Package 'RDML'

May 7, 2026

Type Package

Title Importing Real-Time Thermo Cycler (qPCR) Data from RDML Format Files

Version 1.1

Date 2025-07-09

Description Imports real-time thermo cycler (qPCR) data from Real-time PCR Data Markup Language (RDML) and transforms to the appropriate formats of the 'qpcR' and 'chipPCR' packages, as described in Rodiger et al. (2017) <doi:10.1093/bioinformatics/btx528>. Contains a dendrogram visualization for the structure of RDML object and GUI for RDML editing.

License MIT + file LICENSE

URL https://github.com/PCRuniversum/RDML

Depends R (>= 3.2.0)

Imports checkmate (>= 1.6.2), data.table, pipeR, readxl, rlist (>= 0.4), R6 (>= 2.0.1), stringr, tools (>= 3.2), xml2 (>= 1.0), lubridate (>= 1.6.0)

Collate 'RDML.types.R' 'RDML.R' 'RDML.AsDendrogram.R' 'RDML.AsTable.R' 'RDML.GetFData.R' 'RDML.Merge.R' 'RDML.SetFData.R' 'RDML.init.R' 'functional_wrappers.R' 'rdmlEdit.R'

Suggests chipPCR, magrittr, reshape2, qpcR, dplyr, ggplot2, knitr, kfigr, MBmca, shiny, shinyjs, shinythemes, V8, testthat

RoxygenNote 7.3.2

NeedsCompilation no

Maintainer Konstantin Blagodatskikh <k.blag@yandex.ru>

Repository CRAN

Date/Publication 2025-07-23 11:50:08 UTC

Author Konstantin Blagodatskikh [cre, aut], Stefan Roediger [aut], Michal Burdukiewicz [aut] (ORCID: <https://orcid.org/0000-0001-8926-582X>), Andrej-Nikolai Spiess [aut]

Contents

`adpsType` *adpsType R6 class.*

Description

adpsType R6 class.

adpsType R6 class.

Format

An `R6Class` generator object.

Details

Contains `matrix` of amplification data. Must have three columns:

cyc PCR cycle at which data point was collected (every cycle must have unique number).

tmp temperature in degrees Celsius at the time of measurement (optional).

fluor raw fluorescence intensity measured.

Inherits: rdmlBaseType.

Initialization

```
adpsType$new(fpoints)
```

Fields

`fpoints` assertMatrix. Matrix with amplification data points.

Super class

`RDML::rdmlBaseType` -> `adpsType`

Methods

Public methods:

- adpsType$new()
- adpsType$.asXMLnodes()
- adpsType$clone()

Method new():

Usage:

```
adpsType$new(fpoints)
```

Method .asXMLnodes():

Usage:

```
adpsType$.asXMLnodes(node.name)
```

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
adpsType$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
#cycles
cyc <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
34, 35, 36, 37, 38, 39, 40)
#fluorescence
fluo <- c(2.0172, 2.0131, 2.0035, 2, 2.0024, 2.0056, 2.0105, 2.0179,
2.0272, 2.0488, 2.0922, 2.1925, 2.3937, 2.7499, 3.3072, 4.0966,
5.0637, 6.0621, 7.0239, 7.8457, 8.5449, 9.1282, 9.6022, 9.9995,
10.2657, 10.4989, 10.6813, 10.8209, 10.9158, 10.9668, 11.0053,
11.0318, 11.0446, 11.044, 11.0052, 10.9671, 10.9365, 10.9199,
10.897, 10.8316)
#temperature
temp <- c(55, 55, 55, 55, 54, 54, 55, 55, 55, 55, 55, 55, 55, 55, 55,
55, 55, 55, 55, 55, 55, 55, 55, 56, 55, 55, 55, 55, 55, 55, 55,
55, 55, 55, 55, 55, 55, 55, 55, 55)

#combine all variables into a proper object
data <- data.frame(cyc = cyc, tmp = temp, fluor = fluo)

#create adps object
adpsType$new(data)

#create adps object without temperature data
adpsType$new(data[, -2])
```

annotationType *annotationType R6 class.*

Description

Annotate samples by setting a property and its value. For example, sex could be a property with the possible values M or F. Inherits: rdmlBaseType.

Format

An `R6Class` generator object.

Fields

property checkString. Property name

value checkString. Value

Super class

`RDML::rdmlBaseType` -> `annotationType`

Methods

Public methods:

- `annotationType$new()`
- `annotationType$clone()`

Method `new()`:

Usage:

```
annotationType$new(property, value)
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
annotationType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
#set sex property
annotationType$new(property = "sex", value = "M")
```

as.character.idType *Convert* idType *object to* character

Description

Function to convert idType object to character.

Usage

```
## S3 method for class 'idType'
as.character(x, ...)
```

Arguments

x	idType object.
...	Further arguments to be passed.

as.character.reactIdType *Convert* reactIdType *object to* character

Description

Function to convert reactIdType object to character.

Usage

```
## S3 method for class 'reactIdType'
as.character(x, ...)
```

Arguments

x	reactIdType object.
...	Further arguments to be passed.

AsDendrogram RDML$AsDendrogram() *wrapper*

Description

Read more at RDML.AsDendrogram

Usage

```
AsDendrogram(obj, ...)
```

Arguments

obj	RDML object.
...	AsDendrogram params.

AsTable RDML$AsTable() *wrapper*

Description

Read more at RDML.AsTable

Usage

```
AsTable(obj, ...)
```

Arguments

obj	RDML object.
...	AsTable params.

baseTemperatureType *baseTemperatureType R6 class.*

Description

Parent class for inner usage. Inherits: rdmlBaseType.

Format

An R6Class generator object.

Initialization

```
baseTemperatureType$new(duration,
  temperatureChange = NULL, durationChange = NULL, measure = NULL, ramp =
  NULL)
```

Fields

`duration` checkCount. Duration of this step in seconds.

`temperatureChange` checkNumber. Change of the temperature between two consecutive cycles: actual temperature = temperature + (temperatureChange * cycle counter)

`durationChange` checkCount. Change of the duration between two consecutive cycles: actual duration = duration + (durationChange * cycle counter)

`measure` measureType. Indicates to make a measurement and store it as meltcurve or real-time data.

`ramp` checkNumber. Allowed temperature change between two consecutive cycles in degrees Celsius per second. If unstated, the maximal change rate is assumed.

Super class

RDML::rdmlBaseType -> baseTemperatureType

Methods

Public methods:

- baseTemperatureType$new()
- baseTemperatureType$clone()

Method new():

Usage:

```
baseTemperatureType$new(
  duration,
  temperatureChange = NULL,
  durationChange = NULL,
  measure = NULL,
```

```
  ramp = NULL
)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
baseTemperatureType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`cdnaSynthesisMethodType` *cdnaSynthesisMethodType R6 class.*

Description

Description of the cDNA synthesis method. Inherits: rdmlBaseType.

Format

An `R6Class` generator object.

Initialization

```
cdnaSynthesisMethodType$new(enzyme = NULL,
  primingMethod = NULL, dnaseTreatment = NULL, thermalCyclingConditions =
  NULL)
```

@section Fields:

`enzyme` checkString. Name of the enzyme used for reverse transcription.

`primingMethod` primingMethodType.

`dnaseTreatment` checkFlag if `TRUE`RNA was DNAse treated prior cDNA synthesis.

`thermalCyclingConditions` idReferencesType.

