Package 'openPrimeR'

August 22, 2017

Title Multiplex PCR Primer Design and Analysis

Version 0.99.0

Description An implementation of methods for designing, evaluating, and comparing primer sets for multiplex PCR. Primers are designed by solving a set cover problem such that the number of covered template sequences is maximized with the smallest possible set of primers. To guarantee that high-quality primers are generated, only primers fulfilling constraints on their physicochemical properties are selected. All relevant functions are accessible through a user-friendly Shiny application.

Depends R (>= 3.3.3)

License GPL-2

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

Imports Biostrings (>= 2.38.4),
XML (>= 3.98-1.4),
scales (>= 0.4.0),
reshape2 (>= 1.4.1),
seqinr (>= 3.3-3),
IRanges (>= 2.4.8),
GenomicRanges (>= 1.22.4),
ggplot2 (>= 2.1.0),
plyr (>= 1.8.4),
dplyr (>= 0.5.0),
stringdist (>= 0.9.4.1),
stringr (>= 1.0.0),
RColorBrewer (>= 1.1-2),
DECIPHER (>= 1.16.1),
lpSolveAPI (>= 5.5.2.0-17),
digest (>= 0.6.9),
Hmisc (>= 3.17-4),
ape (>= 3.5),
BiocGenerics (>= 0.16.1),
S4Vectors (>= 0.8.11),
foreach (>= 1.4.3),
magrittr (>= 1.5),

distr (>= 2.6),
distrEx (>= 2.6),
fitdistrplus (>= 1.0-7),
uniqtag (>= 1.0),
openxlsx (>= 4.0.17),
grid (>= 3.1.0),
grDevices (>= 3.1.0),
stats (>= 3.1.0),
utils (>= 3.1.0),
methods (>= 3.1.0)

Suggests shiny (>= 1.0.2),
shinyjs (>= 0.9),
shinyBS (>= 0.61),
DT (>= 0.2),
testthat (>= 1.0.2),
knitr (>= 1.13),
rmarkdown (>= 1.0),
devtools (>= 1.12.0),
doParallel (>= 1.0.10),
pander (>= 0.6.0),
learnr (>= 0.9)

SystemRequirements MAFFT (>= 7.305),
OligoArrayAux (>= 3.8),
ViennaRNA (>= 2.2.4),
MELTING (>= 5.1.1),
Pandoc (>= 1.12.3)

biocViews Software, Technology

VignetteBuilder knitr

Collate 'Comparison.R'
'templates.R'
'primers.R'
'IO.R'
'IO_view.R'
'Ippolito.R'
'RefCoverage.R'
'Tiller.R'
'ambiguity.R'
'check_stop_codons.R'
'con_annealing_temperature.R'
'con_dimerization.R'
'con_gc_clamp.R'
'con_gc_ratio.R'
'con_melting_temperature.R'
'con_primer_coverage.R'
'con_primer_efficiency.R'
'con_primer_secondary_structures.R'
'con_repeats.R'
'con_runs.R'
'con_template_secondary_structures.R'
'constraints.R'
'constraints_eval.R'

'errors.R'
'filters.R'
'helper_functions.R'
'initialize_primers.R'
'initialize_primers_tree.R'
'openPrimeR.R'
'optimization_ILP.R'
'optimization_algo.R'
'optimization_global.R'
'optimization_greedy.R'
'plots_comparison.R'
'settings.R'
'plots_constraints.R'
'plots_coverage.R'
'plots_filtering.R'
'primer_significance.R'
'startApp.R'
'zzz.R'

R topics documented:

`openPrimeR-package` *Multiplex PCR Primer Design and Analysis.*

Description

With openPrimeR you can evaluate existing primers or design novel primers for multiplex polymerase chain reaction that are optimized with respect to the coverage of template sequences and the physicochemical properties of the primers.

Details

For designing primers, you just need the function `design_primers` from **openPrimeR**. As a minimal input, this function requires:

A set of template sequences You an load a `Templates` object with `read_templates`.

Settings for primer design You can load a `DesignSettings` object from a (supplied) XML file with `read_settings`. The settings can be easily customized using the setters `constraints`, `constraintLimits`, `cvg_constraints`, `conOptions`, and `PCR`.

For evaluating existing primers you can load a FASTA or CSV file containing the primers and templates of of interest using `read_primers` and `read_templates`, respectively. After evaluating the properties of the primers using `check_constraints`, you can interpret the results with several functions. For example, you can analyze the coverage of the template sequences using `get_cvg_stats`, determine the deviation from the target constraints using `plot_constraint_deviation`, or create a comprehensive report on the analyzed primers using `create_report`. In order to compare several primer sets with each other, you can create a table of the properties of the primer sets using `get_comparison_table` or create a full report, again using `create_report`.

Package options

openPrimeR uses the following options:

`openPrimeR.constraint_order` The identifiers of constraints in the order they are applied during the filtering procedure. This order is maintained when loading a `DesignSettings` object.

`openPrimeR.relax_order` The identifiers of constraints in the order in which they shall be relaxed during the relaxation procedure when designing primers.

`openPrimeR.plot_abbrev` The maximal number of allowed characters for tick labels in plots.

`openPrimeR.plot_colors` A named vector providing the identifiers of RColorBrewer palettes. Each vector entry provides the plotting colors for a specific type of stratification (i.e. by run, constraint, or primer). The palettes should provide at least eight colors.

Author(s)

Maintainer: Matthias Döring <mdoering@mpi-inf.mpg.de>

Authors:

- Nico Pfeifer <pfeifer@informatik.uni-tuebingen.de>

`adjust_binding_regions` *Adjustment of Existing Binding Regions.*

Description

Adjusts the existing annotation of binding regions by specifying a new binding interval relative to the existing binding region.

Usage

```
adjust_binding_regions(template.df, region.fw, region.rev)
```

Arguments

`template.df`	A `Templates` object providing the template sequences for which the binding regions shall be adjusted.
`region.fw`	Interval of new binding regions relative to the forward binding region defined in `template.df`.
`region.rev`	Interval of new binding regions relative to the reverse binding region defined in `template.df`

Details

The new binding intervals provided by `fw` and `rev` for forward and reverse primers, respectively, are provided relative to the existing definition of binding regions in `template.df`, which can be set using `assign_binding_regions`. When specifying relative positions, position `0` is defined as the first position after the end of the existing binding region. Hence, negative positions relate to regions within the existing binding region while non-negative values extend the binding region further.

Value

A `Templates` object with updated binding regions.

Examples

```
data(Ippolito)
# Extend the binding region by one position
relative.interval <- c(-max(template.df$Allowed_End_fw), 0)
template.df.adj <- adjust_binding_regions(template.df, relative.interval)
# compare old and new annotations:
head(cbind(template.df$Allowed_Start_fw, template.df$Allowed_End_fw))
head(cbind(template.df.adj$Allowed_Start_fw, template.df.adj$Allowed_End_fw))
```

`assign_binding_regions`

Assignment of Template Binding Regions.

Description

Assigns the primer target binding regions to a set of template sequences.

Usage

```
assign_binding_regions(template.df, fw = NULL, rev = NULL,
  optimize.region = FALSE, primer.length = 20, gap.char = "-")
```

Arguments

`template.df`	A `Templates` object containing the sequences for which primer binding regions should be annotated.
`fw`	Binding regions for forward primers. Either a numeric interval indicating a uniform binding range relative to the template 5' end or a path to a FASTA file providing binding sequences for every template. If `fw` is missing, only `rev` is considered.
`rev`	Binding regions for reverse primers. Either a numeric interval indicating a uniform binding range relative to the template 3' end or the path to a FASTA file providing binding sequences for every template. If `rev` is missing, only `fw` is considered.
`optimize.region`	If `TRUE`, the binding regions specified via `fw` and `rev` are adjusted such that binding regions that may form secondary structures are avoided. This feature requires ViennaRNA (see notes). If `FALSE` (the default), the input binding regions are not modified.

primer.length	A numeric scalar providing the probe length that is used for for adjusting the primer binding regions when optimize.region is TRUE.
gap.char	The character indicating gaps in aligned sequences. The default is "-".

Details

The arguments fw and rev provide data describing the binding regions of the forward and reverse primers, respectively. To specify binding regions for each template individually, fw and rev should provide the paths to FASTA files. The headers of these FASTA file should match the headers of the loaded template.df and the sequences in the files specified by fw and rev should indicate the target binding regions.

To specify uniform binding regions, fw and rev should be numeric intervals indicating the allowed binding range for primers in the templates. The fw interval is specified with respect to the 5' end, while the rev interval is specified with respect to the template 3' end. If optimize.region is TRUE, the input binding region is adjusted such that regions forming secondary structures are avoided.

Value

An object of class Templates with assigned binding regions.

Note

The program ViennaRNA (https://www.tbi.univie.ac.at/RNA/) must be installed to perform the computations that are triggered when optimize.region is set to TRUE.

See Also

Other template functions: Templates-class, read_templates, update_template_cvg, write_templates

Examples

```
data(Ippolito)
# Assignment of individual binding regions
l.fasta.file <- system.file("extdata", "IMGT_data", "templates",
     "Homo_sapiens_IGH_functional_leader.fasta", package = "openPrimeR")
template.df.individual <- assign_binding_regions(template.df, l.fasta.file, NULL)
# Assign the first/last 30 bases as forward/reverse binding regions
template.df.uniform <- assign_binding_regions(template.df, c(1,30), c(1,30))
# Optimization of binding regions (requires ViennaRNA)
## Not run: template.df.opti <- assign_binding_regions(template.df, c(1,30), c(1,30),
                     optimize.region = TRUE, primer.length = 20)
## End(Not run)
```

---

check_constraints *Evaluation of Primer Constraints.*

Description

Determines whether a set of primers fulfills the constraints on the properties of the primers.

Usage

```
check_constraints(primer.df, template.df, settings,
  active.constraints = names(constraints(settings)),
  to.compute.constraints = active.constraints, for.shiny = FALSE,
  updateProgress = NULL)
```

Arguments

primer.df	A Primers object containing the primers whose properties are to be checked.
template.df	A Templates object containing the template sequences corresponding to primer.df.
settings	A DesignSettings object containing the constraints that are to be evaluated.
active.constraints	A subset of the constraint identifiers provided by settings that are to be checked for fulfillment. By default active.constraints is NULL such that all constraints found in settings are evaluated. Otherwise, only the constraints specified via active.constraints that are available in settings are considered.
to.compute.constraints	Constraints that are to be computed. By default, to.compute.constraints is set to NULL such that all active.constraints are computed. If to.compute.constraints is a subset of active.constraints, all constraints specified via active.constraints are evaluated for fulfillment, but only the constraints in to.compute.constraints are newly calculated.
for.shiny	Whether the output of the function shall be formatted as HTML. The default setting is FALSE.
updateProgress	Progress callback function for shiny. The defaut is NULL meaning that no progress is monitored via the Shiny interface.

Details

When the optional argument active.constraints is supplied, only a subset of the constraints provided in settings is evaluated. Only constraints that are defined in settings can be computed. For a detailed description of all possible constraints and their options, please consider the ConstraintSettings documentation.

Value

A Primers object that is augmented with columns indicating the results for each evaluated constraint. The constraints_passed column indicates whether all active.constraints were fulfilled. The EVAL_* columns indicate the fulfillment of primer-specific constraints. The T_EVAL_* columns indicate the fulfillment of template-specific (e.g. coverage-based) constraints. For the coverage computations, columns prefixed by Basic_, indicate the results from string matching, while all other results (e.g. primer_coverage) indicate the expected coverage after applying the coverage constraints specified in settings. Columns prefixed by Off_ indicate off-target binding results.

Note

Please note that some constraints can only be computed if additional software is installed, please see the documentation of DesignSettings for an overview.

See Also

Other primer functions: Primers-class, check_restriction_sites, create_report, design_primers, filter_primers, get_initial_primers, primer_significance, score_degen, write_primers

Examples

```
data(Ippolito)
settings.xml <- system.file("extdata", "settings",
                "C_Taq_PCR_high_stringency.xml", package = "openPrimeR")
settings <- read_settings(settings.xml)
# Check all constraints in 'settings' for the first two primers:
constraint.df <- check_constraints(primer.df[1:2,], template.df,
                    settings, active.constraints = names(constraints(settings)))
# Summarize the evaluation results
summary(constraint.df)
```

---

check_restriction_sites

Identification of Sequence Restriction Sites.

Description

Checks a set of primers for the presence of restriction sites. To reduce the number of possible restriction sites, only unambiguous restriction sites are taken into account and only common (typically used) restriction sites are checked if a common restriction site can be found in a sequence.

