Contents

Function Code for Package sesame **5**
Index of Functions . . . 5
.backgroundCorrCh1 . . . 9
.onAttach . . . 10
.optimizeCellComposition . . . 10
.setGroup_betas . . . 11
.setGroup_channel . . . 12
.setGroup_detection . . . 12
.setGroup_dyeBias . . . 12
.setGroup_intensity . . . 12
.setGroup_numProbes . . . 13
addMask . . . 13
aggregateTestEnrichments . . . 13
assemble_plots . . . 13
backgroundCorrectionNoobFit . . . 14
betaMix2States . . . 14
betaMix3States . . . 15
betasCollapseToPfx . . . 15
BetaValueToMValue . . . 16
binReadNumeric . . . 16
binSignals . . . 16
binWriteNumeric . . . 16
bisConversionControl . . . 17
calcEffectSize . . . 17
calcES_Significance . . . 18
calcMode . . . 18
checkLevels . . . 19
chipAddressToSignal . . . 19
cleanRefSet . . . 20
clusterWithinRowGroups . . . 20
clusterWithSampleGrouping . . . 20
clusterWithSignature . . . 20
cnSegmentation . . . 21
cnv_normal_default . . . 22
cnv_plot_extra . . . 22
compareDatbaseSetOverlap . . . 23
compareMouseStrainReference . . . 23
compareMouseTissueReference . . . 24
compareReference . . . 24
controls . . . 25
convertProbeID . . . 26
create_default_mask . . . 27
createDBNetwork . . . 27
createUCSCtrack . . . 27
databases_getMeta . . . 28
dataFrame2sesameQC . . . 28
dbStats . . . 29
deIdentify . . . 30
detectionPnegEcdf . . . 31
dichotomize . . . 31
diffRefSet . . . 31
dmContrasts . . . 31

DMGetProbeInfo . . . 32
DML . . . 32
DMLpredict . . . 33
DMR . . . 33
dmr_combine_pval . . . 34
dmr_merge_cpgs . . . 34
double.transform.f . . . 35
dyeBiasCorr . . . 35
dyeBiasCorrMostBalanced . . . 36
dyeBiasCorrTypeINorm . . . 36
dyeBiasL . . . 37
dyeBiasNL . . . 38
ELBAR . . . 39
errFunc . . . 40
estimateCellComposition . . . 40
estimateLeukocyte . . . 40
exonToCDS . . . 42
expand_url . . . 42
formatVCF . . . 42
genotyper . . . 43
getAFs . . . 43
getAFTypeIbySumAlleles . . . 44
getBetas . . . 44
getBinCoordinates . . . 45
getg0 . . . 45
getMask . . . 45
getRefSet . . . 46
getSignatureU . . . 46
getSignatureUTop . . . 47
guess_chrmorder . . . 47
guess_dbnames . . . 47
imputeBetas . . . 48
imputeBetasByGenomicNeighbors . . . 49
imputeBetasMatrixByMean . . . 49
inferEthnicity . . . 50
inferInfiniumIChannel . . . 50
inferPlatformFromTango . . . 51
inferSex . . . 51
inferSpecies . . . 53
inferStrain . . . 54
inferTissue . . . 55
inferUniverse . . . 56
InfI . . . 56
InfIG . . . 56
InfII . . . 57
InfIR . . . 57
initFileSet . . . 57
KYCG_annoProbes . . . 57
KYCG_buildGeneDBs . . . 58
KYCG_getDBs . . . 59
KYCG_listDBGroups . . . 59
KYCG_loadDBs . . . 60
KYCG_plotBar . . . 61
KYCG_plotDot . . . 61

KYCG_plotEnrichAll . . . 61
KYCG_plotLollipop . . . 62
KYCG_plotManhattan . . . 63
KYCG_plotMeta . . . 64
KYCG_plotMetaEnrichment . . . 64
KYCG_plotPointRange . . . 65
KYCG_plotSetEnrichment . . . 65
KYCG_plotVolcano . . . 65
KYCG_plotWaterfall . . . 66
leftRightMerge1 . . . 67
liftOver . . . 67
listAvailableMasks . . . 68
mapFileSet . . . 68
mapToMammal40 . . . 68
maskIG . . . 68
match1To2_1state . . . 69
match1To2_3states . . . 69
matchDesign . . . 70
meanIntensity . . . 70
medianTotalIntensity . . . 71
mLiftOver . . . 71
model_contrasts . . . 72
mouseBetaToAF . . . 72
MValueToBetaValue . . . 72
negControls . . . 73
noMasked . . . 73
nonuniqMask . . . 73
noob . . . 74
noobSub . . . 74
normalizeSetM . . . 75
normControls . . . 75
normExpSignal . . . 75
oobG . . . 76
oobR . . . 76
openSesame . . . 76
openSesameToFile . . . 77
palgen . . . 77
parseGEOsignalMU . . . 78
plotCytoBand . . . 78
plotMapLines . . . 79
plotTranscript1 . . . 79
plotTranscripts . . . 80
pOOBAH . . . 81
predictAge . . . 81
predictAgeHorvath353 . . . 82
predictAgeSkinBlood . . . 82
predictMouseAgeInMonth . . . 82
prefixMask . . . 82
prefixMaskButC . . . 83
prefixMaskButCG . . . 83
preparePlotDF . . . 83
prepSesame . . . 84
prepSesameList . . . 84
print.DMLSummary . . . 85

print.fileSet . . . 85
probeID_designType . . . 85
probeSuccessRate . . . 85
qualityMask . . . 85
queryCheckPlatform . . . 86
readControls . . . 86
readFileSet . . . 87
readIDAT . . . 87
readIDAT_nonenc . . . 87
readIDAT1 . . . 90
readIDATpair . . . 91
recommendedMaskNames . . . 92
reIdentify . . . 92
resetMask . . . 93
scrub . . . 93
scrubSoft . . . 93
sdf_read_table . . . 94
sdf_write_table . . . 94
SDFcollapseToPfx . . . 94
sdfMsg . . . 94
sdfPlatform . . . 95
searchIDATprefixes . . . 95
segmentBins . . . 96
sesame_checkVersion . . . 96
sesameAnno_attachManifest . . . 96
sesameAnno_buildAddressFile . . . 97
sesameAnno_buildManifestGRanges . . . 97
sesameAnno_download . . . 98
sesameAnno_readManifestTSV . . . 98
sesameQC_calcStats . . . 99
sesameQC_calcStats_betas . . . 99
sesameQC_calcStats_channel . . . 100
sesameQC_calcStats_detection . . . 101
sesameQC_calcStats_dyeBias . . . 101
sesameQC_calcStats_intensity . . . 102
sesameQC_calcStats_numProbes . . . 103
sesameQC_getStats . . . 103
sesameQC_plotBar . . . 104
sesameQC_plotBetaByDesign . . . 104
sesameQC_plotHeatSNPs . . . 105
sesameQC_plotIntensVsBetas . . . 105
sesameQC_plotRedGrnQQ . . . 107
sesameQC_rankStats . . . 107
sesamize . . . 107
setMask . . . 107
SigDF . . . 108
signalMU . . . 108
signalMU_oo . . . 108
SigSetToSigDF . . . 109
sliceFileSet . . . 109
species_ret . . . 110
speciesInfo . . . 110
subsetDBs . . . 110
summaryExtractCf . . . 111

summaryExtractTest . . . 111
testEnrichment . . . 112
testEnrichmentFisher . . . 113
testEnrichmentFisherN . . . 113
testEnrichmentGene . . . 114
testEnrichmentSEA . . . 114
testEnrichmentSEA1 . . . 115
testEnrichmentSpearman . . . 116
totalIntensities . . . 116
train.model.lm . . . 116
twoCompsDiff . . . 116
twoCompsEst2 . . . 117
updateSigDF . . . 118
valid_url . . . 119
valleyDescent . . . 119
vcf_header . . . 119
visualizeGene . . . 119
visualizeProbes . . . 120
visualizeRegion . . . 120
visualizeSegments . . . 121
wrap_openSesame . . . 122
wrap_openSesame1 . . . 123

Function Code for Package sesame

Index of Functions

- .backgroundCorrCh1
- .onAttach
- .optimizeCellComposition
- .setGroup_betas
- .setGroup_channel
- .setGroup_detection
- .setGroup_dyeBias
- .setGroup_intensity
- .setGroup_numProbes
- addMask
- aggregateTestEnrichments
- assemble_plots
- backgroundCorrectionNoobFit
- betaMix2States
- betaMix3States
- betasCollapseToPfx
- BetaValueToMValue
- binReadNumeric
- binSignals
- binWriteNumeric
- bisConversionControl
- calcEffectSize
- calcES_Significance
- calcMode
- checkLevels
- chipAddressToSignal
- cleanRefSet

- clusterWithinRowGroups
- clusterWithSampleGrouping
- clusterWithSignature
- cnSegmentation
- cnv_normal_default
- cnv_plot_extra
- compareDatbaseSetOverlap
- compareMouseStrainReference
- compareMouseTissueReference
- compareReference
- controls
- convertProbeID
- create_default_mask
- createDBNetwork
- createUCSCtrack
- databases_getMeta
- dataFrame2sesameQC
- dbStats
- deIdentify
- detectionPnegEcdf
- dichotomize
- diffRefSet
- dmContrasts
- DMGetProbeInfo
- DML
- DMLpredict
- DMR
- dmr_combine_pval
- dmr_merge_cpgs
- double.transform.f
- dyeBiasCorr
- dyeBiasCorrMostBalanced
- dyeBiasCorrTypeINorm
- dyeBiasL
- dyeBiasNL
- ELBAR
- errFunc
- estimateCellComposition
- estimateLeukocyte
- exonToCDS
- expand_url
- formatVCF
- genotyper
- getAFs
- getAFTypeIbySumAlleles
- getBetas
- getBinCoordinates
- getg0
- getMask
- getRefSet
- getSignatureU
- getSignatureUTop
- guess_chrmorder
- guess_dbnames

- imputeBetas
- imputeBetasByGenomicNeighbors
- imputeBetasMatrixByMean
- inferEthnicity
- inferInfiniumIChannel
- inferPlatformFromTango
- inferSex
- inferSpecies
- inferStrain
- inferTissue
- inferUniverse
- InfI
- InfIG
- InfII
- InfIR
- initFileSet
- KYCG_annoProbes
- KYCG_buildGeneDBs
- KYCG_getDBs
- KYCG_listDBGroups
- KYCG_loadDBs
- KYCG_plotBar
- KYCG_plotDot
- KYCG_plotEnrichAll
- KYCG_plotLollipop
- KYCG_plotManhattan
- KYCG_plotMeta
- KYCG_plotMetaEnrichment
- KYCG_plotPointRange
- KYCG_plotSetEnrichment
- KYCG_plotVolcano
- KYCG_plotWaterfall
- leftRightMerge1
- liftOver
- listAvailableMasks
- mapFileSet
- mapToMammal40
- maskIG
- match1To2_1state
- match1To2_3states
- matchDesign
- meanIntensity
- medianTotalIntensity
- mLiftOver
- model_contrasts
- mouseBetaToAF
- MValueToBetaValue
- negControls
- noMasked
- nonuniqMask
- noob
- noobSub
- normalizeSetM
- normControls

- normExpSignal
- oobG
- oobR
- openSesame
- openSesameToFile
- palgen
- parseGEOsignalMU
- plotCytoBand
- plotMapLines
- plotTranscript1
- plotTranscripts
- pOOBAH
- predictAge
- predictAgeHorvath353
- predictAgeSkinBlood
- predictMouseAgeInMonth
- prefixMask
- prefixMaskButC
- prefixMaskButCG
- preparePlotDF
- prepSesame
- prepSesameList
- print.DMLSummary
- print.fileSet
- probeID_designType
- probeSuccessRate
- qualityMask
- queryCheckPlatform
- readControls
- readFileSet
- readIDAT
- readIDAT_nonenc
- readIDAT1
- readIDATpair
- recommendedMaskNames
- reIdentify
- resetMask
- scrub
- scrubSoft
- sdf_read_table
- sdf_write_table
- SDFcollapseToPfx
- sdfMsg
- sdfPlatform
- searchIDATprefixes
- segmentBins
- sesame_checkVersion
- sesameAnno_attachManifest
- sesameAnno_buildAddressFile
- sesameAnno_buildManifestGRanges
- sesameAnno_download
- sesameAnno_readManifestTSV
- sesameQC_calcStats
- sesameQC_calcStats_betas

- sesameQC_calcStats_channel
- sesameQC_calcStats_detection
- sesameQC_calcStats_dyeBias
- sesameQC_calcStats_intensity
- sesameQC_calcStats_numProbes
- sesameQC_getStats
- sesameQC_plotBar
- sesameQC_plotBetaByDesign
- sesameQC_plotHeatSNPs
- sesameQC_plotIntensVsBetas
- sesameQC_plotRedGrnQQ
- sesameQC_rankStats
- sesamize
- setMask
- SigDF
- signalMU
- signalMU_oo
- SigSetToSigDF
- sliceFileSet
- species_ret
- speciesInfo
- subsetDBs
- summaryExtractCf
- summaryExtractTest
- testEnrichment
- testEnrichmentFisher
- testEnrichmentFisherN
- testEnrichmentGene
- testEnrichmentSEA
- testEnrichmentSEA1
- testEnrichmentSpearman
- totalIntensities
- train.model.lm
- twoCompsDiff
- twoCompsEst2
- updateSigDF
- valid_url
- valleyDescent
- vcf_header
- visualizeGene
- visualizeProbes
- visualizeRegion
- visualizeSegments
- wrap_openSesame
- wrap_openSesame1

.backgroundCorrCh1

```
function (x, pp, alpha, offset = 15)
{
    mu.bg <- pp$mu
    sigma <- pp$sigma
    sigma2 <- sigma * sigma
```

```
    if (alpha <= 0)
        stop("alpha must be positive")
    if (any(na.omit(sigma) <= 0))
        stop("sigma must be positive")
    mu.sf <- x - mu.bg - sigma2/alpha
    signal <- mu.sf + sigma2 * exp(dnorm(0, mean = mu.sf, sd = sigma,
        log = TRUE) - pnorm(0, mean = mu.sf, sd = sigma, lower.tail = FALSE,
        log.p = TRUE))
    o <- !is.na(signal)
    if (any(signal[o] < 0)) {
        warning("Limit of numerical accuracy reached with\nvery low intensity or very high background:\n
        signal[o] <- pmax(signal[o], 1e-06)
    }
    signal.min <- min(signal, na.rm = TRUE)
    signal <- signal - signal.min
    offset + signal
}
<bytecode: 0x0000012a343b6658>
<environment: namespace:sesame>
```

.onAttach

```
function (libname, pkgname)
{
    packageStartupMessage("\n-------------------------------------------------------------\n| SEnsible Step
}
<bytecode: 0x0000012a343b87f0>
<environment: namespace:sesame>
```

.optimizeCellComposition

```
function (g, q, frac0 = NULL, temp = 0.5, maxIter = 1000, delta = 1e-04,
    step.max = 1, verbose = FALSE)
{
    M <- ncol(g)
    if (is.null(frac0)) {
        frac <- c(1, rep(0, M))
    }
    else {
        frac <- frac0
    }
    errcurrent <- errFunc(frac, g, q)
    errmin <- errcurrent
    frac.min <- frac
    niter <- 1
    repeat {
        nu <- sample(seq_len(M + 1), 2)
        step.size <- runif(1) * step.max
        frac.test <- double.transform.f(frac, nu[1], nu[2], step.size)
        if (!is.null(frac.test)) {
            if (verbose) {
                message("errcurrent=", errcurrent, "frac=", paste(lapply(frac,
```

```
                  function(x) sprintf("%1.2f", x)), collapse = "-"),
                  ";stepsize=", step.size, ";temp=", temp, ";best=",
                  paste(lapply(frac.min, function(x) sprintf("%1.2f",
                    x)), collapse = "-"), ";err=", errmin)
            }
            errtest <- errFunc(frac.test, g, q)
            if (errtest < errmin) {
                errmin <- errtest
                frac.min <- frac.test
                if ((errmin - errtest) > errmin * delta)
                  niter <- 1
            }
            else {
                niter <- niter + 1
                if (niter > maxIter) {
                  break
                }
            }
            if (runif(1) < exp(-(errtest - errcurrent)/temp)) {
                errcurrent <- errtest
                frac <- frac.test
            }
        }
    }
    list(frac.min = frac.min, errmin = errmin)
}
<bytecode: 0x0000012a343c0be8>
<environment: namespace:sesame>
```

.setGroup_betas

```
function ()
{
    list(`Beta Value` = c(mean_beta = "Mean Beta           ",
        median_beta = "Median Beta         ", frac_unmeth = "% Beta < 0.3        ",
        frac_meth = "% Beta > 0.7        ", num_na = "N. is.na(Beta)      ",
        frac_na = "% is.na(Beta)       ", mean_beta_cg = "Mean Beta (CG)      ",
        median_beta_cg = "Median Beta (CG)    ", frac_unmeth_cg = "% Beta < 0.3 (CG)   ",
        frac_meth_cg = "% Beta > 0.7 (CG)   ", num_na_cg = "N. is.na(Beta) (CG) ",
        frac_na_cg = "% is.na(Beta) (CG)  ", mean_beta_ch = "Mean Beta (CH)      ",
        median_beta_ch = "Median Beta (CH)    ", frac_unmeth_ch = "% Beta < 0.3 (CH)   ",
        frac_meth_ch = "% Beta > 0.7 (CH)   ", num_na_ch = "N. is.na(Beta) (CH) ",
        frac_na_ch = "% is.na(Beta) (CH)  ", mean_beta_rs = "Mean Beta (RS)      ",
        median_beta_rs = "Median Beta (RS)    ", frac_unmeth_rs = "% Beta < 0.3 (RS)   ",
        frac_meth_rs = "% Beta > 0.7 (RS)   ", num_na_rs = "N. is.na(Beta) (RS) ",
        frac_na_rs = "% is.na(Beta) (RS)  "))
}
<bytecode: 0x0000012a343c8250>
<environment: namespace:sesame>
```

.setGroup_channel

```
function ()
{
    list(`Color Channel` = c(InfI_switch_R2R = "N. Inf.I Probes Red -> Red ",
        InfI_switch_G2G = "N. Inf.I Probes Grn -> Grn ", InfI_switch_R2G = "N. Inf.I Probes Red -> Grn
        InfI_switch_G2R = "N. Inf.I Probes Grn -> Red "))
}
<bytecode: 0x0000012a343cb8d0>
<environment: namespace:sesame>
```

.setGroup_detection

```
function ()
{
    list(Detection = c(num_dtna = "N. Probes w/ Missing Raw Intensity  ",
        frac_dtna = "% Probes w/ Missing Raw Intensity   ", num_dt = "N. Probes w/ Detection Success
        frac_dt = "% Detection Success                  ", num_dt_mk = "N. Detection Succ. (after maskin
        frac_dt_mk = "% Detection Succ. (after masking)   ",
        num_dt_cg = "N. Probes w/ Detection Success (cg) ", frac_dt_cg = "% Detection Success (cg)
        num_dt_ch = "N. Probes w/ Detection Success (ch) ", frac_dt_ch = "% Detection Success (ch)
        num_dt_rs = "N. Probes w/ Detection Success (rs) ", frac_dt_rs = "% Detection Success (rs)
}
<bytecode: 0x0000012a343c5be8>
<environment: namespace:sesame>
```

.setGroup_dyeBias

```
function ()
{
    list(`Dye Bias` = c(medR = "Median Inf.I Intens. Red           ",
        medG = "Median Inf.I Intens. Grn           ", topR = "Median of Top 20 Inf.I Intens. Red ",
        topG = "Median of Top 20 Inf.I Intens. Grn ", RGratio = "Ratio of Red-to-Grn median Intens. ",
        RGdistort = "Ratio of Top vs. Global R/G Ratios "))
}
<bytecode: 0x0000012a343cda68>
<environment: namespace:sesame>
```

.setGroup_intensity

```
function ()
{
    list(`Signal Intensity` = c(mean_intensity = "Mean sig. intensity         ",
        mean_intensity_MU = "Mean sig. intensity (M+U)   ", mean_ii = "Mean sig. intensity (Inf.II)",
        mean_inb_grn = "Mean sig. intens.(I.Grn IB) ", mean_inb_red = "Mean sig. intens.(I.Red IB) ",
        mean_oob_grn = "Mean sig. intens.(I.Grn OOB)", mean_oob_red = "Mean sig. intens.(I.Red OOB)",
        na_intensity_M = "N. NA in M (all probes)     ", na_intensity_U = "N. NA in U (all probes)
        na_intensity_ig = "N. NA in raw intensity (IG) ", na_intensity_ir = "N. NA in raw intensity (IR
        na_intensity_ii = "N. NA in raw intensity (II) "))
}
<bytecode: 0x0000012a343d5ca0>
<environment: namespace:sesame>
```

.setGroup__numProbes

```
function ()
{
    list(`Number of Probes` = c(num_probes = "N. Probes          ",
        num_probes_II = "N. Inf.-II Probes ", num_probes_IR = "N. Inf.-I (Red)   ",
        num_probes_IG = "N. Inf.-I (Grn)   ", num_probes_cg = "N. Probes (CG)    ",
        num_probes_ch = "N. Probes (CH)    ", num_probes_rs = "N. Probes (RS)    "))
}
<bytecode: 0x0000012a343cfab0>
<environment: namespace:sesame>
```

addMask

```
function (sdf, probes)
{
    if (is.logical(probes)) {
        sdf$mask[probes[sdf$Probe_ID]] <- TRUE
    }
    else {
        sdf$mask[match(probes, sdf$Probe_ID)] <- TRUE
    }
    sdf
}
<bytecode: 0x0000012a343d7af0>
<environment: namespace:sesame>
```

aggregateTestEnrichments

```
function (result_list, column = "estimate", return_df = FALSE)
{
    mtx <- do.call(cbind, lapply(result_list[[1]]$dbname, function(db) {
        vapply(result_list, function(x) x$estimate[x$dbname ==
            db], numeric(1))
    }))
    colnames(mtx) <- result_list[[1]]$dbname
    if (return_df) {
        melt(mtx, value.name = column, varnames = c("query",
            "db"))
    }
    else {
        mtx
    }
}
<bytecode: 0x0000012a343dcf68>
<environment: namespace:sesame>
```

assemble__plots

```
function (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)
{
    if (is.null(show.samples.n)) {
        show.samples.n <- ncol(betas)
    }
    if (is.null(heat.height) && length(txns) > 0) {
        heat.height <- 10/length(txns)
    }
    w <- WGrob(plt.txns, name = "txn")
    w <- w + WGrob(plt.mapLines, Beneath(pad = 0, height = mapLine.height))
    w <- w + WHeatmap(t(betas), Beneath(height = heat.height),
        name = "betas", cmp = CMPar(dmin = dmin, dmax = dmax),
        xticklabels = show.probeNames, xticklabel.rotat = 45,
        yticklabels = show.sampleNames, yticklabel.fontsize = sample.name.fontsize,
        yticklabels.n = show.samples.n, xticklabels.n = length(probes))
    w <- w + WGrob(plt.cytoband, TopOf("txn", height = 0.15))
    w
}
<bytecode: 0x0000012a343dadd0>
<environment: namespace:sesame>
```

backgroundCorrectionNoobFit

```
function (ib, bg)
{
    e <- MASS::huber(bg)
    mu <- e$mu
    sigma <- e$s
    alpha <- pmax(MASS::huber(ib)$mu - mu, 10)
    list(mu = mu, sigma = sigma, alpha = alpha)
}
<bytecode: 0x0000012a343e7470>
<environment: namespace:sesame>
```

betaMix2States

```
function (x, n_samples = 10000, th_init = 0.5)
{
    if (sum(!is.na(x)) > n_samples) {
        x1 <- sample(na.omit(x), n_samples)
    }
    else {
        x1 <- na.omit(x)
    }
    m <- matrix(0, nrow = length(x1), ncol = 2)
    m[x1 <= th_init, 1] <- 1
    m[x1 > th_init, 2] <- 1
    fitres <- RPMM::blc(matrix(x1), m, maxiter = 5, tol = 0.001,
        verbose = FALSE)
    m1 <- apply(fitres$w, 1, which.max)
    th <- mean(max(x1[m1 == 1]), min(x1[m1 == 2]))
    m2 <- cut(x, breaks = c(0, th, 1), include.lowest = TRUE)
```

```
    names(m2) <- names(x)
    m2
}
<bytecode: 0x0000012a343ecdb8>
<environment: namespace:sesame>
```

betaMix3States

```
function (x, n_samples = 10000, th_init1 = 0.2, th_init2 = 0.7)
{
    if (sum(!is.na(x)) > n_samples) {
        x1 <- sample(na.omit(x), n_samples)
    }
    else {
        x1 <- na.omit(x)
    }
    m <- matrix(0, nrow = length(x1), ncol = 3)
    m[x1 <= th_init1, 1] <- 1
    m[x1 > th_init1 & x1 <= th_init2, 2] <- 1
    m[x1 > th_init2, 3] <- 1
    fitres <- RPMM::blc(matrix(x1), m, maxiter = 5, tol = 0.001,
        verbose = FALSE)
    m1 <- apply(fitres$w, 1, which.max)
    th1 <- mean(max(x1[m1 == 1]), min(x1[m1 == 2]))
    th2 <- mean(max(x1[m1 == 2]), min(x1[m1 == 3]))
    m2 <- cut(x, breaks = c(0, th1, th2, 1), include.lowest = TRUE)
    names(m2) <- names(x)
    m2
}
<bytecode: 0x0000012a343f2800>
<environment: namespace:sesame>
```

betasCollapseToPfx

```
function (betas, BPPARAM = SerialParam())
{
    if (is.matrix(betas)) {
        pfxes <- vapply(strsplit(rownames(betas), "_"), function(x) x[1],
            character(1))
        out <- do.call(cbind, bplapply(seq_len(ncol(betas)),
            function(i) {
                vapply(split(betas[, i], pfxes), mean, numeric(1),
                  na.rm = TRUE)
            }, BPPARAM = BPPARAM))
        colnames(out) <- colnames(betas)
        out
    }
    else {
        pfxes <- vapply(strsplit(names(betas), "_"), function(x) x[1],
            character(1))
        vapply(split(betas, pfxes), mean, numeric(1), na.rm = TRUE)
    }
```

```
}
<bytecode: 0x0000012a34412918>
<environment: namespace:sesame>
```

BetaValueToMValue

```
function (b)
{
    log2(b/(1 - b))
}
<bytecode: 0x0000012a34414570>
<environment: namespace:sesame>
```

binReadNumeric

```
function (con, s_ind, p_ind, p_len, n = 1, inc = 4)
{
    seek(con, ((s_ind - 1) * p_len + p_ind - 1) * inc, origin = "start")
    readBin(con, "numeric", n, size = inc)
}
<bytecode: 0x0000012a344167e8>
<environment: namespace:sesame>
```

binSignals

```
function (probe.signals, bin.coords, probeCoords)
{
    ov <- GenomicRanges::findOverlaps(probeCoords, bin.coords)
    if (.hasSlot(ov, "queryHits")) {
        .bins <- names(bin.coords)[ov@subjectHits]
        .probe.signals <- probe.signals[names(probeCoords)[ov@queryHits]]
    }
    else {
        .bins <- names(bin.coords)[ov@to]
        .probe.signals <- probe.signals[names(probeCoords)[ov@from]]
    }
    bin.signals <- vapply(split(.probe.signals, .bins), median,
        1, na.rm = TRUE)
    bin.signals
}
<bytecode: 0x0000012a344184b0>
<environment: namespace:sesame>
```

binWriteNumeric

```
function (con, num_array, inc = 4, beg = 0)
{
    seek(con, beg * inc, origin = "start", rw = "write")
    writeBin(as.numeric(num_array), con, size = inc)
}
```

```
<bytecode: 0x0000012a3441d228>
<environment: namespace:sesame>
```

bisConversionControl

```
function (sdf, extR = NULL, extA = NULL, verbose = FALSE)
{
    platform <- sdfPlatform(sdf, verbose = verbose)
    if (platform %in% c("EPICplus", "EPIC", "HM450")) {
        extR <- sesameDataGet(paste0(platform, ".probeInfo"))$typeI.extC
        extA <- sesameDataGet(paste0(platform, ".probeInfo"))$typeI.extT
    }
    stopifnot(!is.null(extR) && !is.null(extA))
    df <- InfIR(sdf)
    extR <- intersect(df$Probe_ID, extR)
    extA <- intersect(df$Probe_ID, extA)
    dR <- df[match(extR, df$Probe_ID), ]
    dA <- df[match(extA, df$Probe_ID), ]
    mean(c(dR$MG, dR$UG), na.rm = TRUE)/mean(c(dA$MG, dA$UG),
        na.rm = TRUE)
}
<bytecode: 0x0000012a3441f238>
<environment: namespace:sesame>
```

calcEffectSize

```
function (pred)
{
    vars <- colnames(colData(pred))
    if (length(vars) == 1) {
        eff <- data.frame(x = apply(assay(pred), 1, function(x) max(x) -
            min(x)))
        colnames(eff) <- vars[[1]]
        rownames(eff) <- rownames(pred)
        return(eff)
    }
    eff <- as.data.frame(do.call(cbind, lapply(vars, function(var) {
        other_vars <- vars[vars != var]
        col_indices <- seq_len(nrow(colData(pred)))
        Reduce(pmax, lapply(split(col_indices, colData(pred)[other_vars]),
            function(x) {
                apply(assay(pred)[, x], 1, function(x) max(x) -
                  min(x))
            }))
    })))
    colnames(eff) <- vars
    rownames(eff) <- rownames(pred)
    eff
}
<bytecode: 0x0000012a3442d008>
<environment: namespace:sesame>
```

calcES__Significance

```
function (dCont, dDisc, permut = 100, precise = FALSE)
{
    dCont <- sort(dCont)
    dContName <- names(dCont)
    dDiscN <- length(dDisc)
    dContN <- length(dCont)
    s <- rep(-1/(dContN - dDiscN), dContN)
    ess <- do.call(rbind, lapply(seq_len(permut), function(i) {
        s[sample.int(dContN, dDiscN)] <- 1/dDiscN
        cs <- cumsum(s)
        data.frame(es_max = max(cs), es_min = min(cs))
    }))
    presence <- names(dCont) %in% dDisc
    s <- ifelse(presence, 1/sum(presence), -1/sum(!presence))
    cs <- cumsum(s)
    es_max <- max(cs)
    es_min <- min(cs)
    res <- list(es_small = es_max, es_large = -es_min, pv_small = 1 -
        ecdf(ess$es_max)(es_max), pv_large = ecdf(ess$es_min)(es_min))
    if (res$pv_small < 0.01 || res$pv_large < 0.01) {
        if (permut < 1000 && precise) {
            res <- calcES_Significance(dCont, dDisc, permut = 1000)
        }
        else {
            if (res$pv_small == 0) {
                res$pv_small <- pnorm(es_max, mean = mean(ess$es_max),
                  sd = sd(ess$es_max), lower.tail = FALSE)
            }
            if (res$pv_large == 0) {
                res$pv_large <- pnorm(es_max, mean = mean(ess$es_max),
                  sd = sd(ess$es_max), lower.tail = TRUE)
            }
        }
    }
    res
}
<bytecode: 0x0000012a3442a738>
<environment: namespace:sesame>
```

