

# Package ‘openPrimeR’

May 20, 2026

**Title** Multiplex PCR Primer Design and Analysis

**Version** 1.35.0

**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. A Shiny app providing a user interface for the functionalities of this package is provided by the 'openPrimeRui' package.

**Depends** R (>= 4.0.0)

**License** GPL-2

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**Imports** Biostrings (>= 2.38.4), palign, 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), unqiqttag (>= 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** 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 (>= 2.4.1), MELTING (>= 5.1.1), Pandoc (>= 1.12.3)

**biocViews** Software, Technology, Coverage, MultipleComparison

**VignetteBuilder** knitr

**Collate** 'AnalysisStats.R' 'Comparison.R' 'Data.R' 'templates.R' 'primers.R' 'IO.R' 'IO\_view.R' 'Input.R' 'Ippolito.R' 'Output.R' 'Plots.R' 'PrimerDesign.R' 'PrimerEvaluation.R' 'RefCoverage.R' 'Scoring.R' 'SettingsDoc.R' 'TemplatesDoc.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'

**git\_url** <https://git.bioconductor.org/packages/openPrimeR>

**git\_branch** devel

**git\_last\_commit** 54d9897

**git\_last\_commit\_date** 2026-04-28

**Repository** Bioconductor 3.24

**Date/Publication** 2026-05-19

**Author** Matthias Döring [aut, cre],  
 Nico Pfeifer [aut]

**Maintainer** Matthias Döring <matthias-doering@gmx.de>

## Contents

openPrimeR-package . . . . .	9
AbstractConstraintSettings-class . . . . .	10
add.coverage.constraints . . . . .	11
add.dimerization.constraints . . . . .	11
add.uniform.leaders.to.seqs . . . . .	12
adjust.ORF.start . . . . .	12
align.seqs . . . . .	13
align.structures . . . . .	13
AnalysisStats . . . . .	14
ancestor_of . . . . .	16
annealing.temp.rule.of.thumb . . . . .	17
annotate.binding.events . . . . .	17
apply.constraint . . . . .	18
apply.constraint.list . . . . .	18
assign_binding_regions.character . . . . .	19
assign_binding_regions.numeric . . . . .	20
augment.primer.cvg . . . . .	21
batchify . . . . .	21
batchify.simple . . . . .	22
batchify.temp . . . . .	22
build.gain.df . . . . .	23
build.ILP.df . . . . .	23
build.tool.overview . . . . .	24
build_leader_df . . . . .	25

call.melt . . . . .	25
call.melt.single . . . . .	26
cascaded.filter . . . . .	27
cascaded.filter.quick . . . . .	28
cbind.Primers . . . . .	29
cbind.Templates . . . . .	29
cbind2,Primers,ANY-method . . . . .	30
cbind2,Templates,ANY-method . . . . .	31
check.3prime.hexamers . . . . .	32
check.3prime.mismatches . . . . .	32
check.init.primers.length . . . . .	33
check.init.primers.length.single . . . . .	34
check.mutations . . . . .	34
check.template.constraints . . . . .	35
check.tool.function . . . . .	36
check.tool.installation . . . . .	36
check_constraints_comparison . . . . .	37
check_constraint_settings_validity . . . . .	38
check_correspondence . . . . .	38
check_cvg_constraints . . . . .	39
check_interval . . . . .	40
check_limits . . . . .	40
check_limit_value . . . . .	41
check_names . . . . .	41
check_report_deps . . . . .	42
check_restriction_sites_single . . . . .	42
check_setting . . . . .	43
check_settings_validity . . . . .	43
combine.binding.events . . . . .	44
combine.strings . . . . .	44
comp . . . . .	45
compare.constraints . . . . .	45
comparison.cvg . . . . .	46
comparison.stats.raw . . . . .	46
complement.sequence . . . . .	47
compute.all.cross.dimers . . . . .	47
compute.all.cross.dimers.frontend . . . . .	48
compute.all.cross.dimers.unfiltered . . . . .	49
compute.all.primers.subsets.ILP . . . . .	50
compute.all.self.dimers . . . . .	50
compute.all.self.dimers.frontend . . . . .	51
compute.basic.details . . . . .	52
compute.constraints . . . . .	53
compute.covered.Ta . . . . .	54
compute.dimer.matrix . . . . .	55
compute.entropy . . . . .	55
compute.entropy.melting.temp . . . . .	56
compute.gc.ratio . . . . .	57
compute.melting.temps . . . . .	57
compute.melting.temps.thermo . . . . .	58
compute.mismatch.table . . . . .	59
compute.primers.efficiencies . . . . .	59

