

`registerClipPath`, `grid.clipPath`, `popContext`.

pushMask *Apply a masking context to the current viewport.*

Description

This function is intended to be used similarly to grid.clip. The key difference is that instead of applying a new clipping context to the viewport, we apply a new masking context.

Usage

```
pushMask(mask = NULL, label = NULL, name = NULL, draw = TRUE)
popMask()
```

Arguments

mask	A mask object, used as the definition of an opacity mask.
label	A label for a defined reference.
name	A character identifier for the grob applying the masking context.
draw	A logical value indicating whether graphics output should be produced.

Details

If label is specified, uses a mask that has been supplied to registerMask. If mask is specified it will be used as the new masking context for the current viewport. If both are specified, it will attempt to define the mask with the given label, as well as applying the mask as the new masking context for the current viewport.

popMask is an alias for popContext.

Value

A pushMask grob. The value is returned invisibly.

Author(s)

Simon Potter

See Also

mask, registerMask, grid.mask, popContext.

`registerFilter` *Create the definition a filter effect.*

Description

A feature of SVG is that elements can be filtered using filter effects defined somewhere in the document. The purpose of this function is to create the definition of a filter effect so that it can be referred to by grobs drawn by gridSVG.

Usage

```
registerFilter(label, filter)
```

Arguments

`label`	A character identifier for the definition.
`filter`	A `filter` object, produced by the `filterEffect` function.

Details

When registering a filter, all locations and dimensions that filter effects refer to become fixed.

Value

None.

Author(s)

Simon Potter

See Also

`grid.filter`, `filterEffect`.

`Retrieve Names Mapped to SVG IDs, CSS Selectors and XPath Expressions`
Retrieving Viewport, Grob, and Reference Names as SVG IDs, CSS Selectors and XPath Expressions

Description

This function gives us SVG IDs (or CSS selectors and XPath expressions) that have been created from a grob, viewport, or referenced name as a result of exporting to SVG.

Usage

```
getSVGMappings(name, type, result = "id")
```

Arguments

`name`	A single element character vector. This should be the name of a grob or viewport (as determined by `type`) present as the grid plot was exported.
`type`	A single element character vector, must be one of `vp`, `grob` or `ref`. This determines whether we are trying to get the IDs of a grob or a viewport or a referenced object like a fill pattern.
`result`	The type of output we want. `id` gives us SVG element IDs. `selector` gives us CSS selectors. `xpath` gives us XPath expressions.

Details

In order to generate unique names for SVG IDs, gridSVG output will not produce the same names as are visible on the grid display list. This function retrieves the SVG IDs associated with grob and viewport names. To use this function first requires importing mapping information, see `gridSVGMappings`.

To make using results easier with existing JavaScript libraries and R packages, CSS selectors and XPath expressions can be returned. This is the case when `result` is specified as one of `selector` or `xpath`. These are targeted to match just the SVG element itself, nothing more.

Value

A character vector representing values that can target specific SVG output.

Author(s)

Simon Potter

`setSVGoptions` *Get and Set Global Options*

Description

Provides access to a predefined set of global options for the **gridSVG** package.

Usage

```
getSVGoption(name)
getSVGoptions()
setSVGoptions(...)
```

Arguments

`name`	The name of one option.
`...`	Named arguments giving a name, value pair for a new option setting.

Details

The options currently available are:

- `id.sep` which controls the separator used between the grob name and the suffix number when **gridSVG** generates id values for SVG elements.
- `gPath.sep` which controls the separator used between elements of a grid `gPath`.
- `vpPath.sep` which controls the separator used between elements of a grid `vpPath`.

Value

`getSVGoption()` returns at most one option setting. `getSVGoptions()` returns all option settings. `setSVGoptions()` returns a list of previous option settings for the options that were changed.

Author(s)

Paul Murrell

See Also

`grid.export`

---

viewportCreate *Recreate a viewport from imported coordinate information.*

Description

Creates a viewport object that is positioned in the same location as a previously exported viewport.

The purpose of this function is so that we can recreate content for later manipulation.

Usage

```
viewportCreate(vpname, newname = NULL,
               vpPath.sep = getSVGoption("vpPath.sep"))
```

Arguments

`vpname`	The name of the viewport to be recreated, as stored in coordinate information. This is most likely a viewport path.
`newname`	The name that is going to be assigned to the viewport as it is re-created. If this parameter is `NULL`, then the name is taken to be the last viewport in listed in `vpname` (because it is usually a viewport path).
`vpPath.sep`	The viewport path separator that was used for `vpname`.

Details

In order to use this function, coordinate information must be available to gridSVG. This means that viewport information must be imported using gridSVGCoords.

The ROOT viewport must also have coordinate information imported because the created viewport is positioned relative to this.

Value

A viewport object.

Author(s)

Simon Potter

Examples

```
## Not run:
  require(grid)

  grid.newpage()

  # Pushing a new VP to draw a rect within
  pushViewport(viewport(x = unit(0.3, "npc"), y = unit(0.2, "npc"),
                        width = unit(0.1, "npc"), height = unit(0.3, "npc"),
                        xscale = c(0, 20), yscale = c(0, 10),
                        name = "testVP"))
  grid.rect()
  grid.export("create-test.svg", exportCoords = "file")

  # Importing coordinate information
  gridSVGCoords(readCoordsJS("create-test.svg.coords.js"))

  # This should appear to be the same rect
  grid.newpage()
  pushViewport(viewportCreate("testVP.1"))
  grid.rect()

  # Let's see if the scales are accurate, should be:
  # xscale: [0, 20]
  # yscale: [0, 10]
  current.viewport()$xscale
  current.viewport()$yscale

## End(Not run)
```

Index