affxparser

April 20, 2011

---

1. Dictionary *1. Dictionary*

Description

This part describes non-obvious terms used in this package.

affxparser The name of this package.

API Application program interface, which describes the functional interface of underlying methods.

block (aka group).

BPMAP A file format containing information related to the design of the tiling arrays.

Calvin A special binary file format.

CDF A file format: chip definition file.

CEL A file format: cell intensity file.

cell (aka feature) A probe.

cell index An integer that identifies a probe uniquely.

chip An array.

chip type An identifier specifying a chip design uniquely, e.g. `"Mapping50K_Xba240"`.

DAT A file format: contains pixel intensity values collected from an Affymetrix GeneArray scanner.

feature A probe.

Fusion SDK Open-source sofware development kit (SDK) provided by Affymetrix to access their data files.

group (aka block) Defines a unique subset of the cells in a unit. Expression arrays typically only have one group per unit, whereas SNP arrays have either two or four groups per unit, one for each of the two allele times possibly repeated for both strands.

MM Mismatch-match, e.g. MM probe.

TPMAP A file format storing the relationship between (PM,MM) pairs (or PM probes) and positions on a set of sequences.

QC Quality control, e.g. QC probes and QC probe sets.

unit A probeset.

XDA A file format, aka as the binary file format.

Author(s)

Henrik Bengtsson (http://www.braju.com/R/)

`2. Cell coordinates and cell indices`

2. Cell coordinates and cell indices

Description

This part describes how Affymetrix *cells*, also known as *probes* or *features*, are addressed.

Cell coordinates

In Affymetrix data files, cells are uniquely identified by there *cell coordinates*, i.e. (x, y). For an array with $N * K$ cells in N rows and K columns, the x coordinate is an integer in $[0, K-1]$, and the y coordinate is an integer in $[0, N-1]$. The cell in the upper-left corner has coordinate $(x, y) = (0, 0)$ and the one in the lower-right corner $(x, y) = (K-1, N-1)$.

Cell indices and cell-index offsets

To simplify addressing of cells, a coordinate-to-index function is used so that each cell can be addressed using a single integer instead (of two). Affymetrix defines the *cell index*, i, of cell (x, y) as

$$i = K * y + x + 1,$$

where one is added to give indices in $[1, N * K]$. Continuing, the above definition means that cells are ordered row by row, that is from left to right and from top to bottom, starting at the upper-left corner. For example, with a chip layout $(N, K) = (1600, 1600)$ the cell at $(x, y) = (0, 0)$ has index i=1, and the cell at $(x, y) = (1599, 1599)$ has index $i = 2560000$. A cell at $(x, y) = (1498, 3)$ has index $i = 6299$.

Given the cell index i, the coordinate (x, y) can be calculated as

$$y = floor((i-1)/K)$$

$$x = (i-1) - K * y.$$

Continuing the above example, the coordinate for cell $i = 1$ is be found to be $(x, y) = (0, 0)$, for cell $i = 2560000$ it is $(x, y) = (1599, 1599)$, for cell $i = 6299$ is it $(x, y) = (1498, 3)$.

Although not needed to use the methods in this package, to get the cell indices for the cell coordinates or vice versa, see `xy2indices()` and `indices2xy()` in the **affy** package.

Author(s)

Henrik Bengtsson (http://www.braju.com/R/)

`affxparser-package` *Package affxparser*

Description

The **affxparser** package provides methods for fast and memory efficient parsing of Affymetrix files [1] using the Affymetrix' Fusion SDK [2]. Both traditional ASCII- and binary (XDA)-based files are supported, as well as Affymetrix future binary format "Calvin". The efficiency of the parsing is dependent on whether a specific file is binary or ASCII.

Currently, there are methods for reading chip definition file (CDF) and a cell intensity file (CEL). These files can be read either in full or in part. For example, probe signals from a few probesets can be extracted very quickly from a set of CEL files into a convenient list structure.

Requirements

This package requires only a standard R installation, that is, it works independently of other CRAN and Bioconductor packages.

To get started

To get started, see:

1. `readCelUnits`() - reads one or several Affymetrix CEL file probeset by probeset.
2. `readCel`() - reads an Affymetrix CEL file. by probe.
3. `readCdf`() - reads an Affymetrix CDF file. by probe.
4. `readCdfUnits`() - reads an Affymetrix CDF file unit by unit.
5. `readCdfCellIndices`() - Like `readCdfUnits()`, but returns cell indices only, which is often enough to read CEL files unit by unit.
6. `applyCdfGroups`() - Re-arranges a CDF structure.
7. `findCdf`() - Locates an Affymetrix CDF file by chip type. This page also describes how to setup default search path for CDF files.

Setting up the CDF search path

Some of the functions in this package search for CDF files automatically by scanning certain directories. To add directories to the default search path, see instructions in `findCdf`().

Future Work

Other Affymetrix files can be parsed using the Fusion SDK. Given sufficient interest we will implement this, e.g. DAT files (image files).

Running examples

In order to run the examples, data files must exists in the current directory. Otherwise, the example scripts will do nothing. Most of the examples requires a CDF file or a CEL file, or both. Make sure the CDF file is of the same chip type as the CEL file.

Affymetrix provides data sets of different types at `http://www.affymetrix.com/support/datasets.affx` that can be used. There are both small are very large data sets available.

Tecnical details

This package implements an interface to the Fusion SDK from Affymetrix.com. This SDK (software development kit) is an open source library used for parsing the various files formats used by the Affymetrix platform.

The intention is to provide interfaces to most if not all file formats which may be parsed using Fusion.

The SDK supports parsing of all the different versions of a specific fileformat. This means that ASCII, binary as well as the new binary format (codename Calvin) used by Affymetrix is supported through a single API. We also expect any future changes to the file formats to be reflected in the SDK, and subsequently in this package.

However, as the current Fusion SDK does not support compressed files, neither does **affxparser**. This is in contrast to some of the existing code in **affy** and relatives (see below for links).

In general we aim to provide functions returning all information in the respective files. Currently it seems that future Affymetrix chip designs may consists of so many features that returning all information will lead to an unnecessary overhead in the case a user only wants access to a subset. We have tried to make this possible.

For older file, certain entries in the files have been removed from newer specifications, and the SDK does not provide utilities for reading these entries. This includes eg. the FEAT column of CDF files.

Currently the package as well as the Fusion SDK is in beta stage. Bugs may be related to either codebase. We are very interested in users being unable to compile/parse files using this library - this includes users with custom chip designs.

In addition, since we aim to return all information stored in the file (and accessible using the Fusion SDK) we would like reports from users being unable to do that.

The efficiency of the underlying code may vary with the version of the file being parsed. For example, we currently report the number of outliers present in a CEL file when reading the header of the file using `readCelHeader`. In order to obtain this information from text based CEL files (version 2), the entire file needs to be read into memory. With version 3 of the file format, this information is stored in the header.

With the introduction of the Fusion SDK (and the next version of their file formats) Affymetrix has made it possible to use multibyte character sets. This implies that character information may be inaccesible if the compiler used to compile the C++ code does not support multibyte character sets (specifically we require that the R installation has defined the macro `SUPPORT_MCBS` in the `Rconfig.h` header file). For example GCC needs to be version 3.4 or greater on Solaris.

In the `info` subdirectory of the package installation, information regarding changes to the Fusion SDK is stored, e.g.

```
pathname <- system.file("info/README", package="affxparser"))
file.show(pathname)
```

Acknowledgments

We would like to thanks Ken Simpson (WEHI, Melbourne) and Seth Falcon (FHCRC, Seattle) for feedback and code contributions.

License

Author(s)

Henrik Bengtsson, <hb@stat.berkeley.edu>, James Bullard, <bullard@stat.berkeley.edu> and Kasper Daniel Hansen, <khansen@stat.berkeley.edu>.

References

[1] Affymetrix Inc, Affymetrix GCOS 1.x compatible file formats, April, 2006. http://www.affymetrix.com/support/developer/
[2] Affymetrix Inc, Fusion Software Developers Kit (SDK), 2006. http://www.affymetrix.com/support/developer/fusion/

---

applyCdfGroupFields

Applies a function to a list of fields of each group in a CDF structure

---

Description

Applies a function to a list of fields of each group in a CDF structure.

Usage

```
applyCdfGroupFields(cdf, fcn, ...)
```

Arguments

cdf	A CDF list structure.
fcn	A function that takes a list structure of fields and returns an updated list of fields.
...	Arguments passed to the fcn function.