calcMode

```
function (x)
{
    dd <- density(na.omit(x))
    dd$x[which.max(dd$y)]
}
<bytecode: 0x0000012a3443f450>
<environment: namespace:sesame>
```

checkLevels

```
function (betas, fc)
{
    stopifnot(is(fc, "factor") || is(fc, "character"))
    apply(betas, 1, function(dt) {
        all(vapply(split(dt, fc), function(x) sum(!is.na(x)) >
            0, logical(1)))
    })
}
<bytecode: 0x0000012a34443550>
<environment: namespace:sesame>
```

chipAddressToSignal

```
function (dm, mft, min_beads = NULL)
{
    mft1 <- mft[!is.na(mft$col), ]
    tmpM <- dm[match(mft1$M, rownames(dm)), ]
    tmpU <- dm[match(mft1$U, rownames(dm)), ]
    sdf <- data.frame(Probe_ID = mft1$Probe_ID, MG = unname(tmpM[,
        "G"]), MR = unname(tmpM[, "R"]), UG = unname(tmpU[, "G"]),
        UR = unname(tmpU[, "R"]), col = mft1$col, mask = FALSE)
    if (!is.null(min_beads)) {
        sdf$mask <- (is.na(tmpM[, "GN"]) | is.na(tmpM[, "RN"]) |
            is.na(tmpU[, "GN"]) | is.na(tmpU[, "RN"]) | tmpM[,
            "GN"] < min_beads | tmpM[, "RN"] < min_beads | tmpU[,
            "GN"] < min_beads | tmpU[, "RN"] < min_beads)
    }
    mft2 <- mft[is.na(mft$col), ]
    if (nrow(mft2) > 0) {
        tmp <- dm[match(mft2$U, rownames(dm)), ]
        s2 <- data.frame(Probe_ID = mft2$Probe_ID, MG = NA, MR = NA,
            UG = unname(tmp[, "G"]), UR = unname(tmp[, "R"]),
            col = "2", mask = FALSE)
        if (!is.null(min_beads)) {
            s2$mask <- (is.na(tmp[, "GN"]) | is.na(tmp[, "RN"]) |
                tmp[, "GN"] < min_beads | tmp[, "RN"] < min_beads)
        }
        sdf <- rbind(sdf, s2)
    }
    sdf$col <- factor(sdf$col, levels = c("G", "R", "2"))
    sdf <- sdf[match(mft$Probe_ID, sdf$Probe_ID), ]
    sdf <- structure(sdf, class = c("SigDF", "data.frame"))
    rownames(sdf) <- NULL
    sdf
}
<bytecode: 0x0000012a34448220>
<environment: namespace:sesame>
```

cleanRefSet

```
function (g, platform = c("EPIC", "HM450", "HM27"))
{
    platform <- match.arg(platform)
    mapinfo <- sesameDataGet(paste0(platform, ".probeInfo"))[[paste0("mapped.probes.hg19")]]
    g <- g[GenomicRanges::intersect(rownames(g), names(mapinfo)),
        , drop = FALSE]
    g.clean <- g[apply(g, 1, function(x) !any(is.na(x))), , drop = FALSE]
    g.clean <- g.clean[rownames(g.clean) %in% names(mapinfo),
        , drop = FALSE]
    g.clean <- g.clean[!(as.vector(GenomicRanges::seqnames(mapinfo[rownames(g.clean)])) %in%
        c("chrX", "chrY", "chrM")), , drop = FALSE]
    g.clean <- g.clean[grep("cg", rownames(g.clean)), , drop = FALSE]
    g.clean
}
<bytecode: 0x0000012a344ae588>
<environment: namespace:sesame>
```

clusterWithinRowGroups

```
function (betas, sigs)
{
    do.call(rbind, lapply(sigs, function(x) {
        row.cluster(betas[x, ])$mat
    }))
}
<bytecode: 0x0000012a344ac690>
<environment: namespace:sesame>
```

clusterWithSampleGrouping

```
function (betas, grouping, groups = unique(grouping))
{
    do.call(cbind, lapply(groups, function(g) {
        column.cluster(betas[, grouping == g])$mat
    }))
}
<bytecode: 0x0000012a344ba310>
<environment: namespace:sesame>
```

clusterWithSignature

```
function (betas, grouping, sigs)
{
    pbs <- do.call(c, lapply(names(sigs), function(g) {
        if (length(sigs[[g]]) > 5)
            rownames(row.cluster(betas[intersect(rownames(betas),
                sigs[[g]]), ])$mat)
        else NULL
    }))
    spl <- do.call(c, lapply(names(sigs), function(g) {
```

```
        colnames(column.cluster(betas[, grouping == g])$mat)
    }))
    betas[pbs, spl]
}
<bytecode: 0x0000012a344b7e30>
<environment: namespace:sesame>
```

cnSegmentation

```
function (sdf, sdfs.normal = NULL, genomeInfo = NULL, probeCoords = NULL,
    tilewidth = 50000, verbose = FALSE, return.probe.signals = FALSE)
{
    stopifnot(is(sdf, "SigDF"))
    platform <- sdfPlatform(sdf, verbose = verbose)
    if (is.null(sdfs.normal)) {
        sdfs.normal <- cnv_normal_default(platform)
    }
    if (is.null(genomeInfo)) {
        genome <- sesameData_check_genome(NULL, platform)
        genomeInfo <- sesameData_getGenomeInfo(genome)
    }
    if (is.null(probeCoords)) {
        genome <- sesameData_check_genome(NULL, platform)
        probeCoords <- sesameData_getManifestGRanges(platform,
            genome = genome)
    }
    seqLength <- genomeInfo$seqLength
    gapInfo <- genomeInfo$gapInfo
    target.intens <- totalIntensities(sdf)
    normal.intens <- do.call(cbind, lapply(sdfs.normal, function(sdf) {
        totalIntensities(sdf)
    }))
    target.intens <- na.omit(target.intens)
    pb <- intersect(rownames(normal.intens), names(target.intens))
    pb <- intersect(names(probeCoords), pb)
    target.intens <- target.intens[pb]
    normal.intens <- normal.intens[pb, ]
    probeCoords <- probeCoords[pb]
    fit <- lm(y ~ ., data = data.frame(y = target.intens, X = normal.intens))
    probe.signals <- setNames(log2(target.intens/pmax(predict(fit),
        1)), pb)
    if (return.probe.signals) {
        probeCoords$cnv <- probe.signals
        return(probeCoords[seqnames(probeCoords) != "*"])
    }
    bin.coords <- getBinCoordinates(seqLength, gapInfo, tilewidth = tilewidth,
        probeCoords)
    bin.signals <- binSignals(probe.signals, bin.coords, probeCoords)
    structure(list(seg.signals = segmentBins(bin.signals, bin.coords),
        bin.coords = bin.coords, bin.signals = bin.signals, genomeInfo = genomeInfo),
        class = "CNSegment")
}
<bytecode: 0x0000012a344c3b80>
```

```
<environment: namespace:sesame>
```

cnv__normal__default

```
function (platform)
{
    if (platform == "EPICv2") {
        sdfs.normal <- sesameDataGet("EPICv2.8.SigDF")
        sdfs.normal[c("GM12878_206909630042_R08C01", "GM12878_206909630040_R03C01")]
    }
    else if (platform == "EPIC") {
        sdfs.normal <- sesameDataGet("EPIC.5.SigDF.normal")
    }
    else {
        stop(sprintf("Please provide sdfs.normal=. No default for %s",
            platform))
    }
}
<bytecode: 0x0000012a344c75f0>
<environment: namespace:sesame>
```

cnv__plot__extra

```
function (seg, genes.to.label, seq.names, seqstart, totlen, p)
{
    cband <- seg$genomeInfo$cytoBand
    cband <- cband[cband$chrom %in% seq.names, ]
    xmins <- (cband$chromStart + seqstart[as.character(cband$chrom)])/totlen
    xmaxs <- (cband$chromEnd + seqstart[as.character(cband$chrom)])/totlen
    requireNamespace("pals")
    cband2col <- setNames(pals::ocean.gray(10)[seq(9, 3)], c("stalk",
        "gneg", "gpos25", "gpos50", "gpos75", "gpos100"))
    cband2col["acen"] <- "red"
    cband2col["gvar"] <- cband2col["gpos75"]
    band_color <- cband2col[as.character(cband$gieStain)]
    band_loc <- min(-2, min(seg$bin.signals, na.rm = TRUE)) -
        0.6
    p <- p + ggplot2::geom_rect(ggplot2::aes(xmin = xmins, xmax = xmaxs,
        ymin = band_loc - 0.5, ymax = band_loc, fill = names(band_color))) +
        ggplot2::scale_fill_manual(values = band_color, guide = "none")
    p <- p + ggplot2::geom_vline(xintercept = c(0, seqstart[-1]/totlen,
        1), linetype = "dotted", alpha = I(0.5))
    p <- p + ggplot2::geom_segment(ggplot2::aes(x = c(0, seqstart[-1]/totlen,
        1), xend = c(0, seqstart[-1]/totlen, 1), y = band_loc -
        0.7, yend = band_loc + 0.2), linewidth = 0.6, color = "black")
    merged.exons <- lapply(genes.to.label, function(x) {
        target.txns <- seg$genomeInfo$txns[GenomicRanges::mcols(seg$genomeInfo$txns)$gene_name ==
            x]
        GenomicRanges::reduce(unlist(target.txns))
    })
    chrm <- vapply(merged.exons, function(x) as.character(GenomicRanges::seqnames(x)[1]),
        character(1))
```

```
    pos <- vapply(merged.exons, function(x) min(GenomicRanges::start(x)),
        numeric(1))
    label_xpos <- (seqstart[chrm] + pos)/totlen
    label_ypos <- max(seg$bin.signals, na.rm = TRUE) * 1.1
    p <- p + ggplot2::geom_text(aes(label_xpos, label_ypos +
        0.6, label = genes.to.label), vjust = 0)
    p <- p + annotate("point", x = label_xpos, y = label_ypos,
        shape = 25, size = 6, color = "#F2AE30", fill = "#F2AE30")
    p <- p + ggplot2::geom_segment(ggplot2::aes(x = label_xpos,
        xend = label_xpos, y = band_loc, yend = label_ypos),
        color = "#F2AE30")
    p
}
<bytecode: 0x0000012a344c0f68>
<environment: namespace:sesame>
```

compareDatbaseSetOverlap

```
function (databases = NA, metric = "Jaccard")
{
    ndatabases <- length(databases)
    names <- names(databases)
    m <- matrix(0, nrow = ndatabases, ncol = ndatabases)
    colnames(m) <- names
    rownames(m) <- names
    for (i in seq(ndatabases - 1)) {
        for (j in seq(i + 1, ndatabases)) {
            message(i, " ", j, "\n")
            m[i, j] <- length(intersect(databases[[i]], databases[[j]]))/length(union(databases[[i]],
                databases[[j]]))
        }
    }
    m
}
<bytecode: 0x0000012a344e26b8>
<environment: namespace:sesame>
```

compareMouseStrainReference

```
function (betas = NULL, show_sample_names = FALSE, query_width = NULL)
{
    se <- sesameDataGet("MM285.addressStrain")$strain_snps
    cd <- as_tibble(SummarizedExperiment::colData(se))
    rd <- as_tibble(SummarizedExperiment::rowData(se))
    md <- metadata(se)
    se <- se[rd$QC != "FAIL", ]
    rd <- rd[rd$QC != "FAIL", ]
    if (!is.null(betas) && is.null(dim(betas))) {
        betas <- cbind(betas)
    }
    afs <- do.call(rbind, lapply(seq_along(rd$flipToAF), function(i) if (xor(rd$flipToAF[i],
        rd$flipForRefBias[i])) {
```

```
        1 - assay(se)[i, ]
    }
    else {
        assay(se)[i, ]
    }))
    rownames(afs) <- rd$Probe_ID
    stops <- c("white", "black")
    g <- WHeatmap(afs, cmp = CMPar(stop.points = stops, dmin = 0,
        dmax = 1), xticklabels = show_sample_names, xticklabels.n = ncol(afs),
        name = "b1")
    if (!is.null(betas)) {
        afs2 <- do.call(rbind, lapply(seq_along(rd$flipToAF),
            function(i) {
                if (xor(rd$flipToAF[i], rd$flipForRefBias[i])) {
                  1 - betas[rd$Probe_ID[i], ]
                }
                else {
                  betas[rd$Probe_ID[i], ]
                }
            }))
        g <- g + WHeatmap(afs2, RightOf("b1", width = query_width),
            cmp = CMPar(stop.points = stops, dmin = 0, dmax = 1),
            name = "b2", xticklabels = TRUE, xticklabels.n = ncol(betas))
        right <- "b2"
    }
    else {
        right <- "b1"
    }
    g <- g + WColorBarV(rd$BranchLong, RightOf(right, width = 0.03),
        cmp = CMPar(label2color = md$strain.colors), name = "bh")
    g <- g + WColorBarH(cd$strain, TopOf("b1", height = 0.03),
        cmp = CMPar(label2color = md$strain.colors), name = "st")
    g <- g + WLegendV("st", TopRightOf("bh", just = c("left",
        "top"), h.pad = 0.02), height = 0.03)
    g + WCustomize(mar.bottom = 0.15, mar.right = 0.06)
}
<bytecode: 0x0000012a344dee98>
<environment: namespace:sesame>
```

compareMouseTissueReference

```
function (betas = NULL, ref = NULL, color = "blueYellow", query_width = 0.3)
{
    .Deprecated("compareReference")
}
<bytecode: 0x0000012a34530ab8>
<environment: namespace:sesame>
```

compareReference

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

```
{
    if (is.null(stop.points)) {
        stop.points <- c("blue", "yellow")
    }
    cd <- as_tibble(colData(ref))
    rd <- as_tibble(rowData(ref))
    md <- metadata(ref)
    if (!is.null(betas) && is.null(dim(betas))) {
        betas <- cbind(betas)
    }
    g <- WHeatmap(assay(ref), cmp = CMPar(stop.points = stop.points,
        dmin = 0, dmax = 1), xticklabels = show_sample_names,
        name = "b1")
    if (!is.null(betas)) {
        g <- g + WHeatmap(betas[rd$Probe_ID, ], RightOf("b1",
            width = query_width), cmp = CMPar(stop.points = stop.points,
            dmin = 0, dmax = 1), name = "b2", xticklabels = show_sample_names,
            xticklabels.n = ncol(betas))
        right <- "b2"
    }
    else {
        right <- "b1"
    }
    g <- g + WColorBarV(rd$branch, RightOf(right, width = 0.03),
        cmp = CMPar(label2color = md$branch_color), name = "bh")
    g <- g + WColorBarH(cd$branch, TopOf("b1", height = 0.03),
        cmp = CMPar(label2color = md$branch_color), name = "ti")
    g <- g + WLegendV("ti", TopRightOf("bh", just = c("left",
        "top"), h.pad = 0.02), height = 0.02)
    g + WCustomize(mar.bottom = 0.15, mar.right = 0.06)
}
<bytecode: 0x0000012a34537098>
<environment: namespace:sesame>
```

controls

```
function (sdf, verbose = FALSE)
{
    stopifnot(is(sdf, "SigDF"))
    if (!is.null(attr(sdf, "controls"))) {
        df <- attr(sdf, "controls")
        return(data.frame(UG = df$G, UR = df$R, Type = df$type))
    }
    else if (sesameDataHas(sprintf("%s.address", sdfPlatform(sdf,
        verbose = verbose)))) {
        df <- sesameDataGet(sprintf("%s.address", sdfPlatform(sdf,
            verbose = verbose)))$controls
        if (is.null(df)) {
            return(sdf[grepl("^ctl", sdf$Probe_ID), ])
        }
        else {
            cbind(df, sdf[match(paste0("ctl_", df$Address), sdf$Probe_ID),
                c("MG", "MR", "UG", "UR")])
```

```
        }
    }
    else {
        return(sdf[grepl("^ctl", sdf$Probe_ID), ])
    }
}
<bytecode: 0x0000012a34539f50>
<environment: namespace:sesame>
```

convertProbeID

```
function (x, target_platform, source_platform = NULL, mapping = NULL,
    target_uniq = TRUE, include_new = FALSE, include_old = FALSE,
    return_mapping = FALSE)
{
    if (is.null(mapping)) {
        source_platform <- sesameData_check_platform(source_platform,
            x)
        dfs <- tibble(ID_source = x)
        dft <- tibble(ID_target = sesameDataGet(sprintf("%s.address",
            target_platform))$ordering$Probe_ID)
        if (target_platform %in% c("EPIC", "HM450", "HM27") &&
            source_platform %in% c("EPICv2", "MSA")) {
            dfs$prefix <- vapply(strsplit(dfs$ID_source, "_"),
                function(xx) xx[1], character(1))
            dft$prefix <- dft$ID_target
        }
        else if (target_platform %in% c("EPICv2", "MSA") && source_platform %in%
            c("EPIC", "HM450", "HM27")) {
            dfs$prefix <- dfs$ID_source
            dft$prefix <- vapply(strsplit(dft$ID_target, "_"),
                function(xx) xx[1], character(1))
        }
        else {
            dfs$prefix <- dfs$ID_source
            dft$prefix <- dft$ID_target
        }
        mapping <- dplyr::full_join(dfs, dft, by = "prefix")
    }
    if (target_uniq) {
        m <- dplyr::distinct(mapping, .data[["ID_target"]], .keep_all = TRUE)
        mapping <- rbind(m[!is.na(m$ID_target), ], mapping[is.na(mapping$ID_target),
            ])
    }
    if (!include_new) {
        mapping <- mapping[!is.na(mapping$ID_source), ]
    }
    if (!include_old) {
        mapping <- mapping[!is.na(mapping$ID_target), ]
    }
    if (return_mapping) {
        mapping
    }
```

```
    else {
        stats::setNames(mapping$ID_target, mapping$ID_source)
    }
}
<bytecode: 0x0000012a34545df0>
<environment: namespace:sesame>
```

create_default_mask

```
function (df)
{
    unmapped <- (is.na(df$mapAS_A) | df$mapAS_A < 35 | (!is.na(df$mapAS_B) &
        df$mapAS_B < 35))
    masks <- data.frame(Probe_ID = df$Probe_ID, nonunique = ((!unmapped) &
        (df$mapQ_A == 0 | (!is.na(df$mapQ_B) & df$mapQ_B == 0))),
        missing_target = ((!unmapped) & (is.na(df$target) | (df$target !=
            "CG")) & grepl("^cg", df$Probe_ID)))
    masks$control <- grepl("^ctl", df$Probe_ID)
    masks$design_issue <- grepl("^uk", df$Probe_ID)
    masks$unmapped <- (unmapped & masks$control != 1 & masks$design_issue !=
        1)
    masks$low_mapq <- ((!is.na(df$mapQ_A)) & (df$mapQ_A < 30 |
        (!is.na(df$mapQ_B) & df$mapQ_B < 30)))
    masks$ref_issue <- (unmapped | masks$missing_target)
    masks[c("Probe_ID", "unmapped", "missing_target", "ref_issue",
        "nonunique", "low_mapq", "control", "design_issue")]
}
<bytecode: 0x0000012a3454cc40>
<environment: namespace:sesame>
```

createDBNetwork

```
function (databases)
{
    m <- compareDatbaseSetOverlap(databases, metric = "jaccard")
    m_melted <- melt(m)
    colnames(m_melted) <- c("gene1", "gene2", "metric")
    m_melted <- m_melted[m_melted$metric != 0, ]
    nodes <- data.frame(id = colnames(m), stringsAsFactors = FALSE)
    edges <- data.frame(source = m_melted$gene1, target = m_melted$gene2,
        weight = m_melted$metric, stringsAsFactors = FALSE)
    list(nodes = nodes, edges = edges)
}
<bytecode: 0x0000012a34556ad0>
<environment: namespace:sesame>
```

createUCSCtrack

```
function (betas, output = NULL, platform = "HM450", genome = "hg38")
{
    probeInfo <- sesameData_getManifestGRanges(platform, genome)
    betas <- betas[names(probeInfo)]
```

```
    df <- data.frame(chrm = GenomicRanges::seqnames(probeInfo),
        beg = GenomicRanges::start(probeInfo) - 1, end = GenomicRanges::end(probeInfo),
        name = names(probeInfo), score = ifelse(is.na(betas),
            0, as.integer(betas * 1000)), strand = GenomicRanges::strand(probeInfo),
        thickStart = GenomicRanges::start(probeInfo) - 1, thickEnd = GenomicRanges::end(probeInfo),
        itemRgb = ifelse(is.na(betas), "0,0,0", ifelse(betas <
            0.3, "0,0,255", ifelse(betas > 0.7, "255,0,0", "50,150,0"))))
    if (is.null(output))
        df
    else write.table(df, file = output, col.names = FALSE, row.names = FALSE,
        quote = FALSE, sep = "\t")
}
<bytecode: 0x0000012a34559bf0>
<environment: namespace:sesame>
```

databases__getMeta

```
function (dbs)
{
    meta <- do.call(bind_rows, lapply(dbs, function(db) {
        m1 <- attributes(db)
        m1 <- m1[names(m1) != "names"]
        if ("meta" %in% names(m1)) {
            m1 <- c(m1[!(names(m1) %in% c("meta"))], m1$meta)
        }
        else {
            m1 <- m1[!(names(m1) %in% c("meta"))]
        }
        if (is.null(m1)) {
            data.frame(hasMeta = FALSE)
        }
        else {
            c(m1, hasMeta = TRUE)
        }
    }))
    meta[, colnames(meta) != "hasMeta"]
}
<bytecode: 0x0000012a3455fde0>
<environment: namespace:sesame>
```

dataFrame2sesameQC

```
function (df)
{
    groups <- c(.setGroup_detection(), .setGroup_numProbes(),
        .setGroup_intensity(), .setGroup_channel(), .setGroup_dyeBias(),
        .setGroup_betas())
    groups <- groups[vapply(groups, function(g) {
        if (all(names(g) %in% colnames(df))) {
            TRUE
        }
        else {
```

```
            FALSE
        }
    }, logical(1))]
    lapply(seq_len(nrow(df)), function(i) {
        new("sesameQC", group = groups, stat = df[i, ])
    })
}
<bytecode: 0x0000012a345cb4d8>
<environment: namespace:sesame>
```

dbStats

```
function (betas, databases, fun = mean, na.rm = TRUE, n_min = NULL,
    f_min = 0.1, long = FALSE)
{
    if (is(betas, "numeric")) {
        betas <- cbind(sample = betas)
    }
    if (is.character(databases)) {
        dbs <- KYCG_getDBs(databases)
    }
    else {
        dbs <- databases
    }
    stats <- do.call(cbind, lapply(dbs, function(db) {
        betas1 <- betas[db[db %in% rownames(betas)], , drop = FALSE]
        n_probes <- nrow(betas1)
        if (n_probes == 0) {
            return(rep(NA, ncol(betas)))
        }
        nacnt <- colSums(!is.na(betas1), na.rm = TRUE)
        stat1 <- apply(betas1, 2, fun, na.rm = na.rm)
        if (is.null(n_min)) {
            n_min1 <- n_probes * f_min
        }
        else {
            n_min1 <- n_min
        }
        stat1[nacnt < n_min1] <- NA
        stat1
    }))
    if (!is.null(names(dbs))) {
        colnames(stats) <- names(dbs)
    }
    else {
        colnames(stats) <- vapply(dbs, function(x) attr(x, "dbname"),
            character(1))
    }
    rownames(stats) <- colnames(betas)
    if (long) {
        stats <- melt(stats, varnames = c("query", "db"), value.name = "value")
    }
    stats
```

```
}
<bytecode: 0x0000012a345d3be0>
<environment: namespace:sesame>
```

deIdentify

```
function (path, out_path = NULL, snps = NULL, mft = NULL, randomize = FALSE)
{
    res <- suppressWarnings(readIDAT(path))
    platform <- inferPlatformFromTango(res)
    if (is.null(out_path)) {
        pfx <- sub(".idat(.gz)?$", "", path)
        if (grepl("_Grn$", pfx)) {
            out_path <- paste0(sub("_Grn$", "", pfx), "_noid_Grn.idat")
        }
        else if (grepl("_Red$", pfx)) {
            out_path <- paste0(sub("_Red$", "", pfx), "_noid_Red.idat")
        }
    }
    if (is.null(mft)) {
        mft <- sesameDataGet(paste0(platform, ".address"))$ordering
    }
    if (is.null(snps)) {
        snps <- grep("^rs", mft$Probe_ID, value = TRUE)
    }
    mft <- mft[mft$Probe_ID %in% snps, ]
    snpsTango <- na.omit(c(mft$M, mft$U))
    qt <- res$Quants
    snpsIdx <- match(snpsTango, rownames(qt))
    dt <- qt[, "Mean"]
    if (randomize) {
        snpsIdx <- snpsIdx[!is.na(snpsIdx)]
        dt[snpsIdx] <- sample(dt[snpsIdx])
    }
    else {
        dt[snpsIdx] <- 0
    }
    if (grepl("\\.gz$", path)) {
        con <- gzfile(path, "rb")
    }
    else {
        con <- file(path, "rb")
    }
    con2 <- file(out_path, "wb")
    writeBin(readBin(con, "raw", n = res$fields["Mean", "byteOffset"]),
        con2)
    writeBin(as.integer(dt), con2, size = 2, endian = "little")
    a <- readBin(con, "raw", n = res$nSNPsRead * 2)
    while (length(a <- readBin(con, "raw", n = 1)) > 0) writeBin(a,
        con2)
    close(con)
    close(con2)
}
```

```
<bytecode: 0x0000012a345d4b50>
<environment: namespace:sesame>
```

detectionPnegEcdf

```
function (sdf, return.pval = FALSE, pval.threshold = 0.05)
{
    stopifnot(is(sdf, "SigDF"))
    negctls <- negControls(sdf)
    funcG <- ecdf(negctls$G)
    funcR <- ecdf(negctls$R)
    pvals <- setNames(pmin(1 - funcR(pmax(sdf$MR, sdf$UR, na.rm = TRUE)),
        1 - funcG(pmax(sdf$MG, sdf$UG, na.rm = TRUE))), sdf$Probe_ID)
    if (return.pval) {
        return(pvals)
    }
    addMask(sdf, pvals > pval.threshold)
}
<bytecode: 0x0000012a345eb9c8>
<environment: namespace:sesame>
```

dichotomize

```
function (q, rmin = 0.15, rmax = 0.85)
{
    q[q > rmax] <- 1
    q[q < rmin] <- 0
    middle <- na.omit(which(q >= rmin & q <= rmax))
    q[middle] <- (q[middle] - rmin)/(rmax - rmin)
    q
}
<bytecode: 0x0000012a345f4290>
<environment: namespace:sesame>
```

diffRefSet

```
function (g)
{
    g <- g[apply(g, 1, function(x) min(x) != max(x)), ]
    message("Reference set is based on ", dim(g)[1], " differential probes from ",
        dim(g)[2], " cell types.")
    g
}
<bytecode: 0x0000012a345ed620>
<environment: namespace:sesame>
```

dmContrasts

```
function (smry)
{
    stopifnot(is(smry, "DMLSummary"))
```

```
    colnames(attr(smry, "model.matrix"))
}
<bytecode: 0x0000012a345f0fe8>
<environment: namespace:sesame>
```

DMGetProbeInfo

```
function (platform, genome)
{
    mft <- sesameData_getManifestGRanges(platform, genome = genome)
    mft <- mft[GenomicRanges::seqnames(mft) != "*"]
    GenomicRanges::mcols(mft) <- NULL
    GenomicRanges::strand(mft) <- "*"
    mft <- sort(mft)
    mft
}
<bytecode: 0x0000012a345ef2e8>
<environment: namespace:sesame>
```

DML

```
function (betas, fm, meta = NULL, BPPARAM = SerialParam())
{
    if (is(betas, "SummarizedExperiment")) {
        betas0 <- betas
        betas <- assay(betas0)
        meta <- colData(betas0)
    }
    stopifnot(nrow(meta) == ncol(betas))
    mm <- model.matrix(fm, meta)
    colnames(mm) <- make.names(colnames(mm))
    contr2lvs <- model_contrasts(mm, meta)
    mm_holdout <- lapply(names(contr2lvs), function(cont) {
        mm[, !(colnames(mm) %in% paste0(cont, contr2lvs[[cont]]))]
    })
    names(mm_holdout) <- names(contr2lvs)
    smry <- BiocParallel::bplapply(seq_len(nrow(betas)), function(i) {
        m0 <- lm(betas[i, ] ~ . + 0, data = as.data.frame(mm))
        sm <- summary(m0)
        sm$cov.unscaled <- NULL
        sm$residuals <- NULL
        sm$terms <- NULL
        sm$Ftest <- do.call(cbind, lapply(mm_holdout, function(mm_) {
            m1 <- lm(betas[i, ] ~ . + 0, data = as.data.frame(mm_))
            anv <- anova(m1, m0)
            c(stat = anv[["F"]][2], pval = anv[["Pr(>F)"]][2])
        }))
        sm
    }, BPPARAM = BPPARAM)
    names(smry) <- rownames(betas)
    class(smry) <- "DMLSummary"
    attr(smry, "model.matrix") <- mm
```

```
    attr(smry, "fm") <- fm
    attr(smry, "contr2lvs") <- contr2lvs
    smry
}
<bytecode: 0x0000012a345fa938>
<environment: namespace:sesame>
```

DMLpredict

```
function (betas, fm, pred = NULL, meta = NULL, BPPARAM = SerialParam())
{
    if (is(betas, "SummarizedExperiment")) {
        betas0 <- betas
        betas <- assay(betas0)
        meta <- colData(betas0)
    }
    mm <- model.matrix(fm, meta)
    colnames(mm) <- make.names(colnames(mm))
    if (is.null(pred)) {
        contr2lvs <- model_contrasts(mm, meta)
        pred <- do.call(expand.grid, contr2lvs)
    }
    mm_pred <- as.data.frame(model.matrix(fm, pred))
    colnames(mm_pred) <- make.names(colnames(mm_pred))
    stopifnot(all(colnames(mm_pred) %in% colnames(mm)))
    res <- do.call(rbind, BiocParallel::bplapply(seq_len(nrow(betas)),
        function(i) {
            m0 <- lm(betas[i, ] ~ . + 0, data = as.data.frame(mm))
            predict(m0, mm_pred)
        }, BPPARAM = BPPARAM))
    rownames(res) <- rownames(betas)
    SummarizedExperiment(res, colData = pred)
}
<bytecode: 0x0000012a3460a3e8>
<environment: namespace:sesame>
```