Super class

`RDML::rdmlBaseType` -> `cdnaSynthesisMethodType`

Methods

Public methods:

- `cdnaSynthesisMethodType$new()`
- `cdnaSynthesisMethodType$clone()`

Method `new()`**:**

Usage:

```
cdnaSynthesisMethodType$new(
  enzyme = NULL,
  primingMethod = NULL,
  dnaseTreatment = NULL,
  thermalCyclingConditions = NULL
)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
cdnaSynthesisMethodType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`commercialAssayType` *commercialAssayType R6 class.*

Description

For some commercial assays, the primer sequences may be unknown. This element allows to describe commercial assays. Inherits: rdmlBaseType.

Format

An `R6Class` generator object.

Initialization

```
commercialAssayType$new(company, orderNumber)
```

@section Fields:

`company` checkString.

`orderNumber` checkString.

Super class

`RDML::rdmlBaseType` -> `commercialAssayType`

Methods

Public methods:

- `commercialAssayType$new()`
- `commercialAssayType$clone()`

Method `new()`**:**

Usage:

```
commercialAssayType$new(company, orderNumber)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
commercialAssayType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`cqDetectionMethodType` *cqDetectionMethodType R6 class.*

Description

The method used to determine the Cq value. Can take values:

"automated threshold and baseline settings"

"manual threshold and baseline settings"

"second derivative maximum"

"other"

Inherits: enumType.

Format

An `R6Class` generator object.

Initialization

```
cqDetectionMethodType$new(value)
```

@section Fields:

`value` checkString.

Super classes

`RDML::rdmlBaseType` -> `RDML::enumType` -> `cqDetectionMethodType`

Methods

Public methods:

- `cqDetectionMethodType$clone()`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
cqDetectionMethodType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

dataCollectionSoftwareType *dataCollectionSoftwareType R6 class.*

Description

Software name and version used to collect and analyze the data. Inherits: rdmlBaseType.

Format

An R6Class generator object.

Initialization

```
dataCollectionSoftwareType$new(name, version)
```

@section Fields:

`name` checkString.

`version` checkString.

Super class

`RDML::rdmlBaseType` -> `dataCollectionSoftwareType`

Methods

Public methods:

- `dataCollectionSoftwareType$new()`
- `dataCollectionSoftwareType$clone()`

Method `new()`:

Usage:

```
dataCollectionSoftwareType$new(name, version)
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
dataCollectionSoftwareType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
dataCollectionSoftwareType$new(name = "ExampleSoft",
                               version = "1.0")
```

`dataType` *dataType R6 class.*

Description

Inherits: rdmlBaseType.

Format

An `R6Class` generator object.

Initialization

```
dataType$new(tar, cq = NULL, excl = NULL,
adp = NULL, mdp = NULL, endPt = NULL, bgFluor = NULL, bgFluorSlp = NULL,
quantFluor = NULL)
```

Fields

`tar` idReferencesType. TargetID - A reference to a target.

`cq` checkNumber. Calculated fractional PCR cycle used for downstream quantification. Negative values express following condition: Not Available: -1.0

`excl` checkString. Excluded. If `excl` is present, this entry should not be evaluated. Do not set this element to `FALSE` if the entry is valid. Instead, leave the entire `excl` element out instead. It may contain a string with a reason for the exclusion. Several reasons for exclusion should be seperated by semicolons ";".

`adp` adpsType.

`mdp` mdpsType.

`endPt` checkNumber. Value of the endpoint measurement.

`bgFluor` checkNumber. Background fluorescence (the y-intercept of the baseline trend based on the estimated background fluorescence).

`bgFluorSlp` checkNumber. Background fluorescence slope - The slope of the baseline trend based on the estimated background fluorescence. The element should be absent to indicate a slope of 0.0; If this element is present without the `bgFluor` element it should be ignored.

`quantFluor` checkNumber. Quantification flourescence - The fluorescence value corresponding to the treshold line.

Methods

`AsDataFrame(dp.type = "adp")` Represents amplification (

`dp.type = "adp"`

) or melting (`dp.type = "mdp"`) data points as `data.frame`

Super class

RDML::rdmlBaseType -> dataType

Methods

Public methods:

- dataType$new()
- dataType$GetFData()
- dataType$clone()

Method new():

Usage:

```
dataType$new(
  tar,
  cq = NULL,
  excl = NULL,
  adp = NULL,
  mdp = NULL,
  endPt = NULL,
  bgFluor = NULL,
  bgFluorSlp = NULL,
  quantFluor = NULL
)
```

Method GetFData():

Usage:

```
dataType$GetFData(dp.type = "adp")
```

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
dataType$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

---

documentationType *documentationType R6 class.*

Description

These elements should be used if the same description applies to many samples, targets or experiments. Inherits: rdmlBaseType.

Format

An R6Class generator object.

Initialization

```
documentationType$new(id, text = NULL)
```

@section Fields:

`id` idType. Identificator.

`text` checkString. Text.

Super class

`RDML::rdmlBaseType` -> `documentationType`

Methods

Public methods:

- `documentationType$new()`
- `documentationType$clone()`

Method `new()`**:**

Usage:

```
documentationType$new(id, text = NULL)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
documentationType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

---

dyeType *dyeType R6 class.*

Description

Detailed information about the dye. Inherits: rdmlBaseType.

Format

An `R6Class` generator object.

Initialization

```
dyeType$new(id, description = NULL)
```

@section Fields:

`id` idType. Identificator.

`description` checkString. Description.

Super class

RDML::rdmlBaseType -> dyeType

Methods

Public methods:

- dyeType$new()
- dyeType$clone()

Method new():

Usage:

```
dyeType$new(id, description = NULL)
```

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
dyeType$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

---

enumType *enumType R6 class.*

Description

Generic class for creating objects thet can take limited list of values.
Inherits: rdmlBaseType.

Format

An R6Class generator object.

Initialization

```
enumType$new(value)
```

@section Fields:

value checkString. Value.

Super class

RDML::rdmlBaseType -> enumType

Methods

Public methods:

- `enumType$new()`
- `enumType$print()`
- `enumType$.asXMLnodes()`
- `enumType$clone()`

Method `new()`:

Usage:

```
enumType$new(value)
```

Method `print()`:

Usage:

```
enumType$print(...)
```

Method `.asXMLnodes()`:

Usage:

```
enumType$.asXMLnodes(node.name)
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
enumType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`experimenterType` *experimenterType R6 class.*

Description

Contact details of the experimenter. Inherits: rdmlBaseType.

Format

An `R6Class` generator object.

Initialization

```
experimenterType$new(id, firstName, lastName,
  email = NULL, labName = NULL, labAddress = NULL)
```

@section Fields:

`id` idType. Identificator.

`firstName` checkString. First name.

`lastName` checkString. Last name.

`email` checkString. Email.

`labName` checkString. Lab name.

`labAddress` checkString. Lab address.

Super class

`RDML::rdmlBaseType` -> `experimenterType`

Methods

Public methods:

- `experimenterType$new()`
- `experimenterType$clone()`

Method `new()`:

Usage:

```
experimenterType$new(
  id,
  firstName,
  lastName,
  email = NULL,
  labName = NULL,
  labAddress = NULL
)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
experimenterType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`experimentType` *experimentType R6 class.*

Description

A qPCR experiment. It may contain several runs (runType). Inherits: rdmlBaseType.

Format

An `R6Class` generator object.

Initialization

```
experimentType$new(id, description = NULL,
  documentation = NULL, run = NULL)
```

@section Fields:

`id` idType.

`description` checkString.

`documentation` `list` of idReferencesType.

`run` `list` of runType.

Methods

`AsDataFrame(dp.type = "adp", long.table = FALSE)` Represents amplification (`dp.type = "adp"`) or melting (`dp.type = "mdp"`) data points as `data.frame`. `long.table = TRUE` means that fluorescence data for all runs and reacts will be at one collumn.

Super class

`RDML::rdmlBaseType` -> `experimentType`

Methods

Public methods:

- `experimentType$new()`
- `experimentType$GetFData()`
- `experimentType$clone()`

Method `new()`:

Usage:

```
experimentType$new(id, description = NULL, documentation = NULL, run = NULL)
```

Method `GetFData()`:

Usage:

```
experimentType$GetFData(dp.type = "adp", long.table = FALSE)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
experimentType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`GetFData` `RDML$GetFData()` *wrapper*

Description

Read more at RDML.GetFData

Usage

```
GetFData(obj, ...)
```

Arguments

`obj`	`RDML` object.
`...`	`GetFData` params.