Value

Returns an updated CDF list structure.

Author(s)

Henrik Bengtsson (http://www.braju.com/R/)

See Also

applyCdfGroups().

`applyCdfGroups` *Applies a function over the groups in a CDF structure*

Description

Applies a function over the groups in a CDF structure.

Usage

```
applyCdfGroups(cdf, fcn, ...)
```

Arguments

`cdf`	A CDF `list` structure.
`fcn`	A `function` that takes a `list` structure of group elements and returns an updated `list` of groups.
`...`	Arguments passed to the `fcn` function.

Value

Returns an updated CDF `list` structure.

Pre-defined restructuring functions

- Generic:
  - `cdfGetFields`() - Gets a subset of groups fields in a CDF structure.
  - `cdfGetGroups`() - Gets a subset of groups in a CDF structure.
  - `cdfOrderBy`() - Orders the fields according to the value of another field in the same CDF group.
  - `cdfOrderColumnsBy`() - Orders the columns of fields according to the values in a certain row of another field in the same CDF group.
- Designed for SNP arrays:
  - `cdfAddBaseMmCounts`() - Adds the number of allele A and allele B mismatching nucleotides of the probes in a CDF structure.
  - `cdfAddProbeOffsets`() - Adds probe offsets to the groups in a CDF structure.
  - `cdfGtypeCelToPQ`() - Function to immitate Affymetrix' `gtype_cel_to_pq` software.
  - `cdfMergeAlleles`() - Function to join CDF allele A and allele B groups strand by strand.
  - `cdfMergeStrands`() - Function to join CDF groups with the same names.

We appreciate contributions.

Author(s)

Henrik Bengtsson (`http://www.braju.com/R/`)

Examples

```
##############################################################
if (require("AffymetrixDataTestFiles")) {            # START #
##############################################################

cdfFile <- findCdf("Mapping10K_Xba131")

# Identify the unit index from the unit name
unitName <- "SNP_A-1509436"
unit <- which(readCdfUnitNames(cdfFile) == unitName)

# Read the CDF file
cdf0 <- readCdfUnits(cdfFile, units=unit, stratifyBy="pmmm", readType=FALSE, readDirectio
cat("Default CDF structure:\n")
print(cdf0)

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Tabulate the information in each group
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cdf <- readCdfUnits(cdfFile, units=unit)
cdf <- applyCdfGroups(cdf, lapply, as.data.frame)
print(cdf)

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Infer the (true or the relative) offset for probe quartets.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cdf <- applyCdfGroups(cdf0, cdfAddProbeOffsets)
cat("Probe offsets:\n")
print(cdf)

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Identify the number of nucleotides that mismatch the
# allele A and the allele B sequences, respectively.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cdf <- applyCdfGroups(cdf, cdfAddBaseMmCounts)
cat("Allele A & B target sequence mismatch counts:\n")
print(cdf)



# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Combine the signals from  the sense and the anti-sense
# strands in a SNP CEL files.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# First, join the strands in the CDF structure.
cdf <- applyCdfGroups(cdf, cdfMergeStrands)
cat("Joined CDF structure:\n")
print(cdf)


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Rearrange values of group fields into quartets.  This
# requires that the values are already arranged as PMs and MMs.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
cdf <- applyCdfGroups(cdf0, cdfMergeAlleles)
cat("Probe quartets:\n")
```

```
print(cdf)


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Get the x and y cell locations (note, zero-based)
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
x <- unlist(applyCdfGroups(cdf, cdfGetFields, "x"), use.names=FALSE)
y <- unlist(applyCdfGroups(cdf, cdfGetFields, "y"), use.names=FALSE)

# Validate
ncol <- readCdfHeader(cdfFile)$cols
cells <- as.integer(y*ncol+x+1)
cells <- sort(cells)

cells0 <- readCdfCellIndices(cdfFile, units=unit)
cells0 <- unlist(cells0, use.names=FALSE)
cells0 <- sort(cells0)

stopifnot(identical(cells0, cells))

##############################################################
}                                                    # STOP #
##############################################################
```

`compareCdfs` *Compares the contents of two CDF files*

Description

Compares the contents of two CDF files.

Usage

```
compareCdfs(pathname, other, quick=FALSE, verbose=0, ...)
```

Arguments

`pathname`	The pathname of the first CDF file.
`other`	The pathname of the seconds CDF file.
`quick`	If `TRUE`, only a subset of the units are compared, otherwise all units are compared.
`verbose`	An `integer`. The larger the more details are printed.
`...`	Not used.

Details

The comparison is done with an upper-limit memory usage, regardless of the size of the CDFs.

Value

Returns `TRUE` if the two CDF are equal, otherwise `FALSE`. If `FALSE`, the attribute `reason` contains a string explaining what difference was detected, and the attributes `value1` and `value2` contain the two objects/values that differs.

Author(s)

Henrik Bengtsson (http://www.braju.com/R/)

See Also

convertCdf().

---

compareCels *Compares the contents of two CEL files*

Description

Compares the contents of two CEL files.

Usage

```
compareCels(pathname, other, readMap=NULL, otherReadMap=NULL, verbose=0, ...)
```

Arguments

pathname	The pathname of the first CEL file.
other	The pathname of the seconds CEL file.
readMap	An optional read map for the first CEL file.
otherReadMap	An optional read map for the second CEL file.
verbose	An integer. The larger the more details are printed.
...	Not used.

Value

Returns TRUE if the two CELs are equal, otherwise FALSE. If FALSE, the attribute reason contains a string explaining what difference was detected, and the attributes value1 and value2 contain the two objects/values that differs.

Author(s)

Henrik Bengtsson (http://www.braju.com/R/)

See Also

convertCel().

convertCdf *Converts a CDF into the same CDF but with another format*

Description

Converts a CDF into the same CDF but with another format. Currently only CDF files in version 4 (binary/XDA) can be written. However, any input format is recognized.

Usage

```
convertCdf(filename, outFilename, version="4", force=FALSE, ..., .validate=TRUE,
```

Arguments

filename	The pathname of the original CDF file.
outFilename	The pathname of the destination CDF file. If the same as the source file, an exception is thrown.
version	The version of the output file format.
force	If FALSE, and the version of the orignal CDF is the same as the output version, the new CDF will not be generated, otherwise it will.
...	Not used.
.validate	If TRUE, a consistency test between the generated and the original CDF is performed. Note that the memory overhead for this can be quite large, because two complete CDF structures are kept in memory at the same time.
verbose	If TRUE, extra details are written while processing.

Value

Returns (invisibly) TRUE if a new CDF was generated, otherwise FALSE.

Benchmarking of ASCII and binary CDFs

Binary CDFs are much faster to read than ASCII CDFs. Here are some example for reading complete CDFs (the differnce is even larger when reading CDFs in subsets):

- HG-U133A (22283 units): ASCII 11.7s (9.3x), binary 1.20s (1x).
- Hu6800 (7129 units): ASCII 3.5s (6.1x), binary 0.57s (1x).

Confirmed convertions to binary (XDA) CDFs

The following chip types have been converted using convertCdf() and then verified for correctness using compareCdfs(): ASCII-to-binary: HG-U133A, Hu6800. Binary-to-binary: Test3.

Author(s)

Henrik Bengtsson (http://www.braju.com/R/)

See Also

See compareCdfs() to compare two CDF files. writeCdf().

Examples

```
##############################################################
if (require("AffymetrixDataTestFiles")) {            # START #
##############################################################


chipType <- "Test3"
cdfFiles <- findCdf(chipType, firstOnly=FALSE)
cdfFiles <- list(
  ASCII=grep("ASCII", cdfFiles, value=TRUE),
  XDA=grep("XDA", cdfFiles, value=TRUE)
)

outFile <- file.path(tempdir(), sprintf("%s.cdf", chipType))
convertCdf(cdfFiles$ASCII, outFile, verbose=TRUE)

##############################################################
}                                                    # STOP #
##############################################################
```

`convertCel` *Converts a CEL into the same CEL but with another format*

Description

Converts a CEL into the same CEL but with another format. Currently only CEL files in version 4 (binary/XDA) can be written. However, any input format is recognized.