Usage

```
check_restriction_sites(primer.df, template.df, adapter.action = c("warn",
  "rm"), selected = NULL, only.confident.calls = TRUE,
  updateProgress = NULL)
```

Arguments

primer.df	A Primers object containing the primer nucleotide sequences to be checked for restriction sites.
template.df	An object of class Templates containing the templates corresponding to primer.df.
adapter.action	The action to be performed when adapter sequences are found. Either "warn" to issue a warning about adapter sequences or "rm" to remove identified adapter sequences. Currently, only the default setting ("warn") is supported.
selected	Names of restriction sites that are to be checked. By default selected is NULL in which case all REBASE restriction sites are taken into account.
only.confident.calls	Whether only confident calls of restriction sites are returned. All restriction site call is considered *confident* if the restriction site is located in a region that does not match the template sequences. Note that this classification requires that the provided primers are somehow complementary to the provided templates. In contrast, non-confident restriction site calls are based solely on the primer sequences and do not take the templates into account, resulting in more false positive calls of restriction sites.
updateProgress	A Shiny progress callback function. The default is NULL meaning that no progress is tracked via the Shiny app.

Value

A data frame with possible restriction sites found in every primer.

References

Roberts, R.J., Vincze, T., Posfai, J., Macelis, D. (2010) REBASE–a database for DNA restriction and modification: enzymes, genes and genomes. Nucl. Acids Res. 38: D234-D236. http://rebase.neb.com

See Also

Other primer functions: `Primers-class`, `check_constraints`, `create_report`, `design_primers`, `filter_primers`, `get_initial_primers`, `primer_significance`, `score_degen`, `write_primers`

Examples

```
data(Ippolito)
# Check the first primer for restriction sites:
site.df <- check_restriction_sites(primer.df[1,], template.df)
```

---

classify_design_problem

Classification of the Difficulty of a Primer Design Task.

Description

Uses reference beta distributions of primer coverage ratios to classify a primer design task into the groups ranging from *easy* to *hard*. For *easy* tasks, it should not be a problem to design a small primer set. For *hard* tasks, however, a small set of primers may not be achievable.

Usage

```
classify_design_problem(template.df, mode.directionality = c("both", "fw",
  "rev"), primer.length = 18, primer.estimate = FALSE, required.cvg = 1)
```

Arguments

`template.df`	A `Templates` object providing the template sequences for which the difficulty of designing primers shall be estimated.
`mode.directionality`	The directionality of the primers that are to be designed. Either `fw` for forward primers, `rev` for reverse primers, or `both` for primers of both directions. By default, both directions are considered.
`primer.length`	A scalar numeric providing the target length of the designed primers. The default length of generated primers is set to `18`.
`primer.estimate`	Whether the number of required primers shall be estimated. By default (`FALSE`), the number of required primers is not estimated.
`required.cvg`	A scalar numeric in the range [0,1] providing the target coverage ratio for designing primers. The `required.cvg` is used only when `primer.estimate` is set to `TRUE` such that a solution to the set cover problem is required.

Details

The difficulty of a primer design task is evaluated by estimating the distribution of coverage ratios per primer by performing exact string matching with primers of length `primer.length`, which are constructed by extracting template subsequences. Next, a beta distribution is fitted to the estimated coverage distribution, which is then compare to reference distributions representing primer design problems of different difficulties via the total variance distance. The difficulty of the input primer design problem is found by selecting the class of the reference distributions that has the smallest distance to the estimated coverage distribution. An estimate of the required number of primers to reach a given `required.cvg` can be computed by setting `primer.estimate` to `TRUE`. Since this estimate is based solely on perfect matching primers, the number of primers that would actually be required is typically less.

Value

A list with the following fields:

`Classification` The estimated difficulty of the primer design task.

`Class-Distances` The total variance distance of the fitted beta distribution to the reference distribution.

`Confidence` The confidence in the estimate of the design tasks' difficulty as based on the class distances.

`Uncertain` Whether the classification is highly uncertain, that is low-confidence.

`Nbr_primers_fw` **and** `Nbr_primers_rev` The respective number of required forward and reverse primers if `primer.estimate` was set to `TRUE`.

Examples

```
data(Ippolito)
design.estimate <- classify_design_problem(template.df)
# Estimate the number of required primers to amplify the first 10 templates
design.estimate.nbr <- classify_design_problem(template.df[1:10,], mode.directionality = "fw",
                          primer.length = 20, primer.estimate = TRUE)
```

Comparison *Evaluated Primer Data for Comparison.*

Description

Evaluated primer sets targeting the functional human IGH immunoglobulin genes. The sets were generated using the default evaluation settings of openPrimeR. The primer sets were gathered from IMGT and the literature.

Usage

```
data(Comparison)
```

Format

`primer.data` and `template.data` are lists of `Primers` and `Templates` objects, respectively.

References

IMGT®, the international ImMunoGeneTics information system® http://www.imgt.org (founder and director: Marie-Paule Lefranc, Montpellier, France).

Examples

```
# Load 'primer.data' and 'template.data'
data(Comparison)
# Explore the first entry of the primer and template data:
primer.data[[1]]
template.data[[1]]
# Summarize the primer properties:
get_comparison_table(template.data, primer.data)
```

---

conOptions *Getter/Setter for Constraint Options.*

Description

Gets the constraint settings of the provided `DesignSettings` object `x`.

Sets the constraint settings of the provided `DesignSettings` object `x`.

Usage

```
conOptions(x)

## S4 method for signature 'DesignSettings'
conOptions(x)

conOptions(x) <- value

## S4 replacement method for signature 'DesignSettings'
conOptions(x) <- value
```

Arguments

`x`	A `DesignSettings` object.
`value`	A list with constraint options. The permissible fields of the list and their types are documented in the `ConstraintOptions` class.

Value

Gets the constraint options list.

Sets the specified list of constraint options.

See Also

Other settings functions: `ConstraintOptions-class`, `ConstraintSettings-class`, `CoverageConstraints-class`, `DesignSettings-class`, `PCR_Conditions-class`, `PCR`, `constraintLimits`, `constraints`, `cvg_constraints`, `read_settings`

Examples

```
# Load some settings
data(Ippolito)
# View the active constraint options
conOptions(settings)
# Prevent mismatch binding events
conOptions(settings)$allowed_mismatches <- 0
# View available constraint options
settings
```

`constraintLimits` *Getter/Setter for Constraint Limits.*

Description

Gets the constraint limits that are defined in the provided `DesignSettings` object `x`.

Sets the constraint limits of the provided `DesignSettings` object `x`.

Usage

```
constraintLimits(x)

## S4 method for signature 'DesignSettings'
constraintLimits(x)

constraintLimits(x) <- value

## S4 replacement method for signature 'DesignSettings'
constraintLimits(x) <- value
```

Arguments

`x`	A `DesignSettings` object whose constraint limits are to be modified.
`value`	A list with constraint boundaries. The permissible fields of the list are provided in `ConstraintSettings`.

Value

Gets the list of constraint limits.

Sets the list of constraint limits.

See Also

Other settings functions: `ConstraintOptions-class`, `ConstraintSettings-class`, `CoverageConstraints-class`, `DesignSettings-class`, `PCR_Conditions-class`, `PCR`, `conOptions`, `constraints`, `cvg_constraints`, `read_settings`

Examples

```
# Load some settings
data(Ippolito)
# View the active constraint limits
constraintLimits(settings)
# Extend the GC relaxation limit
constraintLimits(settings)$gc_clamp <- c("min" = 0, "max" = 6)
# View available constraints
settings
```

ConstraintOptions-class

Class for Constraint Options.

Description

The `ConstraintOptions` class encapsulates the options for constraint computations.

Value

A `ConstraintOptions` object.

Slots

`status` Named boolean vector indicating which of the possible options are active (`TRUE`) and which are not (`FALSE`).

`settings` Named list with constraint options. The following fields are permissible:

allowed_mismatches: The maximal number of allowed mismatches between a primer and a template sequence. If the number of mismatches of a primer with a template exceeds the specified value, the primer is not considered to cover the corresponding template when the coverage is being computed.

allowed_other_binding_ratio: Ratio of allowed binding events outside the target binding ratio. This value should be in the interval [0,1]. If the specified value is greater than zero, all coverage events outside the primer binding region are reported. If, however, the identified ratio of off-target events should exceed the allowed ratio, a warning is issued. If `allowed_other_binding_ratio` is set to `0`, only on-target primer binding events are reported. The setting of `allowed_other_binding_ratio` is ignored when designing primers, which always uses a value of 0.

allowed_region_definition: The definition of the target binding regions that is used for evaluating the coverage. In case that `allowed_region_definition` is `within`, primers have to lie within the allowed binding region. If `allowed_region_definition` is `any`, primers only have to overlap with the target binding region.

hexamer_coverage: If `hexamer_coverage` is set to "active", primers whose 3' hexamer (the last 6 bases) is fully complementary to the corresponding template region are automatically considered to cover the template. If `hexamer_coverage` is set to `inactive`, hexamer complementarity does not guarantee template coverage.

See Also

Other settings functions: `ConstraintSettings-class`, `CoverageConstraints-class`, `DesignSettings-class`, `PCR_Conditions-class`, `PCR`, `conOptions`, `constraintLimits`, `constraints`, `cvg_constraints`, `read_settings`

Examples

```
# Initialize a new 'ConstraintOptions' object:
constraint.options <- new("ConstraintOptions")
# Retrieve the constraint options from a 'DesignSettings' object:
data(Ippolito) # loads a 'DesignSettings' object into 'settings'
conOptions(settings)
# Prevent off-target binding:
conOptions(settings)$allowed_other_binding_ratio <- 0
```

`constraints` *Getter/Setter for Constraints.*

Description

Gets the active constraints of the provided `DesignSettings` object `x`.

Sets the active constraints of the provided `DesignSettings` object `x`.

Usage

```
constraints(x)

## S4 method for signature 'DesignSettings'
constraints(x)

## S4 method for signature 'AbstractConstraintSettings'
constraints(x)

constraints(x) <- value

## S4 replacement method for signature 'DesignSettings,ANY'
constraints(x) <- value

## S4 replacement method for signature 'AbstractConstraintSettings,list'
constraints(x) <- value
```

Arguments

`x`	A `DesignSettings` object.
`value`	A list with constraint settings. Each list entry should have a permissible name and consist of at most two values providing the minimal and/or maximal allowed values, which have to be denominated via `min` and `max`.

Details

For an overview of permissible constraints, please consider the `ConstraintSettings` documentation.

Value

Gets the list of constraints.

Sets the list of constraints.

See Also

Other settings functions: ConstraintOptions-class, ConstraintSettings-class, CoverageConstraints-class, DesignSettings-class, PCR_Conditions-class, PCR, conOptions, constraintLimits, cvg_constraints, read_settings

Examples

```
# Load some settings
data(Ippolito)
# View the active constraints
constraints(settings)
# Require a minimal GC clamp extent of 0
constraints(settings)$gc_clamp["min"] <- 0
# View available constraints
settings
```

ConstraintSettings-class

Class for Constraint Settings.

Description

The ConstraintSettings class encapsulates the constraints on the physicochemical properties of primers.

Details

You can check whether the constraint settings are fulfilled using check_constraints or filter accordingly using filter_primers.

Value

A ConstraintSettings object.

Slots

`status` Named boolean vector indicating which of the possible constraints are active (TRUE) and which are not (FALSE).