`gradientType` *gradientType R6 class.*

Description

Details of the temperature gradient across the PCR block. Inherits: baseTemperatureType.

Format

An `R6Class` generator object.

Initialization

```
gradientType$new(highTemperature,
  lowTemperature, ...)
```

Fields

`highTemperature` checkNumber. The highest temperature of the gradient in degrees Celsius.

`lowTemperature` checkNumber. The lowest temperature of the gradient in degrees Celsius.

`...` Params of parent class baseTemperatureType.

Super classes

RDML::rdmlBaseType -> RDML::baseTemperatureType -> gradientType

Methods

Public methods:

- gradientType$new()
- gradientType$clone()

Method new():

Usage:

```
gradientType$new(highTemperature, lowTemperature, ...)
```

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
gradientType$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

---

idReferencesType *idReferencesType R6 class.*

Description

Contains id of another RDML object. Inherits: idType.

Format

An R6Class generator object.

Initialization

```
idReferencesType$new(id)
```

Fields

id checkString. Identificator.

Super classes

RDML::rdmlBaseType -> RDML::idType -> idReferencesType

Methods

Public methods:

- idReferencesType$clone()

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
idReferencesType$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

---

idType *idType R6 class.*

Description

Contains identificator for varius RDML types. Inherits: rdmlBaseType.

Format

An R6Class generator object.

Initialization

```
idType$new(id)
```

@section Fields:

id checkString. Identificator.

Super class

RDML::rdmlBaseType -> idType

Methods

Public methods:

- idType$new()
- idType$.asXMLnodes()
- idType$clone()

Method new():

Usage:

```
idType$new(id)
```

Method .asXMLnodes():

Usage:

```
idType$.asXMLnodes(node.name)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
idType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`labelFormatType` *labelFormatType R6 class.*

Description

Label used for pcrFormatType. Can take values:

ABC

123

A1a1

Inherits: enumType.

Format

An `R6Class` generator object.

Initialization

```
labelFormatType$new(value)
```

@section Fields:

`value` checkString.

Super classes

`RDML::rdmlBaseType` -> `RDML::enumType` -> `labelFormatType`

Methods

Public methods:

- `labelFormatType$clone()`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
labelFormatType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`lidOpenType` *lidOpenType R6 class.*

Description

This step waits for the user to open the lid and continues afterwards. It allows to stop the program and to wait for the user to add for example enzymes and continue the program afterwards. The temperature of the previous step is maintained. Inherits: rdmlBaseType.

Format

An `R6Class` generator object.

Initialization

```
lidOpenType$new()
```

Super class

`RDML::rdmlBaseType` -> `lidOpenType`

Methods

Public methods:

- `lidOpenType$new()`
- `lidOpenType$clone()`

Method `new()`:

Usage:

```
lidOpenType$new()
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
lidOpenType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

loopType *loopType R6 class.*

Description

This step allows to form a loop or to exclude some steps. It allows to jump to a certain "goto" step for "repeat" times. If the "goto" step is outside of the loop range, it must have "repeat" value "0". Inherits: rdmlBaseType.

Format

An R6Class generator object.

Initialization

```
loopType$new(goto, repeat.n)
```

Fields

`goto` assertCount. The step to go to to form the loop.

`repeat.n` assertCount. Determines how many times the loop is repeated. The first run through the loop is counted as 0, the last loop is "repeat" - 1.

Super class

RDML::rdmlBaseType -> loopType

Methods

Public methods:

- loopType$new()
- loopType$clone()

Method new():

Usage:

```
loopType$new(goto, repeat.n)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
loopType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

mdpsType *mdpsType R6 class.*

Description

Contains `matrix` of melting data points (single data points measured during amplification).

Format

An `R6Class` generator object.

Details

Columns:

tmp (temperature in degrees Celsius at the time of measurement. Every point must have unique value.

fluor fluorescence intensity measured without any correction (including baselining).

Inherits: rdmlBaseType.

Initialization

```
mdpsType$new(fpoints)
```

@section Fields:

`fpoints` assertMatrix. Matrix with amplification data points.

Super class

`RDML::rdmlBaseType` -> `mdpsType`

Methods

Public methods:

- `mdpsType$new()`
- `mdpsType$.asXMLnodes()`
- `mdpsType$clone()`

Method `new()`:

Usage:

```
mdpsType$new(fpoints)
```

Method `.asXMLnodes()`:

Usage:

```
mdpsType$.asXMLnodes(node.name)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
mdpsType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`measureType` *measureType R6 class.*

Description

Can take values:

real time

meltcurve

Inherits: enumType.

Format

An `R6Class` generator object.

Initialization

```
measureType$new(value)
```

@section Fields:

`value` checkString.

Super classes

`RDML::rdmlBaseType` -> `RDML::enumType` -> `measureType`

Methods

Public methods:

- `measureType$clone()`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
measureType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

MergeRDMLs *Merges* RDML *objects*

Description

Merges list of RDML objects. The first object in the list becomes base object. If experiments or runs have same name they will be combined. Reacts with same id, experiment and run overwrite each other!

Usage

```
MergeRDMLs(to.merge)
```

Arguments

to.merge	RDML objects that should be merged.

Examples

```
PATH <- path.package("RDML")
filename <- paste(PATH, "/extdata/", "lc96_bACTXY.rdml", sep ="")
lc96 <- RDML$new(filename)
filename <- paste(PATH, "/extdata/", "stepone_std.rdml", sep ="")
stepone <- RDML$new(filename)
merged <- MergeRDMLs(list(lc96,stepone))
merged$AsDendrogram()
```

new *Creates new instance of* RDML *class object*

Description

This function has been designed to import data from RDML v1.1 and v1.2 format files or from xls file generated by *Applied Biosystems 7500*. To import from xls this file have to contain Sample Setup and Multicomponent Data sheets!

Arguments

filename	string – path to file
show.progress	logical – show loading progress bar if TRUE
conditions.sep	separator for condition defined at sample name
format	string – input file format. Possible values auto, rdml, abi, excel, csv. See Details.

Details

File format options:

auto Tries to detect format by extension. `.xlsx` – `excel`, `.xls` – `abi`, `.csv` – `csv`, other – `rdml`

abi Reads `.xls` files generated by *ABI 7500 v.2*. To create such files use File>Export; check 'Sample Setup' and 'Multicomponent Data'; select 'One File'

excel `.xls` or `.xslx` file with sheets 'description', 'adp', 'mdp'. See example file `table.xlsx`

csv `.csv` file with first column 'cyc' or 'tmp' and fluorescence data in other columns

rdml `.rdml` or `.lc96p` files

Warning

Although the format RDML claimed as data exchange format, the specific implementation of the format at devices from real manufacturers differ significantly. Currently this function is checked against RDML data from devices: *Bio-Rad CFX96*, *Roche LightCycler 96* and *Applied Biosystems StepOne*.

Author(s)

Konstantin A. Blagodatskikh <k.blag@yandex.ru>, Stefan Roediger <stefan.roediger@b-tu.de>, Michal Burdukiewicz <michalburdukiewicz@gmail.com>

Examples

```
## Not run:
## Import from RDML file
PATH <- path.package("RDML")
filename <- paste(PATH, "/extdata/", "lc96_bACTXY.rdml", sep ="")
lc96 <- RDML$new(filename)

## Some kind of overview for lc96
lc96$AsTable(name.pattern = sample[[react$sample$id]]$description)
lc96$AsDendrogram()

## End(Not run)
```

---

nucleotideType *nucleotideType R6 class.*

Description

Type of nucleic acid used as a template in the experiment. May have following values:

DNA

genomic DNA

cDNA

RNA

Format

An R6Class generator object.