compute.secondary.structures . . . . .	60
compute.sodium.equivalent.conc . . . . .	61
compute.structure.vienna . . . . .	62
compute.Ta . . . . .	63
compute.template.secondary.structures . . . . .	64
compute.Tm.baldino . . . . .	64
compute.Tm.sets . . . . .	65
compute.unique.covered.idx . . . . .	67
compute_annealing_temp . . . . .	67
condition . . . . .	68
consecutive.GC.count . . . . .	69
constraints.to.df . . . . .	69
constraints.xml.format . . . . .	70
constraints_to_unit . . . . .	70
convert.from.iupac . . . . .	71
convert.PCR.units . . . . .	71
convert.temperature . . . . .	72
convert.to.iupac . . . . .	72
con_select . . . . .	73
copy.melt.config . . . . .	73
covered.primers.to.ID.string . . . . .	73
covered.seqs.to.ID.string . . . . .	74
covered.seqs.to.idx . . . . .	74
create.constraint.table . . . . .	75
create.constraint.XML . . . . .	75
create.cvg.text . . . . .	76
create.G.matrix . . . . .	76
create.initial.primers.set . . . . .	77
create.k.mers . . . . .	78
create.kmer . . . . .	78
create.options.table . . . . .	79
create.other.table . . . . .	79
create.PCR.table . . . . .	80
create.primers.ranges . . . . .	80
create.primers.naive . . . . .	81
create.primers.tree . . . . .	82
create.Tm.brackets . . . . .	83
create.uniform.leaders . . . . .	83
create_fulfilled_counts . . . . .	84
create_report,list,list-method . . . . .	84
create_report,Primers,Templates-method . . . . .	85
Data . . . . .	86
design_primers.single . . . . .	87
detect.gap.columns . . . . .	89
dimerization.table . . . . .	89
dir.copy . . . . .	90
disambiguate.primers . . . . .	90
estimate.cvg . . . . .	91
estimate.cvg.dir . . . . .	91
eval.comparison.primers . . . . .	92
eval.constraints . . . . .	92
evaluate.basic.cvg . . . . .	93

evaluate.constrained.cvg . . . . .	94
evaluate.cvg . . . . .	94
evaluate.diff.primer.cvg . . . . .	95
evaluate.fw.rev.combinations . . . . .	96
evaluate.GC.clamp . . . . .	96
evaluate.primer.cvg . . . . .	97
evaluate.template.constraints . . . . .	97
exclude.cols . . . . .	98
filter.by.constraints . . . . .	98
filter.comparison.primers . . . . .	99
filter.primer.candidates . . . . .	100
filter.primer.set.opti . . . . .	100
filterLimits . . . . .	101
filters . . . . .	102
filter_primers.by.Tm.delta . . . . .	102
fix_constraint_boundaries . . . . .	103
format.constraints . . . . .	103
format.seq.ali . . . . .	104
format.seqs.tex . . . . .	104
get.3prime.mismatch.pos . . . . .	105
get.analysis.mode . . . . .	105
get.consensus.seq . . . . .	106
get.constraint.value.idx . . . . .	106
get.constraint.values . . . . .	107
get.coverage.matrix . . . . .	107
get.covered.templates . . . . .	108
get.cross.dimers . . . . .	108
get.cvg.constraint.settings . . . . .	109
get.cvg.gain . . . . .	110
get.delta.G . . . . .	110
get.dimer.data . . . . .	111
get.duplex.energies . . . . .	111
get.eval.cols . . . . .	112
get.extension . . . . .	112
get.ILP.vars . . . . .	113
get.init.file.name . . . . .	113
get.leader.exon.regions . . . . .	114
get.leader.exon.regions.single . . . . .	115
get.matches . . . . .	115
get.melting.temp.diff . . . . .	116
get.merge.idx . . . . .	116
get.missing.df . . . . .	117
get.ORFs . . . . .	117
get.other.constraint.settings . . . . .	118
get.PCR.settings . . . . .	118
get.plot.height . . . . .	119
get.primer.binding.idx . . . . .	120
get.primer.identifier.string . . . . .	120
get.redundant.cols . . . . .	121
get.relative.binding.pos . . . . .	122
get.run.names . . . . .	122
get.self.dimers . . . . .	123