DMR

```
function (betas, smry, contrast, platform = NULL, probe.coords = NULL,
    dist.cutoff = NULL, seg.per.locus = 0.5)
{
    stopifnot(is(smry, "DMLSummary"))
    if (is(betas, "SummarizedExperiment")) {
        betas <- assay(betas)
    }
    if (is.null(probe.coords)) {
        if (is.null(platform)) {
            platform <- inferPlatformFromProbeIDs(rownames(betas))
        }
        genome <- sesameData_check_genome(NULL, platform)
        probe.coords <- DMGetProbeInfo(platform, genome)
    }
```

```
    message("Merging correlated CpGs ... ", appendLF = FALSE)
    segs <- dmr_merge_cpgs(betas, probe.coords, dist.cutoff,
        seg.per.locus)
    message(sprintf("Generated %d segments.", segs$id[length(segs$id)]))
    message("Combine p-values ... ")
    cf <- summaryExtractCf(smry, contrast)
    stopifnot(all(segs$cpg.ids %in% rownames(cf)))
    cf <- dmr_combine_pval(cf, segs)
    message("Done.")
    cf
}
<bytecode: 0x0000012a3462fcd8>
<environment: namespace:sesame>
```

dmr_combine_pval

```
function (cf, segs)
{
    seg.est <- as.vector(tapply(cf[segs$cpg.ids, "Estimate"],
        segs$id, function(x) mean(x, na.rm = TRUE)))
    seg.pval <- as.vector(tapply(cf[segs$cpg.ids, "Pr(>|t|)"],
        segs$id, function(x) pnorm(sum(qnorm(x))/sqrt(length(x)))))
    seg.pval.adj <- p.adjust(seg.pval, method = "BH")
    seg.ids.cf <- match(rownames(cf), segs$cpg.ids)
    s <- segs$id[seg.ids.cf]
    cf <- cbind(data.frame(Seg_ID = s, Seg_Chrm = segs$chrm[s],
        Seg_Start = segs$start[s], Seg_End = segs$end[s], Seg_Est = seg.est[s],
        Seg_Pval = seg.pval[s], Seg_Pval_adj = seg.pval.adj[s],
        Probe_ID = rownames(cf)), as.data.frame(cf))
    message(sprintf(" - %d significant segments.", sum(seg.pval <
        0.05, na.rm = TRUE)))
    message(sprintf(" - %d significant segments (after BH).",
        sum(seg.pval.adj < 0.05, na.rm = TRUE)))
    cf <- cf[order(cf$Seg_Est, cf$Seg_Chrm, cf$Seg_Start), ]
    rownames(cf) <- NULL
    cf
}
<bytecode: 0x0000012a34635de8>
<environment: namespace:sesame>
```

dmr_merge_cpgs

```
function (betas, probe.coords, dist.cutoff, seg.per.locus)
{
    betas.noNA <- betas[!apply(betas, 1, function(x) all(is.na(x))),
        ]
    cpg.ids <- intersect(rownames(betas.noNA), names(probe.coords))
    probe.coords <- GenomicRanges::sort(probe.coords[cpg.ids])
    betas.coord.srt <- betas.noNA[names(probe.coords), ]
    cpg.ids <- rownames(betas.coord.srt)
    cpg.coords <- probe.coords[cpg.ids]
    cpg.chrm <- as.vector(GenomicRanges::seqnames(cpg.coords))
```

```
    cpg.start <- GenomicRanges::start(cpg.coords)
    cpg.end <- GenomicRanges::end(cpg.coords)
    n.cpg <- length(cpg.ids)
    beta.dist <- vapply(seq_len(n.cpg - 1), function(i) sum((betas.coord.srt[i,
        ] - betas.coord.srt[i + 1, ])^2, na.rm = TRUE), 1)
    chrm.changed <- (cpg.chrm[-1] != cpg.chrm[-n.cpg])
    if (is.null(dist.cutoff)) {
        dist.cutoff <- quantile(beta.dist, 1 - seg.per.locus)
    }
    change.points <- (beta.dist > dist.cutoff | chrm.changed)
    seg.ids <- cumsum(c(TRUE, change.points))
    message("Done.")
    all.cpg.ids <- rownames(betas)
    unmapped <- all.cpg.ids[!(all.cpg.ids %in% cpg.ids)]
    cpg.ids <- c(cpg.ids, unmapped)
    cpg.chrm <- c(cpg.chrm, rep("*", length(unmapped)))
    cpg.start <- c(cpg.start, rep(NA, length(unmapped)))
    cpg.end <- c(cpg.end, rep(NA, length(unmapped)))
    seg.ids <- c(seg.ids, seq.int(from = seg.ids[length(seg.ids)] +
        1, length.out = length(unmapped)))
    seg.chrm <- as.vector(tapply(cpg.chrm, seg.ids, function(x) x[1]))
    seg.start <- as.vector(tapply(cpg.start, seg.ids, function(x) x[1]))
    seg.end <- as.vector(tapply(cpg.end, seg.ids, function(x) x[length(x)]))
    list(id = seg.ids, chrm = seg.chrm, start = seg.start, end = seg.end,
        cpg.ids = cpg.ids)
}
<bytecode: 0x0000012a34643e90>
<environment: namespace:sesame>
```

double.transform.f

```
function (f, nu1, nu2, step.size)
{
    if (f[nu1] + step.size > 1)
        return(NULL)
    if (f[nu2] - step.size < 0)
        return(NULL)
    f[nu1] <- f[nu1] + step.size
    f[nu2] <- f[nu2] - step.size
    f[1] <- 1 - sum(f[2:length(f)])
    f
}
<bytecode: 0x0000012a34657570>
<environment: namespace:sesame>
```

dyeBiasCorr

```
function (sdf, ref = NULL)
{
    stopifnot(is(sdf, "SigDF"))
    if (is.null(ref)) {
        ref <- meanIntensity(sdf)
```

```
    }
    normctl <- normControls(sdf, average = TRUE)
    fR <- ref/normctl["R"]
    fG <- ref/normctl["G"]
    sdf$MG <- sdf$MG * fG
    sdf$UG <- sdf$UG * fG
    sdf$MR <- sdf$MR * fR
    sdf$UR <- sdf$UR * fR
    sdf
}
<bytecode: 0x0000012a34650628>
<environment: namespace:sesame>
```

dyeBiasCorrMostBalanced

```
function (sdfs)
{
    normctls <- vapply(sdfs, normControls, numeric(2), average = TRUE)
    most.balanced <- which.min(abs(normctls["G", ]/normctls["R",
        ] - 1))
    ref <- mean(normctls[, most.balanced], na.rm = TRUE)
    lapply(sdfs, function(sdf) dyeBiasCorr(sdf, ref))
}
<bytecode: 0x0000012a3465cca8>
<environment: namespace:sesame>
```

dyeBiasCorrTypeINorm

```
function (sdf, mask = TRUE, verbose = FALSE)
{
    stopifnot(is(sdf, "SigDF"))
    rgdistort <- sesameQC_calcStats(sdf, "dyeBias")@stat$RGdistort
    if (is.na(rgdistort) || rgdistort > 10) {
        return(maskIG(sdf))
    }
    if (mask) {
        dG <- InfIG(sdf)
        dR <- InfIR(sdf)
    }
    else {
        dG <- InfIG(noMasked(sdf))
        dR <- InfIR(noMasked(sdf))
    }
    IG0 <- c(dG$MG, dG$UG)
    IR0 <- c(dR$MR, dR$UR)
    maxIG <- max(IG0, na.rm = TRUE)
    minIG <- min(IG0, na.rm = TRUE)
    maxIR <- max(IR0, na.rm = TRUE)
    minIR <- min(IR0, na.rm = TRUE)
    if (maxIG <= 0 || maxIR <= 0) {
        return(sdf)
    }
```

```
    IR1 <- sort(as.numeric(IR0))
    IR2 <- sort(as.vector(normalize.quantiles.use.target(matrix(IR1),
        as.vector(IG0))))
    IRmid <- (IR1 + IR2)/2
    maxIRmid <- max(IRmid)
    minIRmid <- min(IRmid)
    fitfunRed <- function(data) {
        insupp <- data <= maxIR & data >= minIR & (!is.na(data))
        oversupp <- data > maxIR & (!is.na(data))
        undersupp <- data < minIR & (!is.na(data))
        data[insupp] <- approx(x = IR1, y = IRmid, xout = data[insupp],
            ties = mean)$y
        data[oversupp] <- data[oversupp] - maxIR + maxIRmid
        data[undersupp] <- minIRmid/minIR * data[undersupp]
        data
    }
    IG1 <- sort(as.numeric(IG0))
    IG2 <- sort(as.vector(normalize.quantiles.use.target(matrix(IG1),
        as.vector(IR0))))
    IGmid <- (IG1 + IG2)/2
    maxIGmid <- max(IGmid)
    minIGmid <- min(IGmid)
    fitfunGrn <- function(data) {
        insupp <- data <= maxIG & data >= minIG & (!is.na(data))
        oversupp <- data > maxIG & (!is.na(data))
        undersupp <- data < minIG & (!is.na(data))
        data[insupp] <- approx(x = IG1, y = IGmid, xout = data[insupp],
            ties = mean)$y
        data[oversupp] <- data[oversupp] - maxIG + maxIGmid
        data[undersupp] <- minIGmid/minIG * data[undersupp]
        data
    }
    sdf$MR <- fitfunRed(sdf$MR)
    sdf$UR <- fitfunRed(sdf$UR)
    sdf$MG <- fitfunGrn(sdf$MG)
    sdf$UG <- fitfunGrn(sdf$UG)
    sdf
}
<bytecode: 0x0000012a346641c0>
<environment: namespace:sesame>
```

dyeBiasL

```
function (sdf, ref = NULL)
{
    stopifnot(is(sdf, "SigDF"))
    if (is.null(ref)) {
        ref <- meanIntensity(sdf)
    }
    muR <- signalMU(InfIR(sdf))
    fR <- ref/median(muR$M, muR$U, na.rm = TRUE)
    muG <- signalMU(InfIG(sdf))
    fG <- ref/median(muG$M, muG$U, na.rm = TRUE)
```

```
    sdf$MG <- sdf$MG * fG
    sdf$UG <- sdf$UG * fG
    sdf$MR <- sdf$MR * fR
    sdf$UR <- sdf$UR * fR
    sdf
}
<bytecode: 0x0000012a3468fd20>
<environment: namespace:sesame>
```

dyeBiasNL

```
function (sdf, mask = TRUE, verbose = FALSE)
{
    stopifnot(is(sdf, "SigDF"))
    rgdistort <- sesameQC_calcStats(sdf, "dyeBias")@stat$RGdistort
    if (is.na(rgdistort) || rgdistort > 10) {
        return(maskIG(sdf))
    }
    if (mask) {
        dG <- InfIG(sdf)
        dR <- InfIR(sdf)
    }
    else {
        dG <- InfIG(noMasked(sdf))
        dR <- InfIR(noMasked(sdf))
    }
    IG0 <- c(dG$MG, dG$UG)
    IR0 <- c(dR$MR, dR$UR)
    maxIG <- max(IG0, na.rm = TRUE)
    minIG <- min(IG0, na.rm = TRUE)
    maxIR <- max(IR0, na.rm = TRUE)
    minIR <- min(IR0, na.rm = TRUE)
    if (maxIG <= 0 || maxIR <= 0) {
        return(sdf)
    }
    IR1 <- sort(as.numeric(IR0))
    IR2 <- sort(as.vector(normalize.quantiles.use.target(matrix(IR1),
        as.vector(IG0))))
    IRmid <- (IR1 + IR2)/2
    maxIRmid <- max(IRmid)
    minIRmid <- min(IRmid)
    fitfunRed <- function(data) {
        insupp <- data <= maxIR & data >= minIR & (!is.na(data))
        oversupp <- data > maxIR & (!is.na(data))
        undersupp <- data < minIR & (!is.na(data))
        data[insupp] <- approx(x = IR1, y = IRmid, xout = data[insupp],
            ties = mean)$y
        data[oversupp] <- data[oversupp] - maxIR + maxIRmid
        data[undersupp] <- minIRmid/minIR * data[undersupp]
        data
    }
    IG1 <- sort(as.numeric(IG0))
    IG2 <- sort(as.vector(normalize.quantiles.use.target(matrix(IG1),
```

```
        as.vector(IR0))))
    IGmid <- (IG1 + IG2)/2
    maxIGmid <- max(IGmid)
    minIGmid <- min(IGmid)
    fitfunGrn <- function(data) {
        insupp <- data <= maxIG & data >= minIG & (!is.na(data))
        oversupp <- data > maxIG & (!is.na(data))
        undersupp <- data < minIG & (!is.na(data))
        data[insupp] <- approx(x = IG1, y = IGmid, xout = data[insupp],
            ties = mean)$y
        data[oversupp] <- data[oversupp] - maxIG + maxIGmid
        data[undersupp] <- minIGmid/minIG * data[undersupp]
        data
    }
    sdf$MR <- fitfunRed(sdf$MR)
    sdf$UR <- fitfunRed(sdf$UR)
    sdf$MG <- fitfunGrn(sdf$MG)
    sdf$UG <- fitfunGrn(sdf$UG)
    sdf
}
<bytecode: 0x0000012a34693ab8>
<environment: namespace:sesame>
```

ELBAR

```
function (sdf, return.pval = FALSE, pval.threshold = 0.05, margin = 0.05,
    capMU = 3000, delta.beta = 0.2, n.windows = 500)
{
    df <- rbind(signalMU(sdf, mask = FALSE, MU = TRUE), signalMU_oo(sdf,
        MU = TRUE))
    df$beta <- df$M/(df$M + df$U)
    df <- df[order(df$MU), ]
    df <- df[!is.na(df$MU) & !is.nan(df$beta), ]
    thres <- 2^(seq(log2(max(1, df$MU[1] - 1)), log2(df$MU[nrow(df)] +
        1), length.out = n.windows))
    rngs <- vapply(thres, function(t1) {
        bt <- df$beta[df$MU > t1][seq_len(500)]
        quantile(bt, c(margin, 1 - margin), na.rm = TRUE)
    }, numeric(2))
    if (rngs[2, 1] - rngs[1, 1] > 0.5) {
        warning(sprintf("Background signal is dichotomous. \n%s\n",
            "Consider running noob+dyeBiasNL (BD) before this step."))
        maxMU <- df$MU[10]
    }
    else {
        t1 <- thres[rngs[1, ] - rngs[1, 1] < -delta.beta | rngs[2,
            ] - rngs[2, 1] > +delta.beta][1]
        maxMU <- df$MU[df$MU > t1][500]
    }
    maxMU <- min(maxMU, capMU, na.rm = TRUE)
    df1 <- df[df$MU <= maxMU, ]
    bgs <- pmax(df1$M, df1$U, na.rm = TRUE)
    rngs_bg <- quantile(bgs, c(0.1, 0.9), na.rm = TRUE)
```

```
    if (is.na(rngs_bg[1]) || rngs_bg[2] - rngs_bg[1] < 10) {
        warning(sprintf("Background signal lacks variation. \n%s\n%s\n",
            "The detection masking may not be stringent enough.",
            "Consider running noob+dyeBiasNL (BD) before this step."))
    }
    df <- signalMU(sdf, mask = FALSE)
    pvals <- setNames(1 - ecdf(bgs)(pmax(df$M, df$U)), df$Probe_ID)
    pvals[is.na(pvals)] <- 1
    if (return.pval) {
        return(pvals)
    }
    addMask(sdf, pvals > pval.threshold)
}
<bytecode: 0x0000012a346a79d0>
<environment: namespace:sesame>
```

errFunc

```
function (f, g, q)
{
    gamma <- q - g %*% f[2:length(f)]
    sum(ifelse(gamma < f[1]/2, abs(gamma), abs(gamma - f[1])),
        na.rm = TRUE)
}
<bytecode: 0x0000012a346b8bf0>
<environment: namespace:sesame>
```

estimateCellComposition

```
function (g, q, refine = TRUE, dichotomize = FALSE, ...)
{
    if (dichotomize) {
        q <- dichotomize(q)
    }
    res <- .optimizeCellComposition(g, q, step.max = 1, ...)
    res <- .optimizeCellComposition(g, q, frac0 = res$frac.min,
        step.max = 0.05, ...)
    list(frac = setNames(res$frac.min, c("unknown", colnames(g))),
        err = res$errmin, g0 = getg0(res$frac.min, g, q))
}
<bytecode: 0x0000012a346b4690>
<environment: namespace:sesame>
```

estimateLeukocyte

```
function (betas.tissue, betas.leuko = NULL, betas.tumor = NULL,
    platform = c("EPIC", "HM450", "HM27"))
{
    platform <- match.arg(platform)
    if (!is.matrix(betas.tissue)) {
        betas.tissue <- as.matrix(betas.tissue)
    }
```

```
    if (is.null(betas.leuko)) {
        betas.leuko <- sesameDataGet("leukocyte.betas")[[platform]]
    }
    if (!is.matrix(betas.leuko)) {
        betas.leuko <- as.matrix(betas.leuko)
    }
    ave.leuko <- rowMeans(betas.leuko, na.rm = TRUE)
    ave.tissue <- rowMeans(betas.tissue, na.rm = TRUE)
    probes <- intersect(names(ave.leuko), names(ave.tissue))
    ave.leuko <- ave.leuko[probes]
    ave.tissue <- ave.tissue[probes]
    if (toupper(platform) %in% c("HM450", "EPIC")) {
        nprobes <- 1000
    }
    else if (toupper(platform) == "HM27") {
        nprobes <- 100
    }
    tt <- sort(ave.leuko - ave.tissue)
    probes.leuko.lo <- names(head(tt, n = nprobes))
    probes.leuko.hi <- names(tail(tt, n = nprobes))
    if (!is.null(betas.tumor)) {
        if (!is.matrix(betas.tumor))
            betas.tumor <- as.matrix(betas.tumor)
        betas.tissue <- betas.tumor
    }
    if (dim(betas.tissue)[2] >= 10) {
        t.hi <- apply(betas.tissue[probes.leuko.hi, ], 1, min,
            na.rm = TRUE)
        t.lo <- apply(betas.tissue[probes.leuko.lo, ], 1, max,
            na.rm = TRUE)
    }
    else {
        t.hi <- rep(0, length(probes.leuko.hi))
        t.lo <- rep(1, length(probes.leuko.lo))
    }
    l.hi <- as.numeric(as.matrix(ave.leuko[probes.leuko.hi]))
    l.lo <- as.numeric(as.matrix(ave.leuko[probes.leuko.lo]))
    leuko.estimate <- vapply(seq_len(ncol(betas.tissue)), function(i) {
        s.hi <- betas.tissue[probes.leuko.hi, i]
        s.lo <- betas.tissue[probes.leuko.lo, i]
        p <- c((s.hi - t.hi)/(l.hi - t.hi), (s.lo - t.lo)/(l.lo -
            t.lo))
        if (sum(!is.na(p)) < 10)
            return(NA)
        dd <- density(na.omit(p))
        dd$x[which.max(dd$y)]
    }, numeric(1))
    names(leuko.estimate) <- colnames(betas.tissue)
    leuko.estimate
}
<bytecode: 0x0000012a346bf908>
<environment: namespace:sesame>
```

exonToCDS

```
function (exons, cdsStart, cdsEnd)
{
    if (is.na(cdsStart) || is.na(cdsEnd) || cdsEnd <= cdsStart) {
        return(NULL)
    }
    cds <- exons[((GenomicRanges::start(exons) < cdsEnd) & (GenomicRanges::end(exons) >
        cdsStart))]
    GenomicRanges::start(cds) <- pmax(GenomicRanges::start(cds),
        cdsStart)
    GenomicRanges::end(cds) <- pmin(GenomicRanges::end(cds),
        cdsEnd)
    cds
}
<bytecode: 0x0000012a346c8458>
<environment: namespace:sesame>
```

expand_url

```
function (url, base = "https://github.com/zhou-lab/InfiniumAnnotationV1/raw/main/")
{
    if (!any(endsWith(url, c("rds", "tsv.gz")))) {
        url <- sprintf("%s.tsv.gz", url)
    }
    if (!grepl("http", url)) {
        if (!grepl("/", url)) {
            url <- sprintf("Anno/%s/%s", strsplit(url, "\\.")[[1]][1],
                url)
        }
        url <- sprintf("%s/%s", base, url)
    }
    url
}
<bytecode: 0x0000012a346fbdb8>
<environment: namespace:sesame>
```

formatVCF

```
function (sdf, anno, vcf = NULL, genome = "hg38", verbose = FALSE)
{
    platform <- sdfPlatform(sdf, verbose = verbose)
    betas <- getBetas(sdf)[anno$Probe_ID]
    af <- getAFTypeIbySumAlleles(sdf, known.ccs.only = FALSE)
    vafs <- ifelse(anno$U == "ALT", 1 - betas, betas)
    vafs <- ifelse(anno$U == "REF_InfI", af[anno$Probe_ID], vafs)
    gts <- lapply(vafs, genotyper)
    GT <- vapply(gts, function(g) g$GT, character(1))
    GS <- vapply(gts, function(g) g$GS, numeric(1))
    anno$REF[anno$REF == "ACT"] <- "H"
    anno$REF[anno$REF == "AGT"] <- "D"
    anno$ALT[anno$ALT == "ACT"] <- "H"
    anno$ALT[anno$ALT == "AGT"] <- "D"
```

```
    vcflines <- cbind(anno$chrm, anno$end, ".", anno$REF, anno$ALT,
        GS, ifelse(GS > 20, "PASS", "FAIL"), paste0(sprintf("PVF=%1.3f;GT=%s;GS=%d;Probe_ID=%s",
            vafs, GT, GS, anno$Probe_ID), ifelse(is.na(anno$rs),
            "", paste0(";rs_ID=", anno$rs))))
    header <- vcf_header(genome)
    out <- data.frame(vcflines)
    colnames(out) <- c("#CHROM", "POS", "ID", "REF", "ALT", "QUAL",
        "FILTER", "INFO")
    out <- out[order(out[["#CHROM"]], as.numeric(out[["POS"]])),
        ]
    if (is.null(vcf)) {
        return(out)
    }
    else {
        writeLines(header, vcf)
        write.table(out, file = vcf, append = TRUE, sep = "\t",
            row.names = FALSE, col.names = FALSE, quote = FALSE)
    }
}
<bytecode: 0x0000012a34703030>
<environment: namespace:sesame>
```

genotyper

```
function (x, model_background = 0.1, model_nbeads = 40)
{
    GL <- vapply(c(model_background, 0.5, 1 - model_background),
        function(af) {
            dbinom(round(x * model_nbeads), size = model_nbeads,
                prob = af)
        }, numeric(1))
    ind <- which.max(GL)
    GT <- c("0/0", "0/1", "1/1")[ind]
    GS <- floor(-log10(1 - GL[ind]/sum(GL)) * 10)
    list(GT = GT, GS = GS)
}
<bytecode: 0x0000012a3470ea88>
<environment: namespace:sesame>
```

getAFs

```
function (sdf, ...)
{
    betas <- getBetas(sdf, ...)
    c(betas[startsWith(names(betas), "rs")], getAFTypeIbySumAlleles(sdf))
}
<bytecode: 0x0000012a3470fa68>
<environment: namespace:sesame>
```

getAFTypeIbySumAlleles

```
function (sdf, known.ccs.only = TRUE)
{
    stopifnot(all(c("MG", "UG", "MR", "UR") %in% colnames(sdf)))
    dG <- InfIG(sdf)
    dR <- InfIR(sdf)
    af <- c(setNames(pmax(dG$MR + dG$UR, 1)/pmax(dG$MR + dG$UR +
        dG$MG + dG$UG, 2), dG$Probe_ID), setNames(pmax(dR$MG +
        dR$UG, 1)/pmax(dR$MR + dR$UR + dR$MG + dR$UG, 2), dR$Probe_ID))
    if (known.ccs.only) {
        af <- af[intersect(names(af), sesameDataGet("ethnicity.inference")$ccs.probes)]
    }
    af[order(names(af))]
}
<bytecode: 0x0000012a34715bb0>
<environment: namespace:sesame>
```

getBetas

```
function (sdf, mask = TRUE, sum.TypeI = FALSE, collapseToPfx = FALSE,
    collapseMethod = c("mean", "minPval"))
{
    stopifnot(all(c("MG", "UG", "MR", "UR") %in% colnames(sdf)))
    collapseMethod <- match.arg(collapseMethod)
    if (collapseToPfx && collapseMethod == "minPval") {
        sdf <- SDFcollapseToPfx(sdf)
    }
    if (sum.TypeI) {
        d1 <- InfI(sdf)
        d2 <- InfII(sdf)
        betas <- c(setNames(pmax(d1$MG + d1$MR, 1)/pmax(d1$MG +
            d1$MR + d1$UG + d1$UR, 2), d1$Probe_ID), setNames(pmax(d2$UG,
            1)/pmax(d2$UG + d2$UR, 2), d2$Probe_ID))
    }
    else {
        dG <- InfIG(sdf)
        dR <- InfIR(sdf)
        d2 <- InfII(sdf)
        betas <- c(setNames(pmax(dG$MG, 1)/pmax(dG$MG + dG$UG,
            2), dG$Probe_ID), setNames(pmax(dR$MR, 1)/pmax(dR$MR +
            dR$UR, 2), dR$Probe_ID), setNames(pmax(d2$UG, 1)/pmax(d2$UG +
            d2$UR, 2), d2$Probe_ID))
    }
    betas <- setNames(betas[match(sdf$Probe_ID, names(betas))],
        sdf$Probe_ID)
    if (mask) {
        betas[sdf$mask] <- NA
    }
    if (collapseToPfx && collapseMethod == "mean") {
        betas <- betasCollapseToPfx(betas)
    }
    betas
```

```
}
<bytecode: 0x0000012a34719620>
<environment: namespace:sesame>
```

getBinCoordinates

```
function (seqLength, gapInfo, tilewidth = 50000, probeCoords)
{
    tiles <- sort(GenomicRanges::tileGenome(seqLength, tilewidth = tilewidth,
        cut.last.tile.in.chrom = TRUE))
    tiles <- sort(c(GenomicRanges::setdiff(tiles[seq(1, length(tiles),
        2)], gapInfo), GenomicRanges::setdiff(tiles[seq(2, length(tiles),
        2)], gapInfo)))
    GenomicRanges::values(tiles)$probes <- GenomicRanges::countOverlaps(tiles,
        probeCoords)
    bin.coords <- do.call(rbind, lapply(split(tiles, as.vector(GenomicRanges::seqnames(tiles))),
        function(chrom.tiles) leftRightMerge1(GenomicRanges::as.data.frame(GenomicRanges::sort(chrom.ti
    bin.coords <- GenomicRanges::sort(GenomicRanges::GRanges(seqnames = bin.coords$seqnames,
        IRanges::IRanges(start = bin.coords$start, end = bin.coords$end),
        seqinfo = GenomicRanges::seqinfo(tiles)))
    chr.cnts <- table(as.vector(GenomicRanges::seqnames(bin.coords)))
    chr.names <- as.vector(GenomicRanges::seqnames(bin.coords))
    names(bin.coords) <- paste(as.vector(GenomicRanges::seqnames(bin.coords)),
        formatC(unlist(lapply(GenomicRanges::seqnames(bin.coords)@lengths,
            seq_len)), width = nchar(max(chr.cnts)), format = "d",
            flag = "0"), sep = "-")
    bin.coords
}
<bytecode: 0x0000012a347289d0>
<environment: namespace:sesame>
```

getg0

```
function (f, g, q)
{
    gamma <- q - g %*% f[2:length(f)]
    ifelse(gamma < f[1]/2, 0, 1)
}
<bytecode: 0x0000012a34730428>
<environment: namespace:sesame>
```

getMask

```
function (platform = "EPICv2", mask_names = "recommended")
{
    stopifnot(is.character(platform))
    res <- lapply(mask_names, function(mask_name) {
        if (mask_name == "recommended") {
            res <- KYCG_getDBs(sprintf("%s.Mask", platform),
                recommendedMaskNames()[[platform]], silent = TRUE,
                ignore.case = TRUE)
        }
```

```
        else {
            res <- KYCG_getDBs(sprintf("%s.Mask", platform),
                mask_name, silent = TRUE, ignore.case = TRUE)
        }
        if (!is.null(res)) {
            res <- do.call(c, res)
        }
        res
    })
    if (!is.null(res)) {
        res <- unique(do.call(c, res))
    }
    res
}
<bytecode: 0x0000012a34734368>
<environment: namespace:sesame>
```

getRefSet

```
function (cells = NULL, platform = c("EPIC", "HM450"))
{
    platform <- match.arg(platform)
    if (is.null(cells)) {
        cells <- c("CD4T", "CD19B", "CD56NK", "CD14Monocytes",
            "granulocytes")
    }
    refdata <- sesameDataGet("ref.methylation")[cells]
    probes <- Reduce(intersect, lapply(refdata, names))
    g <- do.call(cbind, lapply(refdata, function(x) x[probes]))
    g <- cleanRefSet(g, platform)
    message("Reference set is based on ", dim(g)[1], " probes from ",
        dim(g)[2], " cell types.")
    g
}
<bytecode: 0x0000012a3479cfd8>
<environment: namespace:sesame>
```

getSignatureU

```
function (betas, grouping, u_max = 0.2, m_min = 0.7, max_na_in = 0,
    max_na_out = 0)
{
    groups <- unique(grouping)
    is_na <- is.na(betas)
    sigs <- lapply(groups, function(g) {
        m1 <- rowMeans(betas[, grouping == g], na.rm = TRUE) <
            u_max
        m2 <- rowMeans(betas[, grouping != g], na.rm = TRUE) >
            m_min
        ps1 <- rowSums(is_na[, grouping == g]) <= max_na_in
        ps2 <- rowSums(is_na[, grouping != g]) <= max_na_out
        names(which(m1 & m2 & ps1 & ps2))
```

```
    })
    names(sigs) <- groups
    sigs
}
<bytecode: 0x0000012a347a1960>
<environment: namespace:sesame>
```

getSignatureUTop

```
function (betas, grouping, n = 100, max_na_in = 0, max_na_out = 0)
{
    groups <- unique(grouping)
    is_na <- is.na(betas)
    sigs <- lapply(groups, function(g) {
        mean1 <- rowMeans(betas[, grouping == g], na.rm = TRUE)
        mean0 <- rowMeans(betas[, grouping != g], na.rm = TRUE)
        ps1 <- rowSums(is_na[, grouping == g]) <= max_na_in
        ps2 <- rowSums(is_na[, grouping != g] <= max_na_out)
        head(names(sort((mean1 - mean0)[ps1 & ps2])), n = n)
    })
    names(sigs) <- groups
    sigs
}
<bytecode: 0x0000012a3479ee28>
<environment: namespace:sesame>
```

guess_chrmorder

```
function (chrms)
{
    chrms1 <- chrms[!(chrms %in% c("chrX", "chrY", "chrM"))]
    paste0("chr", c(as.character(seq_len(max(as.integer(str_replace(sort(unique(chrms1)),
        "chr", ""))), na.rm = TRUE))), c("X", "Y", "M")))
}
<bytecode: 0x0000012a347ae380>
<environment: namespace:sesame>
```

guess_dbnames

```
function (nms, platform = NULL, allow_multi = FALSE, type = NULL,
    silent = FALSE, ignore.case = FALSE)
{
    gps <- KYCG_listDBGroups(type = type)
    nms <- do.call(c, lapply(nms, function(nm) {
        if (nm %in% gps$Title) {
            return(nm)
        }
        else if (length(grep(nm, gps$Title, ignore.case = ignore.case)) >=
            1) {
            ret <- grep(nm, gps$Title, value = TRUE, ignore.case = ignore.case)
            if (!allow_multi) {
                ret <- ret[1]
```

```
            }
            return(ret)
        }
        else if (length(grep(nm, gps$Title, ignore.case = ignore.case)) ==
            0) {
            res <- gps$Title[apply(do.call(cbind, lapply(strsplit(nm,
                "\\.")[[1]], function(q1) grepl(q1, gps$Title,
                ignore.case = ignore.case))), 1, all)]
            if (length(res) == 1) {
                return(res[1])
            }
        }
        return(nm)
    }))
    if (!is.null(platform)) {
        nms <- grep(platform, nms, value = TRUE)
    }
    if (!silent) {
        message("Selected the following database groups:")
        invisible(lapply(seq_along(nms), function(i) {
            message(sprintf("%d. %s", i, nms[i]))
        }))
    }
    nms
}
<bytecode: 0x0000012a347abb00>
<environment: namespace:sesame>
```