Details

Inherits: enumType.

Initialization

```
nucleotideType$new(value)
```

@section Fields:

`value` checkString. Value.

Super classes

RDML::rdmlBaseType -> RDML::enumType -> nucleotideType

Methods

Public methods:

- nucleotideType$clone()

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
nucleotideType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

---

oligoType *oligoType R6 class.*

Description

Inherits: rdmlBaseType.

Format

An R6Class generator object.

Initialization

```
oligoType$new(threePrimeTag = NULL,
  fivePrimeTag = NULL, sequence)
```

@section Fields:

`threePrimeTag` checkString. Description of three prime modification (if present).

`fivePrimeTag` checkString. Description of five prime modification (if present).

`sequence` checkString.

Super class

RDML::rdmlBaseType -> oligoType

Methods

Public methods:

- oligoType$new()
- oligoType$clone()

Method new():

Usage:

```
oligoType$new(threePrimeTag = NULL, fivePrimeTag = NULL, sequence)
```

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
oligoType$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

---

pauseType *pauseType R6 class.*

Description

This step allows to pause at a certain temperature. It is typically the last step in an amplification protocol. Inherits: rdmlBaseType.

Format

An R6Class generator object.

Initialization

```
pauseType$new(temperature)
```

Fields

temperature checkNumber. The temperature in degrees Celsius maintained during the pause.

Super class

RDML::rdmlBaseType -> pauseType

Methods

Public methods:

- pauseType$new()
- pauseType$clone()

Method `new()`:

Usage:

```
pauseType$new(temperature)
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
pauseType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`pcrFormatType` *pcrFormatType R6 class.*

Description

The display format of the PCR, analogous to the the qPCR instrument run format. Inherits: rdmlBaseType.

Format

An `R6Class` generator object.

Details

Rotor formats always have 1 column; rows correspond to the number of places in the rotor. Values for common formats are:

Format	rows	columns	rowLabel	columnLabel
single-well	1	1	123	123
48-well plate	6	8	ABC	123
96-well plate	8	12	ABC	123
384-well plate	16	24	ABC	123
1536-well plate	32	48	ABC	123
3072-well array	32	96	A1a1	A1a1
5184-well chip	72	72	ABC	123
32-well rotor	32	1	123	123
72-well rotor	72	1	123	123
100-well rotor	100	1	123	123
free format	-1	1	123	123

If rows field has value -1, the function will not try to reconstruct a plate and just display all run data in a single column. If the columns field has value 1 then the function will not display a column label.

Initialization

```
pcrFormatType$new(rows, columns, rowLabel, columnLabel)
```

@section Fields:

`rows` checkCount.

`columns` checkCount.

`rowLabel` labelFormatType.

`columnLabel` labelFormatType.

Super class

`RDML::rdmlBaseType` -> `pcrFormatType`

Methods

Public methods:

- `pcrFormatType$new()`
- `pcrFormatType$clone()`

Method `new()`**:**

Usage:

```
pcrFormatType$new(rows, columns, rowLabel, columnLabel)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
pcrFormatType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`primingMethodType` *primingMethodType R6 class.*

Description

The primers used in the reverse transcription. Can take values:

oligo-dt

random

target-specific

oligo-dt and random

other

Format

An R6Class generator object.

Details

Inherits: enumType.

Initialization

```
primingMethodType$new(value)
```

@section Fields:

`value` checkString. Value.

Super classes

RDML::rdmlBaseType -> RDML::enumType -> primingMethodType

Methods

Public methods:

- primingMethodType$clone()

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
primingMethodType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

---

quantityType *quantityType R6 class.*

Description

A quantity is always defined by its value and its unit. Inherits: rdmlBaseType.

Format

An R6Class generator object.

Initialization

```
quantityType$new(value, unit)
```

@section Fields:

`value` checkNumber. Value.

`unit` quantityUnitType. Unit.

Super class

RDML::rdmlBaseType -> quantityType

Methods

Public methods:

- quantityType$new()
- quantityType$clone()

Method new():

Usage:

```
quantityType$new(value, unit)
```

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
quantityType$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

---

quantityUnitType *quantityUnitType R6 class.*

Description

The unit the quantity. Can take values:

cop copies per microliter

fold fold change

dil dilution (10 would mean 1:10 dilution)

nMol nanomol per microliter

ng nanogram per microliter

other other unit (must be linear, no exponents or logarithms allowed)

Format

An R6Class generator object.

Details

Inherits: enumType.

Initialization

quantityUnitType$new(value)

@section Fields:

value checkString. Value.

Super classes

RDML::rdmlBaseType -> RDML::enumType -> quantityUnitType

Methods

Public methods:

- quantityUnitType$clone()

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
quantityUnitType$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

RDML *R6 class* RDML *– contains methods to read and overview fluorescence data from RDML v1.1 and v1.2 format files*

Description

This class is a container for RDML format data (Lefever et al. 2009). The data may be further transformed to the appropriate format of the qpcR (Ritz et al. 2008, Spiess et al. 2008) and chipPCR (Roediger et al. 2015) packages (see RDML.new for import details). Real-time PCR Data Markup Language (RDML) is the recommended file format element in the Minimum Information for Publication of Quantitative Real-Time PCR Experiments (MIQE) guidelines (Bustin et al. 2009). The inner structure of imported data faithfully reflects the structure of RDML file v1.2. All data with the exception for fluorescence values can be represented as data.frame by method AsTable. Such possibility of data representation streamlines sample filtering (by targets, types, etc.) and serves as request for GetFData method, which extracts fluorescence data for specified samples.

Format

An R6Class generator object.

Fields

Type, structure of data and description of fields can be viewed at RDML v1.2 file description. Names of fields are first level of XML tree.

Methods

new creates a new instance of RDML class object (see RDML.new)

AsTable represent RDML data as data.frame (see RDML.AsTable)

GetFData gets fluorescence data (see RDML.GetFData)

SetFData sets fluorescence data (see RDML.SetFData)

Merge merges two RDML to one (see MergeRDMLs)

AsDendrogram represents structure of RDML object as dendrogram(see RDML.AsDendrogram)

Super class

RDML::rdmlBaseType -> RDML

Methods

Public methods:

- RDML$AsDendrogram()
- RDML$AsTable()
- RDML$GetFData()
- RDML$new()
- RDML$AsXML()
- RDML$SetFData()
- RDML$clone()

Method AsDendrogram():

Usage:

```
RDML$AsDendrogram(plot.dendrogram = TRUE)
```

Method AsTable():

Usage:

```
RDML$AsTable(
  .default = list(exp.id = experiment$id$id, run.id = run$id$id, react.id = react$id$id,
    position = react$position, sample = react$sample$id, target = data$tar$id,
     target.dyeId = target[[data$tar$id]]$dyeId$id, sample.type =
     sample[[react$sample$id]]$type$value, adp = !is.null(data$adp), mdp =
     !is.null(data$mdp)),
  name.pattern = paste(react$position, react$sample$id,
    private$.sample[[react$sample$id]]$type$value, data$tar$id, sep = "_"),
  add.columns = list(),
  treat.null.as.na = FALSE,
  ...
)
```

Method GetFData():

Usage:

```
RDML$GetFData(request, dp.type = "adp", long.table = FALSE)
```

Method `new()`**:**

Usage:

```
RDML$new(
  filename,
  show.progress = TRUE,
  conditions.sep = NULL,
  format = "auto"
)
```

