

# Package ‘sesame’

April 9, 2025

**Type** Package

**Title** Sensible Step-wise Analysis of DNA METHylation BeadChips

**Description** Tools For analyzing Illumina Infinium DNA methylation arrays. SeSAME provides utilities to support analyses of multiple generations of Infinium DNA methylation BeadChips, including preprocessing, quality control, visualization and inference. SeSAME features accurate detection calling, intelligent inference of ethnicity, sex and advanced quality control routines.

**Version** 1.25.3

**Depends** R (>= 4.5.0), sesameData

**License** MIT + file LICENSE

**RoxygenNote** 7.3.2

**Imports** graphics, BiocParallel, utils, methods, stringr, readr, tibble, MASS, wheatmap (>= 0.2.0), GenomicRanges, IRanges, grid, preprocessCore, S4Vectors, ggplot2, BiocFileCache, GenomeInfoDb, stats, SummarizedExperiment, dplyr, reshape2

**Suggests** scales, BiocManager, knitr, DNACopy, e1071, randomForest, RPMM, rmarkdown, testthat, tidyr, BiocStyle, ggrepel, grDevices, KernSmooth, pals

**Encoding** UTF-8

**VignetteBuilder** knitr

**URL** <https://github.com/zwdzwd/sesame>

**BugReports** <https://github.com/zwdzwd/sesame/issues>

**biocViews** DNAMethylation, MethylationArray, Preprocessing, QualityControl

**Collate** 'readIDAT.R' 'sex.R' 'species.R' 'QC.R' 'GEO.R'  
'SigDFMethods.R' 'sesame.R' 'age.R' 'background.R'  
'cell\_composition.R' 'channel\_inference.R' 'cnv.R' 'impute.R'  
'mLiftOver.R' 'ethnicity.R' 'deidentify.R' 'detection.R' 'dm.R'  
'dye\_bias.R' 'feature\_selection.R' 'fileSet.R' 'mask.R'  
'sesameAnno.R' 'open.R' 'strain.R' 'tissue.R' 'track.R'  
'match\_design.R' 'utils.R' 'vcf.R' 'visualize.R'  
'visualizeHelper.R' 'zzz.R' 'palgen.R'

**git\_url** <https://git.bioconductor.org/packages/sesame>  
**git\_branch** devel  
**git\_last\_commit** 715da26  
**git\_last\_commit\_date** 2025-01-04  
**Repository** Bioconductor 3.21  
**Date/Publication** 2025-04-08  
**Author** Wanding Zhou [aut, cre] (ORCID:  
     <<https://orcid.org/0000-0001-9126-1932>>),  
     Wubin Ding [ctb],  
     David Goldberg [ctb],  
     Ethan Moyer [ctb],  
     Bret Barnes [ctb],  
     Timothy Triche [ctb],  
     Hui Shen [aut]  
**Maintainer** Wanding Zhou <[zhouwanding@gmail.com](mailto:zhouwanding@gmail.com)>

## Contents

sesame-package . . . . .	5
addMask . . . . .	6
assemble_plots . . . . .	6
betasCollapseToPfx . . . . .	7
BetaValueToMValue . . . . .	8
binSignals . . . . .	9
bisConversionControl . . . . .	9
calcEffectSize . . . . .	10
checkLevels . . . . .	11
chipAddressToSignal . . . . .	11
cnSegmentation . . . . .	12
compareMouseStrainReference . . . . .	13
compareMouseTissueReference . . . . .	14
compareReference . . . . .	15
controls . . . . .	16
convertProbeID . . . . .	16
createUCSCtrack . . . . .	17
dataFrame2sesameQC . . . . .	18
deIdentify . . . . .	18
detectionPnegEcdf . . . . .	19
diffRefSet . . . . .	20
dmContrasts . . . . .	20
DML . . . . .	21
DMLpredict . . . . .	22
DMR . . . . .	23
dyeBiasCorr . . . . .	24
dyeBiasCorrMostBalanced . . . . .	25
dyeBiasL . . . . .	25

dyeBiasNL	26
ELBAR	27
estimateLeukocyte	28
formatVCF	29
getAFs	30
getAFTYPEIbySumAlleles	30
getBetas	31
getBinCoordinates	32
getMask	32
getRefSet	33
imputeBetas	34
imputeBetasByGenomicNeighbors	35
imputeBetasMatrixByMean	36
inferEthnicity	36
inferInfiniumIChannel	37
inferSex	38
inferSpecies	39
inferStrain	40
inferTissue	41
initFileSet	42
liftOver	43
listAvailableMasks	43
mapFileSet	44
mapToMammal40	44
matchDesign	45
meanIntensity	46
medianTotalIntensity	46
mLiftOver	47
MValueToBetaValue	50
negControls	50
noMasked	51
noob	51
normControls	52
openSesame	53
openSesameToFile	54
palgen	55
parseGEOsignalMU	55
pOOBAH	56
predictAge	57
predictAgeHorvath353	58
predictAgeSkinBlood	58
predictMouseAgeInMonth	59
prefixMask	60
prefixMaskButC	60
prefixMaskButCG	61
prepSesame	61
prepSesameList	62
print.DMLSummary	63

print.fileSet . . . . .	63
probeID_designType . . . . .	64
probeSuccessRate . . . . .	65
qualityMask . . . . .	65
readFileSet . . . . .	66
readIDATpair . . . . .	67
recommendedMaskNames . . . . .	68
reIdentify . . . . .	68
resetMask . . . . .	69
scrub . . . . .	70
scrubSoft . . . . .	70
SDFcollapseToPfx . . . . .	71
sdfPlatform . . . . .	71
sdf_read_table . . . . .	72
sdf_write_table . . . . .	73
searchIDATprefixes . . . . .	73
segmentBins . . . . .	74
sesameAnno_attachManifest . . . . .	74
sesameAnno_buildAddressFile . . . . .	75
sesameAnno_buildManifestGRanges . . . . .	76
sesameAnno_download . . . . .	77
sesameAnno_readManifestTSV . . . . .	78
sesameQC-class . . . . .	78
sesameQCtoDF . . . . .	79
sesameQC_calcStats . . . . .	79
sesameQC_getStats . . . . .	80
sesameQC_plotBar . . . . .	81
sesameQC_plotBetaByDesign . . . . .	81
sesameQC_plotHeatSNPs . . . . .	82
sesameQC_plotIntensVsBetas . . . . .	83
sesameQC_plotRedGrnQQ . . . . .	84
sesameQC_rankStats . . . . .	84
sesame_checkVersion . . . . .	85
sesamize . . . . .	85
setMask . . . . .	86
SigDF . . . . .	86
signalMU . . . . .	87
sliceFileSet . . . . .	88
summaryExtractTest . . . . .	89
totalIntensities . . . . .	89
twoCompsEst2 . . . . .	90
updateSigDF . . . . .	91
visualizeGene . . . . .	91
visualizeProbes . . . . .	92
visualizeRegion . . . . .	93
visualizeSegments . . . . .	95

---

sesame-package      *Analyze DNA methylation data*

---

## Description

SEnsible and step-wise analysis of DNA methylation data

## Details

This package complements array functionalities that allow processing >10,000 samples in parallel on clusters.

## Value

package

## Author(s)

Wanding Zhou <Wanding.Zhou@vai.org>, Hui Shen <Hui.Shen@vai.org> Timothy J Triche Jr <Tim.Triche@vai.org>

## References

Zhou W, Triche TJ, Laird PW, Shen H (2018)

## See Also

Useful links:

- <https://github.com/zwdzwd/sesame>
- Report bugs at <https://github.com/zwdzwd/sesame/issues>

## Examples

```
sdf <- readIDATpair(sub('_Grn.idat', '', system.file(
  'extdata', '4207113116_A_Grn.idat', package='sesameData')))
```

```
## The OpenSesame pipeline
betas <- openSesame(sdf)
```

addMask *Add probes to mask*

---

**Description**

This function essentially merge existing probe masking with new probes to mask

**Usage**

```
addMask(sdf, probes)
```

**Arguments**

sdf                    a SigDF  
probes                a vector of probe IDs or a logical vector with TRUE representing masked probes

**Value**

a SigDF with added mask

**Examples**

```
sdf <- sesameDataGet('EPIC.1.SigDF')  
sum(sdf$mask)  
sum(addMask(sdf, c("cg14057072", "cg22344912"))$mask)
```

---

assemble\_plots *assemble plots*

---

**Description**

assemble plots

**Usage**

```
assemble_plots(  
  betas,  
  txns,  
  probes,  
  plt.txns,  
  plt.mapLines,  
  plt.cytoband,  
  heat.height = NULL,  
  mapLine.height = 0.2,  
  show.probeNames = TRUE,  
  show.samples.n = NULL,
```

```

    show.sampleNames = TRUE,
    sample.name.fontsize = 10,
    dmin = 0,
    dmax = 1
)

```

### Arguments

betas	beta value
txns	transcripts GRanges
probes	probe GRanges
plt.txns	transcripts plot objects
plt.mapLines	map line plot objects
plt.cytoband	cytoband plot objects
heat.height	heatmap height (auto inferred based on rows)
mapLine.height	height of the map lines
show.probeNames	whether to show probe names
show.samples.n	number of samples to show (default: all)
show.sampleNames	whether to show sample names
sample.name.fontsize	sample name font size
dmin	data min
dmax	data max

### Value

a grid object

---

betasCollapseToPfx      *Collapse betas by averagng probes with common probe ID prefix*

---

### Description

Collapse betas by averagng probes with common probe ID prefix

### Usage

```
betasCollapseToPfx(betas, BPPARAM = SerialParam())
```

**Arguments**

**betas** either a named numeric vector or a numeric matrix (row: probes, column: samples)

**BPPARAM** use MulticoreParam(n) for parallel processing

**Value**

either named numeric vector or a numeric matrix of collapsed beta value matrix

**Examples**

```
## input is a matrix
m <- matrix(seq(0,1,length.out=9), nrow=3)
rownames(m) <- c("cg00004963_TC21", "cg00004963_TC22", "cg00004747_TC21")
colnames(m) <- c("A","B","C")
betasCollapseToPfx(m)

## input is a vector
m <- setNames(seq(0,1,length.out=3),
  c("cg00004963_TC21", "cg00004963_TC22", "cg00004747_TC21"))
betasCollapseToPfx(m)
```

---

BetaValueToMValue      *Convert beta-value to M-value*

---

**Description**

Logit transform a beta value vector to M-value vector.

**Usage**

```
BetaValueToMValue(b)
```

**Arguments**

**b** vector of beta values

**Details**

Convert beta-value to M-value (aka logit transform)

**Value**

a vector of M values

**Examples**

```
BetaValueToMValue(c(0.1, 0.5, 0.9))
```



---

binSignals	<i>Bin signals from probe signals</i>
------------	---------------------------------------

---

**Description**

require GenomicRanges

**Usage**

```
binSignals(probe.signals, bin.coords, probeCoords)
```

**Arguments**

probe.signals	probe signals
bin.coords	bin coordinates
probeCoords	probe coordinates

**Value**

bin signals

---

bisConversionControl	<i>Compute internal bisulfite conversion control</i>
----------------------	--

---

**Description**

Compute GCT score for internal bisulfite conversion control. The function takes a SigSet as input. The higher the GCT score, the more likely the incomplete conversion.

**Usage**

```
bisConversionControl(sdf, extR = NULL, extA = NULL, verbose = FALSE)
```

**Arguments**

sdf	a SigDF
extR	a vector of probe IDs for Infinium-I probes that extend to converted A
extA	a vector of probe IDs for Infinium-I probes that extend to original A
verbose	print more messages

**Value**

GCT score (the higher, the more incomplete conversion)

**Examples**

```

sesameDataCache() # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
bisConversionControl(sdf)

## For more recent platforms like EPICv2, MSA:
## One need extR and extA of other arrays using the sesameAnno
## Not run:
mft = sesameAnno_buildManifestGRanges(sprintf(
  "%s/EPICv2/EPICv2.hg38.manifest.tsv.gz",
  "https://github.com/zhou-lab/InfiniumAnnotationV1/raw/main/Anno/"),
  columns="nextBase")
extR = names(mft)[!is.na(mft$nextBase) & mft$nextBase=="R"]
extA = names(mft)[!is.na(mft$nextBase) & mft$nextBase=="A"]

## End(Not run)

```

---

**calcEffectSize**
*Compute effect size for different variables from prediction matrix*


---

**Description**

The effect size is defined by the maximum variation of a variable with all the other variables controlled constant.

**Usage**

```
calcEffectSize(pred)
```

**Arguments**

pred            predictions

**Value**

a data.frame of effect sizes. Columns are different variables. Rows are different probes.