`settings` Named list containing the settings of the active constraints: The list may contain the following fields:

primer_coverage: The required number of covered template sequences per primer.

primer_specificity: The required required specificity of primers in terms of a ratio in the interval [0,1].

primer_length: The required lengths of primer sequences.

gc_clamp: The desired number of GCs at primer 3' termini.

gc_ratio: The desired ratio of GCs in primers in terms of numbers in the interval [0,1].

no_runs: The accepted length homopolymer runs in a primer.

no_repeats: The accepted length of dinucleotide repeats in a primer.

self_dimerization: The lowest acceptable free energy [kcal/mol] for the interaction of a primer with itself. The identification of self dimers requires the software *OligoArrayAux* (see notes).

melting_temp_range: The desired melting temperature (Celsius) of primers. The accurate computation of melting temperatures requires the software *MELTING* (see notes).

melting_temp_diff: The maximal allowed difference between the melting temperatures (Celsius) of primers contained in the same set. The accurate computation of melting temperatures requires the software *MELTING* (see notes).

cross_dimerization: The lowest acceptable free energy [kcal/mol] for the interaction of a primer with another primer. The identification of cross dimers requires the software *OligoArrayAux* (see notes).

secondary_structure: The lowest acceptable free energy [kcal/mol] for the formation of primer secondary structures. Secondary structures are determined using the software *ViennaRNA* (see notes).

primer_coverage

Computing the primer coverage involves identifying which templates are expected to be amplified (covered) by which primers. The `primer_coverage` constraint determines the minimal and maximal number of coverage events per primer that are required. The computation of primer coverage is governed by the coverage constraints postulated via `CoverageConstraints` and the constraint options defined via `ConstraintOptions`.

primer_specificity

Primer specificity is automatically determined during the primer coverage computations but the constraint is only checked when the `primer_specificity` field is available. The specificity of a primer is defined as its ratio of on-target vs total coverage events (including off-target coverage). Low-specificity primers should be excluded as they may not amplify the target region effectively.

primer_length

The length of a primer is defined by its number of bases. Typical primers have lengths between 18 and 22. Longer primers may guarantee higher specificities.

gc_clamp

The GC clamp refers to the presence of GCs at the 3' end of a primer. For the `gc_clamp` constraint, we consider the number of 3' terminal GCs. For example, the primer *actgaaatttcaccg* has a GC clamp of length 3. The presence of a GC clamp is supposed to aid the stability of the polymerase complex. At the same time, long GC clamps should be avoided.

no_runs

Homopolymer runs (e.g. the primer *aaaaa* has a run of 5 A's) may lead to secondary structure formation and unspecific binding and should therefore be avoided.

no_repeats

Dinucleotide repeats (e.g. the primer *tatata* has 3 TA repeats) should be avoided for the same reason a long homopolymer runs.

self_dimerization

Self dimerization refers to a primer that binds to itself rather than to one of the templates. Primers exhibiting self dimers should be avoided as they may prevent the primer from amplifying the templates. Therefore primers with small free energies of dimerization should be avoided.

melting_temp_range

The melting temperature is the temperature at which 50 are in duplex with templates and 50 Hence, primers exhibiting high melting temperatures have high affinities to the templates, while primers with small melting temperatures have small affinities. The melting temperatures of the primers determine the annealing temperature of the PCR, which is why the melting temperatures of the primers should not deviate too much (see `melting_temp_diff`).

melting_temp_diff

The differences between the melting temperatures of primers in a set of primers should not deviate too much as the annealing temperaturte of a PCR should be based on the smallest melting temperature of a primer in the set. If there are other primers in the set exhibiting considerably higher melting temperatures, these primers may bind inspecifically due to the low annealing temperature.

cross_dimerization

When two different primers bind to each each other rather than to the templates, this is called cross dimerization. Cross dimerization should be prevent at all costs because such primers cannot effectively amplify their target templates. Cross dimerizing primers can be excluding primers exhibiting small free energies of cross dimerization.

secondary_structure

When a primer exhibits secondary structure, this may prevent it from binding to the templates. To prevent this, primers with low free energies of secondary structure formation can be excluded.

Note

External programs are required for computing the following constraints:

MELTING (http://www.ebi.ac.uk/biomodels/tools/melting/): Thermodynamic computations (optional) for determining melting temperatures for the constraints `melting_temp_diff` and `melting_temp_range`

OligoArrayAux (http://unafold.rna.albany.edu/OligoArrayAux.php): Thermodynamic computations used for computing `self_dimerization` and `cross_dimerization`. Also required for computing `primer_coverage` when a constraint based on the free energy of annealing is active.

ViennaRNA (http://www.tbi.univie.ac.at/RNA/): Secondary structure predictions used for the constraint `secondary_structure`

See Also

Other settings functions: `ConstraintOptions-class`, `CoverageConstraints-class`, `DesignSettings-class`, `PCR_Conditions-class`, `PCR`, `conOptions`, `constraintLimits`, `constraints`, `cvg_constraints`, `read_settings`

Examples

```
# Initializing a new 'ConstraintSettings' object:
constraint.settings <- new("ConstraintSettings")
# Retrieving the constraint settings from a 'DesignSettings' object:
data(Ippolito) # loads a 'DesignSettings' object into 'settings'
constraints(settings)
# Modifying the constraint settings:
constraints(settings)$no_runs["max"] <- 10
constraints(settings) <- constraints(settings)[names(constraints(settings)) != "gc_clamp"]
```

`CoverageConstraints-class`

Class for Coverage Constraints.

Description

The `CoverageConstraints` class encapsulates the conditions under which the coverage is evaluated.

Value

A `CoverageConstraints` object.

Slots

`status` Named boolean vector indicating which of the possible options are active (`TRUE`) and which are not (`FALSE`).

`settings` Named list with constraint options. Each list entry should have an entry `min` and/or `max` in order to indicate the minimal and maximal allowed values, respectively. The following identifiers can be used as coverage constraints:

- `primer_efficiency`**:** The desired efficiencies of primer-template amplification events in order to be considered as *covered*. `primer_efficiency` provides a value in the interval [0,1], which is based on **DECIPHER**'s thermodynamic model, which considers the impact of 3' terminal mismatches.
- `annealing_DeltaG`**:** The desired free energies of annealing for putative coverage events between primers and templates. Typically, one would limit the maximally allowed free energy.
- `stop_codon`**:** Whether coverage events introducing stop codons into the amplicons should be allowed or discarded. Here, a value of `1` indicates coverage events that induce stop codons. As such, setting both minimum and maximium to zero will disregard coverage events inducing stop codons, while setting the minimum to zero and the maximum to 1 will allow coverage events that induce stop codons.
- `substitution`**:** Whether coverage events introducing substitutions into the amino acid sequence are considered or discarded. The same encoding as for `stop_codon` is used, that is, the value `1` indicates coverage events inducing substitutions. Hence, to prevent substitutions, the maximal value of `substitution` can be set to zero.
- `terminal_mismatch_pos`**:** The position relative to the primer 3' terminal end for which mismatch binding events should be allowed, where the last base in a primer is indicated by position `1`. For example, setting the minimal value of `terminal_mismatch_pos` to `7` means that only coverage events that do not have a terminal mismatch within the last 6 bases of the primer are allowed.

`coverage_model`**:** Use a logistic regression model combining the free energy of annealing and 3' terminal mismatch positions to determine the expected rate of false positive coverage calls. Using `coverage_model`, you can specify the allowed ratio of falsely predicted coverage events. Typically, one would limit the maximal allowed rate of false positives. Note that setting a small false positive rate will reduce the sensitivity of the coverage calls (i.e. true positives will be missed).

Note

External programs are required for computing the following constraints:

OligoArrayAux (http://unafold.rna.albany.edu/OligoArrayAux.php): Thermodynamic computations used for computing the coverage constraints `annealing_DeltaG`, `primer_efficiency`, and `coverage_model`

See Also

Other settings functions: `ConstraintOptions-class`, `ConstraintSettings-class`, `DesignSettings-class`, `PCR_Conditions-class`, `PCR`, `conOptions`, `constraintLimits`, `constraints`, `cvg_constraints`, `read_settings`

Examples

```
# Initialize a new 'CoverageConstraints' object:
cvg.constraints <- new("CoverageConstraints")
# Retrieving the coverage constraints from a 'DesignSettings' object:
data(Ippolito) # loads a 'DesignSettings' object into 'settings'
cvg_constraints(settings)
# Modifying the coverage constraints
cvg_constraints(settings)$primer_efficiency["min"] <- 0.001
```

---

create_coverage_xls *Creation of a Coverage XLS Spreadsheet.*

Description

Creation of an XLS spreadsheet providing an overview of the covered template sequences for each primer. Each cell in the spreadsheet indicates a coverage event between a primer and template using color codes. Identified coverage events are indicated by green, while primer-template pairs without coverage are indicated by red. In case that a primer binding condition (see `CoverageConstraints`) was active when computing the coverage, the numeric value of the coverage condition is annotated for each cell.

Usage

```
create_coverage_xls(primer.df, template.df, filename, settings)
```

Arguments

`primer.df`	A `Primers` object containg primers with evaluated coverage.
`template.df`	A `Templates` object containing the templates corresponding to `primer.df`.
`filename`	A character vector providing the filename for storing the XLS file.
`settings`	A `DesignSettings` object providing the coverage conditions that are to be shown in the spreadsheet.

Value

Creates a spreadsheet visualizing the primer coverage and stores it under `filename`.

Examples

```
data(Ippolito)
filename <- tempfile("cvg_overview", fileext = ".xls")
# Store coverage of the first 2 primers as an XLS file
create_coverage_xls(primer.df[1:2,], template.df, filename, settings)
```

---

create_report *Creation of a Primer PDF Report.*

Description

Creates a PDF report for analyzed primer sets.

Usage

```
create_report(primers, templates, out.file, settings, sample.name = NULL,
  used.settings = NULL, ...)
```

Arguments

`primers`	To create a report for a single primer set, please provide an evaluated `Primers` object. For creating a report comparing multiple primer sets, please provide a list of `Primers` objects.
`templates`	If `primers` is a `Primers` object, `templates` should be a `Templates` object. If `primers` is a list of `Primers` objects, `templates` should be a list of `Templates` objects of the same length as `primers`.
`out.file`	A character vector giving the path to the file where the report should be stored.
`settings`	The `DesignSettings` object that was used for analyzing the input primers.
`sample.name`	An identifier for your analysis. By default (`NULL`), the sample identifier is selected from the `Run` column of the input templates.
`used.settings`	A named list (with fields `fw` and `rev`) containing the relaxed settings for designing forward/reverse primers. By default (`NULL`), the relaxed settings are not shown in the report.
`...`	`required.cvg` (optional, default: 1), the desired coverage ratio if `primers` is a single primer set.

Value

Creates a PDF file summarizing the results from analyzing one or multiple sets of primers.

Note

Creating the report requires the external programs Pandoc (http://pandoc.org) and LaTeX (http://latex-project.org).

See Also

Other primer functions: Primers-class, check_constraints, check_restriction_sites, design_primers, filter_primers, get_initial_primers, primer_significance, score_degen, write_primers

Examples

```
setting.xml <- system.file("extdata", "settings",
                "C_Taq_PCR_high_stringency.xml", package = "openPrimeR")
settings <- read_settings(setting.xml)
# Creation of a report for a single primer set
data(Ippolito)
out.file.single <- tempfile("evaluation_report", fileext = ".pdf")
create_report(primer.df, template.df, out.file.single, settings)
# Creation of a report for multiple primer sets
data(Comparison)
out.file.comp <- tempfile("comparison_report", fileext = ".pdf")
create_report(primer.data[1:2], template.data[1:2], out.file.comp, settings)
```

---

cvg_constraints *Getter/Setter for Coverage Constraints.*

Description

Gets the coverage constraints of the provided DesignSettings object x.

Sets the coverage constraints of the provided DesignSettings object x.

Usage

```
cvg_constraints(x)

## S4 method for signature 'DesignSettings'
cvg_constraints(x)

cvg_constraints(x) <- value

## S4 replacement method for signature 'DesignSettings'
cvg_constraints(x) <- value
```

Arguments

x	A DesignSettings object.
value	A list with coverage constraints. Each list entry must have a permissible name and contain a numeric vector with at most two components describing the minimal and/or maximal required values that are to be indicated via min and max. The permissible contraint identifiers are documented in the CoverageConstraints class.

Value

Gets the list of coverage constraints.

Sets the list of coverage constraints.

See Also

Other settings functions: ConstraintOptions-class, ConstraintSettings-class, CoverageConstraints-class, DesignSettings-class, PCR_Conditions-class, PCR, conOptions, constraintLimits, constraints, read_settings

Examples

```
# Load some settings
data(Ippolito)
# View all active coverage constraints
cvg_constraints(settings)
# Increase the maximal false positive rate to increase the sensitiviity of coverage predictions
cvg_constraints(settings)$coverage_model <- c("max" = 0.1)
# View available coverage constraints:
settings
```

DesignSettings-class *Class for Primer Design Settings.*

Description

The DesignSettings class encapsulates all settings for designing and evaluating primer sets. Upon loading an XML file, the DesignSettings class checks whether the defined constraints can be applied by identifying whether the requirements for external programs are fulfilled. If the requirements are not fulfilled, the affected constraints are removed from the loaded DesignSettings object and a warning is issued. The loaded constraints are automatically ordered according to the option *openPrimeR.constraint_order* such that the runtime of the design_primers and filter_primers functions is optimized.

Details

Note that the fields Input_Constraints, Input_Constraint_Boundaries, and Coverage_Constraints should contain entries with at most two components using the fields min and/or max.

The Input_Constraint_Boundaries should always be at least as general as the specified Input_Constraints.

Value

A DesignSettings object.

Slots

Input_Constraints A ConstraintSettings object specifying the desired target value ranges for primer properties.