Usage

```
convertCel(filename, outFilename, readMap=NULL, writeMap=NULL, version="4", newC
```

Arguments

`filename`	The pathname of the original CEL file.
`outFilename`	The pathname of the destination CEL file. If the same as the source file, an exception is thrown.
`readMap`	An optional read map for the input CEL file.
`writeMap`	An optional write map for the output CEL file.
`version`	The version of the output file format.
`newChipType`	An optional string for overriding the chip type in the CEL file header.
`...`	Not used.
`.validate`	If `TRUE`, a consistency test between the generated and the original CEL is performed.
`verbose`	If `TRUE`, extra details are written while processing.

Value

Returns (invisibly) `TRUE` if a new CEL was generated, otherwise `FALSE`.

Benchmarking of ASCII and binary CELs

Binary CELs are much faster to read than ASCII CELs. Here are some example for reading complete CELs (the differnce is even larger when reading CELs in subsets):

- To do

Author(s)

Henrik Bengtsson (http://www.braju.com/R/)

See Also

createCel().

Examples

```
##########################################################
if (require("AffymetrixDataTestFiles")) {            # START #
##########################################################

# Search for some available Calvin CEL files
path <- system.file("rawData", package="AffymetrixDataTestFiles")
files <- findFiles(pattern="[.](cel|CEL)$", path=path, recursive=TRUE, firstOnly=FALSE)
files <- grep("FusionSDK_Test3", files, value=TRUE)
files <- grep("Calvin", files, value=TRUE)
file <- files[1]


outFile <- file.path(tempdir(), gsub("[.]CEL$", ",XBA.CEL", basename(file)))
if (file.exists(outFile))
  file.remove(outFile)
convertCel(file, outFile, .validate=TRUE)


##########################################################
}                                                    # STOP #
##########################################################
```

---

createCel *Creates an empty CEL file*

Description

Creates an empty CEL file.

Usage

```
createCel(filename, header, nsubgrids=0, overwrite=FALSE, ..., verbose=FALSE)
```

Arguments

filename	The filename of the CEL file to be created.
header	A list structure describing the CEL header, similar to the structure returned by readCelHeader(). This header can be of any CEL header version.
overwrite	If FALSE and the file already exists, an exception is thrown, otherwise the file is created.
nsubgrids	The number of subgrids.
...	Not used.
verbose	An integer specifying how much verbose details are outputted.

Details

Currently only binary (v4) CEL files are supported. The current version of the method does not make use of the Fusion SDK, but its own code to create the CEL file.

Value

Returns (invisibly) the pathname of the file created.

Redundant fields in the CEL header

There are a few redundant fields in the CEL header. To make sure the CEL header is consistent, redundant fields are cleared and regenerated. For instance, the field for the total number of cells is calculated from the number of cell rows and columns.

Author(s)

Henrik Bengtsson (http://www.braju.com/R/)

Examples

```
##############################################################
if (require("AffymetrixDataTestFiles")) {            # START #
##############################################################

# Search for first available ASCII CEL file
path <- system.file("rawData", package="AffymetrixDataTestFiles")
files <- findFiles(pattern="[.](cel|CEL)$", path=path, recursive=TRUE, firstOnly=FALSE)
files <- grep("ASCII", files, value=TRUE)
file <- files[1]


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Read the CEL header
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
hdr <- readCelHeader(file)

# Assert that we found an ASCII CEL file, but any will do
stopifnot(hdr$version == 3)


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Create a CEL v4 file of the same chip type
```

```
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
outFile <- file.path(tempdir(), "zzz.CEL")
if (file.exists(outFile))
  file.remove(outFile)
createCel(outFile, hdr, overwrite=TRUE)
str(readCelHeader(outFile))

# Verify correctness by update and re-read a few cells
intensities <- as.double(1:100)
indices <- seq(along=intensities)
updateCel(outFile, indices=indices, intensities=intensities)
value <- readCel(outFile, indices=indices)$intensities
stopifnot(identical(intensities, value))


##############################################################
}                                                    # STOP #
##############################################################
```

`findCdf` *Search for CDF files in multiple directories*

Description

Search for CDF files in multiple directories.

Usage

```
findCdf(chipType=NULL, paths=NULL, recursive=TRUE, pattern="[.](c|C)(d|D)(f|F)$"
```

Arguments

`chipType`	A `character` string of the chip type to search for.
`paths`	A `character vector` of paths to be searched. The current directory is always searched at the beginning. If `NULL`, default paths are searched. For more details, see below.
`recursive`	If `TRUE`, directories are searched recursively.
`pattern`	A regular expression file name pattern to match.
`...`	Additional arguments passed to `findFiles`().

Details

Note, the current directory is always searched first, but never recursively (unless it is added to the search path explicitly). This provides an easy way to override other files in the search path.

If `paths` is `NULL`, then a set of default paths are searched. The default search path consitutes:

1. `getOption("AFFX_CDF_PATH")`
2. `Sys.getenv("AFFX_CDF_PATH")`

One of the easiest ways to set system variables for R is to set them in an `.Renviron` file, e.g.

```
# affxparser: Set default CDF path
AFFX_CDF_PATH=${AFFX_CDF_PATH};M:/Affymetrix_2004-100k_trios/cdf
AFFX_CDF_PATH=${AFFX_CDF_PATH};M:/Affymetrix_2005-500k_data/cdf
```

See Startup for more details.

Value

Returns a vector of the full pathnames of the files found.

Author(s)

Henrik Bengtsson (http://www.braju.com/R/)

See Also

This method is used internally by readCelUnits() if the CDF file is not specified.

Examples

```
##############################################################
if (require("AffymetrixDataTestFiles")) {            # START #
##############################################################

# Find a specific CDF file
cdfFile <- findCdf("Mapping10K_Xba131")
print(cdfFile)

# Find the first CDF file (no matter what it is)
cdfFile <- findCdf()
print(cdfFile)

# Find all CDF files in search path and display their headers
cdfFiles <- findCdf(firstOnly=FALSE)
for (cdfFile in cdfFiles) {
  cat("====================================\n")
  hdr <- readCdfHeader(cdfFile)
  str(hdr)
}

##############################################################
}                                                    # STOP #
##############################################################
```

---

readBpmap *Parses a Bpmap file*

Description

Parses (parts of) a Bpmap (binary probe mapping) file from Affymetrix.

Usage

```
readBpmap(filename, seqIndices = NULL, readProbeSeq = TRUE, readSeqInfo
= TRUE, readPMXY = TRUE, readMMXY = TRUE, readStartPos = TRUE,
readCenterPos = FALSE, readStrand = TRUE, readMatchScore = FALSE,
readProbeLength = FALSE, verbose = 0)

readBpmapHeader(filename)

readBpmapSeqinfo(filename, seqIndices = NULL, verbose = 0)
```

Arguments

`filename`	The filename as a character.
`seqIndices`	A vector of integers, detailing the indices of the sequences being read. If `NULL`, the entire file is being read.
`readProbeSeq`	Do we read the probe sequences.
`readSeqInfo`	Do we read the sequence information (a list containing information such as sequence name, number of hits etc.)
`readPMXY`	Do we read the (x,y) coordinates of the PM-probes.
`readMMXY`	Do we read the (x,y) coordinates of the MM-probes (only relevant if the file has MM information)
`readStartPos`	Do we read the start position of the probes.
`readCenterPos`	Do we return the start position of the probes.
`readStrand`	Do we return the strand of the hits.
`readMatchScore`	Do we return the matchscore.
`readProbeLength`	Doe we return the probelength.
`verbose`	How verbose do we want to be.

Details

`readBpmap` reads a BPMAP file, which is a binary file containing information about a given probe's location in a sequence. Here sequence means some kind of reference sequence, typically a chromosome or a scaffold. `readBpmapHeader` reads the header of the BPMAP file, and `readBpmapSeqinfo` reads the sequence info of the sequences (so this function is merely a convinience function).

Value

For `readBpmap`: A list of lists, one list for every sequence read. The components of the sequence lists, depends on the argument of the function call. For `readBpmapheader` a list with two components `version` and `numSequences`. For `readBpmapSeqinfo` a list of lists containing the sequence info.

Author(s)

Kasper Daniel Hansen <khansen@stat.berkeley.edu>

See Also

tpmap2bpmap for information on how to write Bpmap files.

---

readCcgHeader *Reads an the header of an Affymetrix Command Console Generic (CCG) file*

Description

Reads an the header of an Affymetrix Command Console Generic (CCG) file.

Usage

```
readCcgHeader(pathname, verbose=0, .filter=list(fileHeader = TRUE, dataHeader =
```

Arguments

pathname	The pathname of the CCG file.
verbose	An integer specifying the verbose level. If 0, the file is parsed quietly. The higher numbers, the more details.
.filter	A list.
...	Not used.