**Examples**

```

data <- sesameDataGet('HM450.76.TCGA.matched')
res <- DMLpredict(data$betas[1:10,], ~type, meta=data$sampleInfo)
head(calcEffectSize(res))

```

---

checkLevels	<i>filter data matrix by factor completeness only works for discrete factors</i>
-------------	--

---

**Description**

filter data matrix by factor completeness only works for discrete factors

**Usage**

```
checkLevels(betas, fc)
```

**Arguments**

betas	matrix data
fc	factors, or characters

**Value**

a boolean vector whether there is non-NA value for each tested group for each probe

**Examples**

```
se0 <- sesameDataGet("MM285.10.SE.tissue")[1:100,]
se_ok <- checkLevels(SummarizedExperiment::assay(se0),
  SummarizedExperiment::colData(se0)$tissue)
sum(se_ok) # number of good probes
se1 <- se0[se_ok,]

sesameDataGet_resetEnv()
```

---

chipAddressToSignal	<i>Lookup address in one sample</i>
---------------------	-------------------------------------

---

**Description**

Lookup address and transform address to probe

**Usage**

```
chipAddressToSignal(dm, mft, min_beads = NULL)
```

**Arguments**

dm	data frame in chip address, 2 columns: cy3/Grn and cy5/Red
mft	a data frame with columns Probe_ID, M, U and col
min_beads	minimum bead counts, otherwise masked

**Details**

Translate data in chip address to probe address. Type I probes can be separated into Red and Grn channels. The methylated allele and unmethylated allele are at different addresses. For type II probes methylation allele and unmethylated allele are at the same address. Grn channel is for methylated allele and Red channel is for unmethylated allele. The out-of-band signals are type I probes measured using the other channel.

**Value**

a SigDF, indexed by probe ID address

---

cnSegmentation	<i>Perform copy number segmentation</i>
----------------	---

---

**Description**

Perform copy number segmentation using the signals in the signal set. The function takes a SigDF for the target sample and a set of normal SigDF for the normal samples. An optional arguments specifies the version of genome build that the inference will operate on. The function outputs an object of class CNSegment with signals for the segments ( seg.signals), the bin coordinates ( bin.coords) and bin signals (bin.signals).

**Usage**

```
cnSegmentation(
  sdf,
  sdfs.normal = NULL,
  genomeInfo = NULL,
  probeCoords = NULL,
  tilewidth = 50000,
  verbose = FALSE,
  return.probe.signals = FALSE
)
```

**Arguments**

sdf	SigDF
sdfs.normal	a list of SigDFs for normalization, if not given, use the stored normal data from sesameData. However, we do recommend using a matched copy number normal dataset for normalization. assembly
genomeInfo	the genomeInfo files. The default is retrieved from sesameData. Alternative genomeInfo files can be found at <a href="https://github.com/zhou-lab/GenomeInfo">https://github.com/zhou-lab/GenomeInfo</a>
probeCoords	the probe coordinates in the corresponding genome if NULL (default), then the default genome assembly is used. Default genome is given by, e.g., sesameData_check_genome(NULL, "EPIC") For additional mapping, download the GRanges object from <a href="http://zwdzwd.github.io/InfiniumAnnotation">http://zwdzwd.github.io/InfiniumAnnotation</a> and provide

the following argument ..., probeCoords = sesameAnno\_buildManifestGRanges("downloaded\_file"),...  
to this function.

tilewidth        tile width for smoothing  
verbose         print more messages  
return.probe.signals  
                 return probe-level instead of bin-level signal

### Value

an object of CNSegment

### Examples

```
sesameDataCache()

## Not run:
sdfs <- sesameDataGet('EPICv2.8.SigDF')
sdf <- sdfs[["K562_206909630040_R01C01"]]
seg <- cnSegmentation(sdf)
seg <- cnSegmentation(sdf, return.probe.signals=TRUE)
visualizeSegments(seg)

## End(Not run)
```

---

compareMouseStrainReference

*Compare Strain SNPs with a reference panel*

---

### Description

Compare Strain SNPs with a reference panel

### Usage

```
compareMouseStrainReference(
  betas = NULL,
  show_sample_names = FALSE,
  query_width = NULL
)
```

### Arguments

betas            beta value vector or matrix (for multiple samples)  
show\_sample\_names  
                 whether to show sample name  
query\_width     optional argument for adjusting query width

**Value**

grid object that contrast the target sample with pre-built mouse strain reference

**Examples**

```
sesameDataCache() # if not done yet
compareMouseStrainReference()
```

---

compareMouseTissueReference

*Compare mouse array data with mouse tissue references*

---

**Description**

Compare mouse array data with mouse tissue references

**Usage**

```
compareMouseTissueReference(
  betas = NULL,
  ref = NULL,
  color = "blueYellow",
  query_width = 0.3
)
```

**Arguments**

betas	matrix of betas for the target sample This argument is optional. If not given, only the reference will be shown.
ref	the reference beta values in SummarizedExperiment. This argument is optional. If not given, the reference will be downloaded from the sesameData package.
color	either blueYellow or fullJet
query_width	the width of the query beta value matrix

**Value**

grid object that contrast the target sample with pre-built mouse tissue reference

**Examples**

```
cat("Deprecated, see compareReference")
```

---

compareReference	<i>Compare array data with references (e.g., tissue, cell types)</i>
------------------	--

---

### Description

Compare array data with references (e.g., tissue, cell types)

### Usage

```
compareReference(  
  ref,  
  betas = NULL,  
  stop.points = NULL,  
  query_width = 0.3,  
  show_sample_names = FALSE  
)
```

### Arguments

ref	the reference beta values in SummarizedExperiment. One can download them from the sesameData package. See examples.
betas	matrix of betas for the target sample This argument is optional. If not given, only the reference will be shown.
stop.points	stop points for the color palette. Default to blue, yellow.
query_width	the width of the query beta value matrix
show_sample_names	whether to show sample names (default: FALSE)

### Value

grid object that contrast the target sample with references.

### Examples

```
sesameDataCache() # if not done yet  
compareReference(sesameDataGet("MM285.tissueSignature"))  
sesameDataGet_resetEnv()
```

---

controls	<i>get the controls attributes</i>
----------	------------------------------------

---

**Description**

get the controls attributes

**Usage**

```
controls(sdf, verbose = FALSE)
```

**Arguments**

sdf	a SigDF
verbose	print more messages

**Value**

the controls data frame

**Examples**

```
sesameDataCache() # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
head(controls(sdf))
```

---

convertProbeID	<i>Convert Probe ID</i>
----------------	-------------------------

---

**Description**

Convert Probe ID

**Usage**

```
convertProbeID(
  x,
  target_platform,
  source_platform = NULL,
  mapping = NULL,
  target_uniq = TRUE,
  include_new = FALSE,
  include_old = FALSE,
  return_mapping = FALSE
)
```



**Arguments**

x	source probe IDs
target_platform	the platform to take the data to
source_platform	optional source platform
mapping	a liftOver mapping file. Typically this file contains empirical evidence whether a probe mapping is reliable. If given, probe ID-based mapping will be skipped. This is to perform more stringent probe ID mapping.
target_uniq	whether the target Probe ID should be kept unique.
include_new	if true, include mapping of added probes
include_old	if true, include mapping of deleted probes
return_mapping	return mapping table, instead of the target IDs.

**Value**

mapped probe IDs, or mapping table if return\_mapping = T

---

createUCSCtrack	<i>Turn beta values into a UCSC browser track</i>
-----------------	---

---

**Description**

Turn beta values into a UCSC browser track

**Usage**

```
createUCSCtrack(betas, output = NULL, platform = "HM450", genome = "hg38")
```

**Arguments**

betas	a named numeric vector
output	output file name
platform	HM450, EPIC etc.
genome	hg38, mm10, ..., will infer if not given. For additional mapping, download the GRanges object from <a href="http://zwdzwd.github.io/InfiniumAnnotation">http://zwdzwd.github.io/InfiniumAnnotation</a> and provide the following argument ..., genome = sesameAnno_buildManifestGRanges("downloaded_file"),... to this function.

**Value**

when output is null, return a data.frame, otherwise NULL

**Examples**

```

betas.tissue <- sesameDataGet('HM450.1.TCGA.PAAD')$betas
## add output to create an actual file
df <- createUCSCtrack(betas.tissue)

## to convert to bigBed
## sort -k1,1 -k2,2n output.bed >output_sorted.bed
## bedToBigBed output_sorted.bed hg38.chrom output.bb

```

---

dataFrame2sesameQC      *Convert data frame to sesameQC object*

---

**Description**

The function convert a data frame back to a list of sesameQC objects

**Usage**

```
dataFrame2sesameQC(df)
```

**Arguments**

df                      a publicQC data frame

**Value**

a list sesameQC objects

---

deIdentify              *De-identify IDATs by removing SNP probes*

---

**Description**

Mask SNP probe intensity mean by zero.

**Usage**

```
deIdentify(path, out_path = NULL, snps = NULL, mft = NULL, randomize = FALSE)
```

**Arguments**

path                    input IDAT file  
out\_path                output IDAT file  
snps                    SNP definition, if not given, default to SNP probes  
mft                     sesame-compatible manifest if non-standard  
randomize              whether to randomize the SNPs. if TRUE, randomize the signal intensities. one can use set.seed to reidentify the IDAT with the secret seed (see examples). If FALSE, this sets all SNP intensities to zero.

**Value**

NULL, changes made to the IDAT files

**Examples**

```
my_secret <- 13412084
set.seed(my_secret)
temp_out <- tempfile("test")
deIdentify(system.file(
  "extdata", "4207113116_A_Grn.idat", package = "sesameData"),
  temp_out, randomize = TRUE)
unlink(temp_out)
```

---

detectionPnegEcdf	<i>Detection P-value based on ECDF of negative control</i>
-------------------	--

---

**Description**

The function takes a SigDF as input, computes detection p-value using negative control probes' empirical distribution and returns a new SigDF with an updated mask slot.

**Usage**

```
detectionPnegEcdf(sdf, return.pval = FALSE, pval.threshold = 0.05)
```

**Arguments**

sdf                    a SigDF  
 return.pval        whether to return p-values, instead of a masked SigDF  
 pval.threshold    minimum p-value to mask

**Value**

a SigDF, or a p-value vector if return.pval is TRUE

**Examples**

```
sdf <- sesameDataGet("EPIC.1.SigDF")
sum(sdf$mask)
sum(detectionPnegEcdf(sdf)$mask)
```

---

diffRefSet	<i>Restrict refset to differentially methylated probes use with care, might introduce bias</i>
------------	--

---

**Description**

The function takes a matrix with probes on the rows and cell types on the columns and output a subset matrix and only probes that show discordant methylation levels among the cell types.

**Usage**

```
diffRefSet(g)
```

**Arguments**

`g` a matrix with probes on the rows and cell types on the columns

**Value**

`g` a matrix with a subset of input probes (rows)

**Examples**

```
g = diffRefSet(getRefSet(platform='HM450'))
sesameDataGet_resetEnv()
```

---

dmContrasts	<i>List all contrasts of a DMLSummary</i>
-------------	---

---

**Description**

List all contrasts of a DMLSummary

**Usage**

```
dmContrasts(smry)
```

**Arguments**

`smry` a DMLSummary object

**Value**

a character vector of contrasts

**Examples**

```
data <- sesameDataGet('HM450.76.TCGA.matched')
smry <- DML(data$betas[1:10,], ~type, meta=data$sampleInfo)
dmContrasts(smry)

sesameDataGet_resetEnv()
```

DML

*Test differential methylation on each locus***Description**

The function takes a beta value matrix with probes on the rows and samples on the columns. It also takes a sample information data frame (meta) and formula for testing. The function outputs a list of coefficient tables for each factor tested.

**Usage**

```
DML(betas, fm, meta = NULL, BPPARAM = SerialParam())
```

**Arguments**

betas	beta values, matrix or SummarizedExperiment rows are probes and columns are samples.
fm	formula
meta	data frame for sample information, column names are predictor variables (e.g., sex, age, treatment, tumor/normal etc) and are referenced in formula. Rows are samples. When the betas argument is a SummarizedExperiment object, this is ignored. colData(betas) will be used instead. The row order of the data frame must match the column order of the beta value matrix.
BPPARAM	number of cores for parallel processing, default to SerialParam() Use Multi-coreParam(mc.cores) for parallel processing. For Windows, try DoparParam or SnowParam.

**Value**

a list of test summaries, summary.lm objects

**Examples**

```
sesameDataCache() # in case not done yet
data <- sesameDataGet('HM450.76.TCGA.matched')
smry <- DML(data$betas[1:1000,], ~type, meta=data$sampleInfo)

sesameDataGet_resetEnv()
```

DMLpredict

*Predict new data from DML***Description**

This function is also important for investigating factor interactions.

**Usage**

```
DMLpredict(betas, fm, pred = NULL, meta = NULL, BPPARAM = SerialParam())
```

**Arguments**

betas	beta values, matrix or SummarizedExperiment rows are probes and columns are samples.
fm	formula
pred	new data for prediction, useful for studying effect size. This argument is a data.frame to specify new data. If the argument is NULL, all combinations of all contrasts will be used as input. It might not work if there is a continuous variable input. One may need to explicitly provide the input in a data frame.
meta	data frame for sample information, column names are predictor variables (e.g., sex, age, treatment, tumor/normal etc) and are referenced in formula. Rows are samples. When the betas argument is a SummarizedExperiment object, this is ignored. colData(betas) will be used instead.
BPPARAM	number of cores for parallel processing, default to SerialParam() Use Multi-coreParam(mc.cores) for parallel processing. For Windows, try DoparParam or SnowParam.

**Value**

a SummarizedExperiment of predictions. The colData describes the input of the prediction.

**Examples**

```
data <- sesameDataGet('HM450.76.TCGA.matched')

## use all contrasts as new input
res <- DMLpredict(data$betas[1:10,], ~type, meta=data$sampleInfo)

## specify new input
res <- DMLpredict(data$betas[1:10,], ~type, meta=data$sampleInfo,
  pred = data.frame(type=c("Normal", "Tumour")))

## note that the prediction needs to be a factor of the same
## level structure as the original training data.
pred = data.frame(type=factor(c("Normal"), levels=c("Normal", "Tumour")))
res <- DMLpredict(data$betas[1:10,], ~type,
```

```
meta=data$sampleInfo, pred = pred)
```

DMR

*Find Differentially Methylated Region (DMR)***Description**

This subroutine uses Euclidean distance to group CpGs and then combine p-values for each segment. The function performs DML test first if `cf` is `NULL`. It groups the probe testing results into differential methylated regions in a coefficient table with additional columns designating the segment ID and statistical significance (P-value) testing the segment.