Input_Constraint_Boundaries A ConstraintSettings object specifying the limits for relaxing the constraints during the primer design procedure. This slot may contain the same fields as the Input_Constraints slot, but the specified desired ranges should be at least as general as those specified in the Input_Constraints slot.

Coverage_Constraints A CoverageConstraints object specifying the constraints for computing the primer coverage.

PCR_conditions A PCR_Conditions object specifying the PCR-related settings.

constraint_settings A ConstraintSettings object providing options for the computation of individual physicochemical properties.

See Also

read_settings for reading settings from XML files, write_settings for storing settings as XML files, constraints for accessing constraints, constraintLimits for accessing constraint boundaries, cvg_constraints for accessing coverage constraints, conOptions for accessing constraint options, PCR for accessing the PCR conditions.

Other settings functions: ConstraintOptions-class, ConstraintSettings-class, CoverageConstraints-class, PCR_Conditions-class, PCR, conOptions, constraintLimits, constraints, cvg_constraints, read_settings

Examples

```
# Load a settings object
filename <- system.file("extdata", "settings",
                "C_Taq_PCR_high_stringency.xml", package = "openPrimeR")
settings <- read_settings(filename)
# Modify the constraints
constraints(settings)$gc_clamp["min"] <- 0
# Modify the constraint limits for designing primers
constraintLimits(settings)$gc_clamp["max"] <- 6
# Modify the coverage constraints
cvg_constraints(settings)$primer_efficiency["min"] <- 0.001
# Modify the PCR conditions
PCR(settings)$Na_concentration <- 0.0001
# Modify the constraint options
conOptions(settings)$allowed_mismatches <- 0
```

---

design_primers *Design of Multiplex PCR Primers.*

Description

Designs a primer set maximizing the number of covered templates using the smallest possible number of primers. The algorithm tries to ensure that the designed set of primers achieves a coverage ratio not lower than required.cvg. To this end, the constraints for designing primers may be relaxed.

Usage

```
design_primers(template.df, mode.directionality = c("both", "fw", "rev"),
  settings, init.algo = c("naive", "tree"), opti.algo = c("Greedy", "ILP"),
  required.cvg = 1, timeout = Inf, max.degen = 16, conservation = 1,
  sample.name = NULL, cur.results.loc = NULL, primer.df = NULL,
  updateProgress = NULL)
```

Arguments

template.df	A Templates object containing the template sequences and target regions for designing primers.
mode.directionality	The template strand for which primers shall be designed. Primers can be designed either for forward strands ("fw"), for reverse strands ("rev"), or for both strands ("both"). The default setting is "both".

`settings` A `DesignSettings` object specifying the constraint settings for filtering and optimization.

`init.algo` The algorithm to be used for initializing primers. If `init.algo` is "naive", then primers are constructed from substrings of the input template sequences. If `init.algo` is "tree", phylogenetic trees are used to form degenerate primers whose degeneracy is bounded by `max.degen`. This option requires an installation of MAFFT (see notes). The default `init.algo` is "naive".

`opti.algo` The algorithm to be used for solving the primer set covering problem. If `opti.algo` is "Greedy" a greedy algorithm is used to solve the set cover problem. If `opti.algo` is "ILP" an integer linear programming formulation is used. The default `opti.algo` is "Greedy".

`required.cvg` The desired ratio of of covered template sequences. If the target coverage ratio cannot be reached, the constraint settings are relaxed according to the the constraint limits in order to reach the target coverage. The default `required.cvg` is set to 1, indicating that 100% of the templates are to be covered.

`timeout` Timeout in seconds. Only applicable when `opti.algo` is "ILP". The default is `Inf`, which does not limit the runtime.

`max.degen` The maximal degeneracy of primer candidates. This setting is particularly relevant when `init.algo` is set to "tree". The default setting is 16, which means that at most 4 maximally degenerate positions are allowed per primer.

`conservation` Restrict the percentile of considered regions according to their conservation. Only applicable for the tree-based primer initialization. At the its default of 1, all available binding regions are considered.

`sample.name` An identifier for the primer design task. The default setting is `NULL`, which means that the run identifier provided in `template.df` is used.

`cur.results.loc`
Directory for storing the results of the primer design procedure. The default setting is `NULL` such that no output is stored.

`primer.df` An optional `Primers` object. If an evaluated `primer.df` is provided, the primer design procedure only optimizes `primer.df` and does not perform the initialization and filtering steps. The default is `NULL` such that primers are initialized and filtered from scratch.

`updateProgress` Shiny progress callback function. The default is `NULL` such that no progress is logged.

Value

A list with the following fields:

`opti`: A `Primers` object providing the designed primer set.

`used_constraints`: A list with `DesignSettings` objects for each primer direction providing the (possibly relaxed) constraints used for designing the optimal primers.

`all_results`: A list containing objects of class `Primers`. Each list entry corresponds to an optimal primer set for a given melting temperature.

`all_used_constraints`: A list containing `DesignSettings` object for each optimized set in `all_results`.

`filtered`: A list with data providing information on the results of the filtering procedure.

1. Initialization

The primer design algorithm consists of three steps: primer initialization, filtering, and optimization. The method for initializing a set of candidate primers is determined via `init.algo`. If `init.algo` is set to *naive*, primers are created by extracting substrings from all input template sequences. If `init.algo` is set to *tree*, degenerate primers are created by merging similar subsequences by forming their consensus sequence up to a degeneracy of at most `max.degen`. The tree-based initialization is recommended for related sequences.

2. Filtering

The candidate primer set is filtered according to the constraints specified in the `settings` object. In some cases, it is necessary to relax the constraints in order to reach the desired `required.cvg`. In these cases, primers that fail the input constraints may be selected. If you would like to skip the initialization and filtering stages, you can provide an evaluated `Primers` object via `primer.df`.

3. Optimization

Optimizing a primer set entails finding the smallest subset of primers maximizing the coverage, which is done by solving the set cover problem. If melting temperature differences are a constraint, the optimization procedure automatically samples ranges of melting temperatures to find optimal sets for all possible temperatures. You can select the used optimization algorithm via `optia.algo`, where you can set "Greedy" for a greedy algorithm or "ILP for an integer linear program formulation (ILP). While the worst-case runtime of the greedy algorithm is shorter than the worst-case runtime of the ILP, the greedy solution may yield larger primer sets than the ILP solution.

Note

Some constraints specified in the `settings` object can only be computed if additional software is installed, please see the documentation of `DesignSettings` for an overview of all possible settings and the `ConstraintSettings` documentation for an overview of all possible constraints. Usage of `init.algo = "tree"` requires an installation of the multiple alignment program MAFFT (http://mafft.cbrc.jp/alignment/software/).

See Also

Other primer functions: `Primers-class`, `check_constraints`, `check_restriction_sites`, `create_report`, `filter_primers`, `get_initial_primers`, `primer_significance`, `score_degen`, `write_primers`

Examples

```
# Define PCR settings and primer criteria
data(Ippolito)
constraints(settings)$primer_length <- c("min" = 18, "max" = 18)
# Design only forward primers using a greedy algorithm
optimal.primers.greedy <- design_primers(template.df[1:2,], "both", settings, init.algo = "naive")
# Usage of the tree-based initialization strategy (requires MAFFT)
## Not run:
out.dir <- tempdir()
optimal.primers.tree <- design_primers(template.df[1:2,], "both", settings,
                          init.algo = "tree", opti.algo = "ILP",
                          max.degen = 16,
                          cur.results.loc = out.dir)

## End(Not run)
```

filter_primers *Filter a Set of Primers.*

Description

Filters a primer set according to the constraints specified via `settings` and `active.constraints` such that all primers that do not fulfill the constraints are removed from `primer.df`.

Usage

```
filter_primers(primer.df, template.df, settings,
  active.constraints = names(constraints(settings)))
```

Arguments

`primer.df`	A `Primers` object containing the primers to be filtered.
`template.df`	A `Templates` object with the template sequences that are to be covered by `primer.df`.
`settings`	A `DesignSettings` object specifying the parameters for filtering the primers.
`active.constraints`	The constraints that are to be used for filtering primers. By default, `active.constraints` is set to `NULL` such that all active constraints are used.

Value

A `Primers` object containing only those primers fulfilling all specified constraints.

Note

Please note that some constraints can only be computed if additional software is installed, please see `DesignSettings` for an overview.

See Also

Other primer functions: `Primers-class`, `check_constraints`, `check_restriction_sites`, `create_report`, `design_primers`, `get_initial_primers`, `primer_significance`, `score_degen`, `write_primers`

Examples

```
data(Ippolito)
filename <- system.file("extdata", "settings",
             "C_Taq_PCR_high_stringency.xml", package = "openPrimeR")
settings <- read_settings(filename)
# Only retain the primers fulfilling the GC clamp constraint:
filtered.df <- filter_primers(primer.df, template.df, settings,
                 active.constraints = c("gc_ratio"))
```

get_comparison_table *Overview of Primer Set Properties.*

Description

Creates an overview of the properties of multiple primer sets by providing the inter-quartile range of primer properties in bracket notation.

Usage

```
get_comparison_table(template.data, primer.data, sample.name = NULL)
```

Arguments

template.data	List with Template objects corresponding to primer.data.
primer.data	List with evaluated Primers objects whose properties are to be summarized.
sample.name	Either a single identifier or identifiers for every Templates object in template.data. By default, sample.name is NULL such that the Run annotations in the Templates objects provided by template.data are used.

Value

A data frame summarizing the properties of each primer set.

Examples

```
data(Comparison)
tab <- get_comparison_table(template.data[1:3], primer.data[1:3], "IGH")
```

get_cvg_ratio *Determination of the Ratio of Covered Templates.*

Description

Determines the ratio of template sequences that are covered by the evaluated input primers. The ratio is in the interval [0,1] where 0 indicates 0% coverage (no templates covered) and 1 indicates 100% coverage (all templates covered).

Usage

```
get_cvg_ratio(primers, template.df, allowed.mismatches = NULL,
  cvg.definition = c("constrained", "basic"), mode.directionality = NULL,
  as.char = FALSE)
```

Arguments

primers	A Primers object containing the primers for which the coverage should be evaluated.
template.df	A Templates object containing the template sequences corresponding to primers.
allowed.mismatches	The number of allowed mismatches for determining the coverage of the templates. By default, allowed.mismatches is set to NULL such that all annotated coverage events are considered.
cvg.definition	Whether to output the coverage obtained from string matching ("basic") or the expected coverage ("constrained"), which is constructed by applying the coverage constraints. By default, cvg.definition is set to "constrained".
mode.directionality	If mode.directionality is provided, the coverage of templates is computed for a specific direction of primers. Either "fw" (forward coverage only), "rev" (reverse coverage only), or "both" for both directions. By default, mode.directionality is NULL such that the directionality of the primers is determined automatically.
as.char	Whether the coverage ratio should be outputted as a percentage-formatted character vector. By default, as.char is set to FALSE such that a numeric is returned.

Details

The manner in which the coverage ratio is evaluated depends on the directionality of the input primers. If either only forward or reverse primers are inputted, the individual coverage of each primer is used to determine the overall coverage. If, however, forward and reverse primers are inputted at the same time, the coverage is defined by the intersection of binding events from both, forward and reverse primers.

Value

By default, a numeric providing the expected primer coverage ratio. If as.char is TRUE, the output is provided as a percentage-formatted character vector. The attributes no_covered, no_templates, and covered_templates provide the number of covered templates, the total number of templates, and the IDs of covered templates, respectively.

Examples

```
data(Ippolito)
cvg.ratio <- get_cvg_ratio(primer.df, template.df)
# determine the identitity coverage ratio
cvg.ratio.0 <- get_cvg_ratio(primer.df, template.df, allowed.mismatches = 0)
```

---

get_cvg_stats *Coverage Ratios per Group of Templates.*

Description

Retrieve statistics on covered templates, either for a single primer set or for multiple primer sets.

Usage

```
get_cvg_stats(primers, templates, for.viewing = FALSE,
  total.percentages = FALSE, allowed.mismatches = Inf,
  cvg.definition = c("constrained", "basic"))
```

Arguments

primers	To retrieve coverage statistics for a single primer set, please provide an object of class Primers containing primers with evaluated coverage. To retrieve coverage statistics for multiple primer sets, pelase provide a list with evaluated Primers objects.
templates	If primers is an object of class Primers, please provide an object of class Templates containing the template sequences targeted by primers. If primers is a list, templates should be a list of Template objects.
for.viewing	Whether the table should be formatted to be human-readable. By default, for.viewing is FALSE.
total.percentages	Whether group coverage percentages should be computed in relation to the total number of template sequences or in relation to the number of templates belonging to a specific group. By default, total.percentages is FALSE suc that the percentages are group-specific.
allowed.mismatches	The maximal allowed number of mismatches. By default, allowed.mismatches is set to Inf such that the number of mismatches is not restricted additionally.
cvg.definition	If cvg.definition is set to "constrained", the statistics for the expected coverage (after applying the coverage constraints) are retrieved. If cvg.definition is set to "basic", the coverage is determined solely by string matching (i.e. without applying the coverage constraints). By default, cvg.definition is set to "constrained".