get.sets.from.decisions	123
get.static.tool.info	124
get.tree.seqs	124
get.unlist.idx	125
get_constraint_deviation_data	125
get_covered.vanilla	126
get_cvg_stats,list-method	126
get_cvg_stats,Primers-method	127
get_max_set_coverage	128
get_plot_primer_data	129
get_primer_cvg_mm_plot_df	129
get_report_fname	130
get_template_cvg_data	130
hclust.tree	131
highlight.mismatch	131
html.format.structure	132
I.cvg	132
ILPConstrained	133
initialize.primer.set	133
Input	134
insert_str	139
interleave	140
J.cvg	140
joule.to.cal	141
listToXml	141
merge.ambig.primers	142
merge.binding.information	142
merge.primer.entries	143
merge.primer.entries.single	144
merge.select	144
merge.template.decisions	145
mismatch.info	145
mismatch.mutation.check	146
mismatch.string.to.list	146
modify.col.rep	147
my.disambiguate	147
my.error	148
my.read.fasta	148
my.warning	149
my_ggsave	149
my_rbind	150
nbr.of.repeats	150
nbr.of.runs	151
opti	151
optiLimits	152
optimize.ILP	152
optimize.primer.cvg	154
optimize.template.binding.regions.dir	155
optimize.template.binding.regions.single	155
Output	156
pair_primers	158
parse.constraints	159

parse.header . . . . .	159
parse.IMGT.gene.groups . . . . .	160
parse.oligo.results . . . . .	160
plot.all.cvg.info . . . . .	161
plot.all.filtering.stats . . . . .	162
plot.Delta.DeltaG . . . . .	162
plot.dimer.dist . . . . .	163
plot.excluded.hist . . . . .	163
plot.filtering.runtime . . . . .	164
plot.filtering.stats . . . . .	164
plot.filtering.stats.cvg . . . . .	165
Plots . . . . .	165
plot_constraint,list-method . . . . .	170
plot_constraint,Primers-method . . . . .	171
plot_constraint.histogram . . . . .	172
plot_constraint.histogram.nbr.mismatches . . . . .	173
plot_constraint.histogram.primers.efficiencies . . . . .	173
plot_constraint_deviation,list-method . . . . .	174
plot_constraint_deviation,Primers-method . . . . .	174
plot_constraint_fulfillment,list-method . . . . .	175
plot_constraint_fulfillment,Primers-method . . . . .	176
plot_cvg_constraints,list-method . . . . .	176
plot_cvg_constraints,Primers-method . . . . .	177
plot_primer.comparison.box . . . . .	177
plot_primer.comparison.mismatches . . . . .	178
plot_primer_binding_regions,list,list-method . . . . .	179
plot_primer_binding_regions,Primers,Templates-method . . . . .	180
plot_primer_cvg,list,list-method . . . . .	180
plot_primer_cvg,Primers,Templates-method . . . . .	181
plot_primer_cvg_mismatches . . . . .	181
plot_primer_cvg_unstratified . . . . .	182
plot_template_cvg,list,list-method . . . . .	183
plot_template_cvg,Primers,Templates-method . . . . .	183
plot_template_cvg_comparison_mismatch . . . . .	184
plot_template_cvg_comparison_unstratified . . . . .	184
plot_template_cvg_mismatches . . . . .	185
plot_template_cvg_unstratified . . . . .	185
plot_template_structure . . . . .	186
pos.to.range . . . . .	186
predict_coverage . . . . .	187
prefilter.primers.candidates . . . . .	187
prepare.constraint.plot . . . . .	188
prepare.dimer.seqs . . . . .	188
prepare_mm_plot . . . . .	189
prepare_template_cvg_mm_data . . . . .	189
primers.binding.regions.data . . . . .	190
primers.coverage.for.groups . . . . .	190
primers.set.parameter.stats . . . . .	191
PrimerDesign . . . . .	191
PrimerEval . . . . .	195
rbind.primers.data . . . . .	198
rbind.Primers . . . . .	199