**Usage**

```
DMR(
  betas,
  smry,
  contrast,
  platform = NULL,
  probe.coords = NULL,
  dist.cutoff = NULL,
  seg.per.locus = 0.5
)
```

**Arguments**

<code>betas</code>	beta values for distance calculation
<code>smry</code>	DML
<code>contrast</code>	the pair-wise comparison or contrast check <code>colnames(attr(smry, "model.matrix"))</code> if uncertain
<code>platform</code>	EPIC, HM450, MM285, ...
<code>probe.coords</code>	<code>GRanges</code> object that defines CG coordinates if <code>NULL</code> (default), then the default genome assembly is used. Default genome is given by, e.g., <code>sesameData_check_genome(NULL, "EPIC")</code> For additional mapping, download the <code>GRanges</code> object from <a href="http://zwdzwd.github.io/InfiniumAnnotation">http://zwdzwd.github.io/InfiniumAnnotation</a> and provide the following argument ..., <code>probe.coords = sesameAnno_buildManifestGRanges("downloaded_file"),...</code> to this function.
<code>dist.cutoff</code>	cutoff of beta value differences for two neighboring CGs to be considered the same DMR (by default it's determined using the quantile function on <code>seg.per.locus</code> )
<code>seg.per.locus</code>	number of segments per locus higher value leads to more segments

**Value**

coefficient table with segment ID and segment P-value each row is a locus, multiple loci may share a segment ID if they are merged to the same segment. Records are ordered by `Seg_Est`.

## Examples

```
sesameDataCache() # in case not done yet

data <- sesameDataGet('HM450.76.TCGA.matched')
smry <- DML(data$betas[1:1000,], ~type, meta=data$sampleInfo)
colnames(attr(smry, "model.matrix")) # pick a contrast from here
## showing on a small set of 100 CGs
merged_segs <- DMR(data$betas[1:1000,], smry, "typeTumour", platform="HM450")

sesameDataGet_resetEnv()
```

---

dyeBiasCorr

*Correct dye bias in by linear scaling.*

---

## Description

The function takes a SigDF as input and scale both the Grn and Red signal to a reference (ref) level. If the reference level is not given, it is set to the mean intensity of all the in-band signals. The function returns a SigDF with dye bias corrected.

## Usage

```
dyeBiasCorr(sdf, ref = NULL)
```

## Arguments

sdf	a SigDF
ref	reference signal level

## Value

a normalized SigDF

## Examples

```
sesameDataCache() # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
sdf.db <- dyeBiasCorr(sdf)
```



---

`dyeBiasCorrMostBalanced`*Correct dye bias using most balanced sample as the reference*

---

**Description**

The function chose the reference signal level from a list of SigDF. The chosen sample has the smallest difference in Grn and Red signal intensity as measured using the normalization control probes. In practice, it doesn't matter which sample is chosen as long as the reference level does not deviate much. The function returns a list of SigDFs with dye bias corrected.

**Usage**

```
dyeBiasCorrMostBalanced(sdfs)
```

**Arguments**

sdfs                    a list of normalized SigDFs

**Value**

a list of normalized SigDFs

**Examples**

```
sesameDataCache() # if not done yet
sdfs <- sesameDataGet('HM450.10.SigDF')[1:2]
sdfs.db <- dyeBiasCorrMostBalanced(sdfs)
```

---

`dyeBiasL`*Correct dye bias in by linear scaling.*

---

**Description**

The function takes a SigDF as input and scale both the Grn and Red signal to a reference (ref) level. If the reference level is not given, it is set to the mean intensity of all the in-band signals. The function returns a SigDF with dye bias corrected.

**Usage**

```
dyeBiasL(sdf, ref = NULL)
```

**Arguments**

sdf                    a SigDF  
ref                    reference signal level

**Value**

a normalized SigDF

**Examples**

```
sesameDataCache() # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
sdf.db <- dyeBiasL(sdf)
```

---

dyeBiasNL

*Dye bias correction by matching green and red to mid point*

---

**Description**

This function compares the Type-I Red probes and Type-I Grn probes and generates and mapping to correct signal of the two channels to the middle. The function takes one single SigDF and returns a SigDF with dye bias corrected.

**Usage**

```
dyeBiasNL(sdf, mask = TRUE, verbose = FALSE)
```

```
dyeBiasCorrTypeINorm(sdf, mask = TRUE, verbose = FALSE)
```

**Arguments**

sdf	a SigDF
mask	include masked probes in Infinium-I probes. No big difference is noted in practice. More probes are generally better.
verbose	print more messages

**Value**

a SigDF after dye bias correction.

**Examples**

```
sesameDataCache() # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
sdf.db <- dyeBiasNL(sdf)
sdf <- sesameDataGet('EPIC.1.SigDF')
sdf <- dyeBiasCorrTypeINorm(sdf)
```

---

**ELBAR***ELiminate BAcground-dominated Reading (ELBAR)*

---

**Description**

ELiminate BAcground-dominated Reading (ELBAR)

**Usage**

```
ELBAR(  
  sdf,  
  return.pval = FALSE,  
  pval.threshold = 0.05,  
  margin = 0.05,  
  capMU = 3000,  
  delta.beta = 0.2,  
  n.windows = 500  
)
```

**Arguments**

sdf	a SigDF
return.pval	whether to return p-values, instead of a SigDF
pval.threshold	minimum p-value to mask
margin	the percentile margin to define envelope, the smaller the value the more aggressive the masking.
capMU	the maximum M+U to search for intermediate betas
delta.beta	maximum beta value change from sheer background-dominated readings
n.windows	number of windows for smoothing

**Value**

a SigDF with mask added

**Examples**

```
sdf <- sesameDataGet("EPIC.1.SigDF")  
sum(sdf$mask)  
sum(ELBAR(sdf)$mask)
```

---

estimateLeukocyte	<i>Estimate leukocyte fraction using a two-component model</i>
-------------------	--

---

### Description

The method assumes only two components in the mixture: the leukocyte component and the target tissue component. The function takes the beta values matrix of the target tissue and the beta value matrix of the leukocyte. Both matrices have probes on the row and samples on the column. Row names should have probe IDs from the platform. The function outputs a single numeric describing the fraction of leukocyte.

### Usage

```
estimateLeukocyte(  
  betas.tissue,  
  betas.leuko = NULL,  
  betas.tumor = NULL,  
  platform = c("EPIC", "HM450", "HM27")  
)
```

### Arguments

betas.tissue	tissue beta value matrix (#probes X #samples)
betas.leuko	leukocyte beta value matrix, if missing, use the SeSAmE default by infinium platform
betas.tumor	optional, tumor beta value matrix
platform	"HM450", "HM27" or "EPIC"

### Value

leukocyte estimate, a numeric vector

### Examples

```
betas.tissue <- sesameDataGet('HM450.1.TCGA.PAAD')$betas  
estimateLeukocyte(betas.tissue)  
sesameDataGet_resetEnv()
```

---

formatVCF	<i>Convert SNP from Infinium array to VCF file</i>
-----------	--

---

## Description

Convert SNP from Infinium array to VCF file

## Usage

```
formatVCF(sdf, anno, vcf = NULL, genome = "hg38", verbose = FALSE)
```

## Arguments

sdf	SigDF
anno	SNP variant annotation, available at <a href="https://github.com/zhou-lab/InfiniumAnnotationV1/tree/main/Anno/EPIC.hg38.snp.tsv.gz">https://github.com/zhou-lab/InfiniumAnnotationV1/tree/main/Anno/EPIC.hg38.snp.tsv.gz</a>
vcf	output VCF file path, if NULL output to console
genome	genome
verbose	print more messages

## Value

VCF file. If vcf is NULL, a data.frame is output to console. The data.frame does not contain VCF headers. Note the output vcf is not sorted.

## Examples

```
sesameDataCacheAll() # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')

## Not run:
## download anno from
## http://zwdzwd.github.io/InfiniumAnnotation
## output to console
anno = read_tsv(sesameAnno_download("EPICv2.hg38.snp.tsv.gz"))
head(formatVCF(sdf, anno))

## End(Not run)
```

getAFs

*Get allele frequency*

---

**Description**

Get allele frequency

**Usage**

```
getAFs(sdf, ...)
```

**Arguments**

sdf	SigDF
...	additional options to getBetas

**Value**

allele frequency

**Examples**

```
sesameDataCache() # if not done yet  
sdf <- sesameDataGet('EPIC.1.SigDF')  
af <- getAFs(sdf)
```

---

getAFTypeIbySumAlleles

*Get allele frequency treating type I by summing alleles*

---

**Description**

Takes a SigDF as input and returns a numeric vector containing extra allele frequencies based on Color-Channel-Switching (CCS) probes. If no CCS probes exist in the SigDF, then an numeric(0) is returned.

**Usage**

```
getAFTypeIbySumAlleles(sdf, known.ccs.only = TRUE)
```

**Arguments**

sdf	SigDF
known.ccs.only	consider only known CCS probes

**Value**

beta values

**Examples**

```
sesameDataCache() # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
af <- getAFTYPEIbySumAlleles(sdf)
```

---

getBetas

*Get beta Values*

---

**Description**

sum.typeI is used for rescuing beta values on Color-Channel-Switching CCS probes. The function takes a SigDF and returns beta value except that Type-I in-band signal and out-of-band signal are combined. This prevents color-channel switching due to SNPs.

**Usage**

```
getBetas(
  sdf,
  mask = TRUE,
  sum.TypeI = FALSE,
  collapseToPfx = FALSE,
  collapseMethod = c("mean", "minPval")
)
```

**Arguments**

sdf	SigDF
mask	whether to use mask
sum.TypeI	whether to sum type I channels
collapseToPfx	remove replicate to prefix (e.g., cg number) and remove the suffix
collapseMethod	mean or minPval

**Value**

a numeric vector, beta values

**Examples**

```
sesameDataCache() # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
betas <- getBetas(sdf)
```

---

getBinCoordinates      *Get bin coordinates*

---

**Description**

requires GenomicRanges, IRanges

**Usage**

```
getBinCoordinates(seqLength, gapInfo, tilewidth = 50000, probeCoords)
```

**Arguments**

seqLength	chromosome information object
gapInfo	chromosome gap information
tilewidth	tile width for smoothing
probeCoords	probe coordinates

**Value**

bin.coords

---

getMask      *get probe masking by mask names*

---

**Description**

get probe masking by mask names

**Usage**

```
getMask(platform = "EPICv2", mask_names = "recommended")
```

**Arguments**

platform	EPICv2, EPIC, HM450, HM27, ...
mask_names	mask names (see listAvailableMasks) by default: "recommended" see recommendedMaskNames() for detail.

**Value**

a vector of probe ID



## Examples

```
length(getMask("MSA", "recommended"))
length(getMask("EPICv2", "recommended"))
length(getMask("EPICv2", c("recommended", "M_SNPcommon_1pt")))
length(getMask("EPICv2", "M_mapping"))
length(getMask("EPIC"))
length(getMask("HM450"))
length(getMask("MM285"))
```

---

getRefSet	<i>Retrieve reference set</i>
-----------	-------------------------------

---

## Description

The function retrieves the curated reference DNA methylation status for a set of cell type names under the Infinium platform. Supported cell types include "CD4T", "CD19B", "CD56NK", "CD14Monocytes", "granulocytes", "scFat", "skin" etc. See package `sesameData` for more details. The function output a matrix with probes on the rows and specified cell types on the columns. 0 suggests unmethylation and 1 suggests methylation. Intermediate methylation and nonclusive calls are left with NA.

## Usage

```
getRefSet(cells = NULL, platform = c("EPIC", "HM450"))
```

## Arguments

cells	reference cell types
platform	EPIC or HM450

## Value

g, a 0/1 matrix with probes on the rows and specified cell types on the columns.

## Examples

```
betas = getRefSet('CD4T', platform='HM450')
sesameDataGet_resetEnv()
```

---

imputeBetas	<i>Impute of missing data of specific platform</i>
-------------	--

---

### Description

Impute of missing data of specific platform

### Usage

```
imputeBetas(  
  betas,  
  platform = NULL,  
  BPPARAM = SerialParam(),  
  celltype = NULL,  
  sd_max = 999  
)
```

### Arguments

betas	named vector of beta values
platform	platform
BPPARAM	use MulticoreParam(n) for parallel processing
celltype	celltype/tissue context of imputation, if not given, will use nearest neighbor to determine.
sd_max	maximum standard deviation in imputation confidence

### Value

imputed data, vector or matrix

### Examples

```
betas = openSesame(sesameDataGet("EPIC.1.SigDF"))  
sum(is.na(betas))  
betas2 = imputeBetas(betas, "EPIC")  
sum(is.na(betas2))
```

---

`imputeBetasByGenomicNeighbors`*Impute missing data based on genomic neighbors.*

---

## Description

Impute missing data based on genomic neighbors.