imputeBetas

```
function (betas, platform = NULL, BPPARAM = SerialParam(), celltype = NULL,
    sd_max = 999)
{
    if (is.matrix(betas)) {
        betas <- do.call(cbind, bplapply(seq_len(ncol(betas)),
            function(i) {
                imputeBetas(betas[, i], platform = NULL, celltype = celltype,
                  sd_max = sd_max)
            }, BPPARAM = BPPARAM))
        colnames(betas) <- colnames(betas)
        return(betas)
    }
    platform <- sesameData_check_platform(platform, names(betas))
    df <- sesameDataGet(sprintf("%s.imputationDefault", platform))
    d2q <- match(names(betas), df$Probe_ID)
    celltype <- names(which.max(vapply(df$data, function(x) cor(betas,
        x$median[d2q], use = "na.or.complete"), numeric(1))))
    if (is.null(celltype)) {
        celltype <- "Blood"
    }
    idx <- is.na(betas)
    mn <- df$data[[celltype]]$median[d2q][idx]
    sd <- df$data[[celltype]]$sd[d2q][idx]
```

```
    mn[sd > sd_max] <- NA
    betas[idx] <- mn
    betas
}
<bytecode: 0x0000012a347be0f0>
<environment: namespace:sesame>
```

imputeBetasByGenomicNeighbors

```
function (betas, platform = NULL, BPPARAM = SerialParam(), max_neighbors = 3,
    max_dist = 10000)
{
    platform <- sesameData_check_platform(platform, names(betas))
    mft <- sesameData_getManifestGRanges(platform)
    mft_missing <- mft[names(mft) %in% names(which(is.na(betas)))]
    mft_nonmiss <- mft[names(which(!is.na(betas)))]
    index <- findOverlaps(resize(mft_missing, max_dist), mft_nonmiss)
    gm <- mft_missing[queryHits(index)]
    gn <- mft_nonmiss[subjectHits(index)]
    df <- tibble(cg = names(gm), beg_m = start(gm), end_m = end(gm),
        cg_n = names(gn), beg_n = start(gn), end_n = end(gn))
    df$d1 <- df$beg_m - df$end_n - 1
    df$d2 <- df$beg_n - df$end_m - 1
    df$betas <- betas[df$cg_n]
    df$dist <- pmax(df$d1, df$d2)
    df <- summarize(slice_min(group_by(df, .data[["cg"]]), n = max_neighbors,
        order_by = .data[["dist"]]), mbetas = mean(.data[["betas"]]))
    betas[df$cg] <- df$mbetas
    betas
}
<bytecode: 0x0000012a347c67c0>
<environment: namespace:sesame>
```

imputeBetasMatrixByMean

```
function (mx, axis = 1)
{
    stopifnot(is.matrix(mx))
    if (axis == 1) {
        t(apply(mx, 1, function(x) {
            x[is.na(x)] <- mean(x, na.rm = TRUE)
            x
        }))
    }
    else if (axis == 2) {
        apply(mx, 2, function(x) {
            x[is.na(x)] <- mean(x, na.rm = TRUE)
            x
        })
    }
    else {
        stop("Invalid axis. Use 1 for columns or 2 for rows.")
```

```
    }
}
<bytecode: 0x0000012a347c8418>
<environment: namespace:sesame>
```

inferEthnicity

```
function (sdf, verbose = FALSE)
{
    .Deprecated("Please use CytoMethIC::cmi_classify.")
}
<bytecode: 0x0000012a347ccb38>
<environment: namespace:sesame>
```

inferInfiniumIChannel

```
function (sdf, switch_failed = FALSE, mask_failed = FALSE, verbose = FALSE,
    summary = FALSE)
{
    inf1_idx <- which(sdf$col != "2")
    sdf1 <- sdf[inf1_idx, ]
    red_max <- pmax(sdf1$MR, sdf1$UR)
    grn_max <- pmax(sdf1$MG, sdf1$UG)
    new_col <- factor(ifelse(red_max > grn_max, "R", "G"), levels = c("G",
        "R", "2"))
    d1R <- sdf1[new_col == "R", ]
    d1G <- sdf1[new_col == "G", ]
    bg_max <- quantile(c(d1R$MG, d1R$UG, d1G$MR, d1G$UR), 0.95,
        na.rm = TRUE)
    idx <- (is.na(red_max) | is.na(grn_max) | pmax(red_max, grn_max) <
        bg_max)
    if (!switch_failed) {
        new_col[idx] <- sdf1$col[idx]
    }
    if (mask_failed) {
        sdf$mask[inf1_idx[idx]] <- TRUE
    }
    sdf$col[inf1_idx] <- factor(new_col, levels = c("G", "R",
        "2"))
    smry <- c(R2R = sum(sdf1$col == "R" & new_col == "R", na.rm = TRUE),
        G2G = sum(sdf1$col == "G" & new_col == "G", na.rm = TRUE),
        R2G = sum(sdf1$col == "R" & new_col == "G", na.rm = TRUE),
        G2R = sum(sdf1$col == "G" & new_col == "R", na.rm = TRUE))
    if (summary) {
        return(smry)
    }
    sdfMsg(sdf, verbose, "%s: R>R:%d;G>G:%d;R>G:%d;G>R:%d", "Infinium-I color channel reset",
        smry["R2R"], smry["G2G"], smry["R2G"], smry["G2R"])
}
<bytecode: 0x0000012a347d5048>
<environment: namespace:sesame>
```

inferPlatformFromTango

```
function (res)
{
    sig <- sesameDataGet("idatSignature")
    cnts <- vapply(sig, function(x) sum(x %in% rownames(res$Quants)),
        integer(1))
    if (max(cnts) < min(vapply(sig, length, numeric(1)))) {
        return(NULL)
    }
    names(which.max(cnts))
}
<bytecode: 0x0000012a34801388>
<environment: namespace:sesame>
```

inferSex

```
function (betas, platform = NULL)
{
    hypoMALE <- c("cg21983484", "cg23696472", "cg11673471", "cg01742836",
        "cg13574945", "cg08059778", "cg24186901", "cg26023405",
        "cg15977272", "cg13023833", "cg20766178", "cg20455959",
        "cg26584339", "cg13130271", "cg13244998", "cg05872808",
        "cg21290550", "cg05806018", "cg07861180", "cg20015269",
        "cg12576145", "cg10991108", "cg02333283", "cg16357225",
        "cg25206026", "cg20749341", "cg03773146", "cg04872051",
        "cg16590821", "cg09520212", "cg22221554", "cg11152253",
        "cg23429746", "cg00813156", "cg25132467", "cg16221895",
        "cg09307104", "cg15165114", "cg18998000", "cg00723973",
        "cg06041068", "cg10860619", "cg09514431", "cg07912337",
        "cg03334316", "cg17399684", "cg05534333", "cg23493872",
        "cg12413138", "cg05374090", "cg27501007", "cg08855111",
        "cg21159768", "cg16488754", "cg12075609", "cg07446674",
        "cg01342901", "cg02869694", "cg12277627", "cg19992190",
        "cg10717149", "cg14191108", "cg01869765", "cg26505478",
        "cg23685102", "cg02195366", "cg06334238", "cg02615131",
        "cg15565409", "cg15693668", "cg03505772", "cg00845806",
        "cg26439324", "cg12935118", "cg18932686", "cg24264679",
        "cg08782677", "cg13649400", "cg06779802", "cg23554546",
        "cg23951868", "cg00337921", "cg08479532", "cg00114625",
        "cg03391801", "cg22776211", "cg07674503", "cg22452543",
        "cg18140045", "cg15450782", "cg07674075", "cg06510592",
        "cg21137943", "cg24479484", "cg27501723", "cg20439892",
        "cg18107314", "cg08405463", "cg09146364", "cg16894263",
        "cg44822048_BC11", "cg48153389_BC11", "cg48114705_BC11",
        "cg48140091_BC11", "cg47832419_BC11", "cg47450117_BC11",
        "cg47728613_BC11", "cg47583295_TC11", "cg47476627_BC11",
        "cg48109634_BC11", "cg47564226_TC11", "cg47844107_BC11",
        "cg47425903_TC11", "cg47742805_BC21", "cg47855973_BC11",
        "cg47743423_BC11", "cg47906498_TC11", "cg47556267_BC11",
        "cg47744057_TC21", "cg48176284_BC11", "cg48121188_BC11",
        "cg48065865_BC11", "cg47748343_TC21", "cg47424030_BC21",
        "cg47744023_TC11", "cg47440985_TC11", "cg47583387_BC11",
```

```
    "cg47725474_TC21", "cg48024686_TC11", "cg47920249_BC21",
    "cg48114704_TC11", "cg48148849_BC21", "cg47742981_BC11",
    "cg47743136_BC11", "cg48049840_TC11", "cg48111009_TC21",
    "cg48176352_TC11", "cg47655961_BC11", "cg47856861_BC21",
    "cg47826283_TC11", "cg47901233_TC11", "cg48051845_BC11",
    "cg47555978_BC21", "cg47634755_BC11", "cg48147947_TC11",
    "cg47503480_BC11", "cg47740318_BC11", "cg48071477_TC11",
    "cg47643035_TC21", "cg47868567_BC11", "cg47655979_TC21",
    "cg47725912_BC11", "cg47564279_BC11", "cg48016415_TC11",
    "cg47656013_TC21", "cg47873187_TC21", "cg47438865_TC11",
    "cg47906673_BC11", "cg47874829_BC11", "cg47734934_BC21",
    "cg48130287_BC21", "cg47625820_BC21", "cg47505633_TC11",
    "cg48023062_TC21", "cg47744459_TC11", "cg47730002_BC11",
    "cg47663054_BC11", "cg47742655_BC11", "cg48107157_BC11",
    "cg48148824_TC21", "cg47634666_BC11", "cg47832434_TC11",
    "cg48057717_TC21", "cg48106464_BC11", "cg47748082_TC21",
    "cg47897499_BC21", "cg47889728_TC21", "cg47938210_TC21",
    "cg48176806_TC11", "cg47740347_BC11", "cg48021685_BC21",
    "cg47612856_BC11", "cg48139201_BC21", "cg48176811_BC11",
    "cg47741292_TC21", "cg47905796_TC21", "cg47643008_TC21",
    "cg47743984_BC11", "cg47795637_BC21", "cg47667056_TC11",
    "cg48159183_BC21", "cg48164072_TC11", "cg48177792_TC21",
    "cg47743999_TC11", "cg47471551_TC21", "cg47740813_BC21",
    "cg48157924_BC21", "cg47737568_BC11", "cg47724667_BC21",
    "cg47618975_BC11")
hyperMALE <- c("cg26359388", "cg02540440", "cg11049634",
    "cg22874828", "cg09182733", "cg01123965", "cg15822015",
    "cg05130312", "cg17072671", "cg22655232", "cg05695959",
    "cg21010298", "cg06143713", "cg22759686", "cg11143827",
    "cg04303560", "cg11717280", "cg14372935", "cg05533223",
    "cg16405492", "cg15765801", "cg08156775", "cg24183173",
    "cg21797452", "cg03161453", "cg10474871", "cg11516614",
    "cg18813691", "cg08614574", "cg08456555", "cg16440909",
    "cg13326840", "cg16822540", "cg03801901", "cg09039264",
    "cg01383599", "cg14931238", "cg04071644", "cg22208280",
    "cg05559023", "cg23317607", "cg26327984", "cg07801607",
    "cg06870560", "cg24156613", "cg04101819", "cg07422795",
    "cg14261068", "cg12622895", "cg09192294", "cg26695278",
    "cg12653510", "cg03554089", "cg11166197", "cg04032096",
    "cg25047306", "cg07818713", "cg21258987", "cg07981033",
    "cg14492530", "cg18157587", "cg12030638", "cg17498624",
    "cg01816615", "cg08723064", "cg05193067", "cg27167763",
    "cg15521097", "cg25456959", "cg16576300", "cg07318999",
    "cg22417678", "cg22671388", "cg23644934", "cg00267352",
    "cg22223709", "cg23698976", "cg06780606", "cg13920260",
    "cg15861835", "cg10039267", "cg12454245", "cg22067189",
    "cg00150874", "cg08401365", "cg13781721", "cg02931660",
    "cg01316390", "cg14746118", "cg21294096", "cg11871337",
    "cg00408231", "cg09641151", "cg05226646", "cg11291200",
    "cg01109660", "cg23607813", "cg04624564", "cg07452499",
    "cg18123612", "cg48211697_TC11", "cg48222828_BC21", "cg48218650_BC21",
    "cg48219904_BC11", "cg48222534_TC11", "cg48214483_TC21",
    "cg48222923_TC11", "cg48217358_TC12", "cg48217547_BC11",
```

```
        "cg48215035_TC11", "cg48217358_TC11", "cg48215051_TC21",
        "cg48218172_BC21", "cg48218223_TC11", "cg48296014_TC11",
        "cg48218620_TC11", "cg48213060_TC11", "cg48244014_TC11",
        "cg48215477_BC11", "cg48217390_BC11", "cg48272545_BC11",
        "cg48222620_BC11", "cg48309797_TC11", "cg48212920_TC11",
        "cg48218860_TC11", "cg48216374_TC11", "cg48215185_BC21",
        "cg48213802_BC11", "cg48222396_TC11", "cg48214010_BC11",
        "cg48222395_BC11", "cg48218465_TC21", "cg48215216_BC11",
        "cg48216938_BC11", "cg48219858_BC21", "cg48214243_BC11",
        "cg48223281_TC21", "cg48214292_BC21", "cg32022449_BC11",
        "cg48215159_TC21", "cg48222049_BC11", "cg48246403_BC21",
        "cg48214455_BC11", "cg48216569_TC11", "cg48214177_BC21",
        "cg48246617_BC11", "cg48301218_BC11", "cg48214011_BC11",
        "cg48215297_BC21", "cg48217555_BC21", "cg48213764_TC21",
        "cg48222839_TC11", "cg48217418_TC21", "cg48216934_BC21",
        "cg48250058_BC11", "cg48219493_TC21", "cg48222602_TC21",
        "cg48217485_BC12", "cg48218187_TC11", "cg48222171_TC11",
        "cg48217401_BC21", "cg48218225_BC11", "cg48222795_BC11",
        "cg48224019_TC11", "cg48217672_BC11", "cg48217626_TC21",
        "cg48213632_BC21", "cg48216281_TC21", "cg48218341_BC21",
        "cg48222701_BC11", "cg48218522_TC11", "cg48217489_BC11",
        "cg48212144_TC21", "cg48219215_TC21", "cg48218176_BC11",
        "cg48223101_BC11", "cg48222143_TC11", "cg48218124_BC21",
        "cg48218975_BC11", "cg48217449_TC21", "cg48222478_BC21",
        "cg48216323_BC21", "cg48217683_BC11", "cg48215310_TC21",
        "cg48226387_BC11", "cg48218807_BC11", "cg48213481_BC11",
        "cg48224372_BC11", "cg48217446_BC21", "cg48222402_TC11",
        "cg48222222_TC11", "cg48215306_BC21", "cg48219235_BC21",
        "cg48221203_TC11", "cg48216903_BC21", "cg48218631_BC21",
        "cg48220121_TC11", "cg48215553_TC11", "cg48217396_TC11",
        "cg48224236_BC21")
    platform <- sesameData_check_platform(platform, names(betas))
    if (platform != "MM285") {
        betas <- liftOver(betas, "HM450")
    }
    vals <- mean(betas[hyperMALE], na.rm = TRUE) - betas[hypoMALE]
    dd <- density(na.omit(vals))
    if (dd$x[which.max(dd$y)] > 0.4) {
        "MALE"
    }
    else {
        "FEMALE"
    }
}
<bytecode: 0x0000012a3480c6a8>
<environment: namespace:sesame>
```

inferSpecies

```
function (sdf, topN = 1000, threshold.pos = 0.01, threshold.neg = 0.1,
    return.auc = FALSE, return.species = FALSE, verbose = FALSE)
{
    addr <- sesameDataGet(sprintf("%s.addressSpecies", sdfPlatform(sdf,
```

```
        verbose = verbose)))
    df_as <- do.call(cbind, lapply(addr$species, function(x) x$AS))
    rownames(df_as) <- addr$ordering$Probe_ID
    pvalue <- pOOBAH(sdf, return.pval = TRUE)
    pvalue <- pvalue[intersect(names(pvalue), rownames(df_as))]
    pos_probes <- sort(pvalue[pvalue <= threshold.pos], decreasing = FALSE)
    neg_probes <- sort(pvalue[pvalue >= threshold.neg], decreasing = TRUE)
    success.rate <- length(pvalue[pvalue <= 0.05])/length(pvalue)
    topN1 <- min(length(neg_probes), length(pos_probes), topN)
    pos <- pos_probes[seq_len(topN1)]
    neg <- neg_probes[seq_len(topN1)]
    y_true <- structure(c(rep(TRUE, length(pos)), rep(FALSE,
        length(neg))), names = c(names(pos), names(neg)))
    if (length(y_true) == 0) {
        warning("Lack of useful signal. Use reference.")
        return(species_ret(return.auc, return.species, addr$reference,
            NULL, sdf, addr, verbose))
    }
    n1 <- as.numeric(sum(y_true))
    n2 <- as.numeric(sum(!y_true))
    df_as <- df_as[names(y_true), , drop = FALSE]
    auc <- vapply(colnames(df_as), function(s) {
        R1 <- sum(rank(df_as[, s])[seq_along(pos)])
        U1 <- R1 - n1 * (n1 + 1)/2
        U1/(n1 * n2)
    }, numeric(1))
    if (success.rate >= 0.95 || (success.rate >= 0.8 && max(auc) <
        0.5)) {
        sdf <- sdfMsg(sdf, verbose, "Lack of negative probes. Use reference.")
        species <- addr$reference
    }
    else {
        species <- names(which.max(auc))
    }
    species_ret(return.auc, return.species, species, auc, sdf,
        addr, verbose)
}
<bytecode: 0x0000012a34827bb8>
<environment: namespace:sesame>
```

inferStrain

```
function (sdf, return.strain = FALSE, return.probability = FALSE,
    return.pval = FALSE, min_frac_dt = 0.2, verbose = FALSE)
{
    addr <- sesameDataGet("MM285.addressStrain")
    se <- addr$strain_snps
    cd <- SummarizedExperiment::colData(se)
    rd <- SummarizedExperiment::rowData(se)
    md <- metadata(se)
    strain_snps <- rd[, which(colnames(rd) == "C57BL_6J"):ncol(rd)]
    pvals <- pOOBAH(sdf, return.pval = TRUE)
    if (sum(pvals[rd$Probe_ID] < 0.05)/nrow(rd) < min_frac_dt) {
```

```
        if (return.strain) {
            return(NA)
        }
        else if (return.probability) {
            return(rep(NA, ncol(strain_snps)))
        }
        else if (return.pval) {
            return(NA)
        }
        else {
            return(sdfMsg(sdf, verbose, "Abort strain inference for low detection rate."))
        }
    }
    vafs <- getBetas(dyeBiasNL(noob(sdf)), mask = FALSE)[rd$Probe_ID]
    vafs[is.na(vafs)] <- 0.5
    vafs[rd$flipToAF] <- 1 - vafs[rd$flipToAF]
    probes <- intersect(names(vafs), rd$Probe_ID[rd$QC != "FAIL"])
    vafs <- vafs[probes]
    bbloglik <- vapply(strain_snps[match(probes, rd$Probe_ID),
        ], function(x) sum(log(dnorm(x - vafs, mean = 0, sd = 0.8))),
        numeric(1))
    probs <- setNames(exp(bbloglik - max(bbloglik)), colnames(strain_snps))
    best.index <- which.max(probs)
    strain <- names(best.index)
    if (return.strain) {
        strain
    }
    else if (return.probability) {
        probs/sum(probs)
    }
    else if (return.pval) {
        1 - probs[best.index]/sum(probs)
    }
    else {
        updateSigDF(sdf, strain = strain, addr = addr, verbose = verbose)
    }
}
<bytecode: 0x0000012a3483b228>
<environment: namespace:sesame>
```

inferTissue

```
function (betas, reference = NULL, platform = NULL, abs_delta_beta_min = 0.3,
    auc_min = 0.99, coverage_min = 0.8, topN = 15)
{
    stopifnot(is.numeric(betas))
    if (is.null(reference)) {
        if (is.null(platform)) {
            platform <- inferPlatformFromProbeIDs(names(betas))
        }
        stopifnot(platform %in% c("MM285"))
        reference <- sesameDataGet(sprintf("%s.tissueSignature",
            platform))
```

```
    }
    rd <- rowData(reference)
    fracs <- sort(vapply(unique(rd$branch), function(branch) {
        rd1 <- rd[rd$branch == branch & abs(rd$delta_beta) >=
            abs_delta_beta_min, ]
        rd1 <- head(rd1[order(-abs(rd1$delta_beta)), ], n = topN)
        fracs1 <- c(1 - betas[rd1[rd1$delta_beta < 0, "Probe_ID"]],
            betas[rd1[rd1$delta_beta > 0, "Probe_ID"]])
        mean(fracs1, na.rm = TRUE)
    }, numeric(1)), decreasing = TRUE)
    sprintf("[%s](%1.1f) [%s](%1.1f)", names(fracs)[1], fracs[1],
        names(fracs)[2], fracs[2])
}
<bytecode: 0x0000012a3486ae90>
<environment: namespace:sesame>
```

inferUniverse

```
function (platform)
{
    mfts <- c("MM285.address", "EPIC.address", "EPICv2.address",
        "Mammal40.address", "HM450.address", "HM27.address")
    mft <- paste0(platform, ".address")
    if (!(mft %in% mfts)) {
        stop("Platform unsupported. Please provide universe set explicitly.")
    }
    stopifnot()
    sesameDataGet(mft)$ordering$Probe_ID
}
<bytecode: 0x0000012a3486cdc0>
<environment: namespace:sesame>
```

InfI

```
function (sdf)
{
    sdf[sdf$col != "2", , drop = FALSE]
}
<bytecode: 0x0000012a34878988>
<environment: namespace:sesame>
```

InfIG

```
function (sdf)
{
    sdf[sdf$col == "G", , drop = FALSE]
}
<bytecode: 0x0000012a34874e00>
<environment: namespace:sesame>
```

InfII

```
function (sdf)
{
    sdf[sdf$col == "2", , drop = FALSE]
}
<bytecode: 0x0000012a3487d2d8>
<environment: namespace:sesame>
```

InfIR

```
function (sdf)
{
    sdf[sdf$col == "R", , drop = FALSE]
}
<bytecode: 0x0000012a34877740>
<environment: namespace:sesame>
```

initFileSet

```
function (map_path, platform, samples, probes = NULL, inc = 4)
{
    if (is.null(probes)) {
        addr <- sesameDataGet(paste0(platform, ".address"))
        probes <- addr$ordering$Probe_ID
    }
    fset <- structure(list(map_path = map_path, platform = platform,
        probes = probes, samples = sort(samples), inc = inc),
        class = "fileSet")
    fset$n <- length(fset$probes)
    fset$m <- length(fset$samples)
    message("Allocating space for ", fset$m, " ", platform, " samples at ",
        map_path, ".")
    fh <- file(map_path, "wb")
    for (i in seq_along(samples)) {
        writeBin(as.numeric(rep(NA, times = fset$n)), fh, size = inc)
    }
    close(fh)
    saveRDS(fset, paste0(map_path, "_idx.rds"))
    fset
}
<bytecode: 0x0000012a34881ad8>
<environment: namespace:sesame>
```

KYCG_annoProbes

```
function (query, databases, db_names = NULL, platform = NULL,
    sep = ",", indicator = FALSE, silent = FALSE)
{
    platform <- queryCheckPlatform(platform, query, silent = silent)
    if (is.character(databases)) {
        dbs <- KYCG_getDBs(databases, db_names = db_names, platform = platform,
```

```
            silent = silent, type = "categorical")
    }
    else {
        dbs <- databases
        names(dbs) <- vapply(dbs, function(db) {
            paste0(attr(db, "group"), "-", attr(db, "dbname"))
        }, character(1))
    }
    ind <- do.call(cbind, lapply(dbs, function(db) {
        query %in% db
    }))
    if (indicator) {
        rownames(ind) <- query
        colnames(ind) <- names(dbs)
        return(ind)
    }
    else {
        anno <- apply(ind, 1, function(x) paste(names(dbs)[x],
            collapse = sep))
        anno <- ifelse(anno == "", NA, anno)
        names(anno) <- query
        return(anno)
    }
}
<bytecode: 0x0000012a3487fa90>
<environment: namespace:sesame>
```

KYCG_buildGeneDBs

```
function (query = NULL, platform = NULL, genome = NULL, max_distance = 10000,
    silent = FALSE)
{
    platform <- queryCheckPlatform(platform, query, silent = silent)
    genes <- sesameData_getTxnGRanges(sesameData_check_genome(NULL,
        platform), merge2gene = TRUE)
    all_probes <- sesameData_getManifestGRanges(platform, genome = genome)
    if (!is.null(query)) {
        probes <- all_probes[names(all_probes) %in% query]
    }
    genes <- subsetByOverlaps(genes, probes + max_distance, ignore.strand = TRUE)
    hits <- findOverlaps(genes, all_probes + max_distance, ignore.strand = TRUE)
    dbs <- split(names(all_probes)[subjectHits(hits)], names(genes)[queryHits(hits)])
    gene_names <- genes[names(dbs)]$gene_name
    res <- lapply(seq_along(dbs), function(i) {
        d1 <- dbs[[i]]
        attr(d1, "group") <- sprintf("KYCG.%s.gene.00000000",
            platform)
        attr(d1, "dbname") <- names(dbs)[i]
        attr(d1, "gene_name") <- gene_names[i]
        d1
    })
    names(res) <- names(dbs)
    message(sprintf("Building %d gene DBs for %s...", length(res),
```

```
        platform))
    res
}
<bytecode: 0x0000012a34890930>
<environment: namespace:sesame>
```

KYCG_getDBs

```
function (group_nms, db_names = NULL, platform = NULL, summary = FALSE,
    allow_multi = FALSE, ignore.case = FALSE, type = NULL, silent = FALSE)
{
    if (!is.character(group_nms)) {
        return(group_nms)
    }
    group_nms <- guess_dbnames(group_nms, platform = platform,
        allow_multi = TRUE, type = type, silent = silent, ignore.case = ignore.case)
    group_nms <- group_nms[group_nms %in% sesameDataList()$Title]
    if (length(group_nms) == 0) {
        return(NULL)
    }
    res <- do.call(c, lapply(unname(group_nms), function(nm) {
        dbs <- sesameDataGet(nm)
        setNames(lapply(seq_along(dbs), function(ii) {
            db <- dbs[[ii]]
            attr(db, "group") <- nm
            attr(db, "dbname") <- names(dbs)[ii]
            db
        }), names(dbs))
    }))
    if (summary) {
        do.call(bind_rows, lapply(res, attributes))
    }
    else if (is.null(db_names)) {
        res
    }
    else {
        stopifnot(all(db_names %in% names(res)))
        res[db_names]
    }
}
<bytecode: 0x0000012a34898ac0>
<environment: namespace:sesame>
```

KYCG_listDBGroups

```
function (filter = NULL, path = NULL, type = NULL)
{
    if (is.null(path)) {
        gps <- sesameDataList("KYCG", full = TRUE)[, c("Title",
            "Description")]
        gps$type <- vapply(strsplit(gps$Description, " "), function(x) x[2],
            character(1))
```

```
        gps$Description <- str_replace(gps$Description, "KYCG categorical database holding ",
            "")
        if (!is.null(filter)) {
            gps <- gps[grepl(filter, gps$Title), ]
        }
        if (!is.null(type)) {
            gps <- gps[gps$type %in% type, ]
        }
    }
    else {
        gps <- basename(list.files(path, recursive = TRUE))
    }
    gps
}
<bytecode: 0x0000012a3489cd80>
<environment: namespace:sesame>
```

KYCG_loadDBs

```
function (in_paths, group_use_filename = FALSE)
{
    if (length(in_paths) == 1 && dir.exists(in_paths)) {
        groupnms <- grep(".gz$", list.files(in_paths, recursive = TRUE),
            value = TRUE)
        in_paths <- file.path(in_paths, groupnms)
    }
    else {
        groupnms <- basename(in_paths)
    }
    do.call(c, lapply(seq_along(groupnms), function(i) {
        tbl <- read.table(in_paths[i], sep = "\t", header = TRUE)
        dbs <- split(tbl$Probe_ID, tbl$Knowledgebase)
        lapply(names(dbs), function(gp_dbname) {
            gp_dbname_lst <- str_split(gp_dbname, ";", n = 2)[[1]]
            db1 <- dbs[[gp_dbname]]
            if (group_use_filename) {
                attr(db1, "group") <- sub(".gz$", "", groupnms[i])
            }
            else {
                attr(db1, "group") <- gp_dbname_lst[[1]]
            }
            if (length(gp_dbname_lst) > 1) {
                attr(db1, "dbname") <- gp_dbname_lst[[2]]
            }
            else {
                attr(db1, "dbname") <- attr(db1, "group")
            }
            db1
        })
    }))
}
<bytecode: 0x0000012a348a4f10>
<environment: namespace:sesame>
```

KYCG_plotBar

```
function (df, y = "-log10(FDR)", n = 20, order_by = "FDR", label = FALSE)
{
    stopifnot("estimate" %in% colnames(df) && "FDR" %in% colnames(df))
    df1 <- preparePlotDF(df, n, order_by)
    if (y == "-log10(FDR)") {
        df1[["-log10(FDR)"]] <- -log10(df1$FDR)
    }
    p <- ggplot(df1, aes_string("db1", y)) + geom_bar(stat = "identity") +
        coord_flip() + ylab(y) + xlab("CpG Group")
    if (label) {
        df1_label <- df1[df1$FDR < 0.05, ]
        df1_label$pos_label <- df1_label[[y]]/2
        df1_label$label <- sprintf("N=%d", df1_label$overlap)
        p <- p + geom_label(aes_string(x = "db1", y = "pos_label",
            label = "label"), data = df1_label, alpha = 0.6,
            hjust = 0.5)
    }
    p
}
<bytecode: 0x0000012a348d4c68>
<environment: namespace:sesame>
```

KYCG_plotDot

```
function (df, y = "-log10(FDR)", n = 20, order_by = "FDR", title = "Enriched Databases",
    label_by = "dbname", size_by = "overlap", color_by = "estimate",
    short_label = FALSE)
{
    df1 <- preparePlotDF(df, n, order_by, short_label = short_label,
        label_by = label_by)
    if (y == "-log10(FDR)") {
        df1[["-log10(FDR)"]] <- -log10(df1$FDR)
    }
    ggplot(df1) + geom_point(aes(.data[["db1"]], .data[[y]],
        size = .data[[size_by]], color = .data[[color_by]])) +
        coord_flip() + ggtitle(title) + scale_color_gradient(low = "blue",
        high = "red") + ylab(y) + xlab("")
}
<bytecode: 0x0000012a348dc4c8>
<environment: namespace:sesame>
```