Details

Note, the current implementation of this methods does not utilize the Affymetrix Fusion SDK library. Instead, it is implemented in R from the file format definition [1].

Value

A named list structure consisting of ...

Author(s)

Henrik Bengtsson (http://www.braju.com/R/)

References

[1] Affymetrix Inc, Affymetrix GCOS 1.x compatible file formats, April, 2006. http://www.affymetrix.com/support/developer/

See Also

readCcg().

`readCcg` *Reads an Affymetrix Command Console Generic (CCG) Data file*

Description

Reads an Affymetrix Command Console Generic (CCG) Data file. The CCG data file format is also known as the Calvin file format.

Usage

```
readCcg(pathname, verbose=0, .filter=NULL, ...)
```

Arguments

`pathname`	The pathname of the CCG file.
`verbose`	An `integer` specifying the verbose level. If 0, the file is parsed quietly. The higher numbers, the more details.
`.filter`	A `list`.
`...`	Not used.

Details

Note, the current implementation of this methods does not utilize the Affymetrix Fusion SDK library. Instead, it is implemented in R from the file format definition [1].

Value

A named `list` structure consisting of ...

About the CCG file format

A CCG file, consists of a "file header", a "generic data header", and "data" section, as outlined here:

- File Header
- Generic Data Header (for the file)
  1. Generic Data Header (for the files 1st parent)
     (a) Generic Data Header (for the files 1st parents 1st parent)
     (b) Generic Data Header (for the files 1st parents 2nd parent)
     (c) ...
     (d) Generic Data Header (for the files 1st parents Mth parent)
  2. Generic Data Header (for the files 2nd parent)
  3. ...
  4. Generic Data Header (for the files Nth parent)
- Data
  1. Data Group \#1
     (a) Data Set \#1
     – Parameters
     – Column definitions

- Matrix of data

(b) Data Set \#2
(c) ...
(d) Data Set \#L

2. Data Group \#2
3. ...
4. Data Group \#K

Author(s)

Henrik Bengtsson (http://www.braju.com/R/)

References

[1] Affymetrix Inc, Affymetrix GCOS 1.x compatible file formats, April, 2006. http://www.affymetrix.com/support/developer/

See Also

readCcgHeader(). readCdfUnits().

`readCdfCellIndices` *Reads cell indices of units (probesets) in an Affymetrix CDF file*

Description

Reads cell indices of units (probesets) in an Affymetrix CDF file.

Usage

```
readCdfCellIndices(filename, units=NULL, stratifyBy=c("nothing", "pmmm", "pm", "
```

Arguments

`filename`	The filename of the CDF file.
`units`	An `integer vector` of unit indices specifying which units to be read. If `NULL`, all units are read.
`stratifyBy`	A `character` string specifying which and how elements in group fields are returned. If `"nothing"`, elements are returned as is, i.e. as `vector`s. If `"pm"/"mm"`, only elements corresponding to perfect-match (PM) / mismatch (MM) probes are returned (as `vector`s). If `"pmmm"`, elements are returned as a matrix where the first row holds elements corresponding to PM probes and the second corresponding to MM probes. Note that in this case, it is assumed that there are equal number of PMs and MMs; if not, an error is generated. Moreover, the PMs and MMs may not even be paired, i.e. there is no guarantee that the two elements in a column corresponds to a PM-MM pair.
`verbose`	An `integer` specifying the verbose level. If 0, the file is parsed quietly. The higher numbers, the more details.

Value

A named list where the names corresponds to the names of the units read. Each unit element of the list is in turn a list structure with one element groups which in turn is a list. Each group element in groups is a list with a single field named indices. Thus, the structure is

```
cdf
 +- unit #1
 |   +- "groups"
 |       +- group #1
 |       |   +- "indices"
 |       |  group #2
 |       |   +- "indices"
 |       .
 |       +- group #K
 |           +- "indices"
 +- unit #2
 .
 +- unit #J
```

This is structure is compatible with what readCdfUnits() returns.

Author(s)

Henrik Bengtsson (http://www.braju.com/R/)

See Also

readCdfUnits().

---

readCdfGroupNames *Reads group names for a set of units (probesets) in an Affymetrix CDF file*

Description

Reads group names for a set of units (probesets) in an Affymetrix CDF file.

This is for instance useful for SNP arrays where the nucleotides used for the A and B alleles are the same as the group names.

Usage

```
readCdfGroupNames(filename, units=NULL, truncateGroupNames=TRUE, verbose=0)
```

Arguments

filename	The filename of the CDF file.
units	An integer vector of unit indices specifying which units to be read. If NULL, all units are read.

truncateGroupNames
: A logical variable indicating whether unit names should be stripped from the beginning of group names.

verbose
: An integer specifying the verbose level. If 0, the file is parsed quietly. The higher numbers, the more details.

Value

A named list structure where the names of the elements are the names of the units read. Each element is a character vector with group names for the corrsponding unit.

Author(s)

Henrik Bengtsson (http://www.braju.com/R/)

See Also

readCdfUnits().

---

readCdfHeader *Reads the header associated with an Affymetrix CDF file*

Description

Reads the header of an Affymetrix CDF file using the Fusion SDK.

Usage

```
readCdfHeader(filename)
```

Arguments

filename
: name of the CDF file.

Value

A named list with the following components:

rows
: the number of rows on the chip.

cols
: the number of columns on the chip.

probesets
: the number of probesets on the chip.

qcprobesets
: the number of QC probesets on the chip.

reference
: the reference sequence (this component only exists for resequencing chips).

chiptype
: the type of the chip.

filename
: the name of the cdf file.

Author(s)

James Bullard, <bullard@stat.berkeley.edu> and Kasper Daniel Hansen, <khansen@stat.berkeley.e

See Also

readCdfUnits().

Examples

```
for (zzz in 0) {

# Find any CDF file
cdfFile <- findCdf()
if (is.null(cdfFile))
  break

header <- readCdfHeader(cdfFile)
print(header)

} # for (zzz in 0)
```

---

readCdfUnitNames *Reads unit (probeset) names from an Affymetrix CDF file*

Description

Gets the names of all or a subset of units (probesets) in an Affymetrix CDF file. This can be used to get a map between unit names an the internal unit indices used by the CDF file.

Usage

```
readCdfUnitNames(filename, units=NULL, verbose=0)
```

Arguments

filename	The filename of the CDF file.
units	An integer vector of unit indices specifying which units to be read. If NULL, all units are read.
verbose	An integer specifying the verbose level. If 0, the file is parsed quietly. The higher numbers, the more details.

Value

A character vector of unit names.

Author(s)

Henrik Bengtsson (http://www.braju.com/R/)

See Also

readCdfUnits().

Examples

```
## Not run: See help(readCdfUnits) for an example
```

readCdfUnits *Reads units (probesets) from an Affymetrix CDF file*

Description

Reads units (probesets) from an Affymetrix CDF file. Gets all or a subset of units (probesets).

Usage

```
readCdfUnits(filename, units=NULL, readXY=TRUE, readBases=TRUE, readExpos=TRUE,
```

Arguments

filename	The filename of the CDF file.
units	An integer vector of unit indices specifying which units to be read. If NULL, all units are read.
readXY	If TRUE, cell row and column (x,y) coordinates are retrieved, otherwise not.
readBases	If TRUE, cell P and T bases are retrieved, otherwise not.
readExpos	If TRUE, cell "expos" values are retrieved, otherwise not.
readType	If TRUE, unit types are retrieved, otherwise not.
readDirection	If TRUE, unit *and* group directions are retrieved, otherwise not.
stratifyBy	A character string specifying which and how elements in group fields are returned. If "nothing", elements are returned as is, i.e. as vectors. If "pm"/"mm", only elements corresponding to perfect-match (PM) / mismatch (MM) probes are returned (as vectors). If "pmmm", elements are returned as a matrix where the first row holds elements corresponding to PM probes and the second corresponding to MM probes. Note that in this case, it is assumed that there are equal number of PMs and MMs; if not, an error is generated. Moreover, the PMs and MMs may not even be paired, i.e. there is no guarantee that the two elements in a column corresponds to a PM-MM pair.
readIndices	If TRUE, cell indices calculated from the row and column (X,Y) coordinates are retrieved, otherwise not.
verbose	An integer specifying the verbose level. If 0, the file is parsed quietly. The higher numbers, the more details.