## Usage

```
imputeBetasByGenomicNeighbors(  
  betas,  
  platform = NULL,  
  BPPARAM = SerialParam(),  
  max_neighbors = 3,  
  max_dist = 10000  
)
```

## Arguments

<code>betas</code>	named vector of beta values
<code>platform</code>	platform
<code>BPPARAM</code>	use <code>MulticoreParam(n)</code> for parallel processing
<code>max_neighbors</code>	maximum neighbors to use for dense regions
<code>max_dist</code>	maximum distance to count as neighbor

## Value

imputed data, vector or matrix

## Examples

```
betas = openSesame(sesameDataGet("EPICv2.8.SigDF")[[1]])  
sum(is.na(betas))  
betas2 = imputeBetasByGenomicNeighbors(betas, "EPICv2")  
sum(is.na(betas2))
```

---

```
imputeBetasMatrixByMean
```

*Impute Missing Values with Mean This function replaces missing values (NA) in a matrix, default is row means.*

---

### Description

Impute Missing Values with Mean This function replaces missing values (NA) in a matrix, default is row means.

### Usage

```
imputeBetasMatrixByMean(mx, axis = 1)
```

### Arguments

mx	A matrix
axis	A single integer. Use 1 to impute column means (default), and 2 to impute row means.

### Value

A matrix with missing values imputed.

### Examples

```
mx <- cbind(c(1, 2, NA, 4), c(NA, 2, 3, 4))
imputeBetasMatrixByMean(mx, axis = 1)
imputeBetasMatrixByMean(mx, axis = 2)
```

---

```
inferEthnicity
```

*Infer Ethnicity*

---

### Description

This function uses both the built-in rsprobes as well as the type I Color-Channel-Switching probes to infer ethnicity.

### Usage

```
inferEthnicity(sdf, verbose = FALSE)
```

### Arguments

sdf	a SigDF
verbose	print more messages

**Details**

s better be background subtracted and dyebias corrected for best accuracy

Please note: the betas should come from SigDF \*without\* channel inference.

**Value**

string of ethnicity

**Examples**

```
sdf <- sesameDataGet('EPIC.1.SigDF')
## inferEthnicity(sdf)
```

---

`inferInfiniumIChannel` *Infer and reset color channel for Type-I probes instead of using what is specified in manifest. The results are stored to `sdf@extra$IGG` and `sdf@extra$IRR` slot.*

---

**Description**

IGG => Type-I green that is inferred to be green IRR => Type-I red that is inferred to be red

**Usage**

```
inferInfiniumIChannel(
  sdf,
  switch_failed = FALSE,
  mask_failed = FALSE,
  verbose = FALSE,
  summary = FALSE
)
```

**Arguments**

<code>sdf</code>	a SigDF
<code>switch_failed</code>	whether to switch failed probes (default to FALSE)
<code>mask_failed</code>	whether to mask failed probes (default to FALSE)
<code>verbose</code>	whether to print correction summary
<code>summary</code>	return summarized numbers only.

**Value**

a SigDF, or numerics if `summary == TRUE`

**Examples**

```
sdf <- sesameDataGet('EPIC.1.SigDF')
inferInfiniumIChannel(sdf)
```

---

inferSex

*Infer sex.*

---

**Description**

We established our sex calling based on the CpGs hypermethylated in inactive X (XiH), CpGs hypomethylated in inactive X (XiL).

**Usage**

```
inferSex(betas, platform = NULL)
```

**Arguments**

betas	DNA methylation beta
platform	EPICv2, EPIC, HM450, MM285, etc.

**Details**

Note genotype abnormalities such as Dnmt genotype, XXY male (Klinefelter's), 45,X female (Turner's) can confuse the model sometimes. This function works on a single sample.

**Value**

Inferred sex of sample

**Examples**

```
## EPICv2 input
betas = openSesame(sesameDataGet("EPICv2.8.SigDF")[[1]])
inferSex(betas)

## Not run:
## MM285 input
betas = openSesame(sesameDataGet("MM285.1.SigDF"))
inferSex(betas)

## EPIC input
betas = openSesame(sesameDataGet('EPIC.1.SigDF'))
inferSex(betas)

## HM450 input
betas = openSesame(sesameDataGet("HM450.10.SigDF")[[1]])
```

```
inferSex(betas)

## End(Not run)
```

---

inferSpecies	<i>Infer Species</i>
--------------	----------------------

---

### Description

We infer species based on probes pvalues and alignment score. AUC was calculated for each specie,  $y_{true}$  is 1 or 0 for  $pval < threshold.pos$  or  $pval > threshold.neg$ , respectively,

### Usage

```
inferSpecies(
  sdf,
  topN = 1000,
  threshold.pos = 0.01,
  threshold.neg = 0.1,
  return.auc = FALSE,
  return.species = FALSE,
  verbose = FALSE
)
```

### Arguments

sdf	a SigDF
topN	Top n positive and negative probes used to infer species. increase this number can sometimes improve accuracy (DEFAULT: 1000)
threshold.pos	pvalue < threshold.pos are considered positive (default: 0.01).
threshold.neg	pvalue > threshold.neg are considered negative (default: 0.2).
return.auc	return AUC calculated, override return.species
return.species	return a string to represent species
verbose	print more messages

### Value

a SigDF

**Examples**

```
sdf <- sesameDataGet("MM285.1.SigDF")
sdf <- inferSpecies(sdf)

## all available species
all_species <- names(sesameDataGet(sprintf(
  "%s.addressSpecies", sdfPlatform(sdf))))$species)
```

inferStrain

*Infer strain information for mouse array***Description**

Infer strain information for mouse array

**Usage**

```
inferStrain(
  sdf,
  return.strain = FALSE,
  return.probability = FALSE,
  return.pval = FALSE,
  min_frac_dt = 0.2,
  verbose = FALSE
)
```

**Arguments**

sdf	SigDF
return.strain	return strain name
return.probability	return probability vector for all strains
return.pval	return p-value
min_frac_dt	minimum fraction of detected signal (DEFAULT: 0.2) otherwise, we give up strain inference and return NA.
verbose	print more messages

**Value**

a list of best guess, p-value of the best guess and the probabilities of all strains

**Examples**

```
sesameDataCache() # if not done yet
sdf <- sesameDataGet('MM285.1.SigDF')
inferStrain(sdf, return.strain = TRUE)
sdf.strain <- inferStrain(sdf)
```



---

inferTissue	<i>inferTissue infers the tissue of a single sample (as identified through the branchIDs in the row data of the reference) by reporting independent composition through cell type deconvolution.</i>
-------------	--

---

### Description

inferTissue infers the tissue of a single sample (as identified through the branchIDs in the row data of the reference) by reporting independent composition through cell type deconvolution.

### Usage

```
inferTissue(
  betas,
  reference = NULL,
  platform = NULL,
  abs_delta_beta_min = 0.3,
  auc_min = 0.99,
  coverage_min = 0.8,
  topN = 15
)
```

### Arguments

betas	Named vector with probes and their corresponding beta value measurement
reference	Summarized Experiment with either hypomethylated or hypermethylated probe selection (row data), sample selection (column data), meta data, and the betas (assay)
platform	String representing the array type of the betas and reference
abs_delta_beta_min	Numerical value indicating the absolute minimum required delta beta for the probe selection criteria
auc_min	Numeric value corresponding to the minimum AUC value required for a probe to be considered
coverage_min	Numeric value corresponding to the minimum coverage requirement for a probe to be considered. Coverage is defined here as the proportion of samples without an NA value at a given probe.
topN	number of probes to at most use for each branch

### Value

inferred tissue as a string

**Examples**

```
sesameDataCache() # if not done yet
sdf <- sesameDataGet("MM285.1.SigDF")
inferTissue(getBetas(dyeBiasNL(noob(sdf))))

sesameDataGet_resetEnv()
```

---

initFileSet	<i>initialize a fileSet class by allocating appropriate storage</i>
-------------	---

---

**Description**

initialize a fileSet class by allocating appropriate storage

**Usage**

```
initFileSet(map_path, platform, samples, probes = NULL, inc = 4)
```

**Arguments**

map_path	path of file to map
platform	EPIC, HM450 or HM27, consistent with sdfPlatform(sdf)
samples	sample names
probes	probe names
inc	bytes per unit data storage

**Value**

a sesame::fileSet object

**Examples**

```
fset <- initFileSet('mybetas2', 'HM27', c('s1','s2'))
```

---

liftOver	<i>liftOver, see mLiftOver (renamed)</i>
----------	--

---

**Description**

liftOver, see mLiftOver (renamed)

**Usage**

```
liftOver(...)
```

**Arguments**

... see mLiftOver

**Value**

imputed data, vector, matrix, SigDF(s)

---

listAvailableMasks	<i>list existing quality masks for a SigDF</i>
--------------------	--

---

**Description**

list existing quality masks for a SigDF

**Usage**

```
listAvailableMasks(platform, verbose = FALSE)
```

**Arguments**

platform	EPIC, MM285, HM450 etc
verbose	print more messages

**Value**

a tibble of masks

**Examples**

```
listAvailableMasks("EPICv2")
```

---

mapFileSet	<i>Deposit data of one sample to a fileSet (and hence to file)</i>
------------	--

---

**Description**

Deposit data of one sample to a fileSet (and hence to file)

**Usage**

```
mapFileSet(fset, sample, named_values)
```

**Arguments**

fset	a sesame::fileSet, as obtained via readFileSet
sample	sample name as a string
named_values	value vector named by probes

**Value**

a sesame::fileSet

**Examples**

```
## create two samples
fset <- initFileSet('mybetas2', 'HM27', c('s1','s2'))

## a hypothetical numeric array (can be beta values, intensities etc)
hypothetical <- setNames(runif(fset$n), fset$probes)

## map the numeric to file
mapFileSet(fset, 's1', hypothetical)

## get data
sliceFileSet(fset, 's1', 'cg00000292')
```

---

mapToMammal40	<i>Map the SDF (from overlap array platforms) Replicates are merged by picking the best detection</i>
---------------	---

---

**Description**

Map the SDF (from overlap array platforms) Replicates are merged by picking the best detection

**Usage**

```
mapToMammal40(sdf)
```

**Arguments**

sdf                    a SigDF object

**Value**

a named numeric vector for beta values

**Examples**

```
sdf <- sesameDataGet("Mammal40.1.SigDF")
betas <- mapToMammal40(sdf[1:10,])
```

---

matchDesign	<i>normalize Infinium I probe betas to Infinium II</i>
-------------	--

---

**Description**

This is designed to counter tail inflation in Infinium I probes.

**Usage**

```
matchDesign(sdf, min_dbeta = 0.3)
```

**Arguments**

sdf                    SigDF

min\_dbeta            the default algorithm perform 2-state quantile-normalization of the unmethylated and methylated modes separately. However, when the two modes are too close, we fall back to a one-mode normalization. The threshold defines the maximum inter-mode distance.

**Value**

SigDF

**Examples**

```
library(RPMM)
sdf <- sesameDataGet("MM285.1.SigDF")
sesameQC_plotBetaByDesign(sdf)
sesameQC_plotBetaByDesign(matchDesign(sdf))
```

---

meanIntensity	<i>Whole-dataset-wide Mean Intensity</i>
---------------	--

---

### Description

The function takes one single SigDF and computes mean intensity of all the in-band measurements. This includes all Type-I in-band measurements and all Type-II probe measurements. Both methylated and unmethylated alleles are considered. This function outputs a single numeric for the mean.

### Usage

```
meanIntensity(sdf, mask = TRUE)
```

### Arguments

sdf	a SigDF
mask	whether to mask probes using mask column

### Details

Note: mean in this case is more informative than median because methylation level is mostly bimodal.

### Value

mean of all intensities

### Examples

```
sesameDataCache() # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
meanIntensity(sdf)
```

---

medianTotalIntensity	<i>Whole-dataset-wide Median Total Intensity (M+U)</i>
----------------------	--

---

### Description

The function takes one single SigDF and computes median intensity of M+U for each probe. This function outputs a single numeric for the median.

### Usage

```
medianTotalIntensity(sdf, mask = TRUE)
```

**Arguments**

sdf                    a SigDF  
 mask                    whether to mask probes using mask column

**Value**

median of all intensities

**Examples**

```
sesameDataCache() # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
medianTotalIntensity(sdf)
```

---

mLiftOver	<i>Lift over beta values or SigDFs to another Infinium platform This function wraps ID conversion and provide optional imputation functionality.</i>
-----------	--

---

**Description**

Lift over beta values or SigDFs to another Infinium platform This function wraps ID conversion and provide optional imputation functionality.