KYCG_plotEnrichAll

```
function (df, fdr_max = 25, n_label = 15, min_estimate = 0, short_label = TRUE)
{
    gp_size <- sort(table(df$group))
    gp_width <- log(2 + gp_size)
    e1 <- df[order(factor(df$group, levels = names(gp_size)),
        df$dbname), ]
    e1$inc <- (gp_width/gp_size)[e1$group]
    e1$inc1 <- c(0, ifelse(e1$group[-1] != e1$group[-nrow(e1)],
```

```
        1, 0))
    e1$inc2 <- cumsum(e1$inc + e1$inc1)
    if (length(grep("^KYCG", e1$group)) > 0) {
        e1$group <- str_replace(e1$group, "KYCG.", "")
        e1$group <- vapply(strsplit(e1$group, "\\."), function(x) paste0(x[2:(length(x) -
            1)], collapse = "."), character(1))
    }
    if ("gene_name" %in% colnames(e1)) {
        e1$dbname[e1$group == "gene"] <- e1$gene_name[e1$group ==
            "gene"]
    }
    e2 <- e1[e1$estimate > min_estimate & e1$FDR < 0.01, ]
    e2$FDR[e2$FDR < 10^-fdr_max] <- 10^-(fdr_max * 1.1)
    e3 <- rownames_to_column(as.data.frame(do.call(rbind, lapply(split(e1$inc2,
        e1$group), function(x) c(beg = min(x), middle = mean(x),
        end = max(x))))), "group")
    inc2 <- FDR <- estimate <- group <- dbname <- beg <- middle <- NULL
    if (short_label) {
        e2$dbname <- vapply(strsplit(e2$dbname, ";"), function(x) {
            if (length(x) > 1) {
                x[[2]]
            }
            else {
                x[[1]]
            }
        }, character(1))
    }
    requireNamespace("ggrepel")
    ggplot(e2, aes(inc2, -log10(FDR))) + geom_point(aes(size = estimate,
        color = group), alpha = 0.5) + ggrepel::geom_text_repel(data = e2[head(order(e2$FDR),
        n = n_label), ], aes(label = dbname, color = group),
        size = 3, direction = "y", nudge_y = 0.2, max.overlaps = 100) +
        annotate("text", -1, fdr_max * 0.96, label = "Values above this line are capped.",
            hjust = 0, vjust = 1, color = "grey60") + geom_hline(yintercept = fdr_max,
        linetype = "dotted", color = "grey60") + geom_segment(aes(x = beg,
        y = 0, xend = end, yend = 0, color = group), size = 3,
        data = e3) + geom_text(data = e3, aes(middle, -1, label = group,
        color = group), vjust = 1, hjust = 1, angle = 30) + scale_color_discrete(guide = "none") +
        ylim(-6, fdr_max * 1.2) + xlab("") + scale_size_continuous(guide = guide_legend(title = "log2(O
        coord_cartesian(clip = "off") + theme_minimal() + theme(axis.title.x = element_blank(),
        axis.text.x = element_blank(), axis.ticks.x = element_blank(),
        panel.grid.minor.x = element_blank())
}
<bytecode: 0x0000012a348daf70>
<environment: namespace:sesame>
```

KYCG_plotLollipop

```
function (df, label_column = "dbname", n = 20)
{
    estimate <- label <- NULL
    df$label <- df[[label_column]]
    df <- head(df[order(df$estimate, decreasing = TRUE), ], n = n)
```

```
    allest <- df$estimate[!is.infinite(df$estimate)]
    cap <- max(allest) * 1.4
    cap_line <- max(allest) * 1.2
    df$estimate[df$estimate == Inf] <- cap
    ggplot(df, aes_string(x = "label", y = "estimate", label = "label")) +
        geom_hline(yintercept = 0) + geom_segment(aes(x = reorder(label,
        -estimate), y = 0, yend = estimate, xend = label), color = "black") +
        geom_point(fill = "black", stat = "identity", size = 15,
            alpha = 0.95, shape = 21) + scale_fill_gradientn(name = "Log2(OR)",
        colours = c("#2166ac", "#333333", "#b2182b")) + geom_text(color = "white",
        size = 3) + ylab("Log2(OR)") + geom_hline(yintercept = cap_line,
        linetype = "dashed") + ylim(min(min(allest) * 1.3, 0),
        max(max(allest) * 1.5, 0)) + theme_minimal() + theme(axis.title.x = element_blank(),
        axis.text.x = element_blank(), axis.ticks.x = element_blank())
}
<bytecode: 0x0000012a348f7ef8>
<environment: namespace:sesame>
```

KYCG_plotManhattan

```
function (vals, platform = NULL, genome = NULL, title = NULL,
    label_min = 100, col = c("wheat1", "sienna3"), ylabel = "Value")
{
    stopifnot(is(vals, "numeric"))
    if (is.null(platform)) {
        platform <- queryCheckPlatform(platform, query = vals,
            silent = FALSE)
    }
    genome <- sesameData_check_genome(genome, platform)
    gr <- sesameData_getManifestGRanges(platform, genome = genome)
    seqLength <- sesameData_getGenomeInfo(genome)$seqLength
    v <- vals[names(gr)]
    gr <- gr[!is.na(v)]
    SummarizedExperiment::mcols(gr)$val <- v[!is.na(v)]
    cumLength <- setNames(c(0, cumsum(as.numeric(seqLength))[-length(seqLength)]),
        names(seqLength))
    midLength <- cumLength + seqLength/2
    SummarizedExperiment::mcols(gr)$pos <- cumLength[as.character(seqnames(gr))] +
        end(gr)
    SummarizedExperiment::mcols(gr)$Probe_ID <- names(gr)
    df <- as_tibble(gr)
    df$seqnames <- factor(df$seqnames, levels = names(seqLength))
    requireNamespace("ggrepel")
    ggplot(df, aes_string(x = "pos", y = "val")) + geom_point(aes_string(color = "seqnames"),
        alpha = 0.8, size = 1.3) + ggrepel::geom_text_repel(data = df[df$val >
        label_min, ], aes_string(label = "Probe_ID")) + scale_color_manual(values = rep(col,
        length(seqLength))) + scale_x_continuous(labels = names(midLength),
        breaks = midLength) + scale_y_continuous(expand = c(0,
        0)) + theme_bw() + theme(legend.position = "none", panel.border = element_blank(),
        panel.grid.major.x = element_blank(), panel.grid.minor.x = element_blank()) +
        labs(title = title) + xlab("Chromosome") + ylab(ylabel)
}
<bytecode: 0x0000012a348f8c00>
```

```
<environment: namespace:sesame>
```

KYCG_plotMeta

```
function (betas, platform = NULL)
{
    if (!is.matrix(betas)) {
        betas <- cbind(sample = betas)
    }
    if (is.null(platform)) {
        platform <- inferPlatformFromProbeIDs(rownames(betas))
    }
    stopifnot(!is.null(platform))
    dbs <- KYCG_getDBs(sprintf("%s.metagene", platform))
    df <- dbStats(betas, dbs, long = TRUE)
    dflabel <- data.frame(ord = as.integer(names(dbs)), reg = vapply(dbs,
        function(x) attr(x, "label"), character(1)))
    ggplot(df) + annotate("rect", xmin = -1, xmax = 10, ymin = -Inf,
        ymax = Inf, fill = "grey80", alpha = 0.5, color = NA) +
        geom_line(aes_string("db", "value", group = "query")) +
        scale_x_continuous(breaks = dflabel$ord, labels = dflabel$reg) +
        ylab("Mean DNA Methylation Level") + xlab("") + theme(axis.text.x = element_text(angle = 90,
        vjust = 0.5, hjust = 1))
}
<bytecode: 0x0000012a3490df70>
<environment: namespace:sesame>
```

KYCG_plotMetaEnrichment

```
function (result_list)
{
    if (is.data.frame(result_list)) {
        result_list <- list(result_list)
    }
    stopifnot(all(c("dbname", "label") %in% colnames(result_list[[1]])))
    df <- aggregateTestEnrichments(result_list, return_df = TRUE)
    ggplot(df) + annotate("rect", xmin = -1, xmax = 10, ymin = -Inf,
        ymax = Inf, fill = "grey80", alpha = 0.5, color = NA) +
        geom_line(aes_string("db", "estimate", color = "query")) +
        scale_x_continuous(breaks = as.integer(result_list[[1]]$db),
            labels = result_list[[1]]$label) + annotate("text",
        x = min(as.integer(result_list[[1]]$db)), y = 0.05, label = "Enrichment",
        hjust = 0, vjust = 0) + annotate("text", x = min(as.integer(result_list[[1]]$db)),
        y = -0.05, label = "Depletion", hjust = 0, vjust = 1) +
        geom_hline(yintercept = 0, linetype = "dashed") + ylab("Log2 Fold Enrichment") +
        xlab("") + theme(axis.text.x = element_text(angle = 90,
        vjust = 0.5, hjust = 1))
}
<bytecode: 0x0000012a34968d78>
<environment: namespace:sesame>
```

KYCG_plotPointRange

```
function (result_list)
{
    ord <- mean_betas <- state <- est <- NULL
    mtx <- aggregateTestEnrichments(result_list)
    df <- melt(mtx, varnames = c("sample", "state"), value.name = "est")
    df <- summarize(group_by(df, state), ave = mean(pmax(-4,
        est), na.rm = TRUE), sd = sd(pmax(-10, est), na.rm = TRUE))
    df$ymin <- df$ave - df$sd
    df$ymax <- df$ave + df$sd
    df$state <- factor(df$state, levels = df$state[order(df$ave)])
    ggplot(df) + geom_pointrange(aes_string("state", "ave", ymin = "ymin",
        ymax = "ymax")) + geom_hline(yintercept = 0, linetype = "dashed") +
        ylab("Log2 Fold Enrichment") + xlab("") + scale_y_continuous(position = "right") +
        annotate("text", x = 0.5, y = 0.5, label = "Enrichment",
            angle = -90, hjust = 1) + annotate("text", x = 0.5,
        y = -0.5, label = "Depletion", angle = 90, hjust = 0) +
        coord_flip()
}
<bytecode: 0x0000012a3496ac38>
<environment: namespace:sesame>
```

KYCG_plotSetEnrichment

```
function (result, n_sample = 1000, n_presence = 200)
{
    stopifnot("dDisc" %in% names(result))
    dCont <- sort(result$dCont)
    dDisc <- result$dDisc
    presence <- names(dCont) %in% dDisc
    cs <- cumsum(ifelse(presence, 1/sum(presence), -1/sum(!presence)))
    index <- as.integer(seq(1, length(cs), length.out = n_sample))
    pos <- which(presence)
    if (length(pos) > n_presence) {
        pos <- sample(pos, n_presence)
    }
    WGG(ggplot(data.frame(index = index, cs = cs[index])) + geom_segment(data = data.frame(pos = pos),
        aes_string(x = "pos", xend = "pos", y = -0.02, yend = 0.02),
        color = "grey50") + geom_line(aes_string(x = "index",
        y = "cs"), color = "darkred") + xlab("") + ylab("ES(S)")) +
        WGG(ggplot(data.frame(index = index, var = dCont[index]),
            aes_string(x = "index", y = "var")) + geom_area() +
            xlab("CpGs") + ylab("Phenotype Var"), Beneath(height = 0.5))
}
<bytecode: 0x0000012a3496bba8>
<environment: namespace:sesame>
```

KYCG_plotVolcano

```
function (df, label_by = "dbname", alpha = 0.05)
{
    estimate <- FDR <- label <- NULL
```

```
    df <- df[abs(df$estimate) < 1000, ]
    df[["-log10(FDR)"]] <- -log10(df$FDR)
    df$Significance <- ifelse(df$FDR < alpha, "Significant",
        "Not significant")
    g <- ggplot(data = df, aes_string(x = "estimate", y = "-log10(FDR)",
        color = "Significance"))
    g <- g + geom_point() + xlab("log2(OR)")
    g <- g + ylab("-log10 FDR") + scale_colour_manual(name = sprintf("Significance (q < %s)",
        alpha), values = c(Significant = "red", `Not significant` = "black"))
    requireNamespace("ggrepel")
    g <- g + ggrepel::geom_text_repel(data = df[df$FDR < alpha &
        df$estimate > 0, ], aes_string(label = label_by), size = 5,
        box.padding = unit(0.35, "lines"), point.padding = unit(0.3,
            "lines"), show.legend = FALSE)
    g
}
<bytecode: 0x0000012a34974ef0>
<environment: namespace:sesame>
```

KYCG_plotWaterfall

```
function (df, order_by = "Log2(OR)", size_by = "-log10(FDR)",
    label_by = "dbname", n_label = 10)
{
    df$label <- df[[label_by]]
    if (size_by == "-log10(FDR)" || order_by == "-log10(FDR)" ||
        label_by == "-log10(FDR)") {
        df[["-log10(FDR)"]] <- -log10(df$FDR)
    }
    if (df$test[[1]] == "Log2(OR)" && (size_by == "Log2(OR)" ||
        order_by == "Log2(OR)" || label_by == "Log2(OR)")) {
        df[["Log2(OR)"]] <- df$estimate
        message(sprintf("%d extremes are capped.", sum(abs(df[["Log2(OR)"]]) >
            1000)))
        df[["Log2(OR)"]][df[["Log2(OR)"]] > 1000] <- 1000
        df[["Log2(OR)"]][df[["Log2(OR)"]] < -1000] <- -1000
    }
    df <- df[order(df[[order_by]]), ]
    df$index <- seq_len(nrow(df))
    requireNamespace("ggrepel")
    ggplot(df, aes(.data[["index"]], .data[[order_by]])) + geom_point(aes(size = .data[[size_by]]),
        alpha = 0.6) + geom_hline(yintercept = 0, linetype = "dashed",
        color = "grey60") + theme_minimal() + ylab(order_by) +
        xlab("Databases") + ggrepel::geom_text_repel(data = df[head(order(df$log10.p.value),
        n = min(n_label, nrow(df) * 0.5)), ], aes_string(label = "label"),
        nudge_x = -nrow(df)/10, max.overlaps = 999)
}
<bytecode: 0x0000012a34982940>
<environment: namespace:sesame>
```

leftRightMerge1

```
function (chrom.windows, min.probes.per.bin = 20)
{
    while (dim(chrom.windows)[1] > 0 && min(chrom.windows[, "probes"]) <
        min.probes.per.bin) {
        min.window <- which.min(chrom.windows$probes)
        merge.left <- FALSE
        merge.right <- FALSE
        if (min.window > 1 && chrom.windows[min.window, "start"] -
            1 == chrom.windows[min.window - 1, "end"]) {
            merge.left <- TRUE
        }
        if (min.window < dim(chrom.windows)[1] && chrom.windows[min.window,
            "end"] + 1 == chrom.windows[min.window + 1, "start"]) {
            merge.right <- TRUE
        }
        if (merge.left && merge.right) {
            if (chrom.windows[min.window - 1, "probes"] < chrom.windows[min.window +
                1, "probes"]) {
                merge.right <- FALSE
            }
        }
        if (merge.left) {
            chrom.windows[min.window - 1, "end"] <- chrom.windows[min.window,
                "end"]
            chrom.windows[min.window - 1, "probes"] <- chrom.windows[min.window -
                1, "probes"] + chrom.windows[min.window, "probes"]
            chrom.windows <- chrom.windows[-min.window, ]
        }
        else if (merge.right) {
            chrom.windows[min.window + 1, "start"] <- chrom.windows[min.window,
                "start"]
            chrom.windows[min.window + 1, "probes"] <- chrom.windows[min.window +
                1, "probes"] + chrom.windows[min.window, "probes"]
            chrom.windows <- chrom.windows[-min.window, ]
        }
        else {
            chrom.windows <- chrom.windows[-min.window, ]
        }
    }
    chrom.windows
}
<bytecode: 0x0000012a3498e118>
<environment: namespace:sesame>
```

liftOver

```
function (...)
{
    mLiftOver(...)
}
<bytecode: 0x0000012a349941f0>
```

```
<environment: namespace:sesame>
```

listAvailableMasks

```
function (platform, verbose = FALSE)
{
    stopifnot(is.character(platform))
    KYCG_getDBs(sprintf("%s.Mask", platform), summary = TRUE,
        silent = !verbose, ignore.case = TRUE)$dbname
}
<bytecode: 0x0000012a34998868>
<environment: namespace:sesame>
```

mapFileSet

```
function (fset, sample, named_values)
{
    con <- file(fset$map_path, "r+b")
    sample_index <- which(fset$samples == sample)
    if (length(sample_index) != 1) {
        stop("There are no or more than one sample ", sample,
            " in fileSet\n")
    }
    binWriteNumeric(con, named_values[match(fset$probes, names(named_values))],
        beg = (sample_index - 1) * fset$n, inc = fset$inc)
    close(con)
    fset
}
<bytecode: 0x0000012a3499a8e8>
<environment: namespace:sesame>
```

mapToMammal40

```
function (sdf)
{
    addr <- sesameDataGet("Mammal40.address")
    betas <- getBetas(sdf, collapseToPfx = TRUE)[addr$ordering$Probe_ID]
    names(betas) <- addr$ordering$Probe_ID
    betas
}
<bytecode: 0x0000012a3499bb68>
<environment: namespace:sesame>
```

maskIG

```
function (sdf)
{
    sdf$mask[sdf$col == "G"] <- TRUE
    sdf
}
```

```
<bytecode: 0x0000012a34995908>
<environment: namespace:sesame>
```

match1To2_1state

```
function (sdf)
{
    dR <- noMasked(InfIR(sdf))
    bR <- getBetas(dR)
    dG <- noMasked(InfIG(sdf))
    bG <- getBetas(dG)
    d2 <- noMasked(InfII(sdf))
    b2 <- getBetas(d2)
    dG$MG <- normalizeSetM(bG, b2, dG$UG)
    dR$MR <- normalizeSetM(bR, b2, dR$UR)
    sdf2 <- rbind(dR, dG, d2)
    sdf2 <- rbind(sdf2, sdf[!(sdf$Probe_ID %in% sdf2$Probe_ID),
        ])
    sdf2[order(sdf2$Probe_ID), ]
}
<bytecode: 0x0000012a3499d8a0>
<environment: namespace:sesame>
```

match1To2_3states

```
function (sdf)
{
    dR <- noMasked(InfIR(sdf))
    bR <- getBetas(dR)
    dG <- noMasked(InfIG(sdf))
    bG <- getBetas(dG)
    d2 <- noMasked(InfII(sdf))
    b2 <- getBetas(d2)
    mR <- as.integer(betaMix3States(bR))
    mG <- as.integer(betaMix3States(bG))
    m2 <- as.integer(betaMix3States(b2))
    dR$MR[mR == 1] <- normalizeSetM(bR[mR == 1], b2[m2 == 1],
        dR$UR[mR == 1])
    dR$MR[mR == 2] <- normalizeSetM(bR[mR == 2], b2[m2 == 2],
        dR$UR[mR == 2])
    dR$MR[mR == 3] <- normalizeSetM(bR[mR == 3], b2[m2 == 3],
        dR$UR[mR == 3])
    dG$MG[mG == 1] <- normalizeSetM(bG[mG == 1], b2[m2 == 1],
        dG$UG[mG == 1])
    dG$MG[mG == 2] <- normalizeSetM(bG[mG == 2], b2[m2 == 2],
        dG$UG[mG == 2])
    dG$MG[mG == 3] <- normalizeSetM(bG[mG == 3], b2[m2 == 3],
        dG$UG[mG == 3])
    sdf2 <- rbind(dR, dG, d2)
    sdf2 <- rbind(sdf2, sdf[!(sdf$Probe_ID %in% sdf2$Probe_ID),
        ])
    sdf2[order(sdf2$Probe_ID), ]
```

```
}
<bytecode: 0x0000012a349e09f8>
<environment: namespace:sesame>
```

matchDesign

```
function (sdf, min_dbeta = 0.3)
{
    dR <- noMasked(InfIR(sdf))
    dG <- noMasked(InfIG(sdf))
    d2 <- noMasked(InfII(sdf))
    b2 <- getBetas(d2)
    m2 <- as.integer(betaMix2States(b2))
    if (sum(m2 == 1, na.rm = TRUE) > 100 && sum(m2 == 2, na.rm = TRUE) >
        100 && abs(calcMode(b2[m2 == 1]) - calcMode(b2[m2 ==
        2])) > 0.7) {
        return(match1To2_3states(sdf))
    }
    if (sum(m2 == 1, na.rm = TRUE) < 10 || sum(m2 == 2, na.rm = TRUE) <
        10 || valleyDescent(b2[m2 == 1], b2[m2 == 2]) >= 0.8 ||
        abs(calcMode(b2[m2 == 1]) - calcMode(b2[m2 == 2])) <
            min_dbeta) {
        return(match1To2_1state(sdf))
    }
    bR <- getBetas(dR, mask = FALSE)
    mR <- as.integer(betaMix2States(bR))
    bG <- getBetas(dG, mask = FALSE)
    mG <- as.integer(betaMix2States(bG))
    dR$MR[mR == 1] <- normalizeSetM(bR[mR == 1], b2[m2 == 1],
        dR$UR[mR == 1])
    dR$MR[mR == 2] <- normalizeSetM(bR[mR == 2], b2[m2 == 2],
        dR$UR[mR == 2])
    dG$MG[mG == 1] <- normalizeSetM(bG[mG == 1], b2[m2 == 1],
        dG$UG[mG == 1])
    dG$MG[mG == 2] <- normalizeSetM(bG[mG == 2], b2[m2 == 2],
        dG$UG[mG == 2])
    sdf2 <- rbind(dR, dG, d2)
    sdf2 <- rbind(sdf2, sdf[!(sdf$Probe_ID %in% sdf2$Probe_ID),
        ])
    sdf2[order(sdf2$Probe_ID), ]
}
<bytecode: 0x0000012a349ec518>
<environment: namespace:sesame>
```

meanIntensity

```
function (sdf, mask = TRUE)
{
    stopifnot(all(c("MG", "UG", "MR", "UR") %in% colnames(sdf)))
    s <- signalMU(sdf, mask = mask)
    mean(c(s$M, s$U), na.rm = TRUE)
}
```

```
<bytecode: 0x0000012a349ff8d0>
<environment: namespace:sesame>
```

medianTotalIntensity

```
function (sdf, mask = TRUE)
{
    stopifnot(all(c("MG", "UG", "MR", "UR") %in% colnames(sdf)))
    s <- signalMU(sdf, mask = mask)
    median(c(s$M + s$U), na.rm = TRUE)
}
<bytecode: 0x0000012a349fd508>
<environment: namespace:sesame>
```

mLiftOver

```
function (x, target_platform, source_platform = NULL, BPPARAM = SerialParam(),
    mapping = NULL, impute = FALSE, sd_max = 999, celltype = "Blood",
    ...)
{
    if (is.numeric(x)) {
        if (is.matrix(x)) {
            betas <- do.call(cbind, bplapply(seq_len(ncol(x)),
                function(i) {
                  mLiftOver(x[, i], target_platform, source_platform = source_platform,
                    mapping = mapping, impute = impute, sd_max = sd_max,
                    celltype = celltype)
                }, BPPARAM = BPPARAM))
            colnames(betas) <- colnames(x)
        }
        else {
            mapping <- convertProbeID(names(x), target_platform,
                source_platform, mapping = mapping, return_mapping = TRUE,
                include_new = TRUE)
            betas <- setNames(x[mapping$ID_source], mapping$ID_target)
            if (impute) {
                betas <- imputeBetas(betas, target_platform,
                  celltype = celltype, sd_max = sd_max)
            }
        }
        betas
    }
    else if (is(x, "SigDF")) {
        mapping <- convertProbeID(x$Probe_ID, target_platform,
            source_platform, return_mapping = TRUE, target_uniq = TRUE,
            include_new = TRUE)
        x2 <- x[match(mapping$ID_source, x$Probe_ID), ]
        x2$Probe_ID <- mapping$ID_target
        x2 <- x2[order(x2$Probe_ID), ]
        x2$mask[is.na(x2$mask)] <- TRUE
        rownames(x2) <- NULL
        attr(x2, "platform") <- target_platform
```

```
        x2
    }
    else if (is.character(x)) {
        convertProbeID(x, target_platform, source_platform, ...)
    }
    else if (is.list(x) && is(x[[1]], "SigDF")) {
        bplapply(x, mLiftOver, BPPARAM = BPPARAM, target_platform,
            source_platform = source_platform, mapping = mapping,
            impute = impute, sd_max = sd_max, celltype = celltype,
            ...)
    }
}
<bytecode: 0x0000012a349faea0>
<environment: namespace:sesame>
```

model_contrasts

```
function (mm, meta)
{
    contrs <- names(attr(mm, "contrasts"))
    setNames(lapply(contrs, function(cont) {
        x <- make.names(paste0("X", levels(factor(meta[[cont]]))))
        substr(x, 2, nchar(x))
    }), contrs)
}
<bytecode: 0x0000012a34a0deb8>
<environment: namespace:sesame>
```

mouseBetaToAF

```
function (betas)
{
    se <- sesameDataGet("MM285.addressStrain")$strain_snps
    rd <- rowData(se)
    af <- betas[rd$Probe_ID]
    af[rd$flipToAF] <- 1 - af[rd$flipToAF]
    af
}
<bytecode: 0x0000012a34a13078>
<environment: namespace:sesame>
```

MValueToBetaValue

```
function (m)
{
    2^m/(1 + 2^m)
}
<bytecode: 0x0000012a34a1aae8>
<environment: namespace:sesame>
```

negControls

```
function (sdf)
{
    stopifnot(is(sdf, "SigDF"))
    idx <- grep("negative", tolower(sdf$Probe_ID))
    if (length(idx) > 0) {
        negctls <- sdf[idx, c("UG", "UR")]
    }
    else {
        df <- controls(sdf)
        negctls <- df[grep("negative", tolower(df$Type)), c("UG",
            "UR")]
    }
    colnames(negctls) <- c("G", "R")
    negctls
}
<bytecode: 0x0000012a34a18fc8>
<environment: namespace:sesame>
```

noMasked

```
function (sdf)
{
    sdf[!sdf$mask, , drop = FALSE]
}
<bytecode: 0x0000012a34a39c40>
<environment: namespace:sesame>
```

nonuniqMask

```
function (platform, verbose = FALSE)
{
    stopifnot(is.character(platform))
    dbnames <- listAvailableMasks(platform, verbose = verbose)
    if (is.null(dbnames)) {
        return(NULL)
    }
    mask_names <- c("M_nonuniq", "nonunique", "sub35_copy", "multi",
        "design_issue")
    mask_names <- dbnames[dbnames %in% mask_names]
    if (length(mask_names) > 0) {
        do.call(c, KYCG_getDBs(sprintf("%s.Mask", platform),
            mask_names, silent = !verbose))
    }
    else {
        NULL
    }
}
<bytecode: 0x0000012a34a42118>
<environment: namespace:sesame>
```

noob

```
function (sdf, combine.neg = TRUE, offset = 15)
{
    stopifnot(is(sdf, "SigDF"))
    nmk <- sdf[!(sdf$Probe_ID %in% nonuniqMask(sdfPlatform(sdf))),
        ]
    bgG <- oobG(nmk)
    bgR <- oobR(nmk)
    if (combine.neg) {
        neg <- negControls(sdf)
        bgG <- c(bgG, neg$G)
        bgR <- c(bgR, neg$R)
    }
    if (sum(bgG > 0, na.rm = TRUE) < 100 || sum(bgR > 0, na.rm = TRUE) <
        100) {
        return(sdf)
    }
    bgR[bgR == 0] <- 1
    bgG[bgG == 0] <- 1
    bgR <- bgR[bgR < median(bgR, na.rm = TRUE) + 10 * IQR(bgR,
        na.rm = TRUE)]
    bgG <- bgG[bgG < median(bgG, na.rm = TRUE) + 10 * IQR(bgG,
        na.rm = TRUE)]
    ibG <- c(InfIG(nmk)$MG, InfIG(nmk)$UG, InfII(nmk)$UG)
    ibR <- c(InfIR(nmk)$MR, InfIR(nmk)$UR, InfII(nmk)$UR)
    ibG[ibG == 0] <- 1
    ibR[ibR == 0] <- 1
    fitG <- backgroundCorrectionNoobFit(ibG, bgG)
    sdf$MG <- normExpSignal(fitG$mu, fitG$sigma, fitG$alpha,
        sdf$MG) + 15
    sdf$UG <- normExpSignal(fitG$mu, fitG$sigma, fitG$alpha,
        sdf$UG) + 15
    fitR <- backgroundCorrectionNoobFit(ibR, bgR)
    sdf$MR <- normExpSignal(fitR$mu, fitR$sigma, fitR$alpha,
        sdf$MR) + 15
    sdf$UR <- normExpSignal(fitR$mu, fitR$sigma, fitR$alpha,
        sdf$UR) + 15
    sdf
}
<bytecode: 0x0000012a34a3f340>
<environment: namespace:sesame>
```

noobSub

```
function (sig, bg)
{
    e <- MASS::huber(bg)
    mu <- e$mu
    sigma <- e$s
    alpha <- pmax(MASS::huber(sig)$mu - mu, 10)
    normExpSignal(mu, sigma, alpha, sig)
}
```

```
<bytecode: 0x0000012a34a4cc78>
<environment: namespace:sesame>
```

normalizeSetM

```
function (input, ref, U)
{
    bn <- normalize.quantiles.use.target(matrix(input), ref)
    U * bn/(1 - bn)
}
<bytecode: 0x0000012a34a4a4d8>
<environment: namespace:sesame>
```

normControls

```
function (sdf, average = FALSE, verbose = FALSE)
{
    df <- controls(sdf)
    df <- df[grep("norm(_|\\.)", tolower(df$Type)), ]
    if (nrow(df) == 0)
        stop("No normalization control probes found!")
    if (sdfPlatform(sdf, verbose = verbose) == "HM27") {
        df$channel <- ifelse(grepl("norm\\.green", tolower(df$Type)),
            "G", "R")
    }
    else {
        df$channel <- ifelse(grepl("norm_(c|g)", tolower(df$Type)),
            "G", "R")
    }
    if (average) {
        c(G = mean(df[df$channel == "G", "UG"], na.rm = TRUE),
            R = mean(df[df$channel == "R", "UR"], na.rm = TRUE))
    }
    else {
        df
    }
}
<bytecode: 0x0000012a34a565a8>
<environment: namespace:sesame>
```

normExpSignal

```
function (mu, sigma, alpha, x)
{
    sigma2 <- sigma * sigma
    if (alpha <= 0)
        stop("alpha must be positive")
    if (sigma <= 0)
        stop("sigma must be positive")
    mu.sf <- x - mu - sigma2/alpha
    signal <- mu.sf + sigma2 * exp(dnorm(0, mean = mu.sf, sd = sigma,
        log = TRUE) - pnorm(0, mean = mu.sf, sd = sigma, lower.tail = FALSE,
```

```
        log.p = TRUE))
    o <- !is.na(signal)
    if (any(signal[o] < 0)) {
        warning("Limit of numerical accuracy reached with very\nlow intensity or very high background:\n
        signal[o] <- pmax(signal[o], 1e-06)
    }
    signal
}
<bytecode: 0x0000012a34a526d8>
<environment: namespace:sesame>
```