Value

Data frame whose entries provide the coverage of templates belonging to a specific group.

Examples

```
# Coverage statistics for a single primer set
data(Ippolito)
cvg.stats <- get_cvg_stats(primer.df, template.df)
# Coverage statistics for multiple primer sets
data(Comparison)
cvg.stats.comp <- get_cvg_stats(primer.data[1:2], template.data[1:2])
```

---

get_cvg_stats_primer *Statistics on the Number of Coverage Events per Primer.*

Description

Creates a table summarizing the coverage events of each primer according to the number of mismatches between primers and templates.

Usage

```
get_cvg_stats_primer(primer.df, template.df, cvg.definition = c("constrained",
  "basic"))
```

Arguments

primer.df	A Primers object with evaluated coverage providing the set of primers for which the coverage statistics shall be computed.
template.df	A Templates object providing the template sequences for which the primer coverage has been computed.
cvg.definition	If cvg.definition is set to "constrained", the statistics for the expected coverage (after applying the coverage constraints) are retrieved. If cvg.definition is set to "basic", the coverage is determined solely by string matching (i.e. without applying the coverage constraints). By default, cvg.definition is set to "constrained".

Details

Entries in numeric table columns indicate the percentage of coverage events occurring with a certain number of mismatches. For example column *3* provides all coverage events with exactly three mismatches between primers and templates. The column *Group_Coverage* provides a listing of the percentage of covered templates per group.

Value

A data frame listing the number of binding events broken down according to the number of expected mismatches between primers and templates.

Examples

```
data(Ippolito)
primer.cvg.stats <- get_cvg_stats_primer(primer.df, template.df)
```

---

get_initial_primers *Creation of Candidate Primers.*

Description

Creates a set of primer candidates based on the input template sequences. This set of primers can be used to create custom primer design algorithms.

Usage

```
get_initial_primers(sample, template.df, primer.lengths,
  mode.directionality = c("fw", "rev"),
  allowed.region.definition = c("within", "any"), init.algo = c("naive",
  "tree"), max.degen = 16, conservation = 1, updateProgress = NULL)
```

Arguments

sample	Character vector providing an identifier for the templates.
template.df	An object of class Templates providing the template sequences for which an initial set of primers is constructed.
primer.lengths	Numeric interval providing the minimal and maximal allowed lengths of generated primers.
mode.directionality	Character vector giving the direction of primers to be created. Either "fw" to create forward primers or "rev" to create reverse primers. The default is "fw".
allowed.region.definition	A character vector providing the definition of region where primers are to be constructed. If allowed.region.definition is "within", constructed primers lie within the allowed binding region. If allowed.region.definition is "any", primers overlap with the allowed binding region. The default is "within".
init.algo	Algorithm for initializing primers. If you set init.algo to "tree", then initial primers will be generated by forming degenerate consensus sequences using a tree-based approach. This option requires MAFFT for multiple alignments (see notes). If init.algo is set to "naive", initial primers are generated by extracting substrings from the template target regions.
max.degen	A numeric providing the maximal allowed degeneration of created primers.
conservation	The percentile of top-conserved template regions to consider. The value of conservation should be in the range[0,1]. If the conservation is set to 1 (the default), all regions are considered.
updateProgress	A Shiny progress object; by default this is set to NULL such that no callback is used.

Value

A data frame with candidate primers for optimization.

Note

If you want to set init.algo to "tree", please install MAFFT (http://mafft.cbrc.jp/alignment/software/) on your computer.

See Also

Other primer functions: Primers-class, check_constraints, check_restriction_sites, create_report, design_primers, filter_primers, primer_significance, score_degen, write_primers

Examples

```
data(Ippolito)
# Naive primer initialization
init.primers <- get_initial_primers("InitialPrimers", template.df,
                        c(18,18), "fw", init.algo = "naive")
# Tree-based primer initialization (requires MAFFT)
## Not run:
init.primers <- get_initial_primers("InitialPrimers", template.df,
                        c(18,18), "fw", init.algo = "tree")

## End(Not run)
```

Ippolito *Evaluated Primer Data from Ippolito et al.*

Description

Primer and template data for IGHV from Ippolito et al.

Usage

```
data(Ippolito)
```

Format

`primer.df` provides a `Primers` object containing the evaluated set of primers from Tiller et al. `template.df` provides a `Templates` object containing functional, human IGHV templates for, and `settings` provides a `DesignSettings` object providing the used analysis settings.

References

Ippolito GC, Hoi KH, Reddy ST, Carroll SM, Ge X, Rogosch T, Zemlin M, Shultz LD, Ellington AD, VanDenBerg CL, Georgiou G. 2012. Antibody Repertoires in Humanized NOD-scid-IL2R gamma null Mice and Human B Cells Reveals Human-Like Diversification and Tolerance Checkpoints in the Mouse. PLoS One 7:e35497.

Examples

```
data(Ippolito)
# Explore the data:
primer.df
template.df
constraints(settings)
```

parallel_setup *Setup the Parallel Backend.*

Description

Registers the specified number of cores with the parallel backend.

Usage

```
parallel_setup(cores = NULL)
```

Arguments

`cores`	A numeric providing the number of cores to use. The default is `NULL` such that half the number of available cores are used.

Value

Returns `NULL`

Examples

```
## Not run:
# Use two cores for parallel processing:
parallel_setup(2)

## End(Not run)
```

PCR *Getter/Setter for the PCR Conditions.*

Description

Gets the PCR conditions that are defined in the provided `DesignSettings` object `x`.

Sets the PCR conditions that are defined in the provided `DesignSettings` object `x`.

Usage

```
PCR(x)

## S4 method for signature 'DesignSettings'
PCR(x)

PCR(x) <- value

## S4 replacement method for signature 'DesignSettings'
PCR(x) <- value
```

Arguments

`x`	A `DesignSettings` object.
`value`	A named list providing PCR conditions The permissible fields of the list and their types are documented in the `PCR_Conditions` class.

Value

Gets the list of PCR conditions.

Sets the list of PCR conditions.

See Also

Other settings functions: `ConstraintOptions-class`, `ConstraintSettings-class`, `CoverageConstraints-class`, `DesignSettings-class`, `PCR_Conditions-class`, `conOptions`, `constraintLimits`, `constraints`, `cvg_constraints`, `read_settings`

Examples

```
# Load some settings
data(Ippolito)
# View the active PCR conditions
PCR(settings)
# Evaluate primers with a fixed annealing temperature
PCR(settings)$annealing_temperature <- 50 # celsius
# View available PCR conditions
settings
```

PCR_Conditions-class *Class for PCR Conditions.*

Description

The PCR_Conditions class encapsulates the PCR conditions for the computation of primer properties.

Value

A PCR_Conditions object.

Slots

`status` Named boolean vector indicating which of the possible options are active (TRUE) and which are not (FALSE).

`settings` Named list with constraint options. The following fields are possible:

- `use_taq_polymerase`: A logical identifying whether you are performing PCR with a Taq polymerase (TRUE) or not (FALSE).
- `annealing_temp`: The annealing temperature in Celsius that is to be used for evaluating the constraints defined in the ConstraintSettings object. If the annealing temperature field is not provided, a suitable annealing temperature is automatically computed using a rule of thumb (i.e. subtracting 5 from the melting temperature).
- `Na_concentration`: The molar concentration of monovalent sodium ions.
- `Mg_concentration`: The molar concentration of divalent magnesium ions.
- `K_concentration`: The molar concentration of monovalent potassium ions.
- `Tris_concentration`: The molar concentration of the Tris(hydroxymethyl)-aminomethan buffer. Note that this value is the buffer concentration. To determine corresponding Tris ion concentration, the value of the buffer concentration is halved.
- `primer_concentration`: The molar concentration of the PCR primers.
- `template_concentration`: The molar concentration of the PCR templates.

See Also

Other settings functions: ConstraintOptions-class, ConstraintSettings-class, CoverageConstraints-class, DesignSettings-class, PCR, conOptions, constraintLimits, constraints, cvg_constraints, read_settings

Examples

```
# Initialize a new 'PCR_Conditions' object:
PCR.conditions <- new("PCR_Conditions")
# Retrieving the PCR conditions from a 'DesignSettings' object:
data(Ippolito) # loads a 'DesignSettings' object into 'settings'
PCR(settings)
# Modifying the PCR conditions:
PCR(settings)$use_taq_polymerase <- FALSE
```

---

plot_conservation *Plot of Template Sequence Conservation.*

Description

Plots the template sequence conservation (range [0,1]) according to the Shannon entropy of the sequences.

Usage

```
plot_conservation(entropy.df, alignments, template.df, gap.char = "-")
```

Arguments

entropy.df	A data frame with entropies. Each row gives the entropies of a group of related template sequences for all columns of the alignment.
alignments	A list with DNABin alignment objects corresponding to the groups (rows) in the alignment.
template.df	The Templates object for which the conservation has been determined.
gap.char	The gap char in the alignments. By default, gap.char is set to "-".

Value

A plot showing the degree of sequence conservation in the templates.

Note

Computing the conservation scores for the plot requires the MAFFT software for multiple alignments (http://mafft.cbrc.jp/alignment/software/).

Examples

```
## Not run:
data(Ippolito)
# Select binding regions for every group of templates and plot:
template.df <- select_regions_by_conservation(template.df, win.len = 30)
plot_conservation(attr(template.df, "entropies"), attr(template.df, "alignments"), template.df)
# Select binding regions for all templates and plot:
data(Ippolito)
template.df <- select_regions_by_conservation(template.df, by.group = FALSE)
plot_conservation(attr(template.df, "entropies"), attr(template.df, "alignments"), template.df)

## End(Not run)
```

plot_constraint *Plot of Constraint Values.*

Description

Shows the distribution of the primer properties. The current constraint settings are indicated with dashed lines in the plot.

Usage

```
plot_constraint(primers, settings,
  active.constraints = names(constraints(settings)), ...)
```

Arguments

primers	Either an evaluated object of class Primers or a list of Primers objects.
settings	A DesignSettings object containing the settings for the constraints to be plotted.
active.constraints	Identifiers of constraints to be plotted. If active.constraints is not provided, the plotting method automatically plots all constraints defined in settings that are annotated in primers.
...	highlight.set (a character vector identifying the set that is to be highlighted when primers is a list).

Value

A plot showing the distribution of primer properties.

See Also

Other constraint visualizations: plot_constraint_deviation, plot_constraint_fulfillment

Examples

```
# Plot histogram of constraints for a single primer set
data(Ippolito)
plot_constraint(primer.df, settings, active.constraints = c("gc_clamp", "gc_ratio"))
# Compare constraints across multiple primer sets
data(Comparison)
plot_constraint(primer.data[1:3], settings, active.constraints = c("gc_clamp", "gc_ratio"))
```

`plot_constraint_deviation` *Plot of Constraint Deviations.*

Description

Shows the deviation of primer properties from the target ranges.

Usage

```
plot_constraint_deviation(primer.data, settings,
  active.constraints = names(constraints(settings)), ...)
```

Arguments

`primer.data`	An evaluated object of class `Primers` or a list with `Primers` objects.
`settings`	A `DesignSettings` object containing the target ranges for the primer properties.
`active.constraints`	Constraint identifiers to be plotted. By default, all constraints found in `settings` are plotted.
`...`	`deviation.per.primer` (a boolean indicating whether the deviations should be plotted per primer rather than per constraint if `primer.data` is a list)

Details

Deviations are computed in the following way. Let the minimum and maximum allowed constraint values be given by the interval $[s, e]$ and the observed value be p. Then, if $p < s$, we output $-p/|s|$, if $p > e$ we output $p/|e|$, and otherwise, i.e. if $s <= p <= e$, we output 0.

Value

A plot showing the deviations of the primer properties from the targets.

See Also

Other constraint visualizations: `plot_constraint_fulfillment`, `plot_constraint`

Examples

```
# Deviations for a single primer set
data(Ippolito)
plot_constraint_deviation(primer.df, settings)
# Deviations for multiple primer sets
data(Comparison)
plot_constraint_deviation(primer.data, settings)
```

plot_constraint_fulfillment *Constraint Fulfillment Plot.*

Description

Visualizes which which primers pass the constraint settings and which primers break the constraints.