**Usage**

```
mLiftOver(
  x,
  target_platform,
  source_platform = NULL,
  BPPARAM = SerialParam(),
  mapping = NULL,
  impute = FALSE,
  sd_max = 999,
  celltype = "Blood",
  ...
)
```

**Arguments**

x                    either named beta value (vector or matrix), probe IDs or SigDF(s) if input is a matrix, probe IDs should be in the row names if input is a numeric vector, probe IDs should be in the vector names. If input is a character vector, the input will be considered probe IDs.

target\_platform    the platform to take the data to

source_platform	optional information of the source data platform (when there might be ambiguity).
BPPARAM	use MulticoreParam(n) for parallel processing
mapping	a liftOver mapping file. Typically this file contains empirical evidence whether a probe mapping is reliable. If given, probe ID-based mapping will be skipped. This is to perform more stringent probe ID mapping.
impute	whether to impute or not, default is FALSE
sd_max	the maximum standard deviation for filtering low confidence imputation.
celltype	the cell type / tissue context of imputation, if not given, will use nearest neighbor to find out.
...	extra arguments, see ?convertProbeID

### Value

imputed data, vector, matrix, SigDF(s)

### Examples

```
## Not run:
sesameDataCache()

## lift SigDF

sdf = sesameDataGet("EPICv2.8.SigDF")[["GM12878_206909630042_R08C01"]]
dim(mLiftOver(sdf, "EPICv2"))
dim(mLiftOver(sdf, "EPIC"))
dim(mLiftOver(sdf, "HM450"))

sdfs = sesameDataGet("EPICv2.8.SigDF")[1:2]
sdfs_hm450 = mLiftOver(sdfs, "HM450")
## parallel processing
sdfs_hm450 = mLiftOver(sdfs, "HM450", BPPARAM=BiocParallel::MulticoreParam(2))

sdf = sesameDataGet("EPIC.5.SigDF.normal")[["1"]]
dim(mLiftOver(sdf, "EPICv2"))
dim(mLiftOver(sdf, "EPIC"))
dim(mLiftOver(sdf, "HM450"))

sdf = sesameDataGet("HM450.10.SigDF")[["1"]]
dim(mLiftOver(sdf, "EPICv2"))
dim(mLiftOver(sdf, "EPIC"))
dim(mLiftOver(sdf, "HM450"))

## lift beta values

betas = openSesame(sesameDataGet("EPICv2.8.SigDF")[["1"]])
betas_hm450 = mLiftOver(betas, "HM450", impute=TRUE)
length(betas_hm450)
sum(is.na(betas_hm450))
```



```

betas_hm450 <- mLiftOver(betas, "HM450", impute=FALSE)
length(betas_hm450)
sum(is.na(betas_hm450))
betas_epic1 <- mLiftOver(betas, "EPIC", impute=TRUE)
length(betas_epic1)
sum(is.na(betas_epic1))
betas_epic1 <- mLiftOver(betas, "EPIC", impute=FALSE)
length(betas_epic1)
sum(is.na(betas_epic1))

betas_matrix = openSesame(sesameDataGet("EPICv2.8.SigDF")[1:4])
dim(betas_matrix)
betas_matrix_hm450 = mLiftOver(betas_matrix, "HM450", impute=T)
dim(betas_matrix_hm450)
## parallel processing
betas_matrix_hm450 = mLiftOver(betas_matrix, "HM450", impute=T,
BPPARAM=BiocParallel::MulticoreParam(4))

## use empirical evidence in mLiftOver
mapping = sesameDataGet("liftOver.EPICv2ToEPIC")
betas_matrix = openSesame(sesameDataGet("EPICv2.8.SigDF")[1:4])
dim(mLiftOver(betas_matrix, "EPIC", mapping = mapping))
## compare to without using empirical evidence
dim(mLiftOver(betas_matrix, "EPIC"))

betas <- c("cg04707299"=0.2, "cg13380562"=0.9, "cg00000103"=0.1)
head(mLiftOver(betas, "HM450", impute=TRUE))

betas <- c("cg00004963_TC21"=0, "cg00004963_TC22"=0.5, "cg00004747_TC21"=1.0)
betas_hm450 <- mLiftOver(betas, "HM450", impute=TRUE)
head(na.omit(mLiftOver(betas, "HM450", impute=FALSE)))

## lift probe IDs

cg_epic2 = names(sesameData_getManifestGRanges("EPICv2"))
head(mLiftOver(cg_epic2, "HM450"))

cg_epic2 = grep("cg", names(sesameData_getManifestGRanges("EPICv2")), value=T)
head(mLiftOver(cg_epic2, "HM450"))

cg_hm450 = grep("cg", names(sesameData_getManifestGRanges("HM450")), value=T)
head(mLiftOver(cg_hm450, "EPICv2"))

rs_epic2 = grep("rs", names(sesameData_getManifestGRanges("EPICv2")), value=T)
head(mLiftOver(rs_epic2, "HM450", source_platform="EPICv2"))

probes_epic2 = names(sesameData_getManifestGRanges("EPICv2"))
head(mLiftOver(probes_epic2, "EPIC"))
head(mLiftOver(probes_epic2, "EPIC", target_uniq = TRUE))
head(mLiftOver(probes_epic2, "EPIC", include_new = FALSE))
head(mLiftOver(probes_epic2, "EPIC", include_old = FALSE))
head(mLiftOver(probes_epic2, "EPIC", return_mapping=TRUE))

```

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

---

MValueToBetaValue	<i>Convert M-value to beta-value</i>
-------------------	--------------------------------------

---

**Description**

Convert M-value to beta-value (aka inverse logit transform)

**Usage**

```
MValueToBetaValue(m)
```

**Arguments**

m                    a vector of M values

**Value**

a vector of beta values

**Examples**

```
MValueToBetaValue(c(-3, 0, 3))
```

---

negControls	<i>get negative control signal</i>
-------------	------------------------------------

---

**Description**

get negative control signal

**Usage**

```
negControls(sdf)
```

**Arguments**

sdf                    a SigDF

**Value**

a data frame of negative control signals

---

noMasked	<i>remove masked probes from SigDF</i>
----------	--

---

**Description**

remove masked probes from SigDF

**Usage**

```
noMasked(sdf)
```

**Arguments**

sdf	input SigDF object
-----	--------------------

**Value**

a SigDF object without masked probes

**Examples**

```
sesameDataCache()  
sdf <- sesameDataGet("EPIC.1.SigDF")  
sdf <- p00BAH(sdf)  
  
sdf_noMasked <- noMasked(sdf)
```

---

noob	<i>Noob background subtraction</i>
------	------------------------------------

---

**Description**

The function takes a SigDF and returns a modified SigDF with background subtracted. Background was modelled in a normal distribution and true signal in an exponential distribution. The Norm-Exp deconvolution is parameterized using Out-Of-Band (oob) probes. For species-specific processing, one should call inferSpecies on SigDF first. Multi-mapping probes are excluded.

**Usage**

```
noob(sdf, combine.neg = TRUE, offset = 15)
```

**Arguments**

sdf	a SigDF
combine.neg	whether to combine negative control probe.
offset	offset

**Details**

When `combine.neg = TRUE`, background will be parameterized by both negative control and out-of-band probes.

**Value**

a new SigDF with noob background correction

**Examples**

```
sdf <- sesameDataGet('EPIC.1.SigDF')
sdf.nb <- noob(sdf)
```

---

normControls	<i>get normalization control signal</i>
--------------	---

---

**Description**

get normalization control signal from SigDF. The function optionally takes mean for each channel.

**Usage**

```
normControls(sdf, average = FALSE, verbose = FALSE)
```

**Arguments**

sdf	a SigDF
average	whether to average
verbose	print more messages

**Value**

a data frame of normalization control signals

---

 openSesame

*The openSesame pipeline*


---

## Description

This function is a simple wrapper of noob + nonlinear dye bias correction + pOOBAH masking.

## Usage

```
openSesame(
  x,
  prep = "QCDPB",
  prep_args = NULL,
  manifest = NULL,
  func = getBetas,
  BPPARAM = SerialParam(),
  platform = "",
  min_beads = 1,
  ...
)
```

## Arguments

x	SigDF(s), IDAT prefix(es)
prep	preprocessing code, see ?prepSesame
prep_args	optional preprocessing argument list, see ?prepSesame
manifest	optional dynamic manifest
func	either getBetas or getAFs, if NULL, then return SigDF list
BPPARAM	get parallel with MulticoreParam(n)
platform	optional platform string
min_beads	minimum bead number, probes with R or G smaller than this threshold will be masked. If NULL, no filtering based on bead count will be applied. Default to 1.
...	parameters to getBetas

## Details

Please use mask=FALSE to turn off masking.

If the input is an IDAT prefix or a SigDF, the output is the beta value numerics.

## Value

a numeric vector for processed beta values

### Examples

```
in_dir <- system.file("extdata", "", package = "sesameData")
betas <- openSesame(in_dir)
## or
IDATprefixes <- searchIDATprefixes(in_dir)
betas <- openSesame(IDATprefixes)
```

---

openSesameToFile      *openSesame pipeline with file-backed storage*

---

### Description

openSesame pipeline with file-backed storage

### Usage

```
openSesameToFile(map_path, idat_dir, BPPARAM = SerialParam(), inc = 4)
```

### Arguments

map_path	path of file to be mapped (beta values file)
idat_dir	source IDAT directory
BPPARAM	get parallel with MulticoreParam(2)
inc	bytes per item data storage. increase to 8 if precision is important. Most cases 32-bit representation is enough.

### Value

a sesame::fileSet

### Examples

```
openSesameToFile('mybetas',
  system.file('extdata', package='sesameData'))
```

---

palgen	<i>Generate some additional color palettes</i>
--------	--

---

**Description**

Generate some additional color palettes

**Usage**

```
palgen(pal, n = 150, space = "Lab")
```

**Arguments**

pal	a string for adhoc pals
n	the number of colors for interpolation
space	rgb or Lab

**Value**

a palette-generating function

**Examples**

```
library(pals)
pal.bands(palgen("whiteturbo"))
```

---

parseGEOsignalMU	<i>Convert signal M and U to SigDF</i>
------------------	--

---

**Description**

This overcomes the issue of missing IDAT files. However, out-of-band signals will be missing or faked (sampled from a normal distribution).

**Usage**

```
parseGEOsignalMU(
  sigM,
  sigU,
  Probe_IDs,
  oob.mean = 500,
  oob.sd = 300,
  platform = NULL
)
```

**Arguments**

sigM	methylated signal, a numeric vector
sigU	unmethylated signal, a numeric vector
Probe_IDs	probe ID vector
oob.mean	assumed mean for out-of-band signals
oob.sd	assumed standard deviation for out-of-band signals
platform	platform code, will infer if not given

**Value**

SigDF

**Examples**

```
sigM <- c(11436, 6068, 2864)
sigU <- c(1476, 804, 393)
probes <- c("cg07881041", "cg23229610", "cg03513874")
sdf <- parseGEOsignalMU(sigM, sigU, probes, platform = "EPIC")
```

---

pOOBAH

*Detection P-value based on ECDF of out-of-band signal*

---

**Description**

aka pOOBAH (p-val by Out-Of-Band Array Hybridization)

**Usage**

```
pOOBAH(
  sdf,
  return.pval = FALSE,
  combine.neg = TRUE,
  pval.threshold = 0.05,
  verbose = FALSE
)
```

**Arguments**

sdf	a SigDF
return.pval	whether to return p-values, instead of a masked SigDF
combine.neg	whether to combine negative control probes with the out-of-band probes in simulating the signal background
pval.threshold	minimum p-value to mask
verbose	print more messages



**Details**

The function takes a SigDF as input, computes detection p-value using out-of-band probes empirical distribution and returns a new SigDF with an updated mask slot.

**Value**

a SigDF, or a p-value vector if return.pval is TRUE

**Examples**

```
sdf <- sesameDataGet("EPIC.1.SigDF")
sum(sdf$mask)
sum(pOOBAH(sdf)$mask)
```

---

predictAge	<i>Predict age using linear models</i>
------------	--

---

**Description**

The function takes a named numeric vector of beta values. The name attribute contains the probe ID (cg, ch or rs IDs). The function looks for overlapping probes and estimate age using different models.

**Usage**

```
predictAge(betas, model, na_fallback = FALSE, min_nonna = 10)
```

**Arguments**

betas	a probeID-named vector of beta values
model	a model object from sesameDataGet. should contain param, intercept, response2age. default to the Horvath353 model.
na_fallback	use fall back values if na
min_nonna	the minimum number of non-NA values.

**Details**

You can get the models such as the Horvath aging model (Horvath 2013 Genome Biology) from sesameDataGet. The function outputs a single numeric of age in years.

Here are some built-in age models: Anno/HM450/Clock\_Horvath353.rds Anno/HM450/Clock\_Hannum.rds Anno/HM450/Clock\_SkinBlood.rds Anno/EPIC/Clock\_PhenoAge.rds Anno/MM285/Clock\_Zhou347.rds see vignette inferences.html#Age\_\_Epigenetic\_Clock for details

**Value**

age in the unit specified in the model (usually in year, but sometimes can be month, like in the mouse clocks).

### Examples

```
betas <- sesameDataGet('HM450.1.TCGA.PAAD')$betas
## Not run:
## download age models from
## https://github.com/zhou-lab/InfiniumAnnotationV1/tree/main/Anno
## e.g., Anno/HM450/Clock_Horvath353.rds
predictAge(betas, model)

## End(Not run)
```

---

predictAgeHorvath353 *Horvath 353 age predictor*

---

### Description

The function takes a named numeric vector of beta values. The name attribute contains the probe ID (cg, ch or rs IDs). The function looks for overlapping probes and estimate age using Horvath aging model (Horvath 2013 Genome Biology). The function outputs a single numeric of age in years.

### Usage

```
predictAgeHorvath353(betas)
```

### Arguments

betas            a probeID-named vector of beta values

### Value

age in years

### Examples

```
cat("Deprecated. See predictAge")
```

---

predictAgeSkinBlood *Horvath Skin and Blood age predictor*

---

### Description

The function takes a named numeric vector of beta values. The name attribute contains the probe ID (cg, ch or rs IDs). The function looks for overlapping probes and estimate age using Horvath aging model (Horvath et al. 2018 Aging, 391 probes). The function outputs a single numeric of age in years.