Value

A named list where the names corresponds to the names of the units read. Each element of the list is in turn a list structure with three components:

groups	A list with one component for each group (also called block). The information on each group is a list of up to seven components: x, y, pbase, tbase, expos, indices, and direction. All fields but the latter have the same number of values as there are cells in the group. The latter field has only one value indicating the direction for the whole group.
type	An integer specifying the type of the unit, where 1 is "expression", 2 is "genotyping", 3 is "CustomSeq", and 4 "tag".

direction — An integer specifying the direction of the unit, which defines if the probes are interrogating the sense or the anti-sense target, where 0 is "no direction", 1 is "sense", and 2 is "anti-sense".

Author(s)

James Bullard, <bullard@stat.berkeley.edu> and Kasper Daniel Hansen, <khansen@stat.berkeley.e
Modified by Henrik Bengtsson (http://www.braju.com/R/) to read any subset of units and/or subset of parameters, to stratify by PM/MM, and to return cell indices.d

References

[1] Affymetrix Inc, Affymetrix GCOS 1.x compatible file formats, June 14, 2005. http://www.affymetrix.com/support/developer/

See Also

readCdfCellIndices().

Examples

```
##############################################################
if (require("AffymetrixDataTestFiles")) {             # START #
##############################################################

# Find any CDF file
cdfFile <- findCdf()

# Read all units in a CDF file [~20s => 0.34ms/unit]
cdf0 <- readCdfUnits(cdfFile, readXY=FALSE, readExpos=FALSE)

# Read a subset of units in a CDF file [~6ms => 0.06ms/unit]
units1 <- c(5, 100:109, 34)
cdf1 <- readCdfUnits(cdfFile, units=units1, readXY=FALSE, readExpos=FALSE)
stopifnot(identical(cdf1, cdf0[units1]))
rm(cdf0)

# Create a unit name to index map
names <- readCdfUnitNames(cdfFile)
units2 <- match(names(cdf1), names)
stopifnot(all.equal(units1, units2))
cdf2 <- readCdfUnits(cdfFile, units=units2, readXY=FALSE, readExpos=FALSE)

stopifnot(identical(cdf1, cdf2))

##############################################################
}                                                     # STOP #
##############################################################
```

`readCelHeader` *Parsing the header of an Affymetrix CEL file*

Description

Reads in the header of an Affymetrix CEL file using the Fusion SDK.

Usage

```
readCelHeader(filename)
```

Arguments

`filename`	the name of the CEL file.

Details

This function returns the header of a CEL file. Affymetrix operates with different versions of this file format. Depending on what version is being read, different information is accessible.

Value

A named list with components described below. The entries are obtained from the Fusion SDK interface functions. We try to obtain all relevant information from the file.

`filename`	the name of the cel file.
`version`	the version of the cel file.
`cols`	the number of columns on the chip.
`rows`	the number of rows on the chip.
`total`	the total number of features on the chip. Usually equal to `rows` times `cols`, but since it is a separate attribute in the SDK we decided to include it anyway.
`algorithm`	the algorithm used to create the CEL file.
`parameters`	the parameters used in the algorithm. Seems to be semi-colon separated.
`chiptype`	the type of the chip.
`header`	the entire header of the CEL file. Only available for non-calvin format files.
`datheader`	the entire dat header of the CEL file. This contains for example a date.
`librarypackage`	the library package name of the file. Empty for older versions.
`cellmargin`	a parameter used to generate the CEL file. According to Affymetrix, it designates the number of pixels to ignore around the feature border when calculating the intensity value (the number of pixels ignored are cellmargin divided by 2).
`noutliers`	the number of features reported as outliers.
`nmasked`	the number of features reported as masked.

Note

Memory usage:the Fusion SDK allocates memory for the entire CEL file, when the file is accessed. The memory footprint of this function will therefore seem to be (rather) large.

Speed: CEL files of version 2 (standard text files) needs to be completely read in order to report the number of outliers and masked features.

Author(s)

James Bullard, <bullard@stat.berkeley.edu> and Kasper Daniel Hansen, <khansen@stat.berkeley.e

See Also

readCel() for reading in the entire CEL file. That function also returns the header. See affxparserInfo for general comments on the package and the Fusion SDK.

Examples

```
# Scan current directory for CEL files
files <- list.files(pattern="[.](c|C)(e|E)(l|L)$")
if (length(files) > 0) {
  header <- readCelHeader(files[1])
  print(header)
  rm(header)
}

# Clean up
rm(files)
```

---

readCelIntensities *Reads the intensities contained in several Affymetrix CEL files*

Description

Reads the intensities of several Affymetrix CEL files (as opposed to readCel() which only reads a single file).

Usage

```
readCelIntensities(filenames, indices = NULL, ..., verbose = 0)
```

Arguments

filenames	the names of the CEL files as a character vector.
indices	a vector of which indices should be read. If the argument is NULL all features will be returned.
...	Additional arguments passed to readCel().
verbose	an integer: how verbose do we want to be, higher means more verbose.

Details

The function will initially allocate a matrix with the same memory footprint as the final object.

Value

A matrix with a number of rows equal to the length of the indices argument (or the number of features on the entire chip), and a number of columns equal to the number of files. The columns are ordered according to the filenames argument.

Note

Currently this function builds on readCel(), and simply calls this function multiple times. If testing yields sufficient reasons for doing so, it may be re-implemented in C++.

Author(s)

James Bullard, <bullard@stat.berkeley.edu> and Kasper Daniel Hansen, <khansen@stat.berkeley.e

See Also

readCel() for a discussion of a more versatile function, particular with details of the indices argument.

Examples

```
# Scan current directory for CEL files
files <- list.files(pattern="[.](c|C)(e|E)(l|L)$")
if (length(files) >= 2) {
  cel <- readCelIntensities(files[1:2])
  str(cel)
  rm(cel)
}

# Clean up
rm(files)
```

---

readCel *Reads an Affymetrix CEL file*

Description

This function reads all or a subset of the data in an Affymetrix CEL file.

Usage

```
readCel(filename,
        indices = NULL,
        readHeader = TRUE,
        readXY = FALSE, readIntensities = TRUE,
        readStdvs = FALSE, readPixels = FALSE,
        readOutliers = TRUE, readMasked = TRUE,
        readMap = NULL,
        verbose = 0,
        .checkArgs = TRUE)
```

Arguments

filename	the name of the CEL file.
indices	a vector of indices indicating which features to read. If the argument is NULL all features will be returned.
readXY	a logical: will the (x,y) coordinates be returned.

`readIntensities`
: a logical: will the intensities be returned.

`readStdvs`
: a logical: will the standard deviations be returned.

`readPixels`
: a logical: will the number of pixels be returned.

`readOutliers`
: a logical: will the outliers be return.

`readMasked`
: a logical: will the masked features be returned.

`readHeader`
: a logical: will the header of the file be returned.

`readMap`
: A `vector` remapping cell indices to file indices. If `NULL`, no mapping is used.

`verbose`
: how verbose do we want to be. 0 is no verbosity, higher numbers mean more verbose output. At the moment the values 0, 1 and 2 are supported.

`.checkArgs`
: If `TRUE`, the arguments will be validated, otherwise not. *Warning: This should only be used if the arguments have been validated elsewhere!*

Value

A CEL files consists of a *header*, a set of *cell values*, and information about *outliers* and `masked` cells.

The cell values, which are values extract for each cell (aka feature or probe), are the (x,y) coordinate, intensity and standard deviation estimates, and the number of pixels in the cell. If `readIndices=NULL`, cell values for all cells are returned, Only cell values specified by argument `readIndices` are returned.

This value returns a named list with compontents described below:

`header`
: The header of the CEL file. Equivalent to the output from `readCelHeader`, see the documentation for that function.

`x,y`
: (cell values) Two `integer` vectors containing the x and y coordinates associated with each feature.

`intensities`
: (cell value) A `numeric` vector containing the intensity associated with each feature.

`stdvs`
: (cell value) A `numeric` vector containing the standard deviation associated with each feature.

`pixels`
: (cell value) An `integer` vector containing the number of pixels associated with each feature.

`outliers`
: An `integer` vector of indices specifying which of the queried cells that are flagged as outliers. Note that there is a difference between `outliers=NULL` and `outliers=integer(0)`; the last case happens when `readOutliers=TRUE` but there are no outliers.