Usage

```
plot_constraint_fulfillment(primers, settings,
  active.constraints = names(constraints(settings)), plot.p.vals = FALSE,
  ...)
```

Arguments

primers	Either an object of class Primers or a list of such objects.
settings	A DesignSettings object containing the constraints to be evaluated.
active.constraints	The identifiers of constraints to be plotted for fulfillment. By default active.constraints is set according to all active constarints defined in settings.
plot.p.vals	An optional logical argument indicating whether p-values computed via primer_significance should be annotated in the plot. The default is FALSE.
...	The optional arguments ncol (a numeric indicating the number of facet columns if primers is a list), highlight.set (the identifier of the primer set to be highlighted if primers is a list)

Value

A plot indicating the constraints that fulfilled by the input primers.

See Also

Other constraint visualizations: plot_constraint_deviation, plot_constraint

Examples

```
# Plot fulfillment for a single primer set:
data(Ippolito)
plot_constraint_fulfillment(primer.df, settings)
# Plot fulfillment for multiple primer sets:
data(Comparison)
plot_constraint_fulfillment(primer.data[1:5], settings)
```

plot_cvg_constraints *Plot of Coverage Constraints.*

Description

Plots the distribution of the coverage constraint values.

Usage

```
plot_cvg_constraints(primers, settings,
  active.constraints = names(cvg_constraints(settings)), ...)
```

Arguments

primers	A Primers object or a list with objects of class Primers.
settings	A DesignSettings object.
active.constraints	Names of coverage constraints to be plotted. By default, all active coverage constraints in settings are plotted.
...	highlight.set (a character vector identifying the set that is to be highlighted when primers is a list).

Value

A plot showing the distribution of the coverage constraint values.

Examples

```
# Plot coverage constraints of a single primer set
data(Ippolito)
plot_cvg_constraints(primer.df, settings)
# Plot coverage constraints for mulitple primer sets
data(Comparison)
plot_cvg_constraints(primer.data[1:2], settings)
```

plot_cvg_vs_set_size *Plot of Primer Coverage Ratio vs Set Size.*

Description

Plots the coverage ratios of the input primer sets against the size of the sets.

Usage

```
plot_cvg_vs_set_size(primer.data, template.data, show.labels = TRUE,
  highlight.set = NULL)
```

Arguments

primer.data	List with objects of class Primers containing the primer sets that are to be compared.
template.data	List with objects of class Templates containing the templates corresponding to primer.data.
show.labels	Whether the identifiers of the primer sets should be annotated in the plot. The default is TRUE.
highlight.set	A character vector providing the identifiers of primer sets to highlight. By default, highlight.set is NULL such that no highlighting takes place.

Value

A plot of coverage vs set size.

See Also

Other comparison visualizations: plot_penalty_vs_set_size

Examples

```
data(Comparison)
plot_cvg_vs_set_size(primer.data, template.data)
```

---

plot_penalty_vs_set_size

Plot of Primer Penalties vs Set Size.

Description

Plots the penalties of the input primer sets against the number of primers contained in each set. The penalties are computed using score_primers where more information is provided on how to set alpha.

Usage

```
plot_penalty_vs_set_size(primer.data, settings,
  active.constraints = names(constraints(settings)), alpha = 0)
```

Arguments

primer.data	List with objects of class Primers.
settings	An object of class DesignSettings.
active.constraints	A character vector with constraint identifiers to be considered for generating the plot.
alpha	A numeric in the range [0,1] defining the trade-off between the maximal deviation of a constraint (large codealpha) and all constraint deviations (large alpha). By default, alpha is set to 0 such that the absolute deviation across all constraints is considered.

Value

A plot showing the association between primer penalties and the size of the primer sets.

See Also

Other comparison visualizations: plot_cvg_vs_set_size

Examples

```
data(Comparison)
plot_penalty_vs_set_size(primer.data, settings)
```

---

plot_primer *Primer View Plot.*

Description

Visualizes the binding positions of every primer relative to the target binding region in the corresponding template sequences.

Usage

```
plot_primer(primer.df, template.df, identifier = NULL, relation = c("fw",
  "rev"), region.names = c("Binding region", "Amplification region"))
```

Arguments

primer.df	An object of class Primers containing primers with evaluated primer coverage.
template.df	An object of class Templates with template sequences corresponding to primer.df.
identifier	Identifiers of primers that are to be considered. If identifier is set to NULL (the default), all primers are considered.
relation	Compute binding positions relative to forward ("fw") or reverse ("rev") binding regions. The default is "fw".
region.names	Character vector of length 2 providing the names of the binding and amplification region.

Value

A plot of primer binding sites in the templates.

See Also

Other coverage visualizations: plot_primer_binding_regions, plot_primer_cvg, plot_primer_subsets, plot_template_cvg

Examples

```
data(Ippolito)
plot_primer(primer.df[1,], template.df[1:30,])
```

plot_primer_binding_regions *Plot of Primer Binding Regions.*

Description

Visualizes the number of binding events of the input primers with respect to the allowed binding regions in the templates.

Usage

```
plot_primer_binding_regions(primers, templates, direction = c("both", "fw",
  "rev"), group = NULL, relation = c("fw", "rev"),
  region.names = c("Binding region", "Amplification region"), ...)
```

Arguments

primers	Either a single Primers object or a list with Primers objects.
templates	If primers is a primers object, please supply a Templates object. If primers is a list, please supply a corresponding list of Templates objects.
direction	The directionality of primers to be plotted. This can either be "both" to plot primers of both directions (the default), "fw" to plot only forward primers, or "rev" to plot only reverse primers.
group	Optional identifiers of template groups for which binding events should be determined. By default, group is set to NULL such that all templates are considered.
relation	An optional character vector specifying whether binding region data shall be plotted relative to the forward (fw) or reverse (rev) target binding regions.
region.names	An optional, two-component character vector specifying the identifiers for the primer binding region and the amplified region.
...	highlight.set (the identifiers of primer sets to be highlighted, if primers is a list)

Value

A plot for primer binding region comparison.

See Also

Other coverage visualizations: plot_primer_cvg, plot_primer_subsets, plot_primer, plot_template_cvg

Examples

```
# Primer binding regions of a single primer set
data(Ippolito)
plot_primer_binding_regions(primer.df, template.df)
# Primer binding regions of multiple primer sets
data(Comparison)
plot_primer_binding_regions(primer.data[1:3], template.data[1:3])
```

plot_primer_cvg *Plot of Primer Coverage.*

Description

Shows which groups of templates are covered by individual primers.

Usage

```
plot_primer_cvg(primers, templates, per.mismatch = FALSE, ...)
```

Arguments

primers	An object of class Primers or a list with with objects of class Primers.
templates	If primers is an object of class Primers, please supply a Templates object. If primers is a list, please supply a corresponding list with Templates objects.
per.mismatch	A logical identifiying whether the coverage should be plotted for individual settings of allowed mismatches. By default per.mismatch is set to FALSE such that the overall coverage is plotted.
...	groups (the identifiers of template groups to be excluded from the plot if primers is a single primer set)

Value

A plot showing the coverage of individual primers.

See Also

Other coverage visualizations: plot_primer_binding_regions, plot_primer_subsets, plot_primer, plot_template_cvg

Examples

```
# Plot expected coverage per primer
data(Ippolito)
plot_primer_cvg(primer.df, template.df)
# Plot coverage stratified by allowed mismatches:
plot_primer_cvg(primer.df, template.df, per.mismatch = TRUE)
# Plot coverage of multiple primer sets
data(Comparison)
plot_primer_cvg(primer.data[1:3], template.data[1:3])
```

plot_primer_subsets *Plot of Primer Subset Coverage.*

Description

Visualizes the coverage of optimized primer subsets.

Usage

```
plot_primer_subsets(primer.subsets, template.df, required.cvg = 1)
```

Arguments

primer.subsets	A list with optimal primer subsets, each of class Primers. The *k*-th list entry should correspond to an object of class Primers representing the primer subset of size *k* whose coverage ratio is the largest among all possible subsets of size *k*.
template.df	An object of class Templates containing the template sequences corresponding to the primers specified in primer.subsets.
required.cvg	The required coverage ratio. The default is 100%; this value is plotted as a horizontal line.

Details

The input for the primer.subsets argument can be computed using subset_primer_set. The line plot indicates the ratio of covered templates when considering all primers in a primer set of a given size. The bar plots indicate the coverage ratios of individual primers in a set. The target coverage ratio is indicated by a horizontal line. Bars exceeding the target ratio possibly indicate the existence of redundant coverage events.

Value

Plot of the coverages of the primer subsets in primer.subsets.

See Also

Other coverage visualizations: plot_primer_binding_regions, plot_primer_cvg, plot_primer, plot_template_cvg

Examples

```
data(Ippolito)
primer.subsets <- subset_primer_set(primer.df, template.df)
# Plot the coverage of optimal primer subsets
plot_primer_subsets(primer.subsets, template.df)
```

plot_template_cvg *Bar Plot of Template Coverage.*

Description

Creates a bar plot showing the coverage for every group of template sequences.

Usage

```
plot_template_cvg(primers, templates, per.mismatch = FALSE, ...)
```

Arguments

primers	Either a Primers object with evaluated primer coverage or a list containing Primers objects.
templates	If primers is a Primers object, templates should be a Templates object. If primers is a list, then templates should be a list of Templates objects.
per.mismatch	A logical specifying whether the visualization should be stratified according to the allowed number of mismatches. By default, per.mismatch is set to FALSE such that the overall coverage is plotted.
...	Optional arguments groups (a character vector of groups to be plotted when primers is a single primer set), highlight.set (the identifier of a primer set to be highlighted when primers is a list)

Value

A plot showing the number of covered template sequences.

See Also

Other coverage visualizations: plot_primer_binding_regions, plot_primer_cvg, plot_primer_subsets, plot_primer

Examples

```
# Visualize the template coverage of a single primer set
data(Ippolito)
plot_template_cvg(primer.df, template.df)
# Stratify by allowed mismatches:
plot_template_cvg(primer.df, template.df, per.mismatch = TRUE)
# Compare the coverage of multiple primer sets
data(Comparison)
plot_template_cvg(primer.data[1:3], template.data[1:3])
# Stratify by allowed mismatches:
plot_template_cvg(primer.data[1:3], template.data[1:3], per.mismatch = TRUE)
```

`Primers-class` *The Primers Class.*

Description

The `Primers` class encapsulates a data frame representing a set of primers. Objects of this class store all properties associated with a set of primers, for example the results from evaluating the properties of a primer set or from determining its coverage.

Usage

```
Primers(...)
```

Arguments

`...`	A data frame fulfilling the structural requirements for initializing a `Primers` object.

Value

A `Primers` object, an instance of a data frame.

Basic columns

In the following you can find a description of the most important columns that can be found in objects of class `Primers`. Note that angular brackets indicate the existence of multiple possibilities. The following columns are present when a set of primers is loaded from a FASTA file using `read_primers`:

`ID` The identifiers of the primers.

`Identifier` The internal identifiers of the primers.

`Forward` The sequences of forward primers.

`Reverse` The sequences of reverse primers.

`primer_length<fw|rev>` The lengths of forward and reverse primer sequences, respectively.

`Direction` Either 'fw' for forward primers, 'rev' for reverse primers, or 'both' for a primer pair.

`Degeneracy_<fw|rev>` The degeneracy (ambiguity) of forward and reverse primers, respectively.

`Run` An identifier describing the primer set.

Coverage-related columns

The following columns are only available after primer coverage has been computed, that is after `check_constraints` has been called with the active `primer_coverage` constraint. Computed coverage values relating solely to string matching are indicated by the prefix `Basic_`, while columns without this prefix relate to the coverage after applying the constraints formulated via `CoverageConstraints`. Information on off-target coverage events are indicated by the `Off_` prefix, while on-target coverage events do not carry this prefix.

`primer_coverage` The number of templates that are covered by the primers. Note that if a primer set contains primers of both directions, a template is only considered covered if it is covered by primers of both directions.

`Coverage_Ratio` The ratio of templates that are covered by the primers.

`Binding_Position_Start_<fw|rev>` The upstream position in the templates where forward and reverse primers respectively bind.

`Binding_Position_End_<fw|rev>` The downstream position in the templates where forward and reverse primers respectively bind.

`Relative_<Forward|Reverse>_Binding_Position_<Start|End>_<fw|rev>` The binding upstream (`Start`) or downstream (`End`) positions of the primers relative to the forward (`Forward`) or reverse (`Reverse`) binding regions, either for forward (`fw`) or reverse primers (`rev`).

`Binding_Region_Allowed` Whether a coverage event occurred in the target binding region or not. If the allowed off-target ratio was set to 0 only coverage events within the the target region are reported.