**Usage**

```
predictAgeSkinBlood(betas)
```

**Arguments**

betas                    a probeID-named vector of beta values

**Value**

age in years

**Examples**

```
cat("Deprecated. See predictAge")
```

---

predictMouseAgeInMonth  
*Mouse age predictor*

---

**Description**

The function takes a named numeric vector of beta values. The name attribute contains the probe ID. The function looks for overlapping probes and estimate age using an aging model built from 321 MM285 probes. The function outputs a single numeric of age in months. The clock is most accurate with the sesame preprocessing.

**Usage**

```
predictMouseAgeInMonth(betas, na_fallback = TRUE)
```

**Arguments**

betas                    a probeID-named vector of beta values  
na\_fallback              use the fallback default for NAs.

**Value**

age in month

**Examples**

```
cat("Deprecated. See predictAge")
```

---

prefixMask                      *Mask SigDF by probe ID prefix*

---

**Description**

Mask SigDF by probe ID prefix

**Usage**

```
prefixMask(sdf, prefixes = NULL, invert = FALSE)
```

**Arguments**

sdf	SigDF
prefixes	prefix characters
invert	use the complement set

**Value**

SigDF

**Examples**

```
sdf <- resetMask(sesameDataGet("MM285.1.SigDF"))
sum(prefixMask(sdf, c("ctl", "rs"))$mask)
sum(prefixMask(sdf, c("ctl"))$mask)
sum(prefixMask(sdf, c("ctl", "rs", "ch"))$mask)
```

---

prefixMaskButC                      *Mask all but C probes in SigDF*

---

**Description**

Mask all but C probes in SigDF

**Usage**

```
prefixMaskButC(sdf)
```

**Arguments**

sdf	SigDF
-----	-------

**Value**

SigDF

**Examples**

```
sdf <- resetMask(sesameDataGet("MM285.1.SigDF"))
sum(prefixMaskButC(sdf)$mask)
```

---

prefixMaskButCG	<i>Mask all but CG probes in SigDF</i>
-----------------	--

---

**Description**

Mask all but CG probes in SigDF

**Usage**

```
prefixMaskButCG(sdf)
```

**Arguments**

sdf	SigDF
-----	-------

**Value**

SigDF

**Examples**

```
sdf <- resetMask(sesameDataGet("MM285.1.SigDF"))
sum(prefixMaskButCG(sdf)$mask)
```

---

prepSesame	<i>Apply a chain of sesame preprocessing functions in an arbitrary order</i>
------------	--

---

**Description**

Notes on the order of operation: 1. qualityMask and inferSpecies should go before noob and pOOBAH, otherwise the background is too high because of Multi, uk and other probes 2. dyeBias correction needs to happen early 3. channel inference before dyebias 4. noob should happen last, pOOBAH before noob because noob modifies oob

**Usage**

```
prepSesame(sdf, prep = "QCDPB", prep_args = NULL)
```

**Arguments**

sdf	SigDF
prep	code that indicates preprocessing functions and their execution order (functions on the left is executed first).
prep_args	optional argument list to individual functions, e.g., prepSesame(sdf, prep_args=list(Q=list(mask_names = "design_issue"))) sets qualityMask(sdf, mask_names = "design_issue")

**Value**

SigDF

**Examples**

```
sdf <- sesameDataGet("MM285.1.SigDF")  
sdf1 <- prepSesame(sdf, "QCDPB")
```

---

prepSesameList	<i>List supported prepSesame functions</i>
----------------	--

---

**Description**

List supported prepSesame functions

**Usage**

```
prepSesameList()
```

**Value**

a data frame with code, func, description

**Examples**

```
prepSesameList()
```

---

print.DMLSummary      *Print DMLSummary object*

---

**Description**

Print DMLSummary object

**Usage**

```
## S3 method for class 'DMLSummary'  
print(x, ...)
```

**Arguments**

x                    a DMLSummary object  
...                  extra parameter for print

**Value**

print DMLSummary result on screen

**Examples**

```
sesameDataCache() # in case not done yet  
data <- sesameDataGet('HM450.76.TCGA.matched')  
## test the first 10  
smry <- DML(data$betas[1:10,], ~type, meta=data$sampleInfo)  
smry  
  
sesameDataGet_resetEnv()
```

---

print.fileSet      *Print a fileSet*

---

**Description**

Print a fileSet

**Usage**

```
## S3 method for class 'fileSet'  
print(x, ...)
```

**Arguments**

x                    a sesame::fileSet  
...                  stuff for print

**Value**

string representation

**Examples**

```
fset <- initFileSet('mybetas2', 'HM27', c('s1','s2'))
fset
```

---

probeID_designType	<i>Extract the probe type field from probe ID This only works with the new probe ID system. See <a href="https://github.com/zhou-lab/InfiniumAnnotation">https://github.com/zhou-lab/InfiniumAnnotation</a> for illustration</i>
--------------------	--

---

**Description**

Extract the probe type field from probe ID This only works with the new probe ID system. See <https://github.com/zhou-lab/InfiniumAnnotation> for illustration

**Usage**

```
probeID_designType(Probe_ID)
```

**Arguments**

Probe_ID	Probe ID
----------	----------

**Value**

a vector of '1' and '2' suggesting Infinium-I and Infinium-II

**Examples**

```
probeID_designType("cg36609548_TC21")
```



---

probeSuccessRate	<i>Whole-dataset-wide Probe Success Rate</i>
------------------	--

---

**Description**

This function calculates the probe success rate using pOOBAH detection p-values. Probes that has a detection p-value higher than a specific threshold are considered failed probes.

**Usage**

```
probeSuccessRate(sdf, mask = TRUE, max_pval = 0.05)
```

**Arguments**

sdf	a SigDF
mask	whether or not we count the masked probes in SigDF
max_pval	the maximum p-value to consider detection success

**Value**

a fraction number as probe success rate

**Examples**

```
sesameDataCache() # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
probeSuccessRate(sdf)
```

---

qualityMask	<i>Mask beta values by design quality</i>
-------------	---

---

**Description**

Currently quality masking only supports three platforms see also listAvailableMasks(sdfPlatform(sdf))

**Usage**

```
qualityMask(sdf, mask_names = "recommended", verbose = TRUE)
```

**Arguments**

sdf	a SigDF object
mask_names	a vector of masking groups, see listAvailableMasks use "recommended" for recommended masking. One can also combine "recommended" with other masking groups by specifying a vector, e.g., c("recommended", "M_mapping")
verbose	be verbose

**Value**

a filtered SigDF

**Examples**

```
sesameDataCache() # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
sum(sdf$mask)
sum(qualityMask(sdf)$mask)
sum(qualityMask(sdf, mask_names = NULL)$mask)

## list available masks, the dbname column
listAvailableMasks(sdfPlatform(sdf))
listAvailableMasks("EPICv2")
```

---

readFileSet	<i>Read an existing fileSet from storage</i>
-------------	--

---

**Description**

This function only reads the meta-data.

**Usage**

```
readFileSet(map_path)
```

**Arguments**

map\_path            path of file to map (should contain valid \_idx.rds index)

**Value**

a sesame::fileSet object

**Examples**

```
## create two samples
fset <- initFileSet('mybetas2', 'HM27', c('s1','s2'))

## a hypothetical numeric array (can be beta values, intensities etc)
hypothetical <- setNames(runif(fset$n), fset$probes)

## map the numeric to file
mapFileSet(fset, 's1', hypothetical)

## read it from file
fset <- readFileSet('mybetas2')
```

```
## get data
sliceFileSet(fset, 's1', 'cg00000292')
```

---

readIDATpair	<i>Import a pair of IDATs from one sample</i>
--------------	---

---

### Description

The function takes a prefix string that are shared with `_Grn.idat` and `_Red.idat`. The function returns a `SigDF`.

### Usage

```
readIDATpair(  
  prefix.path,  
  manifest = NULL,  
  platform = "",  
  min_beads = NULL,  
  controls = NULL,  
  verbose = FALSE  
)
```

### Arguments

<code>prefix.path</code>	sample prefix without <code>_Grn.idat</code> and <code>_Red.idat</code>
<code>manifest</code>	optional design manifest file
<code>platform</code>	EPIC, HM450 and HM27 etc.
<code>min_beads</code>	minimum bead number, probes with R or G smaller than this threshold will be masked. If <code>NULL</code> , no filtering based on bead count will be applied.
<code>controls</code>	optional control probe manifest file
<code>verbose</code>	be verbose? ( <code>FALSE</code> )

### Value

a `SigDF`

### Examples

```
sdf <- readIDATpair(sub('_Grn.idat', '', system.file(  
  "extdata", "4207113116_A_Grn.idat", package = "sesameData")))
```

---

recommendedMaskNames    *Recommended mask names for each Infinium platform*

---

### Description

The returned name is the db name used in KYCG.mask

### Usage

```
recommendedMaskNames()
```

### Value

a named list of mask names

### Examples

```
recommendedMaskNames()[["EPICv2"]]
recommendedMaskNames()[["EPIC"]]
```

---

reIdentify                    *Re-identify IDATs by restoring scrambled SNP intensities*

---

### Description

This requires setting a seed with a secret number that was used to de-identify the IDAT (see example). This requires a secret number that was used to de-identify the IDAT

### Usage

```
reIdentify(path, out_path = NULL, snps = NULL, mft = NULL)
```

### Arguments

path	input IDAT file
out_path	output IDAT file
snps	SNP definition, if not given, default to SNP probes
mft	sesame-compatible manifest if non-standard

### Value

NULL, changes made to the IDAT files

**Examples**

```
temp_out <- tempfile("test")

set.seed(123)
reIdentify(system.file(
  "extdata", "4207113116_A_Grn.idat", package = "sesameData"), temp_out)
unlink(temp_out)
```

---

resetMask	<i>Reset Masking</i>
-----------	----------------------

---

**Description**

Reset Masking

**Usage**

```
resetMask(sdf, verbose = FALSE)
```

**Arguments**

sdf	a SigDF
verbose	print more messages

**Value**

a new SigDF with mask reset to all FALSE

**Examples**

```
sesameDataCache() # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
sum(sdf$mask)
sdf <- addMask(sdf, c("cg14057072", "cg22344912"))
sum(sdf$mask)
sum(resetMask(sdf)$mask)
```

scrub                      *SCRUB background correction*

---

### **Description**

This function takes a SigDF and returns a modified SigDF with background subtracted. scrub subtracts residual background using background median

### **Usage**

```
scrub(sdf)
```

### **Arguments**

sdf                      a SigDF

### **Details**

This function is meant to be used after noob.

### **Value**

a new SigDF with noob background correction

### **Examples**

```
sdf <- sesameDataGet('EPIC.1.SigDF')
sdf.nb <- noob(sdf)
sdf.nb.scrub <- scrub(sdf.nb)
```

---

scrubSoft                      *SCRUB background correction*

---

### **Description**

This function takes a SigDF and returns a modified SigDF with background subtracted. scrubSoft subtracts residual background using a noob-like procedure.

### **Usage**

```
scrubSoft(sdf)
```

### **Arguments**

sdf                      a SigDF

**Details**

This function is meant to be used after noob.

**Value**

a new SigDF with noob background correction

**Examples**

```
sdf <- sesameDataGet('EPIC.1.SigDF')
sdf.nb <- noob(sdf)
sdf.nb.scrubSoft <- scrubSoft(sdf.nb)
```

---

SDFcollapseToPfx	<i>collapse to probe prefix</i>
------------------	---------------------------------

---

**Description**

collapse to probe prefix

**Usage**

```
SDFcollapseToPfx(sdf)
```

**Arguments**

sdf	a SigDF object
-----	----------------

**Value**

a data frame with updated Probe\_ID

---

sdfPlatform	<i>Convenience function to output platform attribute of SigDF</i>
-------------	---

---

**Description**

Convenience function to output platform attribute of SigDF

**Usage**

```
sdfPlatform(sdf, verbose = FALSE)
```

**Arguments**

sdf	a SigDF object
verbose	print more messages

**Value**

the platform string for the SigDF object

**Examples**

```
sesameDataCache()
sdf <- sesameDataGet('EPIC.1.SigDF')
sdfPlatform(sdf)
```

---

sdf_read_table	<i>read a table file to SigDF</i>
----------------	-----------------------------------

---

**Description**

read a table file to SigDF

**Usage**

```
sdf_read_table(fname, platform = NULL, verbose = FALSE, ...)
```

**Arguments**

fname	file name
platform	array platform (will infer if not given)
verbose	print more information
...	additional argument to read.table

**Value**

read table file to SigDF

**Examples**

```
sesameDataCache() # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
fname <- sprintf("%s/sigdf.txt", tempdir())
sdf_write_table(sdf, file=fname)
sdf2 <- sdf_read_table(fname)
```



---

sdf_write_table	<i>write SigDF to table file</i>
-----------------	----------------------------------

---

**Description**

write SigDF to table file

**Usage**

```
sdf_write_table(sdf, ...)
```

**Arguments**

sdf	the SigDF to output
...	additional argument to write.table

**Value**

write SigDF to table file

**Examples**

```
sesameDataCache() # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
sdf_write_table(sdf, file=sprintf("%s/sigdf.txt", tempdir()))
```

---

searchIDATprefixes	<i>Identify IDATs from a directory</i>
--------------------	--

---

**Description**

The input is the directory name as a string. The function identifies all the IDAT files under the directory. The function returns a vector of such IDAT prefixes under the directory.