`masked`
: An `integer` vector of indices specifying which of the queried cells that are flagged as masked. Note that there is a difference between `masked=NULL` and `masked=integer(0)`; the last case happens when `readMasked=TRUE` but there are no masked features.

The elements of the cell values are ordered according to argument `indices`. The lengths of the cell-value elements equals the number of cells read.

Which of the above elements that are returned are controlled by the `readNnn` arguments. If `FALSE`, the corresponding element above is `NULL`, e.g. if `readStdvs=FALSE` then `stdvs` is `NULL`.

Outliers and masked cells

The Affymetrix image analysis software flags cells as outliers and masked. This method does not return these flags, but instead vectors of cell indices listing which cells *of the queried cells* are outliers and masked, respectively. The current community view seems to be that this should be done based on statistical modelling of the actual probe intensities and should be based on the choice of preprocessing algorithm. Most algorithms are only using the intensities from the CEL file.

Memory usage

The Fusion SDK allocates memory for the entire CEL file, when the file is accessed (but does not actually read the file into memory). Using the `indices` argument will therefore only affect the memory use of the final object (as well as speed), not the memory allocated in the C function used to parse the file. This should be a minor problem however.

Troubleshooting

It is considered a bug if the file contains information not accessible by this function, please report it.

Author(s)

James Bullard, <bullard@stat.berkeley.edu> and Kasper Daniel Hansen, <khansen@stat.berkeley.e

See Also

`readCelHeader()` for a description of the header output. Often a user only wants to read the intensities, look at `readCelIntensities()` for a function specialized for that use.

Examples

```
for (zzz in 0) {  # Only so that 'break' can be used

# Scan current directory for CEL files
celFiles <- list.files(pattern="[.](c|C)(e|E)(l|L)$")
if (length(celFiles) == 0)
  break;

celFile <- celFiles[1]

# Read a subset of cells
idxs <- c(1:5, 1250:1500, 450:440)
cel <- readCel(celFile, indices=idxs, readOutliers=TRUE)
str(cel)

# Clean up
rm(celFiles, celFile, cel)

} # for (zzz in 0)
```

`readCelRectangle` *Reads a spatial subset of probe-level data from Affymetrix CEL files*

Description

Reads a spatial subset of probe-level data from Affymetrix CEL files.

Usage

```
readCelRectangle(filename, xrange=c(0, Inf), yrange=c(0, Inf), ..., asMatrix=TRU
```

Arguments

`filename`	The pathname of the CEL file.
`xrange`	A `numeric vector` of length two giving the left and right coordinates of the cells to be returned.
`yrange`	A `numeric vector` of length two giving the top and bottom coordinates of the cells to be returned.
`...`	Additional arguments passed to `readCel`().
`asMatrix`	If `TRUE`, the CEL data fields are returned as matrices with element (1,1) corresponding to cell (xrange[1],yrange[1]).

Value

A named `list` CEL structure similar to what `readCel`(). In addition, if `asMatrix` is `TRUE`, the CEL data fields are returned as matrices, otherwise not.

Author(s)

Henrik Bengtsson (`http://www.braju.com/R/`)

See Also

The `readCel`() method is used internally.

Examples

```
##############################################################
if (require("AffymetrixDataTestFiles")) {            # START #
##############################################################

rotate270 <- function(x, ...) {
  x <- t(x)
  nc <- ncol(x)
  if (nc < 2) return(x)
  x[,nc:1,drop=FALSE]
}


# Search for some available CEL files
path <- system.file("rawData", package="AffymetrixDataTestFiles")
file <- findFiles(pattern="[.](cel|CEL)$", path=path, recursive=TRUE)
```

```
# Read CEL intensities in the upper left corner
cel <- readCelRectangle(file, xrange=c(0,250), yrange=c(0,250))
z <- rotate270(cel$intensities)
sub <- paste("Chip type:", cel$header$chiptype)
image(z, col=gray.colors(256), axes=FALSE, main=basename(file), sub=sub)
text(x=0, y=1, labels="(0,0)", adj=c(0,-0.7), cex=0.8, xpd=TRUE)
text(x=1, y=0, labels="(250,250)", adj=c(1,1.2), cex=0.8, xpd=TRUE)

# Clean up
rm(rotate270, files, file, cel, z, sub)


##############################################################
}                                                    # STOP #
##############################################################
```

`readCelUnits` *Reads probe-level data ordered as units (probesets) from one or several Affymetrix CEL files*

Description

Reads probe-level data ordered as units (probesets) from one or several Affymetrix CEL files by using the unit and group definitions in the corresponding Affymetrix CDF file.

Usage

```
readCelUnits(filenames, units=NULL, stratifyBy=c("nothing", "pmmm", "pm", "mm"),
```

Arguments

`filenames`	The filenames of the CEL files.
`units`	An `integer vector` of unit indices specifying which units to be read. If `NULL`, all units are read.
`stratifyBy`	Argument passed to low-level method `readCdfUnits`.
`cdf`	A `character` filename of a CDF file, or a CDF `list` structure. If `NULL`, the CDF file is searched for by `findCdf`() first starting from the current directory and then from the directory where the first CEL file is.
`...`	Arguments passed to low-level method `readCel`, e.g. `readXY` and `readStdvs`.
`addDimnames`	If `TRUE`, dimension names are added to arrays, otherwise not. The size of the returned CEL structure in bytes increases by 30-40% with dimension names.
`dropArrayDim`	If `TRUE` and only one array is read, the elements of the group field do *not* have an array dimension.
`transforms`	A `list` of exactly `length(filenames)` `function`s. If `NULL`, no transformation is performed. Intensities read are passed through the corresponding transform function before being returned.
`readMap`	A `vector` remapping cell indices to file indices. If `NULL`, no mapping is used.

verbose — Either a logical, a numeric, or a Verbose object specifying how much verbose/debug information is written to standard output. If a Verbose object, how detailed the information is is specified by the threshold level of the object. If a numeric, the value is used to set the threshold of a new Verbose object. If TRUE, the threshold is set to -1 (minimal). If FALSE, no output is written (and neither is the **R.utils** package required).

Value

A named list with one element for each unit read. The names corresponds to the names of the units read. Each unit element is in turn a list structure with groups (aka blocks). Each group contains requested fields, e.g. intensities, stdvs, and pixels. If more than one CEL file is read, an extra dimension is added to each of the fields corresponding, which can be used to subset by CEL file.

Note that neither CEL headers nor information about outliers and masked cells are returned. To access these, use readCelHeader() and readCel().

Author(s)

Henrik Bengtsson (http://www.braju.com/R/)

References

[1] Affymetrix Inc, Affymetrix GCOS 1.x compatible file formats, June 14, 2005. http://www.affymetrix.com/support/developer/

See Also

Internally, readCelHeader(), readCdfUnits() and readCel() are used.

Examples

```
##############################################################
if (require("AffymetrixDataTestFiles")) {             # START #
##############################################################

# Search for some available CEL files
path <- system.file("rawData", package="AffymetrixDataTestFiles")
files <- findFiles(pattern="[.](cel|CEL)$", path=path, recursive=TRUE, firstOnly=FALSE)
files <- grep("FusionSDK_Test3", files, value=TRUE)
files <- grep("Calvin", files, value=TRUE)

# Fake more CEL files if not enough
files <- rep(files, length.out=5)
print(files);
rm(files);


##############################################################
}                                                     # STOP #
##############################################################
```

readChp *A function to read Affymetrix CHP files*

Description

This function will parse any type of CHP file and return the results in a list. The contents of the list will depend on the type of CHP file that is parsed and readers are referred to Affymetrix documentation of what should be there, and how to interpret it.

Usage

```
readChp(filename, withQuant = TRUE)
```

Arguments

filename	The name of the CHP file to read.
withQuant	A boolean value, currently largely unused.

Details

This is an interface to the Affymetrix Fusion SDK. The Affymetrix documentation should be consulted for explicit details.

Value

A list is returned. The contents of the list depend on the type of CHP file that was read. Users may want to translate the different outputs into specific containers.

Troubleshooting

It is considered a bug if the file contains information not accessible by this function, please report it.