rbind.Templates . . . . .	199
read.leaders . . . . .	200
read.secondary.structure.raw . . . . .	200
read.sequences . . . . .	201
read_primers.internal . . . . .	201
read_primers_csv . . . . .	202
read_primers_multiple . . . . .	202
read_templates_csv . . . . .	203
read_templates_fasta . . . . .	203
read_templates_multiple . . . . .	204
read_templates_single . . . . .	205
relax.constraints . . . . .	206
relax.opti.constraints . . . . .	207
remove.redundant.cols . . . . .	208
remove.seqs.by.keyword . . . . .	208
rename.constraint.options . . . . .	209
render_report . . . . .	209
reorder.primer.table . . . . .	210
restriction_ali . . . . .	210
restriction_hits . . . . .	211
restriction_match . . . . .	211
retrieve.leader.region . . . . .	212
rev.comp.sequence . . . . .	212
rev.sequence . . . . .	213
runTutorial . . . . .	213
sanitize_path . . . . .	214
score.conservaion . . . . .	214
Scoring . . . . .	215
select.allowed.binding.events . . . . .	217
select.best.ILP . . . . .	217
select.best.opti.result . . . . .	218
select.best.primer.idx . . . . .	218
select.best.primer.set . . . . .	219
select.binding.events . . . . .	220
select.constraints . . . . .	220
select.min.cross.idx . . . . .	221
select.primer.region.by.conservaion . . . . .	221
select.primers.by.cvg . . . . .	222
select_best_binding . . . . .	223
selenium.installed . . . . .	224
set.new.constraint.value . . . . .	224
set.new.limits . . . . .	225
Settings . . . . .	225
shannon.entropy . . . . .	234
solve.ILP . . . . .	234
split_str_by_index . . . . .	235
stats_plot_data . . . . .	236
store.filtering.sets . . . . .	236
string.list.format . . . . .	237
string.list.format.total . . . . .	237
string.to.IQR . . . . .	238
subset.ILP . . . . .	238

TemplatesFunctions . . . . .	239
ungap_sequence . . . . .	241
unify.leaders . . . . .	242
update.binding.ranges.by.conservaion . . . . .	242
update.binding.regions . . . . .	243
update.constraint.values . . . . .	243
update.cvg.data . . . . .	244
update.individual.binding.region . . . . .	244
update.opti.results . . . . .	245
update_primer_binding_regions . . . . .	246
update_primer_cvg . . . . .	246
validate_primers . . . . .	247
validate_templates . . . . .	247
view.cvg.primers . . . . .	248
view.dimer.df . . . . .	248
view.input.primers . . . . .	249
view.primers . . . . .	249
view.primers.report . . . . .	250
visualize.all.results . . . . .	250
visualize.filtering.results . . . . .	251
were.constraints.relaxed . . . . .	252
write.out.primer.info . . . . .	253
xmlToChar . . . . .	253

**Index****255**


---

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 can 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 <matthias-doering@gmx.de>

Authors:

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

---

AbstractConstraintSettings-class

*AbstractClass for Constraint Settings.*

---

**Description**

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

**Value**

An AbstractConstraintSettings 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

---

`add.coverage.constraints`*Addition of Coverage Constraints.*

---

**Description**

Adds coverage constraints to ILP instance.

**Usage**

```
add.coverage.constraints(lprec, covered.templates, template.coverage)
```

**Arguments**

<code>lprec</code>	An ILP instance.
<code>covered.templates</code>	Indices of covered template sequences.
<code>template.coverage</code>	List containing the indices of covering primers for each template.

**Value**

`lprec` with coverage constraints.

---

`add.dimerization.constraints`*Addition of Dimerization Constraints*

---

**Description**

Updates ILP formulation with dimerization constraints.

**Usage**

```
add.dimerization.constraints(lprec, D.idx, indices)
```

**Arguments**

<code>lprec</code>	An ILP instance.
<code>D.idx</code>	Data frame giving the indices of dimerizing primer pairs.
<code>indices</code>	Row indices for setting the dimerization constraints in <code>lprec</code> .

**Value**

`lprec` with added dimerization constraints.

---

```
add.uniform.leaders.to.seqs
```

*Add Uniform Binding Regions.*

---

**Description**

Augments a template data frame with uniform binding regions.

**Usage**

```
add.uniform.leaders.to.seqs(lex.seq, leaders)
```

**Arguments**

<code>lex.seq</code>	Template data frame
<code>leaders</code>	Data frame with uniform binding regions.

**Value**

Template data frame with updated binding regions.

---

```
adjust.ORF.start
```

*Adjust ORF position*

---

**Description**

Adjusts the reading frame according to the position at which we consider a subsequence.

**Usage**

```
adjust.ORF.start(ORFs, seq.start)
```

**Arguments**

<code>ORFs</code>	the reading frames (0,1,2).
<code>seq.start</code>	the position where a sequence is extracted .