Method `AsXML()`**:**

Usage:

```
RDML$AsXML(file.name)
```

Method `SetFData()`**:**

Usage:

```
RDML$SetFData(fdata, description, fdata.type = "adp")
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
RDML$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Konstantin A. Blagodatskikh <k.blag@yandex.ru>, Stefan Roediger <stefan.roediger@b-tu.de>, Michal Burdukiewicz <michalburdukiewicz@gmail.com>

References

RDML format http://www.rdml.org/ `R6` package http://cran.r-project.org/web/packages/R6/index.html

`qpcR` package http://cran.r-project.org/web/packages/qpcR/index.html

`chipPCR` package: http://cran.r-project.org/web/packages/chipPCR/index.html

Roediger S, Burdukiewicz M and Schierack P (2015). chipPCR: an R Package to Pre-Process Raw Data of Amplification Curves. *Bioinformatics* first published online April 24, 2015 doi:10.1093/bioinformatics/btv205

Ritz, C., Spiess, A.-N., 2008. qpcR: an R package for sigmoidal model selection in quantitative real-time polymerase chain reaction analysis. *Bioinformatics* 24, 1549–1551. doi:10.1093/bioinformatics/btn227

Spiess, A.-N., Feig, C., Ritz, C., 2008. Highly accurate sigmoidal fitting of real-time PCR data by introducing a parameter for asymmetry. *BMC Bioinformatics* 9, 221. doi:10.1186/1471-2105-9-221

Bustin, S.A., Benes, V., Garson, J.A., Hellemans, J., Huggett, J., Kubista, M., Mueller, R., Nolan, T., Pfaffl, M.W., Shipley, G.L., Vandesompele, J., Wittwer, C.T., 2009. The MIQE guidelines: minimum information for publication of quantitative real-time PCR experiments. *Clin. Chem.* 55, 611–622. doi:10.1373/clinchem.2008.112797

Lefever, S., Hellemans, J., Pattyn, F., Przybylski, D.R., Taylor, C., Geurts, R., Untergasser, A., Vandesompele, J., RDML consortium, 2009. RDML: structured language and reporting guidelines for real-time quantitative PCR data. *Nucleic Acids Res.* 37, 2065–2069. doi:10.1093/nar/gkp056

Examples

```
## EXAMPLE 1:
## internal dataset lc96_bACTXY.rdml (in 'data' directory)
## generated by Roche LightCycler 96. Contains qPCR data
## with four targets and two types.
## Import with default settings.
PATH <- path.package("RDML")
filename <- paste(PATH, "/extdata/", "lc96_bACTXY.rdml", sep ="")
lc96 <- RDML$new(filename)

tab <- lc96$AsTable(name.pattern = paste(sample[[react$sample$id]]$description,
                                         react$id$id),
                    quantity = sample[[react$sample$id]]$quantity$value)
## Show dyes names
unique(tab$target.dyeId)
## Show types of the samples for dye 'FAM'
library(dplyr)
unique(filter(tab, target.dyeId == "FAM")$sample.type)

## Show template quantities for dye 'FAM' type 'std'
COPIES <- filter(tab, target.dyeId == "FAM", sample.type == "std")$quantity
## Define calibration curves (type of the samples - 'std').
## No replicates.
library(qpcR)
CAL <- modlist(lc96$GetFDData(filter(tab,
                                     target.dyeId == "FAM",
                                     sample.type == "std")),
               baseline="lin", basecyc=8:15)
## Define samples to predict (first two samples with the type - 'unkn').
PRED <- modlist(lc96$GetFDData(filter(tab,
                                     target.dyeId == "FAM",
                                     sample.type == "unkn")),
               baseline="lin", basecyc=8:15)
## Conduct quantification.
calib(refcurve = CAL, predcurve = PRED, thresh = "cpD2",
      dil = COPIES)

## EXAMPLE 2:

## internal dataset lc96_bACTXY.rdml (in 'data' directory)
## generated by Roche LightCycler 96. Contains qPCR data
## with four targets and two types.
## Import with default settings.
library(chipPCR)
PATH <- path.package("RDML")
filename <- paste(PATH, "/extdata/", "lc96_bACTXY.rdml", sep ="")
lc96 <- RDML$new(filename)

tab <- lc96$AsTable(name.pattern = paste(sample[[react$sample$id]]$description,
                                         react$id$id),
                    quantity = sample[[react$sample$id]]$quantity$value)
## Show targets names
```

```
unique(tab$target)
## Fetch cycle dependent fluorescence for HEX chanel
tmp <- lc96$GetFDData(filter(tab, target == "bACT", sample.type == "std"))
## Fetch vector of dillutions
dilution <- filter(tab, target.dyeId == "FAM", sample.type == "std")$quantity

## Use plotCurves function from the chipPCR package to
## get an overview of the amplification curves
tmp <- as.data.frame(tmp)
plotCurves(tmp[,1], tmp[,-1])
oldpar <- par()
par(mfrow = c(1,1))
## Use inder function from the chipPCR package to
## calculate the Cq (second derivative maximum, SDM)
SDMout <- sapply(2L:ncol(tmp), function(i) {
  SDM <- summary(inder(tmp[, 1], tmp[, i]), print = FALSE)[2]
})

## Use the effcalc function from the chipPCR package and
## plot the results for the calculation of the amplification
## efficiency analysis.
plot(effcalc(dilution, SDMout), CI = TRUE)
par(oldpar)

## EXAMPLE 3:
## internal dataset BioRad_qPCR_melt.rdml (in 'data' directory)
## generated by Bio-Rad CFX96. Contains qPCR and melting data.
## Import with custom name pattern.
PATH <- path.package("RDML")
filename <- paste(PATH, "/extdata/", "BioRad_qPCR_melt.rdml", sep ="")
cfx96 <- RDML$new(filename)
## Use plotCurves function from the chipPCR package to
## get an overview of the amplification curves
library(chipPCR)
## Extract all qPCR data
tab <- cfx96$AsTable()
cfx96.qPCR <- as.data.frame(cfx96$GetFDData(tab))
plotCurves(cfx96.qPCR[,1], cfx96.qPCR[,-1], type = "l")

## Extract all melting data
cfx96.melt <- cfx96$GetFDData(tab, dp.type = "mdp")
## Show some generated names for samples.
colnames(cfx96.melt)[2L:5]
## Select columns that contain
## samples with dye 'EvaGreen' and have type 'pos'
## using filtering by names.
cols <- cfx96$GetFDData(filter(tab, grepl("pos_EvaGreen$", fdata.name)),
                        dp.type = "mdp")
## Conduct melting curve analysis.
library(qpcR)
invisible(meltcurve(cols, fluos = 2,
          temps = 1))
```

RDML.AsDendrogram *Represents structure of* RDML *file as dendrogram*

Description

Plots and/or returns the structure of RDML file as dendrogram (tree-like structure.)

Arguments

plot.dendrogram	plots dendrogram if TRUE

Value

dendrogram object

Author(s)

Konstantin A. Blagodatskikh <k.blag@yandex.ru>, Stefan Roediger <stefan.roediger@b-tu.de>, Michal Burdukiewicz <michalburdukiewicz@gmail.com>

Examples

```
PATH <- path.package("RDML")
filename <- paste(PATH, "/extdata/", "BioRad_qPCR_melt.rdml", sep ="")
cfx96 <- RDML$new(filename)
#plot dendrogram
cfx96$AsDendrogram()
#assign dendrogram to the object
dendr <- cfx96$AsDendrogram(plot.dendrogram = FALSE)
```

RDML.AsTable *Represents fields of* RDML *object as* data.frame

Description

Formats particular fields of RDML object as data.frames, filters or passes them to RDML.GetFData and RDML.SetFData functions.

Arguments

.default	list of default columns
name.pattern	expression to form fdata.name (see Examples)
add.columns	list of additional columns
treat.null.as.na	if value is NULL then convert it to NA. Helps to deal with incomplete records.
...	additional columns

Details

By default input this function forms `data.frame` with following columns:

exp.id experiment$id

run.id run$id

react.id react$id

position react$position

sample react$sample

target datatarid

target.dyeId target[[data$id]]$dyeId

sample.type sample[[react$sample]]$type

You can overload default columns list by parameter `.default` but note that columns