oobG

```
function (sdf)
{
    dR <- InfIR(sdf)
    c(dR$MG, dR$UG)
}
<bytecode: 0x0000012a34a5f440>
<environment: namespace:sesame>
```

oobR

```
function (sdf)
{
    dG <- InfIG(sdf)
    c(dG$MR, dG$UR)
}
<bytecode: 0x0000012a34a5d778>
<environment: namespace:sesame>
```

openSesame

```
function (x, prep = "QCDPB", prep_args = NULL, manifest = NULL,
    func = getBetas, BPPARAM = SerialParam(), platform = "",
    min_beads = 1, ...)
{
    if (length(x) == 1 && is(x, "character") && dir.exists(x)) {
        x <- searchIDATprefixes(x)
    }
    if (is(x, "SigDF")) {
        wrap_openSesame1(func, prepSesame(x, prep, prep_args),
            ...)
    }
    else if (is(x, "character")) {
        if (length(x) == 1) {
            wrap_openSesame1(func, prepSesame(readIDATpair(x,
                platform = platform, manifest = manifest, min_beads = min_beads),
                prep, prep_args), ...)
        }
        else {
            wrap_openSesame(x, bplapply(x, openSesame, platform = platform,
```

```
                prep = prep, prep_args = prep_args, func = func,
                manifest = manifest, BPPARAM = BPPARAM, ...))
        }
    }
    else if (is(x, "list") && is(x[[1]], "SigDF")) {
        wrap_openSesame(x, bplapply(x, openSesame, platform = platform,
            prep = prep, prep_args = prep_args, fun = func, manifest = manifest,
            BPPARAM = BPPARAM, ...))
    }
    else {
        stop("Unsupported input")
    }
}
<bytecode: 0x0000012a34a59758>
<environment: namespace:sesame>
```

openSesameToFile

```
function (map_path, idat_dir, BPPARAM = SerialParam(), inc = 4)
{
    samples <- basename(searchIDATprefixes(idat_dir))
    sdf <- readIDATpair(file.path(idat_dir, samples[1]))
    fset <- initFileSet(map_path, sdfPlatform(sdf), samples,
        inc = inc)
    message("Mapping ", length(samples), " ", sdfPlatform(sdf),
        " samples to ", map_path, ".")
    returned <- bplapply(samples, function(sample) {
        try({
            betas <- openSesame(file.path(idat_dir, sample))
            mapFileSet(fset, sample, betas)
            TRUE
        })
    }, BPPARAM = BPPARAM)
    succeeded <- !vapply(returned, function(x) inherits(x, "try-error"),
        logical(1))
    message("Successfully processed ", sum(succeeded), " IDATs (",
        sum(!succeeded), " failed).")
    fset
}
<bytecode: 0x0000012a34a60f10>
<environment: namespace:sesame>
```

palgen

```
function (pal, n = 150, space = "Lab")
{
    requireNamespace("pals")
    adhoc_pals <- list(whiteturbo = c("white", "white", pals::turbo(10)[seq(2,
        10)]), whitejet = c("white", "white", "lightblue", "blue",
        "green", "yellow", "orange", "red", "darkred"), whiteblack = c("white",
        "black"))
    if (length(pal) == 1 && is.character(pal) && (pal %in% names(adhoc_pals))) {
```

```
        pal <- adhoc_pals[[pal]]
    }
    if (is.character(pal)) {
        requireNamespace("grDevices")
        grDevices::colorRampPalette(pal, space = space)
    }
    else if (is.function(pal)) {
        pal
    }
    else {
        stop("Please provide the right pal format.")
    }
}
<bytecode: 0x0000012a34a728b8>
<environment: namespace:sesame>
```

parseGEOsignalMU

```
function (sigM, sigU, Probe_IDs, oob.mean = 500, oob.sd = 300,
    platform = NULL)
{
    if (is.null(platform)) {
        platform <- inferPlatformFromProbeIDs(Probe_IDs)
    }
    addr <- sesameDataGet(paste0(platform, ".address"))$ordering
    M <- sigM[match(addr$Probe_ID, Probe_IDs)]
    U <- sigU[match(addr$Probe_ID, Probe_IDs)]
    col <- ifelse(is.na(addr$col), "2", as.character(addr$col))
    oobs <- pmax(50, rnorm(length(col), mean = oob.mean, sd = oob.sd))
    MG <- ifelse(col == "2", NA, ifelse(col == "G", M, oobs))
    MR <- ifelse(col == "2", NA, ifelse(col == "R", M, oobs))
    UG <- ifelse(col == "2", M, ifelse(col == "G", U, oobs))
    UR <- ifelse(col == "2", U, ifelse(col == "R", U, oobs))
    sdf <- data.frame(Probe_ID = addr$Probe_ID, MG = MG, MR = MR,
        UG = UG, UR = UR, col = factor(col, levels = c("G", "R",
            "2")), mask = addr$mask)
    class(sdf) <- c("SigDF", class(sdf))
    sdf
}
<bytecode: 0x0000012a34a752c0>
<environment: namespace:sesame>
```

plotCytoBand

```
function (chrom, beg, end, genomeInfo)
{
    cytoBand <- genomeInfo$cytoBand
    requireNamespace("pals")
    cytoBand2col <- setNames(pals::ocean.gray(10)[seq(9, 3)],
        c("stalk", "gneg", "gpos25", "gpos50", "gpos75", "gpos100"))
    cytoBand2col["acen"] <- "red"
    cytoBand2col["gvar"] <- cytoBand2col["gpos75"]
```

```
    cytoBand.target <- cytoBand[cytoBand$chrom == chrom, ]
    chromEnd <- max(cytoBand.target$chromEnd)
    chromBeg <- min(cytoBand.target$chromStart)
    chromWid <- chromEnd - chromBeg
    bandColor <- cytoBand2col[as.character(cytoBand.target$gieStain)]
    pltx0 <- (c(beg, end) - chromBeg)/chromWid
    gList(grid.text(sprintf("%s:%d-%d", chrom, beg, end), 0,
        0.9, just = c("left", "bottom"), draw = FALSE), grid.rect(0,
        0.35, 1, 0.35, just = c("left", "bottom"), gp = gpar(col = "black",
            lwd = 2, lty = "solid"), draw = FALSE), grid.rect(vapply(cytoBand.target$chromStart,
        function(x) (x - chromBeg)/chromWid, 1), 0.35, (cytoBand.target$chromEnd -
        cytoBand.target$chromStart)/chromWid, 0.35, gp = gpar(fill = bandColor,
        col = bandColor), just = c("left", "bottom"), draw = FALSE),
        grid.segments(x0 = pltx0, y0 = 0.1, x1 = pltx0, y1 = 0.9,
            gp = gpar(col = "red"), draw = FALSE))
}
<bytecode: 0x0000012a34ae5280>
<environment: namespace:sesame>
```

plotMapLines

```
function (probes, beg, end)
{
    nprobes <- length(probes)
    x00 <- ((GenomicRanges::start(probes) - beg)/(end - beg))
    y0 <- rep(0.5, length.out = length(probes))
    x1 <- ((seq_len(nprobes) - 0.5)/nprobes)
    y1 <- rep(0, length.out = nprobes)
    x0 <- c(x00, x00)
    x1 <- c(x1, x00)
    y0 <- c(y0, rep(0.5, length.out = length(probes)))
    y1 <- c(y1, rep(1, length.out = length(probes)))
    grid.segments(x0, y0, x1, y1, draw = FALSE)
}
<bytecode: 0x0000012a34aef7a0>
<environment: namespace:sesame>
```

plotTranscript1

```
function (txn, reg, i, beg, end, isoformHeight, padHeight, txn.font.size)
{
    txn_name <- names(txn)[1]
    exons <- txn[[1]]
    meta <- as.data.frame(GenomicRanges::mcols(txn))
    plt.width <- end - beg
    txn.beg <- max(beg, min(GenomicRanges::start(exons)) - 2000)
    txn.end <- min(end, max(GenomicRanges::end(exons)) + 2000)
    exons <- subsetByOverlaps(exons, reg)
    txn.strand <- as.character(GenomicRanges::strand(exons[1]))
    lined <- (c(txn.beg, txn.end) - beg)/plt.width
    y.bot <- (i - 1) * isoformHeight + padHeight
    y.bot.exon <- y.bot + padHeight
```

```
    y.hei <- isoformHeight - 2 * padHeight
    g <- gList(grid.text(sprintf("%s (%s)", meta$gene_name, txn_name),
        x = mean(lined), y = y.bot + y.hei + padHeight * 0.5,
        just = c("center", "bottom"), gp = gpar(fontsize = txn.font.size),
        draw = FALSE))
    g <- gList(g, gList(grid.lines(x = lined, y = y.bot + y.hei/2,
        arrow = arrow(length = unit(0.06, "inches"), ends = ifelse(txn.strand ==
            "+", "last", "first")), draw = FALSE)))
    g <- gList(g, gList(grid.lines(x = c(0, 1), y = y.bot + y.hei/2,
        gp = gpar(lty = "dotted"), draw = FALSE)))
    g <- gList(g, gList(grid.rect((GenomicRanges::start(exons) -
        beg)/plt.width, y.bot + y.hei/2 - y.hei/3, GenomicRanges::width(exons)/plt.width,
        y.hei/3 * 2, gp = gpar(fill = "grey10", lwd = 0), just = c("left",
            "bottom"), draw = FALSE)))
    cds <- exonToCDS(exons, as.integer(meta$cdsStart), as.integer(meta$cdsEnd))
    if (length(cds) > 0) {
        g <- gList(g, gList(grid.rect((GenomicRanges::start(cds) -
            beg)/plt.width, y.bot + y.hei/2 - y.hei/6, GenomicRanges::width(cds)/plt.width,
            y.hei/6 * 2, gp = gpar(fill = "red", lwd = 0), just = c("left",
                "bottom"), draw = FALSE)))
    }
    g
}
<bytecode: 0x0000012a34af80d8>
<environment: namespace:sesame>
```

plotTranscripts

```
function (txns, reg, beg, end, txn.types = c("protein_coding"),
    txn.font.size = 6)
{
    if (!is.null(txn.types)) {
        txns <- txns[GenomicRanges::mcols(txns)$transcript_type %in%
            txn.types]
    }
    if (length(txns) == 0) {
        return(gList(grid.rect(0, 0.1, 1, 0.8, just = c("left",
            "bottom"), draw = FALSE), grid.text("No transcript found",
            x = 0.5, y = 0.5, draw = FALSE)))
    }
    isoformHeight <- 1/length(txns)
    padHeight <- isoformHeight * 0.2
    do.call(gList, lapply(seq_along(txns), function(i) {
        plotTranscript1(txns[i], reg, i, beg, end, isoformHeight,
            padHeight, txn.font.size)
    }))
}
<bytecode: 0x0000012a34b0d350>
<environment: namespace:sesame>
```

pOOBAH

```
function (sdf, return.pval = FALSE, combine.neg = TRUE, pval.threshold = 0.05,
    verbose = FALSE)
{
    stopifnot(is(sdf, "SigDF"))
    nmk <- sdf[!(sdf$Probe_ID %in% nonuniqMask(sdfPlatform(sdf))),
        ]
    bgG <- oobG(nmk)
    bgR <- oobR(nmk)
    if (combine.neg) {
        neg <- negControls(sdf)
        bgG <- c(bgG, neg$G)
        bgR <- c(bgR, neg$R)
    }
    if (sum(!is.na(bgG)) <= 100) {
        bgG <- seq_len(1000)
    }
    if (sum(!is.na(bgR)) <= 100) {
        bgR <- seq_len(1000)
    }
    funcG <- ecdf(bgG)
    funcR <- ecdf(bgR)
    pvals <- setNames(pmin(1 - funcR(pmax(sdf$MR, sdf$UR, na.rm = TRUE)),
        1 - funcG(pmax(sdf$MG, sdf$UG, na.rm = TRUE))), sdf$Probe_ID)
    if (return.pval) {
        return(pvals)
    }
    addMask(sdf, pvals > pval.threshold)
}
<bytecode: 0x0000012a34b09640>
<environment: namespace:sesame>
```

predictAge

```
function (betas, model, na_fallback = FALSE, min_nonna = 10)
{
    betas <- betas[model$param$Probe_ID]
    if (sum(!is.na(betas)) < min_nonna) {
        stop("Fewer than 10 matching probes left. Age prediction abort.")
    }
    if (sum(is.na(betas)) > 0) {
        if (na_fallback) {
            k <- is.na(betas)
            betas[k] <- model$param$na_fallback[k]
        }
        else {
            probes <- intersect(names(na.omit(betas)), model$param$Probe_ID)
            betas <- betas[probes]
            model$param <- model$param[match(probes, model$param$Probe_ID),
                ]
        }
    }
```

```
    drop(model$response2age(betas %*% model$param$slope + model$intercept))
}
<bytecode: 0x0000012a34b12a18>
<environment: namespace:sesame>
```

predictAgeHorvath353

```
function (betas)
{
    .Deprecated("predictAge")
}
<bytecode: 0x0000012a34b1ab00>
<environment: namespace:sesame>
```

predictAgeSkinBlood

```
function (betas)
{
    .Deprecated("predictAge")
}
<bytecode: 0x0000012a34b19180>
<environment: namespace:sesame>
```

predictMouseAgeInMonth

```
function (betas, na_fallback = TRUE)
{
    .Deprecated("predictAge")
}
<bytecode: 0x0000012a34b1d7f8>
<environment: namespace:sesame>
```

prefixMask

```
function (sdf, prefixes = NULL, invert = FALSE)
{
    idx <- Reduce("|", lapply(prefixes, function(pfx) {
        grepl(sprintf("^%s", pfx), sdf$Probe_ID)
    }))
    if (invert) {
        sdf[!idx, "mask"] <- TRUE
    }
    else {
        sdf[idx, "mask"] <- TRUE
    }
    sdf
}
<bytecode: 0x0000012a34b1fe28>
<environment: namespace:sesame>
```

prefixMaskButC

```
function (sdf)
{
    prefixMask(sdf, c("cg", "ch"), invert = TRUE)
}
<bytecode: 0x0000012a34b40980>
<environment: namespace:sesame>
```

prefixMaskButCG

```
function (sdf)
{
    prefixMask(sdf, "cg", invert = TRUE)
}
<bytecode: 0x0000012a34b3cdf8>
<environment: namespace:sesame>
```

preparePlotDF

```
function (df, n, order_by, short_label = FALSE, label_by = "dbname")
{
    db1 <- FDR <- NULL
    stopifnot("estimate" %in% colnames(df) && "FDR" %in% colnames(df))
    df1 <- df[df$nD > 0, ]
    df1$FDR[df1$FDR == 0] <- .Machine$double.xmin
    if ("group" %in% colnames(df1) && !short_label) {
        gp <- sprintf("%s~", vapply(str_split(df1$group, "\\."),
            function(x) {
                if (length(x) > 3) {
                  x[3]
                }
                else {
                  x[1]
                }
            }, character(1)))
    }
    else {
        gp <- ""
    }
    if (label_by %in% colnames(df1)) {
        df1$db1 <- paste0(gp, df1[[label_by]])
    }
    else if ("feat" %in% colnames(df1)) {
        df1$db1 <- paste0(gp, df1$feat)
    }
    if (length(unique(df1$db1)) != nrow(df1)) {
        df1 <- df1 %>% group_by(db1) %>% slice_min(order_by = FDR,
            n = 1, with_ties = FALSE) %>% ungroup()
    }
    ord <- df1[[order_by]]
    if (order_by == "estimate") {
        ord <- -ord
```

```
    }
    df1 <- df1[order(ord, -df1$estimate), ]
    df1 <- head(df1, n = n)
    df1$db1 <- factor(df1$db1, levels = rev(df1$db1))
    df1
}
<bytecode: 0x0000012a34b45308>
<environment: namespace:sesame>
```

prepSesame

```
function (sdf, prep = "QCDPB", prep_args = NULL)
{
    cfuns <- prepSesameList()
    codes <- str_split(prep, "")[[1]]
    stopifnot(all(codes %in% cfuns$code))
    x <- sdf
    for (c1 in codes) {
        x <- do.call(get(cfuns[cfuns$code == c1, "func"]), c(list(x),
            prep_args[[c1]]))
    }
    x
}
<bytecode: 0x0000012a34b4c078>
<environment: namespace:sesame>
```

prepSesameList

```
function ()
{
    x <- data.frame(rbind(c("0", "resetMask", "Reset mask to all FALSE"),
        c("Q", "qualityMask", "Mask probes of poor design"),
        c("G", "prefixMaskButCG", "Mask all but cg- probes"),
        c("H", "prefixMaskButC", "Mask all but cg- and ch-probes"),
        c("C", "inferInfiniumIChannel", "Infer channel for Infinium-I probes"),
        c("D", "dyeBiasNL", "Dye bias correction (non-linear)"),
        c("E", "dyeBiasL", "Dye bias correction (linear)"), c("P",
            "pOOBAH", "Detection p-value masking using oob"),
        c("I", "ELBAR", "Mask background-dominated readings"),
        c("B", "noob", "Background subtraction using oob"), c("S",
            "inferSpecies", "Set species-specific mask"), c("T",
            "inferStrain", "Set strain-specific mask (mouse)"),
        c("M", "matchDesign", "Match Inf-I/II in beta distribution")))
    colnames(x) <- c("code", "func", "description")
    x
}
<bytecode: 0x0000012a34b51598>
<environment: namespace:sesame>
```

print.DMLSummary

```
function (x, ...)
{
    mm <- attr(x, "model.matrix")
    message(sprintf("DMLSummary Object with %d Loci, %d samples.\n",
        length(x), nrow(mm)))
    contrast_names <- paste0(names(attr(mm, "contrasts")), collapse = ", ")
    message("Contrasts: ", contrast_names, "\n")
}
<bytecode: 0x0000012a34b5a350>
<environment: namespace:sesame>
```

print.fileSet

```
function (x, ...)
{
    message("File Set for", x$n, "probes and", x$m, "samples.\n")
}
<bytecode: 0x0000012a34b5bfa8>
<environment: namespace:sesame>
```

probeID_designType

```
function (Probe_ID)
{
    stopifnot(all(grepl("_", Probe_ID)))
    vapply(Probe_ID, function(x) substr(strsplit(x, "_")[[1]][2],
        3, 3), character(1))
}
<bytecode: 0x0000012a34b562f8>
<environment: namespace:sesame>
```

probeSuccessRate

```
function (sdf, mask = TRUE, max_pval = 0.05)
{
    pval <- pOOBAH(sdf, return.pval = TRUE)
    if (mask) {
        pval <- pval[!sdf$mask]
    }
    pval <- na.omit(pval)
    stopifnot(length(pval) > 100)
    sum(pval < max_pval)/length(pval)
}
<bytecode: 0x0000012a34b65b08>
<environment: namespace:sesame>
```

qualityMask

```
function (sdf, mask_names = "recommended", verbose = TRUE)
{
    platform <- sdfPlatform(sdf, verbose = verbose)
    masks <- getMask(platform, mask_names = mask_names)
    if (is.null(masks)) {
        return(sdf)
    }
    else {
        addMask(sdf, masks)
    }
}
<bytecode: 0x0000012a34b615a8>
<environment: namespace:sesame>
```

queryCheckPlatform

```
function (platform, query = NULL, silent = FALSE)
{
    if (is.null(platform)) {
        stopifnot(!is.null(query))
        if (is.numeric(query)) {
            platform <- inferPlatformFromProbeIDs(names(query),
                silent = silent)
        }
        else {
            platform <- inferPlatformFromProbeIDs(query, silent = silent)
        }
    }
    platform
}
<bytecode: 0x0000012a34b63350>
<environment: namespace:sesame>
```

readControls

```
function (dm, controls)
{
    if ("Color_Channel" %in% colnames(controls)) {
        ctl <- as.data.frame(dm[match(controls$Address, rownames(dm)),
            ])
        rownames(ctl) <- make.names(controls$Name, unique = TRUE)
        ctl <- cbind(ctl, controls[, c("Color_Channel", "Type")])
        colnames(ctl) <- c("G", "R", "col", "type")
        ctl <- ctl[!(is.na(ctl$G) | is.na(ctl$R)), ]
    }
    else {
        ctl <- as.data.frame(chipAddressToSignal(dm, controls))
    }
    ctl
}
<bytecode: 0x0000012a34b6cfb0>
<environment: namespace:sesame>
```

readFileSet

```
function (map_path)
{
    fset <- readRDS(paste0(map_path, "_idx.rds"))
    fset$map_path <- map_path
    fset
}
<bytecode: 0x0000012a34b69700>
<environment: namespace:sesame>
```

readIDAT

```
function (file)
{
    stopifnot(is.character(file) || length(file) != 0)
    file <- path.expand(file)
    if (!file.exists(file)) {
        stop("Unable to find file ", file)
    }
    if (grepl("\\.gz", file))
        con <- gzfile(file, "rb")
    else con <- file(file, "rb")
    on.exit({
        close(con)
    })
    magic <- readChar(con, nchars = 4)
    if (magic != "IDAT") {
        stop("Cannot read IDAT file. File format error. Unknown magic: ",
            magic)
    }
    version <- readBin(con, what = "integer", size = 4, n = 1,
        signed = TRUE, endian = "little")
    if (version == 3) {
        res <- readIDAT_nonenc(file)
    }
    else {
        stop("Cannot read IDAT file. Unsupported IDAT file format version: ",
            version)
    }
    res
}
<bytecode: 0x0000012a34b75840>
<environment: namespace:sesame>
```

readIDAT_nonenc

```
function (file, what = c("all", "IlluminaID", "nSNPsRead"))
{
    readByte <- function(con, n = 1, ...) {
        readBin(con, what = "integer", n = n, size = 1, endian = "little",
            signed = FALSE)
    }
```

```
    readShort <- function(con, n = 1, ...) {
        readBin(con, what = "integer", n = n, size = 2, endian = "little",
            signed = FALSE)
    }
    readInt <- function(con, n = 1, ...) {
        readBin(con, what = "integer", n = n, size = 4, endian = "little",
            signed = TRUE)
    }
    readLong <- function(con, n = 1, ...) {
        readBin(con, what = "integer", n = n, size = 8, endian = "little",
            signed = TRUE)
    }
    readString <- function(con, ...) {
        m <- readByte(con, n = 1)
        n <- m%%128
        shift <- 0L
        while (m%/%128 == 1) {
            m <- readByte(con, n = 1)
            shift <- shift + 7L
            k <- (m%%128) * 2^shift
            n <- n + k
        }
        readChar(con, nchars = n, useBytes = TRUE)
    }
    readField <- function(con, field) {
        switch(field, IlluminaID = readInt(con = con, n = nSNPsRead),
            SD = readShort(con = con, n = nSNPsRead), Mean = readShort(con = con,
                n = nSNPsRead), NBeads = readByte(con = con,
                n = nSNPsRead), MidBlock = {
                nMidBlockEntries <- readInt(con = con, n = 1)
                MidBlock <- readInt(con = con, n = nMidBlockEntries)
            }, RedGreen = readInt(con = con, n = 1), MostlyNull = readString(con = con),
            Barcode = readString(con = con), ChipType = readString(con = con),
            MostlyA = readString(con = con), Unknown.1 = readString(con = con),
            Unknown.2 = readString(con = con), Unknown.3 = readString(con = con),
            Unknown.4 = readString(con = con), Unknown.5 = readString(con = con),
            Unknown.6 = readString(con = con), Unknown.7 = readString(con = con),
            RunInfo = {
                nRunInfoBlocks <- max(0, readInt(con = con, n = 1))
                naValue <- as.character(NA)
                RunInfo <- matrix(naValue, nrow = nRunInfoBlocks,
                  ncol = 5)
                colnames(RunInfo) <- c("RunTime", "BlockType",
                  "BlockPars", "BlockCode", "CodeVersion")
                for (ii in seq_len(nRunInfoBlocks)) {
                  for (jj in seq_len(5)) {
                    RunInfo[ii, jj] <- readString(con = con)
                  }
                }
                RunInfo
            }, stop("readIDAT_nonenc: unknown field"))
    }
    if (!(is.character(file) || try(isOpen(file))))
```

```
        stop("argument 'file' needs to be either a character or an open, seekable connection")
    what <- match.arg(what)
    if (is.character(file)) {
        stopifnot(length(file) == 1)
        file <- path.expand(file)
        stopifnot(file.exists(file))
        fileSize <- file.info(file)$size
        if (grepl("\\.gz$", file))
            con <- gzfile(file, "rb")
        else con <- file(file, "rb")
        on.exit({
            close(con)
        })
    }
    else {
        con <- file
        fileSize <- 0
    }
    if (!isSeekable(con))
        stop("The file connection needs to be seekable")
    magic <- readChar(con, nchars = 4)
    if (magic != "IDAT") {
        stop("Cannot read IDAT file. File format error. Unknown magic: ",
            magic)
    }
    version <- readLong(con, n = 1)
    if (version < 3) {
        stop("Cannot read IDAT file. Unsupported IDAT file format version: ",
            version)
    }
    nFields <- readInt(con, n = 1)
    fields <- matrix(0, nrow = nFields, ncol = 3)
    colnames(fields) <- c("fieldCode", "byteOffset", "Bytes")
    for (ii in seq_len(nFields)) {
        fields[ii, "fieldCode"] <- readShort(con, n = 1)
        fields[ii, "byteOffset"] <- readLong(con, n = 1)
    }
    knownCodes <- c(nSNPsRead = 1000, IlluminaID = 102, SD = 103,
        Mean = 104, NBeads = 107, MidBlock = 200, RunInfo = 300,
        RedGreen = 400, MostlyNull = 401, Barcode = 402, ChipType = 403,
        MostlyA = 404, Unknown.1 = 405, Unknown.2 = 406, Unknown.3 = 407,
        Unknown.4 = 408, Unknown.5 = 409, Unknown.6 = 410, Unknown.7 = 510)
    nNewFields <- 1
    rownames(fields) <- paste("Null", seq_len(nFields))
    for (ii in seq_len(nFields)) {
        temp <- match(fields[ii, "fieldCode"], knownCodes)
        if (!is.na(temp)) {
            rownames(fields)[ii] <- names(knownCodes)[temp]
        }
        else {
            rownames(fields)[ii] <- paste("newField", nNewFields,
                sep = ".")
            nNewFields <- nNewFields + 1
```

```
        }
    }
    stopifnot(min(fields[, "byteOffset"]) == fields["nSNPsRead",
        "byteOffset"])
    seek(con, where = fields["nSNPsRead", "byteOffset"], origin = "start")
    nSNPsRead <- readInt(con, n = 1)
    if (what == "nSNPsRead")
        return(nSNPsRead)
    if (what == "IlluminaID") {
        where <- fields["IlluminaID", "byteOffset"]
        seek(con, where = where, origin = "start")
        res <- readField(con = con, field = "IlluminaID")
        return(as.character(res))
    }
    res <- rownames(fields)
    names(res) <- res
    res <- res[order(fields[res, "byteOffset"])]
    res <- res[names(res) != "nSNPsRead"]
    res <- res[c("IlluminaID", "SD", "Mean", "NBeads")]
    res <- lapply(res, function(xx) {
        where <- fields[xx, "byteOffset"]
        seek(con, where = where, origin = "start")
        readField(con = con, field = xx)
    })
    Unknowns <- list(MostlyNull = res$MostlyNull, MostlyA = res$MostlyA,
        Unknown.1 = res$Unknown.1, Unknown.2 = res$Unknown.2,
        Unknown.3 = res$Unknown.3, Unknown.4 = res$Unknown.4,
        Unknown.5 = res$Unknown.5)
    Quants <- cbind(res$Mean, res$SD, res$NBeads)
    colnames(Quants) <- c("Mean", "SD", "NBeads")
    rownames(Quants) <- as.character(res$IlluminaID)
    res <- list(fileSize = fileSize, versionNumber = version,
        nFields = nFields, fields = fields, nSNPsRead = nSNPsRead,
        Quants = Quants, MidBlock = res$MidBlock, RedGreen = res$RedGreen,
        Barcode = res$Barcode, ChipType = res$ChipType, RunInfo = res$RunInfo,
        Unknowns = Unknowns)
    res
}
<bytecode: 0x0000012a34b70010>
<environment: namespace:sesame>
```

readIDAT1

```
function (grn.name, red.name)
{
    ida.grn <- suppressWarnings(readIDAT(grn.name))
    ida.red <- suppressWarnings(readIDAT(red.name))
    d <- cbind(cy3 = ida.grn$Quants[, "Mean"], cy5 = ida.red$Quants[,
        "Mean"], cy3n = ida.grn$Quants[, "NBeads"], cy5n = ida.red$Quants[,
        "NBeads"])
    colnames(d) <- c("G", "R", "GN", "RN")
    attr(d, "platform") <- inferPlatformFromTango(ida.red)
    d
```

```
}
<bytecode: 0x0000012a34bcf8b0>
<environment: namespace:sesame>
```

readIDATpair

```
function (prefix.path, manifest = NULL, platform = "", min_beads = NULL,
    controls = NULL, verbose = FALSE)
{
    if (file.exists(paste0(prefix.path, "_Grn.idat"))) {
        grn.name <- paste0(prefix.path, "_Grn.idat")
    }
    else if (file.exists(paste0(prefix.path, "_Grn.idat.gz"))) {
        grn.name <- paste0(prefix.path, "_Grn.idat.gz")
    }
    else {
        stop("Grn IDAT does not exist")
    }
    if (file.exists(paste0(prefix.path, "_Red.idat"))) {
        red.name <- paste0(prefix.path, "_Red.idat")
    }
    else if (file.exists(paste0(prefix.path, "_Red.idat.gz"))) {
        red.name <- paste0(prefix.path, "_Red.idat.gz")
    }
    else {
        stop("Red IDAT does not exist")
    }
    if (verbose == TRUE) {
        message("Reading IDATs for ", basename(prefix.path),
            "...")
    }
    dm <- readIDAT1(grn.name, red.name)
    if (platform != "") {
        attr(dm, "platform") <- platform
    }
    else if (is.null(attr(dm, "platform"))) {
        if (!is.null(manifest)) {
            attr(dm, "platform") <- "custom"
        }
        else {
            stop("Cannot infer platform. Please provide custom manifest.")
        }
    }
    if (is.null(manifest)) {
        df_address <- sesameDataGet(paste0(attr(dm, "platform"),
            ".address"))
        manifest <- df_address$ordering
        controls <- df_address$controls
    }
    sdf <- sdfMsg(chipAddressToSignal(dm, manifest, min_beads),
        verbose, "IDAT platform: %s", attr(dm, "platform"))
    attr(sdf, "platform") <- attr(dm, "platform")
    if (!is.null(controls) && nrow(controls) > 0) {
```

```
        attr(sdf, "controls") <- readControls(dm, controls)
    }
    sdf
}
<bytecode: 0x0000012a34bcc480>
<environment: namespace:sesame>
```

recommendedMaskNames

```
function ()
{
    list(EPICv2 = c("M_1baseSwitchSNPcommon_5pt", "M_2extBase_SNPcommon_5pt",
        "M_mapping", "M_nonuniq", "M_SNPcommon_5pt"), MM285 = c("ref_issue",
        "nonunique", "design_issue"), EPIC = c("mapping", "channel_switch",
        "snp5_GMAF1p", "extension", "sub30_copy"), HM450 = c("mapping",
        "channel_switch", "snp5_GMAF1p", "extension", "sub30_copy"),
        HM27 = c("mask"))
}
<bytecode: 0x0000012a34bd8438>
<environment: namespace:sesame>
```

reIdentify

```
function (path, out_path = NULL, snps = NULL, mft = NULL)
{
    res <- suppressWarnings(readIDAT(path))
    platform <- inferPlatformFromTango(res)
    if (is.null(out_path)) {
        pfx <- sub(".idat(.gz)?$", "", path)
        if (grepl("_Grn$", pfx)) {
            out_path <- paste0(sub("_Grn$", "", pfx), "_reid_Grn.idat")
        }
        else if (grepl("_Red$", pfx)) {
            out_path <- paste0(sub("_Red$", "", pfx), "_reid_Red.idat")
        }
    }
    if (is.null(mft)) {
        mft <- sesameDataGet(paste0(platform, ".address"))$ordering
    }
    if (is.null(snps)) {
        snps <- grep("^rs", mft$Probe_ID, value = TRUE)
    }
    mft <- mft[mft$Probe_ID %in% snps, ]
    snpsTango <- na.omit(c(mft$M, mft$U))
    qt <- res$Quants
    snpsIdx <- match(snpsTango, rownames(qt))
    dt <- qt[, "Mean"]
    snpsIdx <- snpsIdx[!is.na(snpsIdx)]
    idx <- seq_along(snpsIdx)
    dt[snpsIdx] <- dt[snpsIdx[match(idx, sample(idx))]]
    if (grepl("\\.gz$", path)) {
        con <- gzfile(path, "rb")
```

```
    }
    else {
        con <- file(path, "rb")
    }
    con2 <- file(out_path, "wb")
    writeBin(readBin(con, "raw", n = res$fields["Mean", "byteOffset"]),
        con2)
    writeBin(as.integer(dt), con2, size = 2, endian = "little")
    a <- readBin(con, "raw", n = res$nSNPsRead * 2)
    while (length(a <- readBin(con, "raw", n = 1)) > 0) writeBin(a,
        con2)
    close(con)
    close(con2)
}
<bytecode: 0x0000012a34bda1e0>
<environment: namespace:sesame>
```

resetMask

```
function (sdf, verbose = FALSE)
{
    sdf$mask <- FALSE
    sdf
}
<bytecode: 0x0000012a34bed578>
<environment: namespace:sesame>
```

scrub

```
function (sdf)
{
    bG <- median(oobG(noMasked(sdf)), na.rm = TRUE)
    bR <- median(oobR(noMasked(sdf)), na.rm = TRUE)
    sdf$MG <- pmax(sdf$MG - bG, 1)
    sdf$MR <- pmax(sdf$MR - bR, 1)
    sdf$UG <- pmax(sdf$UG - bG, 1)
    sdf$UR <- pmax(sdf$UR - bR, 1)
    sdf
}
<bytecode: 0x0000012a34be9910>
<environment: namespace:sesame>
```

scrubSoft

```
function (sdf)
{
    bgR <- oobR(noMasked(sdf))
    bgG <- oobG(noMasked(sdf))
    sdf$MG <- noobSub(sdf$MG, bgG)
    sdf$MR <- noobSub(sdf$MR, bgR)
    sdf$UG <- noobSub(sdf$UG, bgG)
    sdf$UR <- noobSub(sdf$UR, bgR)
```

```
    sdf
}
<bytecode: 0x0000012a34c06150>
<environment: namespace:sesame>
```

sdf_read_table

```
function (fname, platform = NULL, verbose = FALSE, ...)
{
    df <- read.table(fname, header = TRUE, ...)
    sdf <- structure(df, class = c("SigDF", "data.frame"))
    sdf$col <- factor(sdf$col, levels = c("G", "R", "2"))
    sdf$mask <- as.logical(sdf$mask)
    if (is.null(platform)) {
        attr(sdf, "platform") <- inferPlatformFromProbeIDs(sdf$Probe_ID,
            silent = !verbose)
    }
    sdf
}
<bytecode: 0x0000012a34c0ac98>
<environment: namespace:sesame>
```

sdf_write_table

```
function (sdf, ...)
{
    write.table(sdf, row.names = FALSE, ...)
}
<bytecode: 0x0000012a34c0bc08>
<environment: namespace:sesame>
```