**Value**

The adjusted reading frames for the given start positions.

---

`align.seqs`*Multiple Sequence Alignment*

---

**Description**

Computes a multiple sequence alignment using MAFFT.

**Usage**

```
align.seqs(seqs, names)
```

**Arguments**

seqs	The sequences to be aligned.
names	The identifiers of the sequences.

**Value**

An alignment object as created from seqinr's read.alignment method.

**References**

Katoh, Misawa, Kuma, Miyata 2002 (Nucleic Acids Res. 30:3059-3066) MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform.

---

`align.structures`*Formatting of Dimerization Structures.*

---

**Description**

Formats the given dimerization structures nicely.

**Usage**

```
align.structures(structs)
```

**Arguments**

s1	structs A character vector, where a block of 4 elements contains: sequence 1 (with removed overlaps), part of sequence 1 overlapping with sequence 2, part of sequence 2 overlapping with sequence 1, and sequence 2 (with removed overlaps).
----	---

**Value**

A list of two elements giving the conformation of the first and the second sequence, respectively.

**Description**

`get_cvg_ratio` 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).

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

`get_cvg_stats_primer` Creates a table summarizing the coverage events of individual primers.

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

**Usage**

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

get_comparison_table(templates, primers, sample.name = NULL)

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

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

**Arguments**

`primer.df` A Primers object containing the primers.

`template.df` A Templates object containing the template sequences corresponding to `primer.df`.

`allowed.mismatches`  
The number of allowed mismatches for determining the coverage of the templates. By default, all annotated coverage events are considered.

<code>cvg.definition</code>	If <code>cvg.definition</code> is set to "constrained", the statistics for the expected coverage (after applying the coverage constraints) are retrieved. If <code>cvg.definition</code> is set to "basic", the coverage is determined solely by string matching (i.e. without applying the coverage constraints). By default, <code>cvg.definition</code> is set to "constrained".
<code>mode.directionality</code>	If <code>mode.directionality</code> 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, <code>mode.directionality</code> is NULL such that the directionality of the primers is determined automatically.
<code>as.char</code>	Whether the coverage ratio should be outputted as a percentage-formatted character vector. By default, <code>as.char</code> is set to FALSE such that a numeric is returned.
<code>templates</code>	If <code>primers</code> is an object of class <code>Primers</code> , please provide an object of class <code>Templates</code> containing the template sequences targeted by primers. If <code>primers</code> is a list, <code>templates</code> should be a list of <code>Template</code> objects.
<code>primers</code>	To retrieve statistics for a single primer set, please provide an object of class <code>Primers</code> containing a set of evaluated primers. To retrieve statistics for multiple primer sets, please provide a list with evaluated <code>Primers</code> objects.
<code>sample.name</code>	Either a single identifier or a character vector of identifiers for every <code>Templates</code> object in <code>templates</code> . By default, <code>sample.name</code> is NULL such that the Run annotations in the provided <code>Templates</code> objects are used.
<code>for.viewing</code>	Whether the table should be formatted to be human-readable. By default, <code>for.viewing</code> is FALSE.
<code>total.percentages</code>	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, <code>total.percentages</code> is FALSE such that the percentages are group-specific.

## Details

The manner in which `get_cvg_ratio` determines the coverage ratio 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.

For `get_cvg_stats_primer`, the cells corresponding to columns with numeric identifiers indicate the percentage of coverage events occurring with a certain number of mismatches. For example column 3 provides the number of coverage events where there are exactly three mismatches between primers and templates. The column `Group_Coverage` provides a listing of the percentage of covered templates per group.

## Value

By default, `get_cvg_ratio` returns 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.

`get_comparison_table` returns a data frame summarizing the properties of the provided primer data sets.

get\_cvgs\_stats\_primer returns a list with the following entries. cvg\_per\_nbr\_mismatches contains a data frame listing the number of binding events broken down according to the number of expected mismatches between primers and templates. cvg\_per\_group contains a data frame listing the the coverage of individual primers per group of templates.

get\_cvgs\_stats returns a data frame whose entries provide the coverage of templates per group of templates.