Author(s)

R. Gentleman

See Also

readCel

Examples

```
if (require("AffymetrixDataTestFiles")) {
path <- system.file("rawData", package="AffymetrixDataTestFiles")
files <- findFiles(pattern="[.](chp|CHP)$", path=path,
                   recursive=TRUE, firstOnly=FALSE)

s1 = readChp(files[1])
length(s1)
names(s1)
names(s1[[7]])
}
```

`readClfEnv` *Parsing a CLF file using Affymetrix Fusion SDK*

Description

This function parses a CLF file using the Affymetrix Fusion SDK. CLF (chip layout) files contain information associating probe ids with chip x- and y- coordinates.

Usage

```
readClfEnv(file, readBody = TRUE)
```

Arguments

`file`	`character(1)` providing a path to the CLF file to be input.
`readBody`	`logical(1)` indicating whether the entire file should be parsed (`TRUE`) or only the file header information describing the chips to which the file is relevant.

Value

An enviroment. The `header` element is always present; the remainder are present when `readBody=TRUE`.

`header`	A list with information about the CLF file. The list contains elements described in the CLF file format document referenced below.
`dims`	A length-two integer vector of chip x- and y-coordinates.
`id`	An integer vector of length `prod(dims)` containing probe identifiers.
`x`	An integer vector of length `prod(dims)` containing x-coordinates corresponding to the entries in `id`.
`y`	An integer vector of length `prod(dims)` containing y-coordinates corresponding to the entries in `id`.

Author(s)

Martin Morgan mtmorgan@fhcrc.org

See Also

https://www.affymetrix.com/support/developer/fusion/File_Format_CLF_aptv161.pdf describes CLF file content.

readClfHeader *Read the header of a CLF file.*

Description

Reads the header of a CLF file. The exact information stored in this file can be viewed in the readClfEnv documentation which reads the header in addition to the body.

Usage

```
readClfHeader(file)
```

Arguments

file	file a CLF file

Value

A list of header elements.

readClf *Parsing a CLF file using Affymetrix Fusion SDK*

Description

This function parses a CLF file using the Affymetrix Fusion SDK. CLF (chip layout) files contain information associating probe ids with chip x- and y- coordinates.

Usage

```
readClf(file)
```

Arguments

file	character(1) providing a path to the CLF file to be input.

Value

An list. The header element is always present.

header	A list with information about the CLF file. The list contains elements described in the CLF file format document referenced below.
dims	A length-two integer vector of chip x- and y-coordinates.
id	An integer vector of length prod(dims) containing probe identifiers.
x	An integer vector of length prod(dims) containing x-coordinates corresponding to the entries in id.
y	An integer vector of length prod(dims) containing y-coordinates corresponding to the entries in id.

Author(s)

Martin Morgan mtmorgan@fhcrc.org

See Also

https://www.affymetrix.com/support/developer/fusion/File_Format_CLF_aptv161.pdf describes CLF file content.

`readPgfEnv` *Parsing a PGF file using Affymetrix Fusion SDK*

Description

This function parses a PGF file using the Affymetrix Fusion SDK. PGF (probe group) files describe probes present within probe sets, including the type (e.g., pm, mm) of the probe and probeset.

Usage

```
readPgfEnv(file, readBody = TRUE, indices = NULL)
```

Arguments

`file`	`character(1)` providing a path to the PGF file to be input.
`readBody`	`logical(1)` indicating whether the entire file should be parsed (`TRUE`) or only the file header information describing the chips to which the file is relevant.
`indices`	`integer(n)` vector of positive integers indicating which probesets to read. These integers must be sorted (increasing) and unique.

Value

An environment. The `header` element is always present; the remainder are present when `readBody=TRUE`.

The elements present when `readBody=TRUE` describe probe sets, atoms, and probes. Elements within probe sets, for instance, are coordinated such that the `i`th index of one vector (e.g., `probesetId`) corresponds to the `i`th index of a second vector (e.g., `probesetType`). The atoms contained within probeset `i` are in positions `probesetStartAtom[i]:(probesetStartAtom[i+1]-1)` of the atom vectors. A similar map applies to probes within atoms, using `atomStartProbe` as the index.

The PGF file format includes optional elements; these elements are always present in the environment, but with appropriate default values.

`header`	A list with information about the PGF file. The list contains elements described in the PGF file format document referenced below.
`probesetId`	integer vector of probeset identifiers.
`probesetType`	character vector of probeset types. Types are described in the PGF file format document.
`probesetName`	character vector of probeset names.

probesetStartAtom
: integer vector of the start index (e.g., in the element atomId of atoms belonging to this probeset).

atomId
: integer vector of atom identifiers.

atomExonPosition
: integer vector of probe interrogation position relative to the target sequence.

atomStartProbe
: integer vector of the start index (e.g., in the element probeId of probes belonging to this atom).

probeId
: integer vector of probe identifiers.

probeType
: character vector of probe types. Types are described in the PGF file format document.

probeGcCount
: integer vector of probe GC content.

probeLength
: integer vector of probe lengths.

probeInterrogationPosition
: integer vector of the position, within the probe, at which interrogation occurs.

probeSequence
: character vector of the probe sequence.

Author(s)

Martin Morgan mtmorgan@fhcrc.org

See Also

https://www.affymetrix.com/support/developer/fusion/File_Format_PGF_aptv161.pdf describes PGF file content.

The internal function .pgfProbeIndexFromProbesetIndex provides a map between the indicies of probe set entries and the indicies of the probes contained in the probe set.

---

readPgfHeader *Read the header of a PGF file into a list.*

Description

This function reads the header of a PGF file into a list more details on what the exact fields are can be found in the details section.

Usage

```
readPgfHeader(file)
```

Arguments

file
: file:A file in PGF format

Details

https://www.affymetrix.com/support/developer/fusion/File_Format_PGF_aptv161.pdf

Value

A list corresponding to the elements in the header.

`readPgf` *Parsing a PGF file using Affymetrix Fusion SDK*

Description

This function parses a PGF file using the Affymetrix Fusion SDK. PGF (probe group) files describe probes present within probe sets, including the type (e.g., pm, mm) of the probe and probeset.

Usage

```
readPgf(file, indices = NULL)
```

Arguments

`file`	`character(1)` providing a path to the PGF file to be input.
`indices`	`integer(n)` a vector of indices of the probesets to be read.

Value

An list. The `header` element is always present; the remainder are present when `readBody=TRUE`.

The elements present when `readBody=TRUE` describe probe sets, atoms, and probes. Elements within probe sets, for instance, are coordinated such that the `i`th index of one vector (e.g., `probesetId`) corresponds to the `i`th index of a second vector (e.g., `probesetType`). The atoms contained within probeset `i` are in positions `probesetStartAtom[i]:(probesetStartAtom[i+1]-1)` of the atom vectors. A similar map applies to probes within atoms, using `atomStartProbe` as the index.

The PGF file format includes optional elements; these elements are always present in the list, but with appropriate default values.

`header`	A list with information about the PGF file. The list contains elements described in the PGF file format document referenced below.
`probesetId`	integer vector of probeset identifiers.
`probesetType`	character vector of probeset types. Types are described in the PGF file format document.
`probesetName`	character vector of probeset names.
`probesetStartAtom`	integer vector of the start index (e.g., in the element `atomId` of atoms belonging to this probeset).
`atomId`	integer vector of atom identifiers.
`atomExonPosition`	integer vector of probe interrogation position relative to the target sequence.
`atomStartProbe`	integer vector of the start index (e.g., in the element `probeId` of probes belonging to this atom).
`probeId`	integer vector of probe identifiers.

probeType character vector of probe types. Types are described in the PGF file format document.

probeGcCount integer vector of probe GC content.

probeLength integer vector of probe lengths.

probeInterrogationPosition integer vector of the position, within the probe, at which interrogation occurs.

probeSequence character vector of the probe sequence.

Author(s)

Martin Morgan mtmorgan@fhcrc.org

See Also

https://www.affymetrix.com/support/developer/fusion/File_Format_PGF_aptv161.pdf describes PGF file content.

The internal function .pgfProbeIndexFromProbesetIndex provides a map between the indicies of probe set entries and the indicies of the probes contained in the probe set.

---

updateCel *Updates a CEL file*

Description

Updates a CEL file.

Usage

```
updateCel(filename, indices=NULL, intensities=NULL, stdvs=NULL, pixels=NULL, wri
```

Arguments

filename The filename of the CEL file.

indices A numeric vector of cell (probe) indices specifying which cells to updated. If NULL, all indices are considered.

intensities A numeric vector of intensity values to be stored. Alternatively, it can also be a named data.frame or matrix (or list) where the named columns (elements) are the fields to be updated.

stdvs A optional numeric vector.

pixels A optional numeric vector.

writeMap An optional write map.