SDFcollapseToPfx

```
function (sdf)
{
    sdf$Probe_ID <- vapply(strsplit(sdf$Probe_ID, "_"), function(x) x[1],
        character(1))
    sdf$pval <- pOOBAH(sdf, return.pval = TRUE)
    slice_min(group_by(sdf, .data[["Probe_ID"]]), .data[["mask"]] +
        .data[["pval"]], n = 1, with_ties = FALSE)
}
<bytecode: 0x0000012a34c08198>
<environment: namespace:sesame>
```

sdfMsg

```
function (sdf, verbose, msg, ...)
{
    msg <- sprintf(msg, ...)
    msg <- sprintf("[%s] %s", Sys.time(), msg)
    attr(sdf, "msg") <- c(attr(sdf, "msg"), msg)
```

```
    if (verbose) {
        message(msg)
    }
    sdf
}
<bytecode: 0x0000012a34c1b258>
<environment: namespace:sesame>
```

sdfPlatform

```
function (sdf, verbose = FALSE)
{
    if ("platform" %in% names(attributes(sdf))) {
        attr(sdf, "platform")
    }
    else {
        inferPlatformFromProbeIDs(sdf$Probe_ID, silent = !verbose)
    }
}
<bytecode: 0x0000012a34c14d58>
<environment: namespace:sesame>
```

searchIDATprefixes

```
function (dir.name, recursive = TRUE, use.basename = TRUE)
{
    stopifnot(dir.exists(dir.name))
    paths <- list.files(dir.name, "\\.idat(.gz)?$", recursive = recursive)
    prefixes <- unique(sub("_(Grn|Red).idat(.gz)?", "", paths))
    df <- data.frame(paths = paths, prefix = sub("_(Grn|Red).idat.*",
        "", paths), channel = sub(".*_(Grn|Red).idat.*", "\\1",
        paths))
    byprefix <- split(df, df$prefix)
    is.valid <- vapply(byprefix, function(x) all(sort(x[, "channel"]) ==
        c("Grn", "Red")), logical(1))
    prefixes <- names(is.valid)[is.valid]
    if (length(prefixes) == 0)
        stop("No IDAT file found.")
    prefixes <- file.path(dir.name, prefixes)
    if (use.basename) {
        names(prefixes) <- basename(prefixes)
    }
    else {
        names(prefixes) <- prefixes
    }
    prefixes
}
<bytecode: 0x0000012a34c18c28>
<environment: namespace:sesame>
```

segmentBins

```
function (bin.signals, bin.coords)
{
    bin.coords <- bin.coords[names(bin.signals)]
    maplocs <- as.integer((GenomicRanges::start(bin.coords) +
        GenomicRanges::end(bin.coords))/2)
    cna <- DNAcopy::CNA(genomdat = bin.signals, chrom = as.character(GenomicRanges::seqnames(bin.coords)
        maploc = maplocs, data.type = "logratio")
    seg <- DNAcopy::segment(x = cna, min.width = 5, nperm = 10000,
        alpha = 0.001, undo.splits = "sdundo", undo.SD = 2.2,
        verbose = 0)
    summary <- DNAcopy::segments.summary(seg)
    pval <- DNAcopy::segments.p(seg)
    seg.signals <- cbind(summary, pval[, c("pval", "lcl", "ucl")])
    seg.signals$chrom <- as.character(seg.signals$chrom)
    seg.signals
}
<bytecode: 0x0000012a34c1c698>
<environment: namespace:sesame>
```

sesame_checkVersion

```
function ()
{
    rv <- R.Version()
    msg <- paste0("SeSAMe requires matched versions of ", "R, sesame, sesameData and ExperimentHub.\n",
        "Here is the current versions installed:\n", sprintf("R: %s.%s\n",
            rv$major, rv$minor), sprintf("Bioconductor: %s\n",
            BiocManager::version()), sprintf("sesame: %s\n",
            packageVersion("sesame")), sprintf("sesameData: %s\n",
            packageVersion("sesameData")), sprintf("ExperimentHub: %s\n",
            packageVersion("ExperimentHub")))
    message(msg)
}
<bytecode: 0x0000012a34c2a820>
<environment: namespace:sesame>
```

sesameAnno_attachManifest

```
function (df, probe_id = "Probe_ID", platform = NULL, genome = NULL)
{
    df <- as.data.frame(df)
    stopifnot(is(df, "data.frame"))
    stopifnot(probe_id %in% colnames(df))
    if (is.null(platform)) {
        platform <- inferPlatformFromProbeIDs(df[[probe_id]])
    }
    genome <- sesameData_check_genome(genome, platform)
    mft <- sesameAnno_readManifestTSV(sprintf("%s.%s.manifest",
        platform, genome))
    if (platform %in% c("HM27", "HM450")) {
        mft_probeid <- "probeID"
```

```
    }
    else {
        mft_probeid <- "Probe_ID"
    }
    cbind(df, as.data.frame(mft)[match(df[[probe_id]], mft[[mft_probeid]]),
        ])
}
<bytecode: 0x0000012a34c27d58>
<environment: namespace:sesame>
```

sesameAnno_buildAddressFile

```
function (tsv)
{
    if (is.character(tsv)) {
        tsv <- sesameAnno_readManifestTSV(tsv)
    }
    ordering <- data.frame(Probe_ID = tsv$Probe_ID, M = tsv$address_B,
        U = tsv$address_A, col = factor(tsv$channel, levels = c("G",
            "R")), mask = FALSE)
    ordering$mask <- create_default_mask(tsv)$ref_issue
    message(sprintf("%d probes masked", sum(ordering$mask)))
    message(sprintf("%d probes/rows in ordering", nrow(ordering)))
    message(sprintf("%d probes masked", sum(ordering$mask)))
    message(sprintf("%d red probes", sum(na.omit(ordering$col ==
        "R"))))
    message(sprintf("%d grn probes", sum(na.omit(ordering$col ==
        "G"))))
    ordering
}
<bytecode: 0x0000012a34c325c0>
<environment: namespace:sesame>
```

sesameAnno_buildManifestGRanges

```
function (tsv, genome = NULL, decoy = FALSE, columns = NULL)
{
    if (is.character(tsv)) {
        tsv <- sesameAnno_readManifestTSV(tsv)
    }
    chrms <- tsv$CpG_chrm
    chrms <- chrms[!is.na(chrms)]
    if (!is.null(genome) && genome %in% c("mm10", "mm39", "hg19",
        "hg38")) {
        if (decoy) {
            chrms <- c(guess_chrmorder(chrms[!grepl("_", chrms)]),
                sort(unique(chrms[grepl("_", chrms)])))
        }
        else {
            chrms <- guess_chrmorder(chrms[!grepl("_", chrms)])
        }
    }
```

```
    else {
        chrms <- sort(unique(chrms))
    }
    chrms <- c(chrms, "*")
    idx <- is.na(tsv$CpG_chrm) | !(tsv$CpG_chrm %in% chrms)
    tsv$CpG_chrm[idx] <- "*"
    tsv$CpG_beg[idx] <- -1
    tsv$CpG_end[idx] <- 0
    gr <- GRanges(tsv$CpG_chrm, IRanges::IRanges(tsv$CpG_beg +
        1, tsv$CpG_end), strand = ifelse(is.na(tsv$mapFlag_A),
        "*", ifelse(tsv$mapFlag_A == "0", "+", "-")), seqinfo = Seqinfo(chrms))
    if (length(columns) > 0) {
        SummarizedExperiment::mcols(gr) <- tsv[, columns]
    }
    names(gr) <- tsv$Probe_ID
    metadata(gr)[["genome"]] <- genome
    message(sprintf("%d probes in GRanges.", length(gr)))
    message(sprintf("%d probes belong to chr*.", sum(seqnames(gr) ==
        "*")))
    message(sprintf("%d probes on decoy chr.", sum(grepl("_",
        seqnames(gr)))))
    sort(gr, ignore.strand = TRUE)
}
<bytecode: 0x0000012a34c3c6f0>
<environment: namespace:sesame>
```

sesameAnno_download

```
function (url, destfile = tempfile(basename(url)))
{
    url <- expand_url(url)
    download.file(url, destfile = destfile)
    destfile
}
<bytecode: 0x0000012a34c3e6c8>
<environment: namespace:sesame>
```

sesameAnno_readManifestTSV

```
function (tsv_fn)
{
    if (is.character(tsv_fn) && !file.exists(tsv_fn)) {
        tsv_fn <- expand_url(tsv_fn)
        if (!valid_url(tsv_fn)) {
            stop(sprintf("File %s cannot be found.", tsv_fn))
        }
        return(sesameAnno_readManifestTSV(gzcon(url(tsv_fn))))
    }
    read_tsv(tsv_fn, col_types = cols(CpG_chrm = col_character(),
        CpG_beg = col_integer(), CpG_end = col_integer(), address_A = col_integer(),
        address_B = col_integer(), target = col_character(),
        nextBase = col_character(), channel = col_character(),
```

```
        Probe_ID = col_character(), mapFlag_A = col_integer(),
        mapChrm_A = col_character(), mapPos_A = col_integer(),
        mapQ_A = col_integer(), mapCigar_A = col_character(),
        AlleleA_ProbeSeq = col_character(), mapNM_A = col_character(),
        mapAS_A = col_integer(), mapYD_A = col_character(), mapFlag_B = col_integer(),
        mapChrm_B = col_character(), mapPos_B = col_integer(),
        mapQ_B = col_integer(), mapCigar_B = col_character(),
        AlleleB_ProbeSeq = col_character(), mapNM_B = col_character(),
        mapAS_B = col_integer(), mapYD_B = col_character(), type = col_character()))
}
<bytecode: 0x0000012a34c8c418>
<environment: namespace:sesame>
```

sesameQC_calcStats

```
function (sdf, funs = NULL)
{
    if (is.null(funs)) {
        funs <- c(sesameQC_calcStats_detection, sesameQC_calcStats_intensity,
            sesameQC_calcStats_numProbes, sesameQC_calcStats_channel,
            sesameQC_calcStats_dyeBias, sesameQC_calcStats_betas)
    }
    if (!is(funs, "list")) {
        funs <- c(funs)
    }
    qc <- new("sesameQC")
    for (func in funs) {
        if (is.character(func)) {
            func <- get(paste0("sesameQC_calcStats_", func))
            stopifnot(is(func, "function"))
        }
        qc <- func(sdf, qc = qc)
    }
    qc
}
<bytecode: 0x0000012a34c8e770>
<environment: namespace:sesame>
```

sesameQC_calcStats_betas

```
function (sdf, qc = NULL)
{
    g1 <- .setGroup_betas()
    group_nm <- names(g1)[1]
    if (is.null(qc)) {
        s <- list()
        g <- list()
    }
    else {
        s <- qc@stat
        g <- qc@group
    }
```

```
    if (group_nm %in% names(g)) {
        return(qc)
    }
    g[[group_nm]] <- g1[[group_nm]]
    betas <- getBetas(pOOBAH(noob(dyeBiasNL(sdf))))
    s$mean_beta <- mean(betas, na.rm = TRUE)
    s$median_beta <- median(betas, na.rm = TRUE)
    s$frac_unmeth <- sum(betas < 0.3, na.rm = TRUE)/sum(!is.na(betas))
    s$frac_meth <- sum(betas > 0.7, na.rm = TRUE)/sum(!is.na(betas))
    s$num_na <- sum(is.na(betas))
    s$frac_na <- sum(is.na(betas))/length(betas)
    for (pt in c("cg", "ch", "rs")) {
        b1 <- betas[grep(paste0("^", pt), names(betas))]
        s[[paste0("mean_beta_", pt)]] <- mean(b1, na.rm = TRUE)
        s[[paste0("median_beta_", pt)]] <- median(b1, na.rm = TRUE)
        s[[paste0("frac_unmeth_", pt)]] <- sum(b1 < 0.3, na.rm = TRUE)/sum(!is.na(b1))
        s[[paste0("frac_meth_", pt)]] <- sum(b1 > 0.7, na.rm = TRUE)/sum(!is.na(b1))
        s[[paste0("num_na_", pt)]] <- sum(is.na(b1))
        s[[paste0("frac_na_", pt)]] <- sum(is.na(b1))/length(b1)
    }
    new("sesameQC", stat = s, group = g)
}
<bytecode: 0x0000012a34c95a90>
<environment: namespace:sesame>
```

sesameQC_calcStats_channel

```
function (sdf, qc = NULL)
{
    g1 <- .setGroup_channel()
    group_nm <- names(g1)[1]
    if (is.null(qc)) {
        s <- list()
        g <- list()
    }
    else {
        s <- qc@stat
        g <- qc@group
    }
    if (group_nm %in% names(g)) {
        return(qc)
    }
    g[[group_nm]] <- g1[[group_nm]]
    res <- inferInfiniumIChannel(sdf, summary = TRUE)
    for (nm in names(res)) {
        s[[paste0("InfI_switch_", nm)]] <- unname(res[nm])
    }
    new("sesameQC", stat = s, group = g)
}
<bytecode: 0x0000012a34ca3a20>
<environment: namespace:sesame>
```

sesameQC_calcStats_detection

```
function (sdf, qc = NULL)
{
    g1 <- .setGroup_detection()
    group_nm <- names(g1)[1]
    if (is.null(qc)) {
        s <- list()
        g <- list()
    }
    else {
        s <- qc@stat
        g <- qc@group
    }
    if (group_nm %in% names(g)) {
        return(qc)
    }
    g[[group_nm]] <- g1[[group_nm]]
    pvals0 <- pOOBAH(sdf, return.pval = TRUE)
    pvals <- na.omit(pvals0)
    s$num_dtna <- sum(is.na(pvals0))
    s$frac_dtna <- s$num_dtna/length(pvals0)
    s$num_dt <- sum(pvals <= 0.05)
    s$frac_dt <- s$num_dt/length(pvals)
    idx_mk <- !is.na(pvals0) & !sdf$mask
    s$num_dt_mk <- sum(pvals0[idx_mk] <= 0.05)
    s$frac_dt_mk <- s$num_dt_mk/sum(idx_mk)
    for (pt in c("cg", "ch", "rs")) {
        p1 <- pvals[grep(paste0("^", pt), names(pvals))]
        s[[paste0("num_dt_", pt)]] <- sum(p1 <= 0.05)
        s[[paste0("frac_dt_", pt)]] <- sum(p1 <= 0.05)/length(p1)
    }
    new("sesameQC", stat = s, group = g)
}
<bytecode: 0x0000012a34ca82c8>
<environment: namespace:sesame>
```

sesameQC_calcStats_dyeBias

```
function (sdf, qc = NULL)
{
    g1 <- .setGroup_dyeBias()
    group_nm <- names(g1)[1]
    if (is.null(qc)) {
        s <- list()
        g <- list()
    }
    else {
        s <- qc@stat
        g <- qc@group
    }
    if (group_nm %in% names(g)) {
        return(qc)
```

```
    }
    g[[group_nm]] <- g1[[group_nm]]
    t1 <- InfI(sdf)
    intens <- totalIntensities(sdf)
    s$medR <- median(sort(intens[t1[t1$col == "R", "Probe_ID"]]))
    s$medG <- median(sort(intens[t1[t1$col == "G", "Probe_ID"]]))
    s$topR <- median(tail(sort(intens[t1[t1$col == "R", "Probe_ID"]]),
        n = 20))
    s$topG <- median(tail(sort(intens[t1[t1$col == "G", "Probe_ID"]]),
        n = 20))
    s$RGratio <- s$medR/s$medG
    s$RGdistort <- (s$topR/s$topG)/(s$medR/s$medG)
    new("sesameQC", stat = s, group = g)
}
<bytecode: 0x0000012a34cadca0>
<environment: namespace:sesame>
```

sesameQC_calcStats_intensity

```
function (sdf, qc = NULL)
{
    g1 <- .setGroup_intensity()
    group_nm <- names(g1)[1]
    if (is.null(qc)) {
        s <- list()
        g <- list()
    }
    else {
        s <- qc@stat
        g <- qc@group
    }
    if (group_nm %in% names(g)) {
        return(qc)
    }
    g[[group_nm]] <- g1[[group_nm]]
    dG <- InfIG(sdf)
    dR <- InfIR(sdf)
    d2 <- InfII(sdf)
    s$mean_intensity <- meanIntensity(sdf)
    s$mean_intensity_MU <- mean(totalIntensities(sdf), na.rm = TRUE)
    s$mean_ii <- mean(c(d2$UG, d2$UR), na.rm = TRUE)
    s$mean_inb_grn <- mean(c(dG$MG, dG$UG), na.rm = TRUE)
    s$mean_inb_red <- mean(c(dR$MR, dR$UR), na.rm = TRUE)
    s$mean_oob_grn <- mean(c(dR$MG, dR$UG), na.rm = TRUE)
    s$mean_oob_red <- mean(c(dG$MR, dG$UR), na.rm = TRUE)
    mu <- signalMU(sdf)
    s$na_intensity_M <- sum(is.na(mu$M))
    s$na_intensity_U <- sum(is.na(mu$U))
    s$na_intensity_ig <- sum(is.na(c(dG$MG, dG$MR, dG$UG, dG$UR)))
    s$na_intensity_ir <- sum(is.na(c(dR$MG, dR$MR, dR$UG, dR$UR)))
    s$na_intensity_ii <- sum(is.na(c(d2$UG, d2$UR)))
    new("sesameQC", stat = s, group = g)
}
```

```
<bytecode: 0x0000012a34cb3a68>
<environment: namespace:sesame>
```

sesameQC_calcStats_numProbes

```
function (sdf, qc = NULL)
{
    g1 <- .setGroup_numProbes()
    group_nm <- names(g1)[1]
    if (is.null(qc)) {
        s <- list()
        g <- list()
    }
    else {
        s <- qc@stat
        g <- qc@group
    }
    if (group_nm %in% names(g)) {
        return(qc)
    }
    g[[group_nm]] <- g1[[group_nm]]
    s$num_probes <- nrow(sdf)
    s$num_probes_II <- nrow(InfII(sdf))
    s$num_probes_IR <- nrow(InfIR(sdf))
    s$num_probes_IG <- nrow(InfIG(sdf))
    s$num_probes_cg <- sum(startsWith(sdf$Probe_ID, "cg"))
    s$num_probes_ch <- sum(startsWith(sdf$Probe_ID, "ch"))
    s$num_probes_rs <- sum(startsWith(sdf$Probe_ID, "rs"))
    new("sesameQC", stat = s, group = g)
}
<bytecode: 0x0000012a34cc3b90>
<environment: namespace:sesame>
```

sesameQC_getStats

```
function (qc, stat_names = NULL, drop = TRUE)
{
    if (is.null(stat_names)) {
        stat_names <- names(qc@stat)
    }
    if (length(stat_names) == 1 && drop) {
        qc@stat[[stat_names]]
    }
    else {
        qc@stat[stat_names]
    }
}
<bytecode: 0x0000012a34cc8978>
<environment: namespace:sesame>
```

sesameQC_plotBar

```
function (qcs, keys = NULL)
{
    if (is(qcs, "sesameQC")) {
        qcs <- list(qcs)
    }
    df <- do.call(rbind, lapply(qcs, function(x) as.data.frame(x@stat)))
    g <- qcs[[1]]@group
    display_nms <- do.call(c, lapply(names(g), function(gn) {
        setNames(sprintf("%s | %s", gn, str_trim(g[[gn]])), names(g[[gn]]))
    }))
    if (is.null(keys)) {
        keys <- c("frac_dt", "mean_intensity", "median_beta_cg",
            "median_beta_ch", "RGratio", "RGdistort")
        df <- df[, keys[keys %in% colnames(df)], drop = FALSE]
    }
    if (ncol(df) == 0) {
        stop("There is no QC metrics to plot")
    }
    df$sample_name <- names(qcs)
    plt <- NULL
    for (x in colnames(df)) {
        if (x == "sample_name") {
            next
        }
        p <- ggplot(df) + geom_bar(aes_string("sample_name",
            x), stat = "identity") + ylab("") + xlab("") + ggtitle(display_nms[x]) +
            theme(axis.text.x = element_text(angle = -90, vjust = 0.5,
                hjust = 0))
        if (x == "frac_dt") {
            p <- p + scale_y_continuous(labels = scales::percent)
        }
        else if (x == "median_beta_cg" || x == "median_beta_ch") {
            p <- p + ylim(c(0, 1))
        }
        if (is.null(plt)) {
            plt <- wheatmap::WGG(p)
        }
        else {
            plt <- plt + wheatmap::WGG(p, Beneath(pad = 0))
        }
    }
    plt
}
<bytecode: 0x0000012a34cca640>
<environment: namespace:sesame>
```

sesameQC_plotBetaByDesign

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

```
    if (!is.null(prep)) {
        par(mfrow = c(nchar(prep) + 1, 1), mar = mar)
        for (n in c(0, seq_len(nchar(prep)))) {
            sesameQC_plotBetaByDesign(prepSesame(sdf, substr(prep,
                1, n)), prep = NULL, legend_pos = legend_pos,
                main = sprintf("%s %s", main, substr(prep, 1,
                  n)), ...)
        }
        return(invisible(NULL))
    }
    dA <- density(na.omit(getBetas(sdf)))
    dR <- density(na.omit(getBetas(InfIR(sdf))))
    dG <- density(na.omit(getBetas(InfIG(sdf))))
    d2 <- density(na.omit(getBetas(InfII(sdf))))
    plot(dA, main = main, ylim = c(0, max(dA$y, dR$y, dG$y, d2$y)),
        ...)
    lines(dR, col = "red")
    lines(dG, col = "darkgreen")
    lines(d2, col = "blue")
    legend(legend_pos, legend = c("All", "Infinium-I Red", "Infinium-I Grn",
        "Infinium-II"), col = c("black", "red", "darkgreen",
        "blue"), lty = "solid")
}
<bytecode: 0x0000012a34d17100>
<environment: namespace:sesame>
```

sesameQC_plotHeatSNPs

```
function (sdfs, cluster = TRUE, filter.nonvariant = TRUE)
{
    afs <- openSesame(sdfs, func = getAFs, mask = FALSE)
    if (cluster) {
        afs <- both.cluster(afs)$mat
    }
    if (filter.nonvariant) {
        rg <- apply(afs, 1, function(x) {
            max(x, na.rm = TRUE) - min(x, na.rm = TRUE)
        })
        afs <- afs[rg > 0.3, ]
    }
    stopifnot(nrow(afs) > 0)
    wheatmap::WHeatmap(afs, xticklabels = TRUE, cmp = CMPar(stop.points = c("white",
        "yellow", "red"), dmin = 0, dmax = 1)) + WCustomize(mar.bottom = 0.15)
}
<bytecode: 0x0000012a34d260e0>
<environment: namespace:sesame>
```

sesameQC_plotIntensVsBetas

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

```
    if (use_max) {
        df <- signalMU(sdf)
        intens <- setNames(pmax(df$M, df$U), df$Probe_ID)
    }
    else {
        intens <- totalIntensities(sdf, mask = mask)
    }
    requireNamespace("KernSmooth")
    smoothScatter(log2(intens), getBetas(sdf, mask = mask)[names(intens)],
        xlab = "Total Intensity (Log2(M+U))", ylab = expression(paste(beta,
            " (DNA methylation Level)")), nrpoints = 0, colramp = palgen(pal),
        xlim = intens.range, ...)
    graphics::abline(h = 0.5, lty = "dashed")
    x <- c(seq(1, 100, by = 1), seq(101, 10000, by = 100))
    dG <- InfIG(sdf)
    dR <- InfIR(sdf)
    bG <- median(c(dR$MG, dR$UG), na.rm = TRUE)
    bR <- median(c(dG$MR, dG$UR), na.rm = TRUE)
    if (use_max) {
        lines(log2(pmax(bG, x + bR)), (0 + bG)/(0 + bG + x +
            bR), col = "blue")
        lines(log2(pmax(x + bG, bR)), (x + bG)/(x + bG + 0 +
            bR), col = "blue")
        lines(log2(pmax(bR, x + bR)), (0 + bR)/(x + bR + 0 +
            bR), col = "red")
        lines(log2(pmax(x + bR, bR)), (x + bR)/(x + bR + 0 +
            bR), col = "red")
        lines(log2(pmax(bG, x + bG)), (0 + bG)/(x + bG + 0 +
            bG), col = "green")
        lines(log2(pmax(x + bG, bG)), (x + bG)/(x + bG + 0 +
            bG), col = "green")
    }
    else {
        lines(log2(x + bG + bR), (0 + bG)/(0 + bG + x + bR),
            col = "blue")
        lines(log2(x + bG + bR), (x + bG)/(x + bG + 0 + bR),
            col = "blue")
        lines(log2(x + bR + bR), (0 + bR)/(x + bR + 0 + bR),
            col = "red")
        lines(log2(x + bR + bR), (x + bR)/(x + bR + 0 + bR),
            col = "red")
        lines(log2(x + bG + bG), (0 + bG)/(x + bG + 0 + bG),
            col = "green")
        lines(log2(x + bG + bG), (x + bG)/(x + bG + 0 + bG),
            col = "green")
    }
}
<bytecode: 0x0000012a34d288d0>
<environment: namespace:sesame>
```

sesameQC_plotRedGrnQQ

```
function (sdf, main = "R-G QQ Plot", ...)
{
    dG <- InfIG(noMasked(sdf))
    dR <- InfIR(noMasked(sdf))
    m <- max(c(dR$MR, dR$UR, dG$MG, dG$UG), na.rm = TRUE)
    qqplot(c(dR$MR, dR$UR), c(dG$MG, dG$UG), xlab = "Infinium-I Red Signal",
        ylab = "Infinium-I Grn Signal", main = main, cex = 0.5,
        xlim = c(0, m), ylim = c(0, m), ...)
    graphics::abline(0, 1, lty = "dashed")
}
<bytecode: 0x0000012a34d3c468>
<environment: namespace:sesame>
```

sesameQC_rankStats

```
function (qc, publicQC = NULL, platform = "EPIC")
{
    if (is.null(publicQC)) {
        publicQC <- sesameDataGet(sprintf("%s.publicQC", platform))
    }
    s <- qc@stat
    g <- qc@group
    metrics <- intersect(names(qc@stat), colnames(publicQC))
    if (length(metrics) == 0) {
        return(qc)
    }
    ranks <- lapply(metrics, function(mt) {
        ecdf(publicQC[[mt]])(qc@stat[[mt]])
    })
    names(ranks) <- paste0("rank_", metrics)
    s <- c(s, ranks)
    s$rankN <- nrow(publicQC)
    new("sesameQC", stat = s, group = g)
}
<bytecode: 0x0000012a34d3f518>
<environment: namespace:sesame>
```

sesamize

```
function (...)
{
    .Deprecated("https://github.com/zwdzwd/sesamize")
}
<bytecode: 0x0000012a34d45510>
<environment: namespace:sesame>
```

setMask

```
function (sdf, probes)
{
```

```
    addMask(resetMask(sdf), probes)
}
<bytecode: 0x0000012a34d43c58>
<environment: namespace:sesame>
```