`Nbr_of_mismatches_<fw|rev>` The number of mismatches of forward and reverse primer coverage events, respectively.

`Mismatch_pos_<fw|rev>` The position of mismatches for forward and reverse coverage events, respectively. Mismatch positions are reported relative to the 3' end, that is, position 1 indicates a mismatch in the last base of a primer.

`primer_specificity` The specificity of a primer as determined by its ratio of off-target binding events.

Constraint-related columns

Each constraint that is considered when calling `check_constraints` gives rise to at least one column in the provided `Primers` object. Due to the large number of possible constraints, we will limit our description to the `gc_clamp` constraint. Once the GC clamp property has been computed, the `gc_clamp_fw` column contains the length of the GC clamp for forward primers and `gc_clamp_rev` the corresponding length for reverse primers. Whether the desired extent of the GC clamp was obtained by a primer is indicated by the `EVAL_gc_clamp` column. It contains `TRUE` when the GC clamp constraint was fulfilled and `FALSE` when it was broken. To identify whether all required constraints were fulfilled by a primer, the `constraints_passed` column can be used. It contains `TRUE` if all `active.constraints` used by `check_constraints` were fulfilled and `FALSE` otherwise.

See Also

`read_primers` for loading a primer set, `score_degen` for scoring the degeneracy of a primer, `primer_significance` for determining the significance of a primer set, `get_initial_primers` for computing an initial set of primers, `design_primers` for designing primer sets, `check_constraints` for determining the properties of a primer set, `filter_primers` for filtering a primer set, `check_restriction_sites` to search for restriction sites, `get_cvg_ratio` to determine the coverage ratio of a primer set, `create_report` to create a PDF report for a primer set.

Other primer functions: `check_constraints`, `check_restriction_sites`, `create_report`, `design_primers`, `filter_primers`, `get_initial_primers`, `primer_significance`, `score_degen`, `write_primers`

Examples

```
primer.location <- system.file("extdata", "IMGT_data", "primers", "IGHV",
                    "Ippolito2012.fasta", package = "openPrimeR")
primer.df <- read_primers(primer.location, "_fw", "_rev")
```

primer_significance *Significance of a Primer Set.*

Description

Uses Fisher's exact test to determine the significance of a primer set according to its ratio of fulfilled constraints on the primer properties.

Usage

```
primer_significance(primer.df, set.name = NULL, active.constraints = NULL)
```

Arguments

primer.df	An object of class Primers for which the significance of physicochemical properties shall be determined.
set.name	An identifier for the input primers. If NULL, the run identifier is used.
active.constraints	Identifiers of the constraints contained in primer.df to consider. By default (NULL) all constraints available in primer.df are considered for determining the significance.

Details

The significance is computed by comparing the total count of fulfilled and failed constraints with the corresponding counts of primer sets from the literature. Significant p-values indicate primer sets whose rate of constraint fulfillment is higher compared to the reference sets.

Value

The p-value of the primer set according to Fisher's exact test. The returned value has the following attributes:

test The results of the significance test

tab The confusion matrix for Fisher's exact test

constraints The names of the considered constraints

See Also

Other primer functions: Primers-class, check_constraints, check_restriction_sites, create_report, design_primers, filter_primers, get_initial_primers, score_degen, write_primers

Examples

```
data(Ippolito)
p.data <- primer_significance(primer.df, "Ippolito")
attr(p.data,"tab") # the confusion matrix
attr(p.data, "test") # results from Fisher's test
attr(p.data, "constraints") # considered constraints for the test
```

read_primers *Input of Primers.*

Description

Reads one or multiple input files with primer sequences. The input can either be in FASTA or in CSV format.

Usage

```
read_primers(primer.location, fw.id = "_fw", rev.id = "_rev",
  merge.ambig = c("none", "merge", "unmerge"), max.degen = 16,
  template.df = NULL, adapter.action = c("warn", "rm"),
  sample.name = NULL, updateProgress = NULL)
```

Arguments

primer.location	Path to a single or multiple primer FASTA or CSV files.
fw.id	For FASTA input, the identifier for forward primers in the FASTA headers.
rev.id	For FASTA input, the identifier for reverse primers in the FASTA headers.
merge.ambig	Indicates whether similar primers should be merged ("merge") using IUPAC ambiguity codes or whether primers should be disambiguated ("unmerge"). By default merge.ambig is set to "none", leaving primers as they are.
max.degen	A scalar numeric providing the maximum allowed degeneracy for merging primers if merge.ambig is set to "merge". Degeneracy is defined by the number of disambiguated sequences that are represented by a degenerate primer.
template.df	An object of class Templates. If template.df is provided the primers are checked for restriction sites upon input. By default template.df is NULL such that the primers are not checked for restriction sites.
adapter.action	The action to be performed when template.df is provided for identifying adapter sequences. Either "warn" to issue warning about adapter sequences or "rm" to remove identified adapter sequences. The default is "warn".
sample.name	An identifier for the input primers.
updateProgress	A Shiny progress callback function. This is NULL by default such that no progress is tracked.

Details

The input arguments fw.id, rev.id, merge.ambig, and max.degen are only used for loading primers from a FASTA file. If you want to load a FASTA file, please ensure that fw.id and rev.id are set according to the keywords indicating the primer directionalities in the FASTA file. When loading a CSV file, the format of the file should adhere to the structure defined by the Primers class. You can easily store a Primers objects as a CSV file using the write_primers function.

Value

An object of class Primers for a single primer.location or a list of such objects for multiple locations.

Examples

```
primer.fasta <- system.file("extdata", "IMGT_data", "primers", "IGHV",
                    "Ippolito2012.fasta", package = "openPrimeR")
primer.df <- read_primers(primer.fasta, "_fw", "_rev")
# Read multiple FASTA files
fasta.files <- list.files(system.file("extdata", "IMGT_data", "primers",
                "IGHV", package = "openPrimeR"), pattern = "*\\.fasta",
                full.names = TRUE)[1:3]
primer.data <- read_primers(fasta.files)
# Read primers from a CSV file
primer.csv <- system.file("extdata", "IMGT_data", "comparison",
             "primer_sets", "IGL", "IGL_openPrimeR2017.csv",  package = "openPrimeR")
primer.df <- read_primers(primer.csv)
# Read multiple primer CSV files
primer.files <- list.files(path = system.file("extdata", "IMGT_data", "comparison",
                        "primer_sets", "IGH", package = "openPrimeR"),
                         pattern = "*\\.csv", full.names = TRUE)[1:3]
primer.data <- read_primers(primer.files)
# Read a mixture of FASTA/CSV files:
mixed.primers <- c(primer.fasta, primer.csv)
primer.data <- read_primers(mixed.primers)
```

---

read_settings *Loading of Analysis Settings.*

Description

Loads primer analysis settings from an XML file.

Usage

```
read_settings(filename = list.files(system.file("extdata", "settings", package
  = "openPrimeR"), pattern = "*.xml", full.names = TRUE), frontend = FALSE)
```

Arguments

filename	Path to a valid XML file containing the primer analysis settings. By default, filename is set to all settings that are shipped with openPrimeR and the lexicographically first file is loaded.
frontend	Indicates whether settings shall be loaded for the Shiny frontend. In this case no unit conversions for the PCR settings are performed. The default setting is FALSE such that the correct units are used.

Details

If filename is not provided, a default XMl settings file is loaded. Please review the function's examples to learn more about the default settings. If you want to load custom settings, you can store a modified DesignSettings object as an XML file using write_settings.

Value

An object of class DesignSettings.

See Also

Other settings functions: ConstraintOptions-class, ConstraintSettings-class, CoverageConstraints-class, DesignSettings-class, PCR_Conditions-class, PCR, conOptions, constraintLimits, constraints, cvg_constraints

Examples

```
# Select the available settings
#' the available supplied settings by calling
available.settings <- list.files(
     system.file("extdata", "settings", package = "openPrimeR"),
     pattern = "*.xml", full.names = TRUE)
# Select one of the settings and load them
filename <- available.settings[1]
settings <- read_settings(filename)
# Modify, store, and read a settings object:
constraints(settings)$gc_clamp <- c("min" = 0, "max" = 5)
out.file <- tempfile("my_settings", fileext = ".xml")
write_settings(settings, out.file)
my_settings <- read_settings(out.file)
```

---

read_templates *Input of Template Sequences.*

Description

Read one or multiple files with template sequences in FASTA or CSV format.

Usage

```
read_templates(template.file, hdr.structure = NULL, delim = NULL,
  id.column = NULL, rm.keywords = NULL, remove.duplicates = FALSE,
  fw.region = c(1, 30), rev.region = c(1, 30), gap.character = "-",
  run = NULL)
```

Arguments

template.file	Path to one or multiple FASTA or CSV files containing the template sequences.
hdr.structure	A character vector describing the information contained in the FASTA headers. In case that the headers of fasta.file contain template group information, please include the keyword "GROUP" in hdr.structure. If the numer of elements provided via hdr.structure is shorter than the actual header structure, the missing fields are ignored.
delim	Delimiter for the information in the FASTA headers.
id.column	Field in the header to be used as the identifier of individual template sequences.
rm.keywords	A vector of keywords that are used to remove templates whose headers contain any of the keywords.
remove.duplicates	Whether duplicate sequence shall be removed.

fw.region	The positional interval from the template 5' end specifying the binding sites for forward primers. The default fw.region is set to the first 30 bases of the templates.
rev.region	The positional interval from the template 3' end specifying the binding sites for reverse primers. The default rev.region is set to the last 30 bases of the templates.
gap.character	The character in the input file representing gaps. Gaps are automatically removed upon input and the default character is "-".
run	An identifier for the set of template sequences. By default, run is NULL and its value is set via template.file.

Details

When supplying a FASTA file with template sequences, the input arguments hdr.structure, delim, id.column, rm.keywords, remove.duplicates, fw.region, rev.region, gap.character, and run are utilized. Most importantly, hdr.structure and delim should match the FASTA header structure. To learn more about setting the primer binding regions, consider the assign_binding_regions function. In contrast, when supplying a CSV file with template sequences, the data are loaded without performing any modifications because the CSV file should represent an object of class Templates, which can be stored using the write_templates function.

Value

An object of class Templates.

See Also

Other template functions: Templates-class, assign_binding_regions, update_template_cvg, write_templates

Examples

```
# Load templates from a FASTA file
fasta.file <- system.file("extdata", "IMGT_data", "templates",
          "Homo_sapiens_IGH_functional_exon.fasta", package = "openPrimeR")
hdr.structure <- c("ACCESSION", "GROUP", "SPECIES", "FUNCTION")
template.df.fasta <- read_templates(fasta.file, hdr.structure, "|", "GROUP")
# Load mutliple FASTA files
fasta.files <- c(fasta.file, fasta.file)
template.df.fastas <- read_templates(fasta.files, hdr.structure, "|", "GROUP")
# Load templates from a previously stored CSV file
csv.file <- system.file("extdata", "IMGT_data", "comparison",
               "templates", "IGH_templates.csv", package = "openPrimeR")
template.df.csv <- read_templates(csv.file)
# Load multiple CSV files:
csv.files <- c(csv.file, csv.file)
template.df.csvs <- read_templates(csv.files)
# Load a mixture of FASTA/CSV files:
mixed.files <- c(csv.file, fasta.file)
template.data <- read_templates(mixed.files)
```

RefCoverage *Experimental Coverage Data.*

Description

Experimental Coverage Data.

Usage

```
data(RefCoverage)
```

Format

The `feature.matrix` data frame contains the properties of the primer set from Tiller et al. as well as a primer set that was designed by openPrimeR. The column `Experimental_Coverage` indicates the experimentally determined coverage, while the other columns relate to properties of the primers that were computed with openPrimeR. The `ref.data` list contains the raw experimental coverage of individual primers from the primer sets from Tiller and openPrimeR, which both target templates from the IGH locus. The rows of the data frames indicate primers and the columns indicate IGH templates for which experimental coverage was determined. The cell entries are hex codes. Each hex code represents a color indicating a certain experimental coverage status. Hex codes representing red shades indicate no or little amplification, while hex codes for green shades indicate high yields.

References

Tiller, Thomas, et al. "Efficient generation of monoclonal antibodies from single human B cells by single cell RT-PCR and expression vector cloning." Journal of immunological methods 329.1 (2008): 112-124.

Examples

```
data(RefCoverage)
```

runTutorial *The openPrimeR Tutorial.*

Description

Starts a Shiny app containing the openPrimeR tutorial, which was built using the learnr package. The application starts locally and should open a new tab in your default browser. If no browser is opened, please consider the console output to identify the local port on which the server is running.