```
exp.id, run.id, react.id, target
```

are necessary for usage `AsTable` output as input for `GetFData` and `SetFData`.
Additional columns can be introduced by specifying them at input parameter `...` (see Examples).
All default and additional columns accession expressions must be named.

Experiment, run, react and data to which belongs each fluorescence data vector can be accessed by `experiment`, `run`, `react`, `data` (see Examples).

Result table does not contain data from experiments with ids starting with '.'!

Author(s)

Konstantin A. Blagodatskikh <k.blag@yandex.ru>, Stefan Roediger <stefan.roediger@b-tu.de>, Michal Burdukiewicz <michalburdukiewicz@gmail.com>

Examples

```
## internal dataset stepone_std.rdml (in 'data' directory)
## generated by Applied Biosystems Step-One. Contains qPCR data.
library(chipPCR)
PATH <- path.package("RDML")
filename <- paste(PATH, "/extdata/", "stepone_std.rdml", sep ="")
stepone <- RDML$new(filename)
## Mark fluorescense data which Cq > 30 and add quantities to
## AsTable output.
## Names for fluorescense data will contain sample name and react
## positions
tab <- stepone$AsTable(
         name.pattern = paste(react$sample$id, react$position),
         add.columns = list(cq30 = if(data$cq >= 30) ">=30" else "<30",
         quantity = sample[[react$sample$id]]$quantity$value)
         )
## Show cq30 and quantities
tab[, c("cq30", "quantity")]
## Get fluorescence values for 'std' type samples
```

```
## in format ready for ggplot function
library(dplyr)
fdata <- stepone$GetFData(
           filter(tab, sample.type == "std"),
           long.table = TRUE)
## Plot fdata with colour by cq30 and shape by quantity
library(ggplot2)
ggplot(fdata, aes(x = cyc, y = fluor,
                  group = fdata.name,
                  colour = cq30,
                  shape = as.factor(quantity))) +
                  geom_line() + geom_point()
```

RDML.GetFData *Gets fluorescence data vectors from* RDML *object*

Description

Gets fluorescence data vectors from RDML object for specified method of experiment.

Arguments

request	Output from AsTable method(RDML.AsTable)
dp.type	Type of fluorescence data (i.e. 'adp' for qPCR or 'mdp' for melting)
long.table	Output table is ready for ggplot (See RDML.AsTable for example)

Value

matrix which contains selected fluorescence data and additional information fromm request if long.table = TRUE.

Author(s)

Konstantin A. Blagodatskikh <k.blag@yandex.ru>, Stefan Roediger <stefan.roediger@b-tu.de>, Michal Burdukiewicz <michalburdukiewicz@gmail.com>

Examples

```
## internal dataset BioRad_qPCR_melt.rdml (in 'data' directory)
## generated by Bio-Rad CFX96. Contains qPCR and melting data.
## Import without splitting by targets/types and with
## custom name pattern.
PATH <- path.package("RDML")
filename <- paste(PATH, "/extdata/", "BioRad_qPCR_melt.rdml", sep ="")
cfx96 <- RDML$new(filename)
## Select melting fluorescence data with sample.type 'unkn'.
library(dplyr)
tab <- cfx96$AsTable()
fdata <- cfx96$GetFData(filter(tab, sample.type == "unkn"),
```

```
                          dp.type = "adp")
## Show names for obtained fdata
colnames(fdata)
```

RDML.SetFData *Sets fluorescence data vectors to* RDML *object*

Description

Sets fluorescence data vectors to RDML object for specified method of experiment.

Arguments

data	matrix containing in the first column data corresponding to all fluorescence values in the following columns. The name of the first column is the name of variable and names of other column are fdata.names (links to rows at description).
description	output from AsTable function that describes fluorescence data.
fdata.type	'adp' for qPCR, 'mdp' for melting data.

Examples

```
PATH <- path.package("RDML")
filename <- paste0(PATH, "/extdata/", "stepone_std.rdml")
cfx96 <- RDML$new(filename)
## Use plotCurves function from the chipPCR package to
## get an overview of the amplification curves
library(chipPCR)
## Extract all qPCR data
tab <- cfx96$AsTable()
tab2 <- tab
tab2$run.id <- "cpp"
cfx96.qPCR <- as.data.frame(cfx96$GetFData(tab))
cpp <- cbind(cyc = cfx96.qPCR[, 1],
 apply(cfx96.qPCR[, -1], 2,
   function(y) CPP(x = cfx96.qPCR[, 1], y = y)$y.norm))
cfx96$SetFData(cpp, tab2)
library(ggplot2)
cfx96.gg <- cfx96$GetFData(tab, long.table = TRUE)
cpp.gg <- cfx96$GetFData(tab2,
                         long.table = TRUE)
ggplot(cfx96.gg, aes(x = cyc, y = fluor,
                group=fdata.name)) +
                 geom_line() +
                 ggtitle("Raw data")
ggplot(cpp.gg, aes(x = cyc, y = fluor,
                group=fdata.name)) +
                 geom_line() +
                 ggtitle("CPP processed data")
```

rdmlBaseType *Base R6 class for RDML package.*

Description

Most classes from RDML package inherit this class. It is designed for internal usage and should not be directly accessed.

Format

An R6Class generator object.

Initialization

```
rdmlBaseType$new()
```

Methods

`.asXMLnodes(node.name)` Represents object as XML nodes. Should not be called directly. `node.name` – name of the root node for the generated XML tree.

`print(...)` prints object

Methods

Public methods:

- `rdmlBaseType$.asXMLnodes()`
- `rdmlBaseType$copy()`
- `rdmlBaseType$print()`
- `rdmlBaseType$clone()`

Method `.asXMLnodes()`:

Usage:

```
rdmlBaseType$.asXMLnodes(node.name)
```

Method `copy()`:

Usage:

```
rdmlBaseType$copy()
```

Method `print()`:

Usage:

```
rdmlBaseType$print(...)
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
rdmlBaseType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

rdmlEdit *RDML Editor Graphical User Interface*

Description

Launches graphical user interface that can edit RDML metadata and show qPCR or melting curves.

Usage

```
rdmlEdit()
```

rdmlIdType *rdmlIdType R6 class.*

Description

This element can be used to assign a publisher and id to the RDML file.
Inherits: rdmlBaseType.

Format

An R6Class generator object.

Initialization

```
rdmlIdType$new(publisher, serialNumber,
  MD5Hash = NULL)
```

Fields

`publisher` checkString. RDML file publisher.

`serialNumber` checkString. Serial number.

`MD5Hash` checkString. An MD5Hash calculated over the complete file after removing all rdmlIDTypes and all whitespaces between elements.

Super class

RDML::rdmlBaseType -> rdmlIdType

Methods

Public methods:

- rdmlIdType$new()
- rdmlIdType$clone()

Method new():

Usage:

```
rdmlIdType$new(publisher, serialNumber, MD5Hash = NULL)
```

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
rdmlIdType$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

---

reactIdType *reactIdType R6 class.*

Description

Contains identificator for reactType. Inherits: rdmlBaseType.

Format

An R6Class generator object.

Initialization

```
reactIdType$new(id)
```

@section Fields:

id checkCount. Identificator.