... Not used.

verbose An integer specifying how much verbose details are outputted.

Details

Currently only binary (v4) CEL files are supported. The current version of the method does not make use of the Fusion SDK, but its own code to navigate and update the CEL file.

Value

Returns (invisibly) the pathname of the file updated.

Author(s)

Henrik Bengtsson (http://www.braju.com/R/)

Examples

```
##############################################################
if (require("AffymetrixDataTestFiles")) {            # START #
##############################################################

# Search for some available Calvin CEL files
path <- system.file("rawData", package="AffymetrixDataTestFiles")
files <- findFiles(pattern="[.](cel|CEL)$", path=path, recursive=TRUE, firstOnly=FALSE)
files <- grep("FusionSDK_HG-U133A", files, value=TRUE)
files <- grep("Calvin", files, value=TRUE)
file <- files[1]

# Convert to an XDA CEL file
filename <- file.path(tempdir(), basename(file))
if (file.exists(filename))
  file.remove(filename)
convertCel(file, filename)


fields <- c("intensities", "stdvs", "pixels")

# Cells to be updated
idxs <- 1:2

# Get CEL header
hdr <- readCelHeader(filename)

# Get the original data
cel <- readCel(filename, indices=idxs, readStdvs=TRUE, readPixels=TRUE)
print(cel[fields])
cel0 <- cel

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Square-root the intensities
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
updateCel(filename, indices=idxs, intensities=sqrt(cel$intensities))
cel <- readCel(filename, indices=idxs, readStdvs=TRUE, readPixels=TRUE)
print(cel[fields])


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Update a few cell values by a data frame
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
data <- data.frame(
  intensities=cel0$intensities,
  stdvs=c(201.1, 3086.1)+0.5,
  pixels=c(9,9+1)
)
```

```
updateCel(filename, indices=idxs, data)

# Assert correctness of update
cel <- readCel(filename, indices=idxs, readStdvs=TRUE, readPixels=TRUE)
print(cel[fields])
for (ff in fields) {
#  stopifnot(all.equal(cel[[ff]], data[[ff]], .Machine$double.eps^0.25))
}

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Update a region of the CEL file
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Load pre-defined data
side <- 306
pathname <- system.file("extras/easternEgg.gz", package="affxparser")
con <- gzfile(pathname, open="rb")
z <- readBin(con=con, what="integer", size=1, signed=FALSE, n=side^2)
close(con)
z <- matrix(z, nrow=side)
side <- min(hdr$cols - 2*22, side)
z <- as.double(z[1:side,1:side])
x <- matrix(22+0:(side-1), nrow=side, ncol=side, byrow=TRUE)
idxs <- as.vector((1 + x) + hdr$cols*t(x))
# Load current data in the same region
z0 <- readCel(filename, indices=idxs)$intensities
# Mix the two data sets
z <- (0.3*z^2 + 0.7*z0)
# Update the CEL file
updateCel(filename, indices=idxs, intensities=z)

# Make some spatial changes
rotate270 <- function(x, ...) {
  x <- t(x)
  nc <- ncol(x)
  if (nc < 2) return(x)
  x[,nc:1,drop=FALSE]
}

# Display a spatial image of the updated CEL file
cel <- readCelRectangle(filename, xrange=c(0,350), yrange=c(0,350))
z <- rotate270(cel$intensities)
sub <- paste("Chip type:", cel$header$chiptype)
image(z, col=gray.colors(256), axes=FALSE, main=basename(filename), sub=sub)
text(x=0, y=1, labels="(0,0)", adj=c(0,-0.7), cex=0.8, xpd=TRUE)
text(x=1, y=0, labels="(350,350)", adj=c(1,1.2), cex=0.8, xpd=TRUE)


# Clean up
file.remove(filename)
rm(files, cel, cel0, idxs, data, ff, fields, rotate270)


############################################################
}                                                  # STOP #
############################################################
```

`updateCelUnits` *Updates a CEL file unit by unit*

Description

Updates a CEL file unit by unit.

Please note that, contrary to `readCelUnits()`*, this method can only update a single CEL file at the time.*

Usage

```
updateCelUnits(filename, cdf=NULL, data, ..., verbose=0)
```

Arguments

`filename`	The filename of the CEL file.
`cdf`	A (optional) CDF `list` structure either with field `indices` or fields `x` and `y`. If `NULL`, the unit names (and from there the cell indices) are inferred from the names of the elements in `data`.
`data`	A `list` structure in a format similar to what is returned by `readCelUnits`() for *a single CEL file only*.
`...`	Optional arguments passed to `readCdfCellIndices`(), which is called if `cdf` is not given.
`verbose`	An `integer` specifying how much verbose details are outputted.

Value

Returns what `updateCel`() returns.

Working with re-arranged CDF structures

Note that if the `cdf` structure is specified the CDF file is *not* queried, but all information about cell x and y locations, that is, cell indices is expected to be in this structure. This can be very useful when one work with a cdf structure that originates from the underlying CDF file, but has been restructured for instance through the `applyCdfGroups`() method, and `data` correspondingly. This update method knows how to update such structures too.

Author(s)

Henrik Bengtsson (`http://www.braju.com/R/`)

See Also

Internally, `updateCel`() is used.

Examples

```
##############################################################
if (require("AffymetrixDataTestFiles")) {            # START #
##############################################################

# Search for some available Calvin CEL files
path <- system.file("rawData", package="AffymetrixDataTestFiles")
files <- findFiles(pattern="[.](cel|CEL)$", path=path, recursive=TRUE, firstOnly=FALSE)
files <- grep("FusionSDK_Test3", files, value=TRUE)
files <- grep("Calvin", files, value=TRUE)
file <- files[1]

# Convert to an XDA CEL file
pathname <- file.path(tempdir(), basename(file))
if (file.exists(pathname))
  file.remove(pathname)
convertCel(file, pathname)



# Check for the CDF file
hdr <- readCelHeader(pathname)
cdfFile <- findCdf(hdr$chiptype)

hdr <- readCdfHeader(cdfFile)
nbrOfUnits <- hdr$nunits
print(nbrOfUnits);

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Example: Read and re-write the same data
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
units <- c(101, 51)
data1 <- readCelUnits(pathname, units=units, readStdvs=TRUE)
cat("Original data:\n")
str(data1)
updateCelUnits(pathname, data=data1)
data2 <- readCelUnits(pathname, units=units, readStdvs=TRUE)
cat("Updated data:\n")
str(data2)
stopifnot(identical(data1, data2))


# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Example: Random read and re-write "stress test"
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
for (kk in 1:10) {
  nunits <- sample(min(1000,nbrOfUnits), size=1)
  units <- sample(nbrOfUnits, size=nunits)
  cat(sprintf("%02d. Selected %d random units: reading", kk, nunits));
  t <- system.time({
    data1 <- readCelUnits(pathname, units=units, readStdvs=TRUE)
  }, gcFirst=TRUE)[3]
  cat(sprintf(" [%.2fs=%.2fs/unit], updating", t, t/nunits))
  t <- system.time({
    updateCelUnits(pathname, data=data1)
```

```
  }, gcFirst=TRUE)[3]
  cat(sprintf(" [%.2fs=%.2fs/unit], validating", t, t/nunits))
  data2 <- readCelUnits(pathname, units=units, readStdvs=TRUE)
  stopifnot(identical(data1, data2))
  cat(". done\n")
}

##############################################################
}                                                    # STOP #
##############################################################
```

`writeTpmap` *Writes BPMAP and TPMAP files.*

Description

Writes BPMAP and TPMAP files.

Usage

```
writeTpmap(filename, bpmaplist, verbose = 0)

tpmap2bpmap(tpmapname, bpmapname, verbose = 0)
```

Arguments

`filename`	The filename.
`bpmaplist`	A list structure similar to the result of `readBpmap`.
`tpmapname`	Filename of the TPMAP file.
`bpmapname`	Filename of the BPMAP file.
`verbose`	How verbose do we want to be.

Details

`writeTpmap` writes a text probe map file, while `tpmap2bpmap` converts such a file to a binary probe mapping file. Somehow Affymetrix has different names for the same structure, depending on whether the file is binary or text. I have seen many TPMAP files referred to as BPMAP files.

Value

These functions are called for their side effects (creating files).

Author(s)

Kasper Daniel Hansen <khansen@stat.berkeley.edu>

See Also

`readBpmap`

Index