**Usage**

```
searchIDATprefixes(dir.name, recursive = TRUE, use.basename = TRUE)
```

**Arguments**

dir.name	the directory containing the IDAT files.
recursive	search IDAT files recursively
use.basename	basename of each IDAT path is used as sample name This won't work in rare situation where there are duplicate IDAT files.

**Value**

the IDAT prefixes (a vector of character strings).

**Examples**

```
## only search what are directly under
IDATprefixes <- searchIDATprefixes(
  system.file("extdata", "", package = "sesameData"))

## search files recursively is by default
IDATprefixes <- searchIDATprefixes(
  system.file(package = "sesameData"), recursive=TRUE)
```

---

segmentBins	<i>Segment bins using DNACopy</i>
-------------	-----------------------------------

---

**Description**

Segment bins using DNACopy

**Usage**

```
segmentBins(bin.signals, bin.coords)
```

**Arguments**

bin.signals	bin signals (input)
bin.coords	bin coordinates

**Value**

segment signal data frame

---

sesameAnno_attachManifest	<i>Annotate a data.frame using manifest</i>
---------------------------	---

---

**Description**

Annotate a data.frame using manifest

**Usage**

```
sesameAnno_attachManifest(
  df,
  probe_id = "Probe_ID",
  platform = NULL,
  genome = NULL
)
```

**Arguments**

df	input data frame with Probe_ID as a column
probe_id	the Probe_ID column name, default to "Probe_ID" or rownames
platform	which array platform, guess from probe ID if not given
genome	the genome build, use default if not given

**Value**

a new data.frame with manifest attached

**Examples**

```
## Not run:
df <- data.frame(Probe_ID = c("cg00101675_BC21", "cg00116289_BC21"))
sesameAnno_attachManifest(df)

## End(Not run)
```

---

```
sesameAnno_buildAddressFile
```

*Build sesame ordering address file from tsv*

---

**Description**

Build sesame ordering address file from tsv

**Usage**

```
sesameAnno_buildAddressFile(tsv)
```

**Arguments**

tsv	a platform name, a file path or a tibble/data.frame manifest file
-----	---

**Value**

a list of ordering and controls

## Examples

```
## Not run:
tsv = sesameAnno_download("HM450.hg38.manifest.tsv.gz")
addr <- sesameAnno_buildAddressFile(tsv)

## End(Not run)
```

---

sesameAnno\_buildManifestGRanges  
*Build manifest GRanges from tsv*

---

## Description

manifest tsv files can be downloaded from <http://zwdzwd.github.io/InfiniumAnnotation>

## Usage

```
sesameAnno_buildManifestGRanges(  
  tsv,  
  genome = NULL,  
  decoy = FALSE,  
  columns = NULL  
)
```

## Arguments

tsv	a file path, a platform (e.g., EPIC), or a tibble/data.frame object
genome	a genome string, e.g., hg38, mm10
decoy	consider decoy sequence in chromosome order
columns	the columns to include in the GRanges

## Value

GRanges

## Examples

```
## Not run:
tsv = sesameAnno_download("HM450.hg38.manifest.tsv.gz")
gr <- sesameAnno_buildManifestGRanges(tsv)
## direct access
gr <- sesameAnno_buildManifestGRanges("HM450.hg38.manifest")

## End(Not run)
```

---

sesameAnno\_download     *Download SeSAmE annotation files*

---

## Description

see also <http://zwdzwd.github.io/InfiniumAnnotation>

## Usage

```
sesameAnno_download(url, destfile = tempfile(basename(url)))
```

## Arguments

url	url or title of the annotation file
destfile	download to this file, a temp file if unspecified

## Details

This function acts similarly as `sesameAnno_get` except that it directly download files without invoking `BiocFileCache`. This is needed in some situation because `BiocFileCache` may change the file name and downstream program may depend on the correct file names. It also lets you download files in a cleaner way without routing through `BiocFileCache`

## Value

the path to downloaded file

## Examples

```
## Not run:  
## avoid testing as this function uses external host  
sesameAnno_download("Test/3999492009_R01C01_Grn.idat")  
sesameAnno_download("EPIC.hg38.manifest.tsv.gz")  
sesameAnno_download("EPIC.hg38.snp.tsv.gz")  
  
## End(Not run)
```

sesameAnno\_readManifestTSV  
*Read manifest file to a tsv format*

---

**Description**

Read manifest file to a tsv format

**Usage**

```
sesameAnno_readManifestTSV(tsv_fn)
```

**Arguments**

tsv\_fn            tsv file path

**Value**

a manifest as a tibble

**Examples**

```
## Not run:  
tsv = sesameAnno_download("HM450.hg38.manifest.tsv.gz")  
mft <- sesameAnno_readManifestTSV(tsv)  
## direct access  
mft <- sesameAnno_readManifestTSV("HM450.hg38.manifest")  
  
## End(Not run)
```

---

sesameQC-class            *An S4 class to hold QC statistics*

---

**Description**

An S4 class to hold QC statistics

**Value**

sesameQC object

**Slots**

stat a list to store qc stats

---

sesameQCtoDF	<i>Convert a list of sesameQC to data frame</i>
--------------	---

---

**Description**

Convert a list of sesameQC to data frame

**Usage**

```
sesameQCtoDF(qcs, cols = c("frac_dt_cg", "RGdistort", "RGratio"))
```

**Arguments**

qcs	sesameQCs
cols	QC columns, use NULL to report all

**Value**

a data frame

**Examples**

```
sdf <- sesameDataGet("EPIC.1.SigDF")
qcs <- sesameQC_calcStats(sdf, "detection")
sesameQCtoDF(qcs)
```

---

sesameQC_calcStats	<i>Calculate QC statistics</i>
--------------------	--------------------------------

---

**Description**

It is a function to call one or multiple sesameQC\_calcStats functions

**Usage**

```
sesameQC_calcStats(sdf, funs = NULL)
```

**Arguments**

sdf	a SigDF object
funs	a sesameQC_calcStats_* function or a list of them default to all functions. One can also use a string such as "detection" or c("detection", "intensity") to reduce typing

**Details**

currently supporting: detection, intensity, numProbes, channel, dyeBias, betas

**Value**

a sesameQC object

**Examples**

```
sesameDataCache() # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
sesameQC_calcStats(sdf)
sesameQC_calcStats(sdf, "detection")
sesameQC_calcStats(sdf, c("detection", "channel"))
## retrieve stats as a list
sesameQC_getStats(sesameQC_calcStats(sdf, "detection"))
## or as data frames
as.data.frame(sesameQC_calcStats(sdf, "detection"))
```

---

sesameQC\_getStats      *Get stat numbers from an sesameQC object*

---

**Description**

Get stat numbers from an sesameQC object

**Usage**

```
sesameQC_getStats(qc, stat_names = NULL, drop = TRUE)
```

**Arguments**

qc	a sesameQC object
stat_names	which stat(s) to retrieve, default to all.
drop	whether to drop to a string when stats_names has only one element.

**Value**

a list of named stats to be retrieved

**Examples**

```
sdf <- sesameDataGet("EPIC.1.SigDF")
qc <- sesameQC_calcStats(sdf, "detection")
sesameQC_getStats(qc, "frac_dt")
```



---

sesameQC\_plotBar      *Bar plots for sesameQC*

---

**Description**

By default, it plots median\_beta\_cg, median\_beta\_ch, RGratio, RGdistort, frac\_dt

**Usage**

```
sesameQC_plotBar(qcs, keys = NULL)
```

**Arguments**

qcs                    a list of SigDFs  
keys                   optional, other key to plot, instead of the default keys can be found in the parenthesis of the print output of each sesameQC output.

**Value**

a bar plot comparing different QC metrics

**Examples**

```
sesameDataCache() # if not done yet  
sdfs <- sesameDataGet("EPIC.5.SigDF.normal")[1:2]  
sesameQC_plotBar(lapply(sdfs, sesameQC_calcStats, "detection"))
```

---

sesameQC\_plotBetaByDesign  
*Plot betas distinguishing different Infinium chemistries*

---

**Description**

Plot betas distinguishing different Infinium chemistries

**Usage**

```
sesameQC_plotBetaByDesign(  
  sdf,  
  prep = NULL,  
  legend_pos = "top",  
  mar = c(3, 3, 1, 1),  
  main = "",  
  ...  
)
```

**Arguments**

sdf	SigDF
prep	prep codes to step through
legend_pos	legend position (default: top)
mar	margin of layout when showing steps of prep
main	main title in plots
...	additional options to plot

**Value**

create a density plot

**Examples**

```
sdf <- sesameDataGet("EPIC.1.SigDF")
sesameQC_plotBetaByDesign(sdf, prep="DB")
```

---

sesameQC\_plotHeatSNPs *Plot SNP heatmap*

---

**Description**

Plot SNP heatmap

**Usage**

```
sesameQC_plotHeatSNPs(sdfs, cluster = TRUE, filter.nonvariant = TRUE)
```

**Arguments**

sdfs	beta value matrix, row: probes; column: samples
cluster	show clustered heatmap
filter.nonvariant	whether to filter nonvariant (range < 0.3)

**Value**

a grid graphics object

**Examples**

```
sdfs <- sesameDataGet("EPIC.5.SigDF.normal")[1:2]
plt <- sesameQC_plotHeatSNPs(sdfs, filter.nonvariant = FALSE)
```

---

`sesameQC_plotIntensVsBetas`

*Plot Total Signal Intensities vs Beta Values This plot is helpful in revealing the extent of signal background and dye bias.*

---

## Description

Plot Total Signal Intensities vs Beta Values This plot is helpful in revealing the extent of signal background and dye bias.

## Usage

```
sesameQC_plotIntensVsBetas(  
  sdf,  
  mask = TRUE,  
  use_max = FALSE,  
  intens.range = c(5, 15),  
  pal = "whiteturbo",  
  ...  
)
```

## Arguments

<code>sdf</code>	a SigDF
<code>mask</code>	whether to remove probes that are masked
<code>use_max</code>	to use max(M,U) or M+U
<code>intens.range</code>	plot range of signal intensity
<code>pal</code>	color palette, whiteturbo, whiteblack, whitejet
<code>...</code>	additional arguments to smoothScatter

## Value

create a total signal intensity vs beta value plot

## Examples

```
sesameDataCache() # if not done yet  
sdf <- sesameDataGet('EPIC.1.SigDF')  
sesameQC_plotIntensVsBetas(sdf)
```

---

sesameQC\_plotRedGrnQQ *Plot red-green QQ-Plot using Infinium-I Probes*

---

### Description

Plot red-green QQ-Plot using Infinium-I Probes

### Usage

```
sesameQC_plotRedGrnQQ(sdf, main = "R-G QQ Plot", ...)
```

### Arguments

sdf	a SigDF
main	plot title
...	additional options to qqplot

### Value

create a qqplot

### Examples

```
sesameDataCache() # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
sesameQC_plotRedGrnQQ(sdf)
```

---

sesameQC\_rankStats *This function compares the input sample with public data. Only overlapping metrics will be compared.*

---

### Description

This function compares the input sample with public data. Only overlapping metrics will be compared.

### Usage

```
sesameQC_rankStats(qc, publicQC = NULL, platform = "EPIC")
```

### Arguments

qc	a sesameQC object
publicQC	public QC statistics, filtered from e.g.: EPIC.publicQC, MM285.publicQC and Mammal40.publicQC
platform	EPIC, MM285 or Mammal40, used when publicQC is not given

**Value**

a sesameQC

**Examples**

```
sesameDataCache() # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
sesameQC_rankStats(sesameQC_calcStats(sdf, "intensity"))
```

---

sesame\_checkVersion     *Check SeSAmE versions*

---

**Description**

print package version of sesame and depended packages to help troubleshoot installation issues.

**Usage**

```
sesame_checkVersion()
```

**Value**

print the version of sesame, sesameData, bioconductor and R

**Examples**

```
sesame_checkVersion()
```

---

sesamize	<i>sesamize function is deprecated. Please check <a href="https://github.com/zwdzwd/sesamize">https://github.com/zwdzwd/sesamize</a> for previous scripts</i>
----------	---

---

**Description**

sesamize function is deprecated. Please check <https://github.com/zwdzwd/sesamize> for previous scripts

**Usage**

```
sesamize(...)
```

**Arguments**

...                    arguments for sesamize

**Value**

a message text for deprecated function

**Examples**

```
cat("Deprecated. see https://github.com/zwdzwd/sesamize")
```

---

setMask	<i>Set mask to only the probes specified</i>
---------	--

---

**Description**

Set mask to only the probes specified

**Usage**

```
setMask(sdf, probes)
```

**Arguments**

sdf	a SigDF
probes	a vector of probe IDs or a logical vector with TRUE representing masked probes

**Value**

a SigDF with added mask

**Examples**

```
sdf <- sesameDataGet('EPIC.1.SigDF')
sum(sdf$mask)
sum(setMask(sdf, "cg14959801")$mask)
sum(setMask(sdf, c("cg14057072", "cg22344912"))$mask)
```

---

SigDF	<i>SigDF validation from a plain data frame</i>
-------	---

---

**Description**

SigDF validation from a plain data frame

**Usage**

```
SigDF(df, platform = "EPIC", ctl = NULL)
```

**Arguments**

df a data.frame with Probe\_ID, MG, MR, UG, UR, col and mask  
platform a string to specify the array platform  
ctl optional control probe data frame

**Value**

a SigDF object

**Examples**

```
sesameDataCache() # if not done yet  
sdf <- sesameDataGet('EPIC.1.SigDF')
```

---

signalMU	<i>report M and U for regular probes</i>
----------	--

---

**Description**

report M and U for regular probes

**Usage**

```
signalMU(sdf, mask = TRUE, MU = FALSE)
```

**Arguments**

sdf a SigDF  
mask whether to apply mask  
MU add a column for M+U

**Value**

a data frame of M and U columns

**Examples**

```
sesameDataCache() # if not done yet  
sdf <- sesameDataGet('EPIC.1.SigDF')  
head(signalMU(sdf))
```

---

sliceFileSet	<i>Slice a fileSet with samples and probes</i>
--------------	--

---

### Description

Slice a fileSet with samples and probes

### Usage

```
sliceFileSet(fset, samples = fset$samples, probes = fset$probes, memmax = 10^5)
```

### Arguments

fset	a sesame::fileSet, as obtained via readFileSet
samples	samples to query (default to all samples)
probes	probes to query (default to all probes)
memmax	maximum items to read from file to memory, to protect from accidental memory congestion.