### Examples

```
data(Ippolito)
# Determine the overall coverage
cvg.ratio <- get_cvgs_ratio(primer.df, template.df)
# Determine the identity coverage ratio
cvg.ratio.0 <- get_cvgs_ratio(primer.df, template.df, allowed.mismatches = 0)

# Summarize the properties of multiple primer sets
data(Comparison)
tab <- get_comparison_table(template.data[1:3], primer.data[1:3], "IGH")

data(Ippolito)
# Determine coverage stats per primer
primer.cvg.stats <- get_cvgs_stats_primer(primer.df, template.df)

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

---

ancestor\_of

*Tree Ancestry*

---

### Description

Checks whether ancestor.node is an ancestor to the nodes specified in test.node.

### Usage

```
ancestor_of(tree, ancestor.node, test.node)
```

### Arguments

tree	The phylogenetic tree to be tested.
ancestor.node	A node to be checked for being an ancestor to test.node.
test.node	Possible descendants of ancestor.node.

### Value

TRUE, if ancestor.node is an ancestor to any node in test.node.

---

 annealing.temp.rule.of.thumb

*Rule of thumb for annealing temperature*


---

**Description**

Computes the annealing temperature using a rule of thumb

**Usage**

```
annealing.temp.rule.of.thumb(melting.temp)
```

**Arguments**

melting.temp     Melting temperatures of primers

**Value**

The annealing temperature corresponding to the input melting temperature.

---

annotate.binding.events

*Annotation of Primer Binding Events.*


---

**Description**

Annotates whether primer binding events are in the allowed binding region or not.

**Usage**

```
annotate.binding.events(  
  fw.binding,  
  allowed.range,  
  nbr.primers,  
  allowed.region.definition = c("within", "any")  
)
```

**Arguments**

fw.binding        IRanges with coverage information.  
allowed.range    IRanges of the allowed binding ranges in the templates.  
nbr.primers      Number of primers to consider.  
allowed.region.definition  
                  Definition of the allowed binding region

**Value**

IRanges with annotations of (preliminary) specificity and allowed binding. The field `all_binding` contains all binding regions, `on_target` contains all events in the target region, and `off_target` contains all off-target binding events.

---

apply.constraint      *Application of Constraints*

---

### Description

Checks whether the input values are within the specified limits.

### Usage

```
apply.constraint(  
  gc.ratio.fw,  
  gc.ratio.rev,  
  min.GC,  
  max.GC,  
  fw.idx,  
  rev.idx,  
  mode.directionality = c("fw", "rev", "both")  
)
```

### Arguments

gc.ratio.fw	Forward values.
gc.ratio.rev	Reverse values.
min.GC	Minimal allowed value.
max.GC	Maximal allowed value.
fw.idx	Indices of forward values to consider.
rev.idx	Indices of reverse values to consider.
mode.directionality	Direction of primers

### Value

Data frame with TRUE for values fulfilling the constraints, FALSE otherwise. Also returns FALSE if a data point is not available.

---

apply.constraint.list      *Apply Constraints to a List.*

---

### Description

Checks whether the input values are within the specified limits.

**Usage**

```

apply.constraint.list(
  gc.ratio.fw,
  gc.ratio.rev,
  min.GC,
  max.GC,
  fw.idx,
  rev.idx,
  mode.directionality = c("fw", "rev", "both")
)

```

**Arguments**

gc.ratio.fw	Forward values (comma-separated strings).
gc.ratio.rev	Reverse values (comma-separated strings).
min.GC	Minimal allowed value.
max.GC	Maximal allowed value.
fw.idx	Indices of forward values to consider.
rev.idx	Indices of reverse values to consider.
mode.directionality	Direction of primers

**Details**

Applies a constraint to every element in a vector of comma separated strings. Applied when filtering covered seqs according to primer efficiency.

**Value**

Data frame with TRUE for values fulfilling the constraints, FALSE otherwise.

---

```
assign_binding_regions.character
```

*Character Assignment of Binding Regions*

---

**Description**

Generic method for assigning the binding region using individual binding regions.

**Usage**

```

## S3 method for class 'character'
assign_binding_regions(
  template.df,
  fw,
  rev,
  optimize.region = FALSE,
  primer.length = 20,
  gap.char = "-"
)

```

**Arguments**

template.df	Template data frame.
fw	FASTA file specifying the forward binding regions.
rev	FASTA file specifying the reverse binding regions.
optimize.region	Should the primer binding region be optimized using secondary structure prediction?
primer.length	Probe length for optimizing template secondary structure.
gap.char	The gap character for aligned sequences.

**Value**

Template data frame with assigned binding regions.

---

```
assign_binding_regions.numeric
```

*Numeric Assignment of Binding Regions.*

---

**Description**

Numeric S3 generic case for assigning binding regions.

**Usage**