SigDF

```
function (df, platform = "EPIC", ctl = NULL)
{
    df <- df[, c("Probe_ID", "MG", "MR", "UG", "UR", "col", "mask")]
    if (is.factor(df$col) && length(levels(df$col)) == 2) {
        df$col <- as.character(df$col)
        df$col[is.na(df$col)] <- "2"
        df$col <- factor(df$col, levels = c("G", "R", "2"))
    }
    sdf <- structure(df, class = c("SigDF", "data.frame"))
    attr(sdf, "platform") <- platform
    attr(sdf, "controls") <- ctl
    rownames(sdf) <- NULL
    sdf
}
<bytecode: 0x0000012a34d480a0>
<environment: namespace:sesame>
```

signalMU

```
function (sdf, mask = TRUE, MU = FALSE)
{
    stopifnot(all(c("MG", "UG", "MR", "UR") %in% colnames(sdf)))
    dG <- InfIG(sdf)
    dR <- InfIR(sdf)
    d2 <- InfII(sdf)
    sdf2 <- rbind(data.frame(M = dG$MG, U = dG$UG, Probe_ID = dG$Probe_ID),
        data.frame(M = dR$MR, U = dR$UR, Probe_ID = dR$Probe_ID),
        data.frame(M = d2$UG, U = d2$UR, Probe_ID = d2$Probe_ID))
    sdf2 <- sdf2[match(sdf$Probe_ID, sdf2$Probe_ID), ]
    if (mask) {
        sdf2 <- sdf2[!sdf$mask, ]
    }
    rownames(sdf2) <- NULL
    if (MU) {
        sdf2$MU <- sdf2$M + sdf2$U
    }
    sdf2
}
<bytecode: 0x0000012a34d4df80>
<environment: namespace:sesame>
```

signalMU_oo

```
function (sdf, MU = FALSE)
{
```

```
    stopifnot(all(c("MG", "UG", "MR", "UR") %in% colnames(sdf)))
    dG <- InfIG(sdf)
    dR <- InfIR(sdf)
    sdf2 <- rbind(data.frame(M = dG$MR, U = dG$UR, Probe_ID = dG$Probe_ID),
        data.frame(M = dR$MG, U = dR$UG, Probe_ID = dR$Probe_ID))
    rownames(sdf2) <- NULL
    if (MU) {
        sdf2$MU <- sdf2$M + sdf2$U
    }
    sdf2
}
<bytecode: 0x0000012a34db22e0>
<environment: namespace:sesame>
```

SigSetToSigDF

```
function (sset)
{
    df <- rbind(data.frame(Probe_ID = rownames(sset@IG), MG = sset@IG[,
        "M"], MR = sset@oobR[, "M"], UG = sset@IG[, "U"], UR = sset@oobR[,
        "U"], col = "G", mask = FALSE), data.frame(Probe_ID = rownames(sset@IR),
        MG = sset@oobG[, "M"], MR = sset@IR[, "M"], UG = sset@oobG[,
            "U"], UR = sset@IR[, "U"], col = "R", mask = FALSE),
        data.frame(Probe_ID = rownames(sset@II), MG = NA, MR = NA,
            UG = sset@II[, "M"], UR = sset@II[, "U"], col = "2",
            mask = FALSE))
    sdf <- structure(df, class = c("SigDF", "data.frame"))
    sdf$col <- factor(sdf$col, levels = c("G", "R", "2"))
    attr(sdf, "platform") <- sset@platform
    attr(sdf, "controls") <- sset@ctl
    rownames(sdf) <- NULL
    sdf
}
<bytecode: 0x0000012a34db0c38>
<environment: namespace:sesame>
```

sliceFileSet

```
function (fset, samples = fset$samples, probes = fset$probes,
    memmax = 10^5)
{
    sample_indices <- match(samples, fset$samples)
    if (any(is.na(sample_indices))) {
        message(sum(is.na(sample_indices)), " sample(s) are nonexistent")
        samples <- samples[!is.na(sample_indices)]
        sample_indices <- sample_indices[!is.na(sample_indices)]
    }
    probe_indices <- match(probes, fset$probes)
    if (any(is.na(probe_indices))) {
        message(sum(is.na(probe_indices)), " probe(s) are nonexistent")
        probes <- probes[!is.na(probe_indices)]
        probe_indices <- probe_indices[!is.na(probe_indices)]
```

```
    }
    if (length(probes) * length(samples) > memmax) {
        stop("Too many items retrieved (memmax = ", memmax, "\n")
    }
    con <- file(fset$map_path, "rb")
    res <- do.call(cbind, lapply(setNames(sample_indices, samples),
        function(s_ind) {
            vapply(probe_indices, function(p_ind) {
                binReadNumeric(con, s_ind, p_ind, fset$n, inc = fset$inc)
            }, numeric(1))
        }))
    close(con)
    rownames(res) <- probes
    res
}
<bytecode: 0x0000012a34dc17a8>
<environment: namespace:sesame>
```

species_ret

```
function (return.auc, return.species, species, auc, sdf, addr,
    verbose)
{
    if (return.auc) {
        auc
    }
    else if (return.species) {
        speciesInfo(addr, species)
    }
    else {
        updateSigDF(sdf, species = species, addr = addr, verbose = verbose)
    }
}
<bytecode: 0x0000012a34dc5cd0>
<environment: namespace:sesame>
```

speciesInfo

```
function (addr, species)
{
    res <- addr$species[[species]]
    res[c("scientificName", "taxonID", "commonName", "assembly")]
}
<bytecode: 0x0000012a34dcdc68>
<environment: namespace:sesame>
```

subsetDBs

```
function (dbs, universe)
{
    dbs <- lapply(dbs, function(db) {
        db1 <- intersect(db, universe)
```

```
        attributes(db1) <- attributes(db)
        db1
    })
    dbs <- dbs[length(dbs) > 0]
}
<bytecode: 0x0000012a34dd1ef0>
<environment: namespace:sesame>
```

summaryExtractCf

```
function (smry, contrast)
{
    cf <- do.call(rbind, lapply(smry, function(x) {
        if (x$aliased[contrast]) {
            NA
        }
        else {
            x$coefficients[contrast, ]
        }
    }))
    rownames(cf) <- names(smry)
    cf
}
<bytecode: 0x0000012a34dd95c8>
<environment: namespace:sesame>
```

summaryExtractTest

```
function (smry)
{
    contr2lvs <- attr(smry, "contr2lvs")
    smrylen <- vapply(smry, function(x) {
        nrow(x$coefficients)
    }, numeric(1))
    smry <- smry[smrylen == max(smrylen)]
    est <- do.call(bind_rows, lapply(smry, function(x) {
        x$coefficients[, "Estimate"]
    }))
    colnames(est) <- paste0("Est_", colnames(est))
    pvals <- do.call(bind_rows, lapply(smry, function(x) {
        x$coefficients[, "Pr(>|t|)"]
    }))
    colnames(pvals) <- paste0("Pval_", colnames(pvals))
    if (is.null(smry[[1]]$Ftest)) {
        return(cbind(Probe_ID = names(smry), est, pvals))
    }
    f_pvals <- do.call(rbind, lapply(smry, function(x) {
        x$Ftest["pval", , drop = FALSE]
    }))
    colnames(f_pvals) <- paste0("FPval_", colnames(f_pvals))
    contr2lvs <- contr2lvs[vapply(contr2lvs, function(x) nchar(x[[1]]),
        numeric(1)) > 0]
```

```
    if (length(contr2lvs) > 0) {
        effsize <- do.call(cbind, lapply(names(contr2lvs), function(cont) {
            lvs <- contr2lvs[[cont]]
            lvs <- lvs[2:length(lvs)]
            lvs <- lvs[paste0("Est_", cont, lvs) %in% colnames(est)]
            apply(est[, paste0("Est_", cont, lvs), drop = FALSE],
                1, function(x) {
                  max(x, 0) - min(x, 0)
                })
        }))
        colnames(effsize) <- paste0("Eff_", names(contr2lvs))
    }
    else {
        effsize <- NULL
    }
    bind_cols(Probe_ID = names(smry), est, pvals, f_pvals, effsize)
}
<bytecode: 0x0000012a34dd6978>
<environment: namespace:sesame>
```

testEnrichment

```
function (query, databases = NULL, universe = NULL, alternative = "greater",
    include_genes = FALSE, platform = NULL, silent = FALSE)
{
    platform <- queryCheckPlatform(platform, query, silent = silent)
    if (is.null(databases)) {
        dbs <- c(KYCG_getDBs(KYCG_listDBGroups(platform, type = "categorical")$Title,
            silent = silent))
    }
    else if (is.character(databases)) {
        dbs <- KYCG_getDBs(databases, platform = platform, silent = silent)
    }
    else {
        dbs <- databases
    }
    if (include_genes) {
        dbs <- c(dbs, KYCG_buildGeneDBs(query, platform, silent = silent))
    }
    dbs <- dbs[vapply(dbs, length, integer(1)) > 0]
    if (!silent) {
        message(sprintf("Testing against %d database(s)...",
            length(dbs)))
    }
    if (is.null(universe)) {
        universe <- inferUniverse(platform)
    }
    else {
        dbs <- subsetDBs(dbs, universe)
    }
    res <- do.call(bind_rows, lapply(dbs, function(db) {
        testEnrichmentFisher(query = query, database = db, universe = universe,
            alternative = alternative)
```

```
    }))
    res$FDR <- p.adjust(res$p.value, method = "fdr")
    rownames(res) <- NULL
    res <- cbind(res, databases_getMeta(dbs))
    res[order(res$log10.p.value, -abs(res$estimate)), ]
}
<bytecode: 0x0000012a34de8d88>
<environment: namespace:sesame>
```

testEnrichmentFisher

```
function (query, database, universe, alternative = "greater")
{
    nD <- length(database)
    nQ <- length(query)
    nDQ <- length(intersect(query, database))
    nU <- length(universe)
    testEnrichmentFisherN(nD, nQ, nDQ, nU, alternative = alternative)
}
<bytecode: 0x0000012a34dfa8f8>
<environment: namespace:sesame>
```

testEnrichmentFisherN

```
function (nD, nQ, nDQ, nU, alternative = "greater")
{
    nDmQ <- nD - nDQ
    nQmD <- nQ - nDQ
    nUmDQ <- nU - nQ - nD + nDQ
    if (alternative == "two.sided") {
        pvg <- phyper(nDQ - 1, nDQ + nQmD, nUmDQ + nDmQ, nDmQ +
            nDQ, lower.tail = FALSE, log.p = TRUE)/log(10)
        pvl <- phyper(nDQ, nDQ + nQmD, nUmDQ + nDmQ, nDmQ + nDQ,
            lower.tail = TRUE, log.p = TRUE)/log(10)
        log10.p.value <- pmin(pmin(pvg, pvl) + log(2), 0)/log(10)
    }
    else if (alternative == "greater") {
        log10.p.value <- phyper(nDQ - 1, nDQ + nQmD, nUmDQ +
            nDmQ, nDmQ + nDQ, lower.tail = FALSE, log.p = TRUE)/log(10)
    }
    else if (alternative == "less") {
        log10.p.value <- phyper(nDQ, nDQ + nQmD, nUmDQ + nDmQ,
            nDmQ + nDQ, lower.tail = TRUE, log.p = TRUE)/log(10)
    }
    else {
        stop("alternative must be either greater, less or two-sided.")
    }
    odds_ratio <- nDQ/nQmD/nDmQ * nUmDQ
    odds_ratio[odds_ratio == Inf] <- .Machine$double.xmax
    odds_ratio[odds_ratio == 0] <- .Machine$double.xmin
    data.frame(estimate = log2(odds_ratio), p.value = 10^(log10.p.value),
        log10.p.value = log10.p.value, test = "Log2(OR)", nQ = nQ,
```

```
        nD = nD, overlap = nDQ, cf_Jaccard = nDQ/(nD + nQmD),
        cf_overlap = nDQ/pmin(nD, nQ), cf_NPMI = (log2(nD) +
            log2(nQ) - 2 * log2(nU))/(log2(nDQ) - log2(nU)) -
            1, cf_SorensenDice = 2 * nDQ/(nD + nQ))
}
<bytecode: 0x0000012a34df4350>
<environment: namespace:sesame>
```

testEnrichmentGene

```
function (query, platform = NULL, silent = FALSE, ...)
{
    platform <- queryCheckPlatform(platform, query, silent = silent)
    dbs_gene <- KYCG_buildGeneDBs(query, platform)
    testEnrichment(query, databases = dbs_gene, platform = platform,
        ...)
}
<bytecode: 0x0000012a34e035b0>
<environment: namespace:sesame>
```

testEnrichmentSEA

```
function (query, databases, platform = NULL, silent = FALSE,
    precise = FALSE, prepPlot = FALSE)
{
    platform <- queryCheckPlatform(platform, query, silent = silent)
    stopifnot(!is.null(databases))
    if (is.character(databases)) {
        dbs <- KYCG_getDBs(databases, platform = platform, silent = silent)
    }
    else {
        dbs <- databases
    }
    dbs <- dbs[vapply(dbs, length, integer(1)) > 0]
    if (!silent) {
        message(sprintf("Testing against %d database(s)...",
            length(dbs)))
    }
    if (is.character(query) && all(vapply(dbs, is.numeric, logical(1)))) {
        res <- lapply(dbs, function(db) {
            testEnrichmentSEA1(query = db, database = query,
                precise = precise, full = prepPlot)
        })
    }
    else if (is.numeric(query) && all(vapply(dbs, is.character,
        logical(1)))) {
        res <- lapply(dbs, function(db) {
            testEnrichmentSEA1(query = query, database = db,
                precise = precise, full = prepPlot)
        })
    }
    else {
```

```
        stop("query and db must be one numerical and one categorical")
    }
    if (prepPlot) {
        return(res)
    }
    else {
        res <- do.call(bind_rows, res)
    }
    res$FDR <- p.adjust(res$p.value, method = "fdr")
    rownames(res) <- NULL
    res <- cbind(res, databases_getMeta(dbs))
    res[order(res$p.value, -abs(res$estimate)), ]
}
<bytecode: 0x0000012a34e012c8>
<environment: namespace:sesame>
```

testEnrichmentSEA1

```
function (query, database, precise = FALSE, full = FALSE)
{
    test <- "Set Enrichment Score"
    overlap <- intersect(names(query), database)
    if (length(overlap) != length(database)) {
        warning("Not every data in database has query.")
        warning(sprintf("Using %d in %d data for testing.", length(overlap),
            length(database)))
    }
    if (length(overlap) == 0 || length(overlap) == length(query)) {
        return(data.frame(estimate = 0, p.value = 1, log10.p.value = 0,
            test = test, nQ = length(database), nD = length(query),
            overlap = length(overlap)))
    }
    res <- calcES_Significance(query, overlap, precise = precise)
    if (res$es_large > res$es_small) {
        df <- data.frame(estimate = -res$es_large, p.value = res$pv_large,
            log10.p.value = log10(res$pv_large), test = test,
            nQ = length(database), nD = length(query), overlap = length(overlap))
    }
    else {
        df <- data.frame(estimate = res$es_small, p.value = res$pv_small,
            log10.p.value = log10(res$pv_small), test = test,
            nQ = length(database), nD = length(query), overlap = length(overlap))
    }
    if (full) {
        list(res = df, dCont = query, dDisc = overlap)
    }
    else {
        df
    }
}
<bytecode: 0x0000012a34e0d2f0>
<environment: namespace:sesame>
```

testEnrichmentSpearman

```
function (query, database)
{
    test <- "Spearman's rho"
    if (length(intersect(names(query), names(database))) == 0) {
        return(data.frame(estimate = 0, p.value = 1, log10.p.value = 0,
            test = test, nQ = length(query), nD = length(database),
            overlap = 0))
    }
    database <- database[match(names(query), names(database))]
    res <- cor.test(query, database, method = "spearman")
    data.frame(estimate = res$estimate[[1]], p.value = res$p.value,
        log10.p.value = log10(res$estimate[[1]]), test = test,
        nQ = length(query), nD = length(database), overlap = length(query))
}
<bytecode: 0x0000012a34e14aa8>
<environment: namespace:sesame>
```

totalIntensities

```
function (sdf, mask = FALSE)
{
    stopifnot(all(c("MG", "UG", "MR", "UR") %in% colnames(sdf)))
    s <- signalMU(sdf, mask = mask)
    setNames(s$M + s$U, s$Probe_ID)
}
<bytecode: 0x0000012a34e1b850>
<environment: namespace:sesame>
```

train.model.lm

```
function (input, output)
{
    fitdata <- data.frame(IB = as.vector(input) + 1, LOB = log(as.vector(output) +
        1))
    m <- lm(LOB ~ IB, data = fitdata)
    function(d) {
        force(m)
        pp <- predict(m, newdata = data.frame(IB = as.vector(d)),
            interval = "prediction", level = 0.8)
        list(mu = exp(pp[, "fit"]), sigma = (exp(pp[, "upr"]) -
            exp(pp[, "lwr"]))/10.13)
    }
}
<bytecode: 0x0000012a34e1f2a0>
<environment: namespace:sesame>
```

twoCompsDiff

```
function (pop1, pop2)
{
```

```
    pb <- intersect(rownames(pop1), rownames(pop2))
    pop1 <- pop1[pb, ]
    pop2 <- pop2[pb, ]
    tt <- sort(rowMeans(pop1) - rowMeans(pop2))
    res <- list(diff_1m2u = names(tail(tt, n = 1000)), diff_1u2m = names(head(tt,
        n = 1000)))
    res
}
<bytecode: 0x0000012a34e214e0>
<environment: namespace:sesame>
```

twoCompsEst2

```
function (pop1, pop2, target, use.ave = TRUE, diff_1m2u = NULL,
    diff_1u2m = NULL)
{
    pb <- intersect(intersect(rownames(pop1), rownames(pop2)),
        rownames(target))
    message(length(pb), " probes shared. Starting from there.\n")
    pop1 <- pop1[pb, ]
    pop2 <- pop2[pb, ]
    target <- target[pb, ]
    if (is.null(diff_1m2u) || is.null(diff_1u2m)) {
        if (use.ave) {
            tt <- sort(rowMeans(pop1) - rowMeans(pop2))
            diff_1m2u <- names(tail(tt, n = 1000))
            diff_1u2m <- names(head(tt, n = 1000))
        }
        else {
            diff_1u2m <- names(which(apply(pop1, 1, function(x) {
                all(x < 0.3, na.rm = TRUE) && sum(is.na(x))/length(x) <
                  0.5
            }) & apply(pop2, 1, function(x) {
                all(x > 0.7, na.rm = TRUE) && sum(is.na(x))/length(x) <
                  0.5
            })))
            diff_1m2u <- names(which(apply(pop1, 1, function(x) {
                all(x > 0.7, na.rm = TRUE) && sum(is.na(x))/length(x) <
                  0.5
            }) & apply(pop2, 1, function(x) {
                all(x < 0.3, na.rm = TRUE) && sum(is.na(x))/length(x) <
                  0.5
            })))
        }
    }
    message(length(diff_1u2m), " probes meth. in 2 and unmeth. in 1.\n")
    message(length(diff_1m2u), " probes meth. in 1 and unmeth. in 2.\n")
    d1u2m_hi <- apply(pop2[diff_1u2m, ], 1, max, na.rm = TRUE)
    d1u2m_lo <- apply(pop1[diff_1u2m, ], 1, min, na.rm = TRUE)
    d1m2u_hi <- apply(pop1[diff_1m2u, ], 1, max, na.rm = TRUE)
    d1m2u_lo <- apply(pop2[diff_1m2u, ], 1, min, na.rm = TRUE)
    est <- vapply(seq_len(ncol(target)), function(i) {
        xx <- c((target[diff_1u2m, i] - d1u2m_lo)/(d1u2m_hi -
```

```
            d1u2m_lo), 1 - (target[diff_1m2u, i] - d1m2u_lo)/(d1m2u_hi -
            d1m2u_lo))
        dd <- density(na.omit(xx))
        dd$x[which.max(dd$y)]
    }, numeric(1))
    names(est) <- colnames(target)
    est
}
<bytecode: 0x0000012a34e28950>
<environment: namespace:sesame>
```

updateSigDF

```
function (sdf, species = NULL, strain = NULL, addr = NULL, verbose = FALSE)
{
    if (!is.null(species)) {
        if (is.null(addr)) {
            addr <- sesameDataGet(sprintf("%s.addressSpecies",
                sdfPlatform(sdf, verbose = verbose)))
        }
        stopifnot(species %in% names(addr$species))
        addrS <- addr$species[[species]]
        sdf <- sdfMsg(sdf, verbose, "Update using species: %s",
            species)
    }
    else if (!is.null(strain)) {
        if (is.null(addr)) {
            addr <- sesameDataGet(sprintf("%s.addressStrain",
                sdfPlatform(sdf, verbose = verbose)))
        }
        stopifnot(strain %in% names(addr$strain))
        addrS <- addr$strain[[strain]]
        sdf <- sdfMsg(sdf, verbose, "Update using strain: %s",
            strain)
    }
    else {
        stop("Please specify a species or strain.")
    }
    m <- match(sdf$Probe_ID, addr$ordering$Probe_ID)
    m_idx <- (!is.na(m)) & !is.na(addrS$col[m]) & (sdf$col !=
        "2")
    nc <- as.character(addrS$col[m[m_idx]])
    nc[is.na(nc)] <- "2"
    sdf$col[m_idx] <- factor(nc, levels = c("G", "R", "2"))
    sdf$mask <- sdf$mask | (!is.na(m) & addrS$mask[m])
    sdf
}
<bytecode: 0x0000012a34e48640>
<environment: namespace:sesame>
```

valid_url

```
function (url_in, t = 2)
{
    con <- url(url_in)
    check <- suppressWarnings(try(open.connection(con, open = "rt",
        timeout = t), silent = TRUE)[1])
    suppressWarnings(try(close.connection(con), silent = TRUE))
    ifelse(is.null(check), TRUE, FALSE)
}
<bytecode: 0x0000012a34e50238>
<environment: namespace:sesame>
```

valleyDescent

```
function (x1, x2)
{
    m1 <- calcMode(x1)
    m2 <- calcMode(x2)
    dd <- density(na.omit(c(x1, x2)))
    dfunc <- approxfun(dd$x, dd$y)
    lo <- min(m1, m2)
    hi <- max(m1, m2)
    va <- min(dfunc(c(x1[x1 >= lo & x1 <= hi], x2[x2 >= lo &
        x2 <= hi])), na.rm = TRUE)
    va/min(dfunc(c(lo, hi)), na.rm = TRUE)
}
<bytecode: 0x0000012a34e5bd90>
<environment: namespace:sesame>
```

vcf_header

```
function (genome)
{
    c("##fileformat=VCFv4.0", sprintf("##fileDate=%s", format(Sys.time(),
        "%Y%m%d")), sprintf("##reference=%s", genome), paste0("##INFO=<ID=PVF,Number=1,Type=Float,",
        "Description=\"Pseudo Variant Frequency\">"), paste0("##INFO=<ID=GT,Number=1,Type=String,",
        "Description=\"Genotype\">"), paste0("##INFO=<ID=GS,Number=1,Type=Integer,",
        "Description=\"Genotyping score from 7 to 85\">"), paste0("##INFO=<ID=Probe_ID,Number=1,Type=St
        "Description=\"Infinium Probe ID\">"), paste0("##INFO=<ID=rs_ID,Number=1,Type=String,",
        "Description=\"Overlapping rs ID from dbSNP\">"))
}
<bytecode: 0x0000012a34e5cb08>
<environment: namespace:sesame>
```

visualizeGene

```
function (gene_name, betas, platform = NULL, genome = NULL, upstream = 2000,
    dwstream = 2000, ...)
{
    if (is.null(dim(betas))) {
        betas <- as.matrix(betas)
```

```
    }
    platform <- sesameData_check_platform(platform, rownames(betas))
    genome <- sesameData_check_genome(genome, platform)
    txns <- sesameData_getGenomeInfo(genome)$txns
    target.txns <- txns[GenomicRanges::mcols(txns)$gene_name ==
        gene_name]
    stopifnot(length(target.txns) > 0)
    target.strand <- as.character(GenomicRanges::strand(target.txns[[1]][1]))
    if (target.strand == "+") {
        pad.start <- upstream
        pad.end <- dwstream
    }
    else {
        pad.start <- dwstream
        pad.end <- upstream
    }
    merged.exons <- GenomicRanges::reduce(unlist(target.txns))
    visualizeRegion(as.character(GenomicRanges::seqnames(merged.exons[1])),
        min(GenomicRanges::start(merged.exons)) - pad.start,
        max(GenomicRanges::end(merged.exons)) + pad.end, betas,
        platform = platform, genome = genome, ...)
}
<bytecode: 0x0000012a34e56758>
<environment: namespace:sesame>
```

visualizeProbes

```
function (probeNames, betas, platform = NULL, genome = NULL,
    upstream = 1000, dwstream = 1000, ...)
{
    if (is.null(dim(betas))) {
        betas <- as.matrix(betas)
    }
    platform <- sesameData_check_platform(platform, rownames(betas))
    genome <- sesameData_check_genome(genome, platform)
    probes <- sesameData_getManifestGRanges(platform, genome)
    probeNames <- probeNames[probeNames %in% names(probes)]
    if (length(probeNames) == 0)
        stop("Probes specified are not well mapped.")
    target.probes <- probes[probeNames]
    regBeg <- min(GenomicRanges::start(target.probes)) - upstream
    regEnd <- max(GenomicRanges::end(target.probes)) + dwstream
    visualizeRegion(as.character(GenomicRanges::seqnames(target.probes[1])),
        regBeg, regEnd, betas, platform = platform, genome = genome,
        ...)
}
<bytecode: 0x0000012a34e65e18>
<environment: namespace:sesame>
```

visualizeRegion

```
function (chrm, 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, ...)
{
    if (is.null(dim(betas))) {
        betas <- as.matrix(betas)
    }
    platform <- sesameData_check_platform(platform, rownames(betas))
    genome <- sesameData_check_genome(genome, platform)
    reg <- GRanges(chrm, IRanges::IRanges(beg, end))
    genomeInfo <- sesameData_getGenomeInfo(genome)
    txns <- subsetByOverlaps(genomeInfo$txns, reg)
    probes <- sesameData_getManifestGRanges(platform, genome = genome)
    probes <- subsetByOverlaps(probes, reg)
    probes <- probes[names(probes) %in% rownames(betas)]
    if (na.rm) {
        probes <- probes[apply(betas[names(probes), ], 1, function(x) !all(is.na(x)))]
    }
    if (length(probes) == 0) {
        stop("No probe overlap region ", sprintf("%s:%d-%d",
            chrm, beg, end))
    }
    if (length(probes) > nprobes.max) {
        stop(sprintf("Too many probes (%d). Shrink region?",
            length(probes)))
    }
    plt.txns <- plotTranscripts(txns, reg, beg, end, txn.types = txn.types,
        txn.font.size = txn.font.size)
    plt.mapLines <- plotMapLines(probes, beg, end)
    plt.cytoband <- plotCytoBand(chrm, beg, end, genomeInfo)
    betas <- betas[names(probes), , drop = FALSE]
    if (cluster.samples) {
        betas <- column.cluster(betas[names(probes), , drop = FALSE])$mat
    }
    if (draw) {
        assemble_plots(betas, txns, probes, plt.txns, plt.mapLines,
            plt.cytoband, ...)
    }
    else {
        return(betas)
    }
}
<bytecode: 0x0000012a34e67e28>
<environment: namespace:sesame>
```

visualizeSegments

```
function (seg, to.plot = NULL, genes.to.label = NULL)
{
    stopifnot(is(seg, "CNSegment"))
    bin.coords <- seg$bin.coords
    bin.seqinfo <- seqinfo(bin.coords)
    bin.signals <- seg$bin.signals
```

```
    sigs <- seg$seg.signals
    total.length <- sum(as.numeric(bin.seqinfo@seqlengths), na.rm = TRUE)
    if (is.null(to.plot)) {
        to.plot <- (bin.seqinfo@seqlengths > total.length * 0.01)
    }
    else {
        to.plot <- seqnames(bin.seqinfo) %in% to.plot
    }
    seqlen <- as.numeric(bin.seqinfo@seqlengths[to.plot])
    seq.names <- bin.seqinfo@seqnames[to.plot]
    totlen <- sum(seqlen, na.rm = TRUE)
    seqcumlen <- cumsum(seqlen)
    seqstart <- setNames(c(0, seqcumlen[-length(seqcumlen)]),
        seq.names)
    bin.coords <- bin.coords[as.vector(seqnames(bin.coords)) %in%
        seq.names]
    bin.signals <- bin.signals[names(bin.coords)]
    GenomicRanges::values(bin.coords)$bin.mids <- (start(bin.coords) +
        end(bin.coords))/2
    GenomicRanges::values(bin.coords)$bin.x <- seqstart[as.character(seqnames(bin.coords))] +
        bin.coords$bin.mids
    p <- ggplot2::ggplot() + ggplot2::geom_point(ggplot2::aes(bin.coords$bin.x/totlen,
        bin.signals, color = bin.signals, alpha = I(0.8)))
    seg.beg <- (seqstart[sigs$chrom] + sigs$loc.start)/totlen
    seg.end <- (seqstart[sigs$chrom] + sigs$loc.end)/totlen
    p <- p + ggplot2::geom_segment(ggplot2::aes(x = seg.beg,
        xend = seg.end, y = sigs$seg.mean, yend = sigs$seg.mean),
        linewidth = 1, color = "blue")
    p <- p + ggplot2::scale_x_continuous(labels = seq.names,
        breaks = (seqstart + seqlen/2)/totlen) + ggplot2::theme(axis.text.x = ggplot2::element_text(ang
        hjust = 0.5))
    p <- p + ggplot2::scale_colour_gradient2(limits = c(-0.3,
        0.3), low = "red", mid = "grey", high = "green", oob = scales::squish,
        guide = guide_legend(title = "Log2 Signal Ratio")) +
        ggplot2::xlab("") + ggplot2::ylab("")
    p <- cnv_plot_extra(seg, genes.to.label, seq.names, seqstart,
        totlen, p)
    p + ggplot2::theme(panel.grid.major.x = element_blank(),
        panel.grid.minor.x = element_blank())
}
<bytecode: 0x0000012a34e77930>
<environment: namespace:sesame>
```

wrap_openSesame

```
function (x, ret)
{
    if (all(vapply(ret, is.numeric, logical(1))) && length(unique(vapply(ret,
        length, integer(1)))) == 1) {
        ret <- do.call(cbind, ret)
        if (is.null(colnames(ret)) && is.character(x) && length(x) ==
            ncol(ret)) {
            colnames(ret) <- basename(x)
```

```
        }
        ret
    }
    else {
        if (is.null(names(ret)) && is.character(x) && length(x) ==
            length(ret)) {
            names(ret) <- basename(x)
        }
        ret
    }
}
<bytecode: 0x0000012a34eac4f8>
<environment: namespace:sesame>
```

wrap_openSesame1

```
function (func, ret, ...)
{
    if (is.null(func)) {
        ret
    }
    else {
        func(ret, ...)
    }
}
<bytecode: 0x0000012a34eaac20>
<environment: namespace:sesame>
```