### Value

a numeric matrix of length(samples) columns and length(probes) rows

### Examples

```
## create two samples
fset <- initFileSet('mybetas2', 'HM27', c('s1','s2'))

## a hypothetical numeric array (can be beta values, intensities etc)
hypothetical <- setNames(runif(fset$n), fset$probes)

## map the numeric to file
mapFileSet(fset, 's1', hypothetical)

## get data
sliceFileSet(fset, 's1', 'cg00000292')
```



---

summaryExtractTest	<i>Extract slope information from DMLSummary</i>
--------------------	--

---

**Description**

Extract slope information from DMLSummary

**Usage**

```
summaryExtractTest(smry)
```

**Arguments**

smry	DMLSummary from DML command
------	-----------------------------

**Value**

a table of slope and p-value

**Examples**

```
sesameDataCache() # in case not done yet
data <- sesameDataGet('HM450.76.TCGA.matched')
smry <- DML(data$betas[1:10,], ~type, meta=data$sampleInfo)
slopes <- summaryExtractTest(smry)

sesameDataGet_resetEnv()
```

---

totalIntensities	<i>M+U Intensities Array</i>
------------------	------------------------------

---

**Description**

The function takes one single SigDF and computes total intensity of all the in-band measurements by summing methylated and unmethylated alleles. This function outputs a single numeric for the mean.

**Usage**

```
totalIntensities(sdf, mask = FALSE)
```

**Arguments**

sdf	a SigDF
mask	whether to mask probes using mask column

**Value**

a vector of M+U signal for each probe

**Examples**

```
sesameDataCache() # if not done yet
sdf <- sesameDataGet('EPIC.1.SigDF')
intensities <- totalIntensities(sdf)
```

---

twoCompsEst2

*Estimate the fraction of the 2nd component in a 2-component mixture*

---

**Description**

Estimate the fraction of the 2nd component in a 2-component mixture

**Usage**

```
twoCompsEst2(
  pop1,
  pop2,
  target,
  use.ave = TRUE,
  diff_1m2u = NULL,
  diff_1u2m = NULL
)
```

**Arguments**

pop1	Reference methylation level matrix for population 1
pop2	Reference methylation level matrix for population 2
target	Target methylation level matrix to be analyzed
use.ave	use population average in selecting differentially methylated probes
diff_1m2u	A vector of differentially methylated probes (methylated in population 1 but unmethylated in population 2)
diff_1u2m	A vector of differentially methylated probes (unmethylated in population 1 but methylated in population 2)

**Value**

Estimate of the 2nd component in the 2-component mixture

---

updateSigDF	<i>Set color and mask using strain/species-specific manifest</i>
-------------	--

---

**Description**

also sets attr("species")

**Usage**

```
updateSigDF(sdf, species = NULL, strain = NULL, addr = NULL, verbose = FALSE)
```

**Arguments**

sdf	a SigDF
species	the species the sample is considered to be
strain	the strain the sample is considered to be
addr	species-specific address species, optional
verbose	print more messages

**Value**

a SigDF with updated color channel and mask

**Examples**

```
sdf <- sesameDataGet('Mammal40.1.SigDF')  
sdf_mouse <- updateSigDF(sdf, species="mus_musculus")
```

---

visualizeGene	<i>Visualize Gene</i>
---------------	-----------------------

---

**Description**

Visualize the beta value in heatmaps for a given gene. The function takes a gene name which is taken from the UCSC refGene. It searches all the transcripts for the given gene and optionally extend the span by certain number of base pairs. The function also takes a beta value matrix with sample names on the columns and probe names on the rows. The function can also work on different genome builds (default to hg38, can be hg19).

**Usage**

```
visualizeGene(
  gene_name,
  betas,
  platform = NULL,
  genome = NULL,
  upstream = 2000,
  dwstream = 2000,
  ...
)
```

**Arguments**

gene_name	gene name
betas	beta value matrix (row: probes, column: samples)
platform	HM450, EPIC, or MM285 (default)
genome	hg19, hg38, or mm10 (default)
upstream	distance to extend upstream
dwstream	distance to extend downstream
...	additional options, see visualizeRegion, assemble_plots

**Value**

None

**Examples**

```
betas <- sesameDataGet('HM450.76.TCGA.matched')$betas
visualizeGene('ADA', betas, 'HM450')
```

---

visualizeProbes

*Visualize Region that Contains the Specified Probes*


---

**Description**

Visualize the beta value in heatmaps for the genomic region containing specified probes. The function works only if specified probes can be spanned by a single genomic region. The region can cover more probes than specified. Hence the plotting heatmap may encompass more probes. The function takes as input a string vector of probe IDs (cg/ch/rs-numbers). if draw is FALSE, the function returns the subset beta value matrix otherwise it returns the grid graphics object.

**Usage**

```
visualizeProbes(  
  probeNames,  
  betas,  
  platform = NULL,  
  genome = NULL,  
  upstream = 1000,  
  dwnstream = 1000,  
  ...  
)
```

**Arguments**

probeNames	probe names
betas	beta value matrix (row: probes, column: samples)
platform	HM450, EPIC or MM285 (default)
genome	hg19, hg38 or mm10 (default)
upstream	distance to extend upstream
dwnstream	distance to extend downstream
...	additional options, see visualizeRegion and assemble_plots

**Value**

None

**Examples**

```
betas <- sesameDataGet('HM450.76.TCGA.matched')$betas  
visualizeProbes(c('cg22316575', 'cg16084772', 'cg20622019'), betas, 'HM450')
```

---

visualizeRegion

*Visualize Region*

---

**Description**

The function takes a genomic coordinate (chromosome, start and end) and a beta value matrix (probes on the row and samples on the column). It plots the beta values as a heatmap for all probes falling into the genomic region. If 'draw=TRUE' the function returns the plotted grid graphics object. Otherwise, the selected beta value matrix is returned. 'cluster.samples=TRUE/FALSE' controls whether hierarchical clustering is applied to the subset beta value matrix.

**Usage**

```
visualizeRegion(
  chr,
  beg,
  end,
  betas,
  platform = NULL,
  genome = NULL,
  draw = TRUE,
  cluster.samples = FALSE,
  na.rm = FALSE,
  nprobes.max = 1000,
  txn.types = "protein_coding",
  txn.font.size = 6,
  ...
)
```

**Arguments**

<code>chr</code>	chromosome
<code>beg</code>	begin of the region
<code>end</code>	end of the region
<code>betas</code>	beta value matrix (row: probes, column: samples)
<code>platform</code>	EPIC, HM450, or MM285
<code>genome</code>	hg38, mm10, ..., will infer if not given. For additional mapping, download the GRanges object from <a href="http://zwdzwd.github.io/InfiniumAnnotation">http://zwdzwd.github.io/InfiniumAnnotation</a> and provide the following argument ..., <code>genome = sesameAnno_buildManifestGRanges("downloaded_file"),...</code> to this function.
<code>draw</code>	draw figure or return betas
<code>cluster.samples</code>	whether to cluster samples
<code>na.rm</code>	remove probes with all NA.
<code>nprobes.max</code>	maximum number of probes to plot
<code>txn.types</code>	default to <code>protein_coding</code> , use NULL for all
<code>txn.font.size</code>	transcript name font size
<code>...</code>	additional options, see <code>assemble_plots</code>

**Value**

graphics or a matrix containing the captured beta values

**Examples**

```
betas <- sesameDataGet('HM450.76.TCGA.matched')$betas
visualizeRegion('chr20', 44648623, 44652152, betas, 'HM450')
```

---

visualizeSegments	<i>Visualize segments</i>
-------------------	---------------------------

---

**Description**

The function takes a CNSegment object obtained from cnSegmentation and plot the bin signals and segments (as horizontal lines).

**Usage**

```
visualizeSegments(seg, to.plot = NULL, genes.to.label = NULL)
```

**Arguments**

seg	a CNSegment object
to.plot	chromosome to plot (by default plot all chromosomes)
genes.to.label	gene(s) to label

**Details**

require ggplot2, scales

**Value**

plot graphics

**Examples**

```
sesameDataCache()
## Not run:
sdfs <- sesameDataGet('EPICv2.8.SigDF')
sdf <- sdfs[["K562_206909630040_R01C01"]]
seg <- cnSegmentation(sdf)
seg <- cnSegmentation(sdf, return.probe.signals=TRUE)
visualizeSegments(seg)
visualizeSegments(seg, to.plot=c("chr9","chr22"))
visualizeSegments(seg, genes.to.label=c("ABL1","BCR"))

## End(Not run)

sesameDataGet_resetEnv()
```

# Index

- \* **DNAMethylation**
  - sesame-package, 5
- \* **Microarray**
  - sesame-package, 5
- \* **QualityControl**
  - sesame-package, 5
- addMask, 6
- assemble\_plots, 6
- betasCollapseToPfx, 7
- BetaValueToMValue, 8
- binSignals, 9
- bisConversionControl, 9
- calcEffectSize, 10
- checkLevels, 11
- chipAddressToSignal, 11
- cnSegmentation, 12
- compareMouseStrainReference, 13
- compareMouseTissueReference, 14
- compareReference, 15
- controls, 16
- convertProbeID, 16
- createUCSCtrack, 17
- dataFrame2sesameQC, 18
- deIdentify, 18
- detectionPnegEcdf, 19
- diffRefSet, 20
- dmContrasts, 20
- DML, 21
- DMLpredict, 22
- DMR, 23
- dyeBiasCorr, 24
- dyeBiasCorrMostBalanced, 25
- dyeBiasCorrTypeINorm (dyeBiasNL), 26
- dyeBiasL, 25
- dyeBiasNL, 26
- ELBAR, 27
- estimateLeukocyte, 28
- formatVCF, 29
- getAFs, 30
- getAFTYPEIbySumAlleles, 30
- getBetas, 31
- getBinCoordinates, 32
- getMask, 32
- getRefSet, 33
- imputeBetas, 34
- imputeBetasByGenomicNeighbors, 35
- imputeBetasMatrixByMean, 36
- inferEthnicity, 36
- inferInfiniumIChannel, 37
- inferSex, 38
- inferSpecies, 39
- inferStrain, 40
- inferTissue, 41
- initFileSet, 42
- liftOver, 43
- listAvailableMasks, 43
- mapFileSet, 44
- mapToMammal40, 44
- matchDesign, 45
- meanIntensity, 46
- medianTotalIntensity, 46
- mLiftOver, 47
- MValueToBetaValue, 50
- negControls, 50
- noMasked, 51
- noob, 51
- normControls, 52
- openSesame, 53
- openSesameToFile, 54



palgen, 55  
parseGEOsignalMU, 55  
pOOBAH, 56  
predictAge, 57  
predictAgeHorvath353, 58  
predictAgeSkinBlood, 58  
predictMouseAgeInMonth, 59  
prefixMask, 60  
prefixMaskButC, 60  
prefixMaskButCG, 61  
prepSesame, 61  
prepSesameList, 62  
print.DMLSummary, 63  
print.fileSet, 63  
probeID\_designType, 64  
probeSuccessRate, 65  
  
qualityMask, 65  
  
readFileSet, 66  
readIDATpair, 67  
recommendedMaskNames, 68  
reIdentify, 68  
resetMask, 69  
  
scrub, 70  
scrubSoft, 70  
sdf\_read\_table, 72  
sdf\_write\_table, 73  
SDFcollapseToPfx, 71  
sdfPlatform, 71  
searchIDATprefixes, 73  
segmentBins, 74  
sesame (sesame-package), 5  
sesame-package, 5  
sesame\_checkVersion, 85  
sesameAnno\_attachManifest, 74  
sesameAnno\_buildAddressFile, 75  
sesameAnno\_buildManifestGRanges, 76  
sesameAnno\_download, 77  
sesameAnno\_readManifestTSV, 78  
sesameQC-class, 78  
sesameQC\_calcStats, 79  
sesameQC\_getStats, 80  
sesameQC\_plotBar, 81  
sesameQC\_plotBetaByDesign, 81  
sesameQC\_plotHeatSNPs, 82  
sesameQC\_plotIntensVsBetas, 83  
sesameQC\_plotRedGrnQQ, 84  
sesameQC\_rankStats, 84  
sesameQCtoDF, 79  
sesamize, 85  
setMask, 86  
SigDF, 86  
signalMU, 87  
sliceFileSet, 88  
summaryExtractTest, 89  
  
totalIntensities, 89  
twoCompsEst2, 90  
  
updateSigDF, 91  
  
visualizeGene, 91  
visualizeProbes, 92  
visualizeRegion, 93  
visualizeSegments, 95