Super classes

RDML::rdmlBaseType -> RDML::idType -> reactIdType

Methods

Public methods:

- reactIdType$new()
- reactIdType$clone()

Method new():

Usage:

```
reactIdType$new(id)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
reactIdType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`reactType` *reactType R6 class.*

Description

A reaction is an independent chemical reaction corresponding for example to a well in a 96 well plate, a capillary in a rotor, a through-hole on an array, etc. Inherits: rdmlBaseType.

Format

An `R6Class` generator object.

Details

The ID of this reaction

Schemas :

- rotor : assign IDs according to the position of the sample on the rotor (1 for the 1st sample, 2 for the 2nd, ...)
- plate (96/384/1536 well) : the IDs are assigned in a row-first/column-second manner. For each row, the samples are numbered according to the increasing column number. At the end of a row, the numbering starts at the first column of the next row. An example for this type of plate can be found below :

	1	2	3	...
A	1	2	3	
B	13	14		
...				

or

	1	2	3	...
1	1	2	3	
2	13	14		
...				

- multi-array plate (BioTrove) : the IDs are assigned in a row-first/column-second manner, ignoring the organisation of sub-arrays. For each row, the samples are numbered according to the increasing column number. At the end of a row, the the next row. An example for this type of plate can be found below : todo...

Initialization

`reactType$new(id, sample, data = NULL, pcrFormat = pcrFormatType$new(8, 12, labelFormatType$new("123")`

@section Fields:

`id` reactIdType. See 'Details'.

`sample` idReferencesType. SampleID - A reference to a sample.

`data list` of dataType.

`position` Human readable form of the `react id` (i.e. '13' -> 'B1')..

Methods

`AsDataFrame(dp.type = "adp")` Represents amplification (`dp.type = "adp"`) or melting (`dp.type = "mdp"`) data points of all targets as one `data.frame`

`.recalcPosition(pcrformat)` Converts `react id` to the human readable form (i.e. '13' -> 'B1'). This converted value can be accessed by `position` field. `pcrFormat` is `pcrFormatType`. Currently, only 'ABC' and '123' are supported as labels. For '123' '123' the `Position` will look like 'r01c01', for 'ABC' '123' it will be 'A01' and for '123' 'ABC' it will be 01A. 'ABC' 'ABC' is not currently supported. Note that 'ABC' will result in loss of information if the experiment contains more than 26 rows!

Super class

`RDML::rdmlBaseType` -> `reactType`

Methods

Public methods:

- `reactType$new()`
- `reactType$GetFData()`
- `reactType$.recalcPosition()`
- `reactType$clone()`

Method `new()`:

Usage:

```
reactType$new(
  id,
  sample,
  data = NULL,
  pcrFormat = pcrFormatType$new(8, 12, labelFormatType$new("ABC"),
    labelFormatType$new("123"))
)
```

Method `GetFData()`:

Usage:

```
reactType$GetFData(dp.type = "adp", long.table = FALSE)
```

Method `.recalcPosition()`:

Usage:

```
reactType$.recalcPosition(pcrFormat)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
reactType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`runType` *runType R6 class.*

Description

A run is a set of reactions performed in one "run", for example one plate, one rotor, one array, one chip. Inherits: rdmlBaseType.

Format

An `R6Class` generator object.

Initialization

```
runType$new(id, description = NULL,
  documentation = NULL, experimenter = NULL, instrument = NULL,
  dataCollectionSoftware = NULL, backgroundDeterminationMethod = NULL,
  cqDetectionMethod = NULL, thermalCyclingConditions = NULL, pcrFormat,
  runDate = NULL, react = NULL)
```

Fields

`id` idType.

`description` checkString.

`documentation` `list` of idReferencesType.

`experimenter` `list` of idReferencesType.

`instrument` checkString. Description of the instrument used to aquire the data.

`dataCollectionSoftware` dataCollectionSoftwareType. Description of the software used to analyze/collect the data.

`backgroundDeterminationMethod` checkString. Description of method used to determine the background.

`cqDetectionMethod` cqDetectionMethodType. Description of method used to calculate the quantification cycle.

`thermalCyclingConditions` idReferencesType. The program used to aquire the data.

`pcrFormat` adpsType.

`runDate` adpsType. Time stamp of data acquisition.

`react` `list` of adpsType.

Methods

`AsDataFrame(dp.type = "adp")` Represents amplification (`dp.type = "adp"`) or melting (`dp.type = "mdp"`) data points as `data.frame`

Super class

`RDML::rdmlBaseType` -> `runType`

Methods

Public methods:

- `runType$new()`
- `runType$GetFData()`
- `runType$UpdateReactsPosition()`
- `runType$clone()`

Method `new()`:

Usage:

```
runType$new(
  id,
  description = NULL,
  documentation = NULL,
  experimenter = NULL,
  instrument = NULL,
  dataCollectionSoftware = NULL,
  backgroundDeterminationMethod = NULL,
  cqDetectionMethod = NULL,
  thermalCyclingConditions = NULL,
  pcrFormat,
  runDate = NULL,
  react = NULL
)
```

Method `GetFData()`:

Usage:

```
runType$GetFData(dp.type = "adp", long.table = FALSE)
```

Method `UpdateReactsPosition()`:

Usage:

```
runType$UpdateReactsPosition()
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
runType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`sampleType` *sampleType R6 class.*

Description

A sample is a template solution with defined concentation. Since dilutions of the same material differ in concentration, they are considered different samples. A technical replicate samples should contain the same name (reactions are performed on the same material), and biological replicates should contain different names (the template derived from the different biological replicates is are divergent). Serial dilutions in a standard curve must have different names (preferably stating their dillution). Inherits: rdmlBaseType.

Format

An `R6Class` generator object.

Initialization

```
sampleType$new(id, description = NULL,
 documentation = NULL, xRef =  NULL, annotation = NULL, type =
 sampleTypeType$new("unkn"), interRunCalibrator = FALSE, quantity = NULL,
 calibratorSample = FALSE, cdnaSynthesisMethod = NULL, templateQuantity =
 NULL)
```

@section Fields:

`id` idType. Concentration of the template in nanogram per microliter in the final reaction mix.

`description` checkString.

`documentation` `list` of idReferencesType.

`xRef` `list` of xRefType.

`annotation` `list` of annotationType.

`type` sampleTypeType.

`interRunCalibrator` checkFlag. `TRUE` if this sample is used as inter run calibrator.

`quantity` quantityType. Quantity - The reference quantity of this sample. It should be only used if the sample is part of a standard curve. The provided value will be used to quantify unknown samples in absolute quantification assays. Only the use of positive integers (like 1, 10, 100, 1000) and fractions (e.g. 1, 0.1, 0.01, 0.001) is acceptable. The use of exponents (1, 2, 3, 4 or -1, -2, -3, -4) if forbidden, because it will not be interpreted as 10E1, 10E2, 10E3, 10E4 or 10E-1, 10E-2, 10E-3, 10E-4.

`calibratorSample` checkFlag. `TRUE` if this sample is used as calibrator sample.

`cdnaSynthesisMethod` cdnaSynthesisMethodType.

`templateQuantity` templateQuantityType.

Super class

`RDML::rdmlBaseType` -> `sampleType`

Methods

Public methods:

- sampleType$new()
- sampleType$clone()

Method `new()`**:**

Usage:

```
sampleType$new(
  id,
  description = NULL,
  documentation = NULL,
  xRef = NULL,
  annotation = NULL,
  type = sampleTypeType$new("unkn"),
  interRunCalibrator = FALSE,
  quantity = NULL,
  calibratorSample = FALSE,
  cdnaSynthesisMethod = NULL,
  templateQuantity = NULL
)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
sampleType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`sampleTypeType` *sampleTypeType R6 class.*

Description

Can take values:

unkn unknown sample

ntc non template control

nac no amplification control

std standard sample

ntp no target present

nrt minusRT

pos positive control

opt optical calibrator sample

Format

An R6Class generator object.