Usage

```
runTutorial(dev = FALSE)
```

Arguments

`dev`	A logical indicating whether to start the development version of the tutorial (default: `FALSE`).

Value

Opens the openPrimeR tutorial in a web browser.

Note

The Shiny app can be started only if you fulfill all of the suggested package dependencies for the Shiny framework, so please ensure that you've installed openPrimeR including all suggested dependencies.

Examples

```
# Open the tutorial
## Not run:
runTutorial()

## End(Not run)
```

---

score_conservation *Scoring of Template Conservation.*

Description

Determines the sequence conservation scores of a set of templates using Shannon entropy.

Usage

```
score_conservation(template.df, gap.char = "-", win.len = 30,
  by.group = TRUE)
```

Arguments

template.df	A Templates object providing the sequence conservation shall be determined.
gap.char	The alignment gap character. By default, this is set to "-".
win.len	The size of a window for evaluating conservation. The default window size is set to 30.
by.group	Whether the determination of binding regions should be stratified according to the groups defined in template.df. The default is TRUE.

Value

A list containing Entropies and Alignments. Entropies are given as a data frame with conservation scores. Each column indicates a position in the alignment of template sequences and each row gives the entropies of the sequences belonging to a specific group of template sequences. Alignments are given as lists of DNABin objects, where each object gives the alignment corresponding to one group of template sequences.

Note

Requires the MAFFT software for multiple alignments (http://mafft.cbrc.jp/alignment/software/).

Examples

```
## Not run:
data(Ippolito)
entropy.data <- score_conservation(template.df, gap.char = "-", win.len = 18, by.group = TRUE)

## End(Not run)
```

`score_degen` *Primer Degeneration Score.*

Description

Determines the degeneration score of a sequence.

Usage

```
score_degen(seq, gap.char = "-")
```

Arguments

`seq`	A list of vectors with single characters.
`gap.char`	The gap character in sequences.

Details

The degeneration of an ambiguous sequence is defined as the number of unambiguous sequences that the ambiguous sequence represents. Let a sequence `S` of length `n` be represented by a collection of sets such that

$$S = s_1, s_2, \ldots, s_n$$

where s_i indicates the set of unambiguous bases found at position i of the primer. Then the degeneracy D of a primer can be defined as

$$D = \prod_i |s_i|$$

where $|s_i|$ provides the number of disambiguated bases at position i.

Value

The number of unambiguous sequences represented by `seq`.

See Also

Other primer functions: `Primers-class`, `check_constraints`, `check_restriction_sites`, `create_report`, `design_primers`, `filter_primers`, `get_initial_primers`, `primer_significance`, `write_primers`

score_primers *Scoring of Primers.*

Description

Computes scores for a set of primers based on the deviations of the primers from the constraints.

Usage

```
score_primers(primer.df, settings,
  active.constraints = names(constraints(settings)), alpha = 0.5)
```

Arguments

primer.df	A Primers object containing the primers that are to be scored.
settings	A DesignSettings object containing the settings that are evaluated when computing the deviation.
active.constraints	A character vector of constraint identifiers that are considered for scoring the primers.
alpha	A numeric that is used to determine the trade-off between the impact of the maximal observed deviation and the total deviation. At its default alpha is set to 0.5 such that the maximal deviation and the total deviation have an equal weight when computing the penalties.

Details

The penalty of a primer is computed in the following way. Let d be a vector indicating the absolute deviations from individual constraints and let p be the scalar penalty that is assigned to a primer. We define

$$p = \alpha \cdot \max_i d_i + \sum_i (1 - \alpha) \cdot d_i$$

such that for large values of alpha the maximal deviation dominates giving rise to a local penalty (reflecting the largest absolute deviation) and for small alpha the total deviation dominates giving rise to a global penalty (reflecting the sum of constraint deviations). When alpha is 1 only the most extreme absolute deviation is considered and when alpha is 0 the sum of all absolute deviations is computed.

Value

A data frame containing scores for the primers.

Examples

```
data(Ippolito)
primer.scores <- score_primers(primer.df, settings)
```

`select_regions_by_conservation` *Selection of Primer Binding Regions by Conservation.*

Description

Computes Shannon entropy for putative binding regions and determines the most conserved regions.

Usage

```
select_regions_by_conservation(template.df, gap.char = "-", win.len = 30,
  by.group = TRUE, direction = c("both", "fw", "rev"))
```

Arguments

`template.df`	A `Templates` object containing the template sequences for which the binding regions shall be determined according to conservation.
`gap.char`	The alignment gap character. This is "-" by default.
`win.len`	The extent of the desired primer binding region. This should be smaller than the `allowed.region`. The default is 30.
`by.group`	Shall the determination of binding regions be stratified according to the groups defined in `template.df`. By default, this is set to `TRUE`.
`direction`	Whether regions shall be selected for primers of both directions ("both"), forward primers ("fw"), or reverse primers ("rev"). The default is "both".

Value

A `Templates` object with adjusted binding regions. The attribute `entropies` gives a data frame with positional entropies and the attribute `alignments` gives the alignments of the templates.

Note

Requires the MAFFT software for multiple alignments (http://mafft.cbrc.jp/alignment/software/).

Examples

```
## Not run:
data(Ippolito)
new.template.df <- select_regions_by_conservation(template.df)

## End(Not run)
```

startApp *The openPrimeR Shiny Application.*

Description

Starts the openPrimeR Shiny application. A new tab should open in your default browser. If no browser is opened, please consider the console output to identify the local port on which the server is running and manually open the shown URL.

Usage

```
startApp()
```

Value

Opens the Shiny app in a web browser.

Note

The Shiny app can be started only if you fulfill all of the suggested package dependencies for the Shiny framework, so please ensure that you've installed openPrimeR including all suggested dependencies.

Examples

```
# Start the shiny app
## Not run:
startApp()

## End(Not run)
```

subset_primer_set *Optimal Primer Subsets.*

Description

Determines subsets of the input primer set that are optimal with regard to the number of covered template sequences.

Usage

```
subset_primer_set(primer.df, template.df, k = 1, groups = NULL,
  identifier = NULL, cur.results.loc = NULL)
```

Arguments

primer.df	An objectc of class Primers providing the primers for which optimal subsets should be constructed.
template.df	An object of class Templates providing the template sequences that are targeted by primer.df.
k	The spacing between generated primer subset sizes. By default, k is set to 1 such that all primer subsets are constructed.
groups	The identifiers of template groups according to which coverage should be determined. By default, groups is set to NULL such that all all covered templates are considered.
identifier	An identifier for storing the primer set. By default, identifier is set to NULL.
cur.results.loc	Directory for storing the results. By default, cur.results.loc is set to NULL, which means that no results are stored.

Details

The optimal subsets are identified by solving an integer-linear program. Since the quality of the primers (in terms of properties) is not taken into account when creating the subsets, this method should only be used for primer sets that are already of high quality.

Value

A list with optimal primer subsets, each of class Primers.

Examples

```
data(Ippolito)
primer.subsets <- subset_primer_set(primer.df, template.df)
```

Templates-class *The Templates Class.*

Description

The Templates class encapsulates a data frame containing the sequencs of the templates, their binding regions, as well as additional information (e.g. template coverage).

Usage

```
Templates(...)
```

Arguments

...	A data frame fulfilling the structural requirements for initializing a Templates object.

Details

In the following you can find a description of the most important columns that can be found in an object of class `Templates`. Note that angle brackets in the column names indicate the existence of multiple possibilities.

`ID` The identifiers of the templates.

`Identifier` The internal identifiers of the templates.

`Group` The identifiers of the groups that the templates belong to.

`Allowed_Start_<fw|rev>` The start of the interval in the templates where binding is allowed for forward and reverse primers, respectively.

`Allowed_End_<fw|rev>` The end of the interval in the templates where binding is allowed for forward and reverse primers, respectively.

`Allowed_<fw|rev>` The template sequence where binding is allowed for forward and reverse primers, respectively.

`Run` An identifier for the set of template sequences.

`Covered_By_Primers` The identifiers of primers covering the templates, when the template coverage has been annotated.

`primer_coverage` The number of primers covering the templates, when the template coverage has been annotated.

Value

A `Templates` object, an instance of a data frame.

See Also

`read_templates` for loading template sequences, `assign_binding_regions` for adjusting the primer binding regions, `update_template_cvg` for setting the template coverage, `plot_template_cvg` for plotting the template coverage,

Other template functions: `assign_binding_regions`, `read_templates`, `update_template_cvg`, `write_templates`

Examples

```
fasta.file <- system.file("extdata", "IMGT_data", "templates",
     "Homo_sapiens_IGH_functional_exon.fasta", package = "openPrimeR")
hdr.structure <- c("ACCESSION", "GROUP", "SPECIES", "FUNCTION")
template.df <- read_templates(fasta.file, hdr.structure, "|", "GROUP")
```

Tiller *Evaluated Primer Data from Tiller et al.*

Description

Primer and template data for IGHV from Tiller et al.

Usage

```
data(Tiller)
```

Format

`tiller.primer.df` provides a `Primers` object, `tiller.template.df` provides the corresponding `Templates` object, and `tiller.settings` provides the `DesignSettings` object that was used for evaluating `tiller.primer.df`. DesignSettings object `tiller.settings`.

References

Tiller, Thomas, et al. "Efficient generation of monoclonal antibodies from single human B cells by single cell RT-PCR and expression vector cloning." Journal of immunological methods 329.1 (2008): 112-124.

Examples

```
data(Tiller)
tiller.primer.df
tiller.template.df
constraints(tiller.settings)
```

`update_template_cvg` *Annotation of Template Coverage.*

Description

Annotates the templates with coverage information.

Usage

```
update_template_cvg(template.df, primer.df, mode.directionality = NULL)
```

Arguments

`template.df`	An object of class `Templates`.
`primer.df`	An object of class `Primers` containing primers with annotated coverage that are to be used to update the template coverage in `template.df`.
`mode.directionality`	Directionality of primers. The default is `NULL`, which means that the directionality of primers is identified automatically.

Value

An object of class `Templates` with updated coverage columns.

See Also

Other template functions: `Templates-class`, `assign_binding_regions`, `read_templates`, `write_templates`

Examples

```
data(Ippolito)
template.df <- update_template_cvg(template.df, primer.df)
```

write_primers *Storing Primers to Disk.*

Description

Writes a set of primers to disk, either as a FASTA or CSV file.

Usage

```
write_primers(primer.df, fname, ftype = c("FASTA", "CSV"))
```

Arguments

primer.df	An object of class Primers to be stored to disk.
fname	The path to the file where the primers should be stored.
ftype	A character vector giving the type of the file. This can either be "FASTA" or "CSV" (default: "FASTA").

Value

Stores primers to fname.

See Also

Other primer functions: Primers-class, check_constraints, check_restriction_sites, create_report, design_primers, filter_primers, get_initial_primers, primer_significance, score_degen

Examples

```
data(Ippolito)
# Store primers as FASTA
fname.fasta <- tempfile("my_primers", fileext = ".fasta")
write_primers(primer.df, fname.fasta)
# Store primers as CSV
fname.csv <- tempfile("my_primers", fileext = ".csv")
write_primers(primer.df, fname.csv, "CSV")
```

write_settings *Storing Design Settings to Disk.*

Description

Stores primer analysis settings to a file in XML format.

Usage

```
write_settings(settings, filename)
```

Arguments

settings	A DesignSettings object to be stored to disk.
filename	A character vector specifying the location where the settings should be stored as an XML file.

Value

Outputs the return status from closing the connection.

Examples

```
xml <- settings.xml <- system.file("extdata", "settings",
       "C_Taq_PCR_high_stringency.xml", package = "openPrimeR")
settings <- read_settings(xml)
out.file <- tempfile("my_settings", fileext = ".xml")
write_settings(settings, out.file)
```

---

write_templates *Storing Templates to Disk.*

Description

Stores a set of templates as a FASTA or CSV file.

Usage

```
write_templates(template.df, fname, ftype = c("FASTA", "CSV"))
```

Arguments

template.df	An object of class Templates to be stored to disk.
fname	The path to the file where the templates should be stored.
ftype	A character vector giving the filetype. This can either be "FASTA" or "CSV" (default: "FASTA").

Value

Stores templates to fname.

See Also

Other template functions: Templates-class, assign_binding_regions, read_templates, update_template_cvg

Examples

```
data(Ippolito)
# Store templates as FASTA
fname.fasta <- tempfile("my_templates", fileext = ".fasta")
write_templates(template.df, fname.fasta)
# Store templates as CSV
fname.csv <- tempfile("my_templates", fileext = ".csv")
write_templates(template.df, fname.csv, "CSV")
```

Index