```
## S3 method for class 'numeric'
assign_binding_regions(
  template.df,
  fw,
  rev,
  optimize.region = FALSE,
  primer.length = 20,
  gap.char = "-"
)
```

**Arguments**

template.df	Template data frame.
fw	Binding region data forward primers.
rev	Binding region data for reverse primers.
optimize.region	Should the primer binding region be optimized using secondary structure prediction?
primer.length	Probe length for optimizing template secondary structure.

**Value**

The template data frame with assigned binding regions.

---

augment.primer.cvg      *Augmentation of Primer Coverage.*

---

**Description**

Computes the coverage for the primers in primer.df that is still missing such that the relaxation procedure can adjust appropriate constraints.

**Usage**

```
augment.primer.cvg(
  primer.df,
  template.df,
  settings,
  partial = FALSE,
  constraint = NULL
)
```

**Arguments**

primer.df	A Primers object for which the primer coverage shall be augmented.
template.df	A Templates object.
settings	A DesignSettings object giving the parameters for coverage computations.
partial	Whether all missing primer coverage values should be computed. If partial is TRUE, only the coverage values of the primers that were excluded due to the specified constraint are computed.
constraint	A character vector specifying the exclusion reason for which the partial augmentation should take place.

**Value**

A Primers object with augmented coverage entries.

---

batchify      *Creates multiple Batches for Parallelization.*

---

**Description**

Creates multiple Batches for Parallelization.

**Usage**

```
batchify(tasks, annealing.temps = NULL)
```

**Arguments**

tasks	An integer vector with indices representing individual computations.
annealing.temps	Temperatures according to which to batchify.

**Value**

A list of lists containing indices corresponding to tasks, each list gives a batch.

---

batchify.simple	<i>Simple Batchification</i>
-----------------	------------------------------

---

**Description**

Simple Batchification

**Usage**

```
batchify.simple(tasks)
```

**Arguments**

tasks            The tasks to assign to individual batches.

**Value**

A list of lists containing indices corresponding to tasks, each list gives a batch.

---

batchify.temp	<i>Batchification by Temperature.</i>
---------------	---------------------------------------

---

**Description**

Batchification by Temperature.

**Usage**

```
batchify.temp(tasks, annealing.temp)
```

**Arguments**

tasks            The tasks to assign to individual batches.  
annealing.temp   The annealing temperatures according to which batches are to be created.

**Value**

A list of lists containing indices corresponding to tasks, each list gives a batch.

---

build.gain.df	<i>Gain of Coverage by Excluded Primers.</i>
---------------	--

---

**Description**

Computes a data frame on the excluded sequences per constraint.

**Usage**

```
build.gain.df(candidate.df, constraint.settings, constraint.limits, relax.df)
```

**Arguments**

candidate.df    An object of class `Primers` containing excluded primers that are considered for addition to `filtered.df`.

constraint.settings    A list with the current constraint settings.

constraint.limits    The current constraint limits.

relax.df    Data frame with count of relaxations per constraint.

**Value**

A data frame with exclusion data.

---

build.ILP.df	<i>Construction of ILP Results.</i>
--------------	-------------------------------------

---

**Description**

Constructs a data frame summarizing the properties of an ILP solution.

**Usage**

```
build.ILP.df(  
  ILP,  
  vars,  
  primer.df,  
  template.df,  
  i,  
  target.temp,  
  time = NA,  
  deltaG_Cutoff = NA,  
  deltaG_Limit = NA  
)
```

**Arguments**

ILP	A solved ILP instance.
vars	The ILP decision variables.
primer.df	The primer data frame corresponding to the ILP.
template.df	The template data frame.
i	Index for the ILP.
target.temp	Target melting temperature in Celsius.
time	Runtime of the ILP.
deltaG_Cutoff	Free energy cutoff used for the dimerization constraint.
deltaG_Limit	The free energy boundary for dimerization.

**Value**

Data frame summarizing the ILP solution.

---

build.tool.overview    *Creation of an Overview of Third-Party Tools.*

---

**Description**

Creates a table of required third-party tools and their installation status.

**Usage**

```
build.tool.overview(AVAILABLE.TOOLS, for.shiny = FALSE)
```

**Arguments**

AVAILABLE.TOOLS	A vector whose names give the required tools and whose entries give their installation status as logicals.
If	for.shiny is TRUE, provide the URLs for the tool using HTML.

**Value**

A data frame with information on third-part tools.

---

build_leader_df	<i>Building of Leader Data Frame.</i>
-----------------	---------------------------------------

---

**Description**

Constructs the leader data frame.

**Usage**