Details

Inherits: enumType.

Initialization

```
sampleTypeType$new(value)
```

@section Fields:

value checkString. Value.

Super classes

RDML::rdmlBaseType -> RDML::enumType -> sampleTypeType

Methods

Public methods:

- sampleTypeType$clone()

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
sampleTypeType$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

---

sequencesType *sequencesType R6 class.*

Description

Inherits: rdmlBaseType.

Format

An R6Class generator object.

Initialization

```
sequencesType$new(forwardPrimer = NULL,
reversePrimer = NULL, probe1 = NULL, probe2 = NULL, amplicon = NULL)
```

@section Fields:

`forwardPrimer` oligoType.

`reversePrimer` oligoType.

`probe1` oligoType.

`probe2` oligoType.

`amplicon` oligoType.

Super class

`RDML::rdmlBaseType` -> `sequencesType`

Methods

Public methods:

- `sequencesType$new()`
- `sequencesType$clone()`

Method `new()`:

Usage:

```
sequencesType$new(
  forwardPrimer = NULL,
  reversePrimer = NULL,
  probe1 = NULL,
  probe2 = NULL,
  amplicon = NULL
)
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
sequencesType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

SetFData — RDML$SetFData() *wrapper*

Description

Read more at RDML.SetFData

Usage

```
SetFData(obj, ...)
```

Arguments

obj	RDML object.
...	SetFData params.

stepType — *stepType R6 class.*

Description

Inherits: rdmlBaseType.

Format

An R6Class generator object.

Initialization

```
stepType$new(nr, description = NULL,
  temperature = NULL, gradient = NULL, loop = NULL, pause = NULL, lidOpen =
  NULL)
```

Fields

nr checkCount. The incremental number of the step. First step should have value 1. The increment between steps should be constant and equivalent to 1.

description checkString.

temperature temperatureType.

gradient gradientType.

loop loopType.

pause pauseType.

lidOpen lidOpenType.

Super class

RDML::rdmlBaseType -> stepType

Methods

Public methods:

- stepType$new()
- stepType$clone()

Method new():

Usage:

```
stepType$new(
  nr,
  description = NULL,
  temperature = NULL,
  gradient = NULL,
  loop = NULL,
  pause = NULL,
  lidOpen = NULL
)
```

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
stepType$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

---

targetType *targetType R6 class.*

Description

A target is a PCR reaction with defined set of primers. PCR reactions for the same gene with distinct primer sequences are considered different targets. Inherits: rdmlBaseType.

Format

An R6Class generator object.

Initialization

```
targetType$new(id, description = NULL,
  documentation = NULL, xRef = NULL, type, amplificationEfficiencyMethod =
  NULL, amplificationEfficiency = NULL, amplificationEfficiencySE = NULL,
  detectionLimit = NULL, dyeId, sequences = NULL, commercialAssay = NULL)
```

Fields

`id` idType.
`description` checkString.
`documentation` `list` of idReferencesType.
`xRef` `list` of xRefType.
`type` targetTypeType.
`amplificationEfficiencyMethod` checkString.
`amplificationEfficiency` checkNumber.
`amplificationEfficiencySE` checkNumber.
`detectionLimit` checkNumber.
`dyeId` idReferencesType.
`sequences` sequencesType.
`commercialAssay` commercialAssayType.

Super class

`RDML::rdmlBaseType` -> `targetType`

Methods

Public methods:

- `targetType$new()`
- `targetType$clone()`

Method `new()`**:**

Usage:

```
targetType$new(
  id,
  description = NULL,
  documentation = NULL,
  xRef = NULL,
  type,
  amplificationEfficiencyMethod = NULL,
  amplificationEfficiency = NULL,
  amplificationEfficiencySE = NULL,
  detectionLimit = NULL,
  dyeId,
  sequences = NULL,
  commercialAssay = NULL
)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
targetType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`targetTypeType` *targetTypeType R6 class.*

Description

Can take values:

ref reference target

toi target of interest

Inherits: enumType.

Format

An `R6Class` generator object.

Initialization

```
targetTypeType$new(value)
```

@section Fields:

`value` checkString.

Super classes

`RDML::rdmlBaseType` -> `RDML::enumType` -> `targetTypeType`

Methods

Public methods:

- `targetTypeType$clone()`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
targetTypeType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`temperatureType` *temperatureType R6 class.*

Description

This step keeps a constant temperature on the heat block. Inherits: baseTemperatureType.

Format

An `R6Class` generator object.

Initialization

```
temperatureType$new(temperature, ...)
```

Fields

`temperature` checkNumber. The temperature of the step in degrees Celsius.

`...` Params of parent class baseTemperatureType.

Super classes

`RDML::rdmlBaseType` -> `RDML::baseTemperatureType` -> `temperatureType`

Methods

Public methods:

- `temperatureType$new()`
- `temperatureType$clone()`

Method `new()`:

Usage:

```
temperatureType$new(temperature, ...)
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
temperatureType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

templateQuantityType *templateQuantityType R6 class.*

Description

Inherits: rdmlBaseType.

Format

An R6Class generator object.

Initialization

```
templateQuantityType$new(conc, nucleotide)
```

@section Fields:

conc checkNumber. Concentration of the template in nanogram per microliter in the final reaction mix.

nucleotide nucleotideType.

Super class

RDML::rdmlBaseType -> templateQuantityType

Methods

Public methods:

- templateQuantityType$new()
- templateQuantityType$clone()

Method new():

Usage:

```
templateQuantityType$new(conc, nucleotide)
```

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
templateQuantityType$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

thermalCyclingConditionsType *thermalCyclingConditionsType R6 class.*

Description

A cycling program for PCR or to amplify cDNA. Inherits: rdmlBaseType.

Format

An R6Class generator object.

Initialization

```
thermalCyclingConditionsType$new(id,
  description = NULL, documentation = NULL, lidTemperature = NULL,
  experimenter = NULL, step)
```

Fields

`id` idType.

`description` checkString.

`documentation` `list` of idReferencesType.

`lidTemperature` checkNumber. The temperature in degrees Celsius of the lid during cycling.

`experimenter` `list` of idReferencesType. Reference to the person who made or uses this protocol.

`step` `list` of stepType. The steps a protocol runs through to amplify DNA.

Super class

`RDML::rdmlBaseType` -> `thermalCyclingConditionsType`

Methods

Public methods:

- `thermalCyclingConditionsType$new()`
- `thermalCyclingConditionsType$clone()`

Method `new()`:

Usage:

```
thermalCyclingConditionsType$new(
  id,
  description = NULL,
  documentation = NULL,
  lidTemperature = NULL,
  experimenter = NULL,
  step
)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
thermalCyclingConditionsType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`xRefType` *xRefType R6 class.*

Description

Inherits: rdmlBaseType.

Format

An `R6Class` generator object.

Initialization

```
xRefType$new(name = NULL, id = NULL)
```

@section Fields:

`name` checkString. Reference to an external database, for example "GenBank".

`id` checkString. The ID of the entry within the external database, for example "AJ832138".

Super class

`RDML::rdmlBaseType` -> `xRefType`

Methods

Public methods:

- `xRefType$new()`
- `xRefType$clone()`

Method `new()`**:**

Usage:

```
xRefType$new(name = NULL, id = NULL)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
xRefType$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`[.GetFData` *Extract data points from* `RDML` *object*

Description

Extract data points from `RDML` object as.data.frame.

Usage

```
## S3 method for class 'RDML'
x[i, j, dp.type = "adp"]
```

Arguments

`x`	`RDML` object.
`i, j`	indices.
`dp.type`	Type of fluorescence data (i.e. 'adp' for qPCR or 'mdp' for melting).

Index