```
build_leader_df(  
  direction = c("fw", "rev"),  
  leader,  
  start,  
  end,  
  ali.start,  
  ali.end  
)
```

**Arguments**

direction	The primer direction for which we are annotating binding regions.
leader	The binding region sequence.
start	The start positions of the binding region.
end	The end positions of the binding region.
ali.start	The start positions of the binding region in the aligned input.
ali.end	The end positions of the binding region in the aligned input.

**Value**

A data frame with binding region information.

---

call.melt	<i>Thermodynamic melting temperature computations.</i>
-----------	--

---

**Description**

Computes the melting temperature for the input primers.

**Usage**

```
call.melt(  
  primers,  
  complements,  
  primer_conc,  
  na_salt_conc,  
  mg_salt_conc,  
  k_salt_conc,  
  tris_salt_conc  
)
```

**Arguments**

primers	Character vector of primer strings.
complements	Character vector with complement sequences corresponding to primers.
primer_conc	Primer concentration.
na_salt_conc	Sodium ion concentration.
mg_salt_conc	Magnesium ion concentration.
k_salt_conc	Potassium ion concentration.
tris_salt_conc	Tris buffer concentration.

**Value**

Melting temperature data frame.

**References**

Le Novère N. (2001). MELTING, computing the melting temperature of nucleic acid duplex. *Bioinformatics*, 17: 1226-1227. Dumousseau M., Rodriguez N., Juty N., Le Novère N. (2012) MELTING, a flexible platform to predict the melting temperatures of nucleic acids. *BMC Bioinformatics*, 13: 101.

---

call.melt.single      *Thermodynamic melting temperature computations.*

---

**Description**

Computes the melting temperature for the input primers.

**Usage**

```
call.melt.single(
  primers,
  complements,
  out.file,
  primer_conc,
  na_salt_conc,
  mg_salt_conc,
  k_salt_conc,
  tris_salt_conc,
  ID
)
```

**Arguments**

primers	List of primer strings.
complements	List with corresponding complements.
out.file	Path to the file where MELTING will write the results.
primer_conc	Primer concentration.
na_salt_conc	Sodium ion concentration.

mg_salt_conc	Magnesium ion concentration.
k_salt_conc	Potassium ion concentration.
tris_salt_conc	Tris buffer concentration.
ID	identifiers of the input primers

**Value**

Melting temperature data frame.

---

cascaded.filter      *Filtering for the Optimization*

---

**Description**

Filter primers according to constraints and relax constraints if necessary.

**Usage**

```
cascaded.filter(
  primer.df,
  template.df,
  settings,
  mode.directionality = c("fw", "rev", "both"),
  required.cvg = 1,
  target.temps = NULL,
  updateProgress = NULL,
  results.loc = NULL
)
```

**Arguments**

primer.df	Primer data frame.
template.df	Template data frame.
settings	Settings object.
mode.directionality	Primer direction.
required.cvg	Required ratio of covered templates. If required.cvg is set to 0, the constraints are not relaxed.
target.temps	Target melting temperature of the primers in Celsius. This argument is only required if we try to match the melting temperatures of another primer set, e.g. when first optimizing forward and then optimizing reverse primers.
updateProgress	Progress callback function for shiny.
results.loc	Directory where the filtering results should be stored.

**Details**

Constraints are relaxed if the required.cvg could not be reached with the input constraints.

**Value**

The filtered primer data frame with respect to required.cvg.

---

cascaded.filter.quick *Cascaded Filter*

---

### Description

Filter primers in a cascaded fashion.

### Usage

```
cascaded.filter.quick(  
  primer.df,  
  template.df,  
  settings,  
  to.compute.constraints,  
  mode.directionality = c("fw", "rev", "both"),  
  active.constraints = NULL,  
  no.structures = FALSE,  
  updateProgress = NULL  
)
```

### Arguments

primer.df	Primer data frame.
template.df	Template data frame.
settings	Settings object.
to.compute.constraints	Names of constraints that still have to be computed.
mode.directionality	Primer direction.
active.constraints	The constraints that are to be used for filtering. If active.constraints is NULL, all filtering constraints are used.
no.structures	Whether dimerization structures shall be computed.
updateProgress	Progress callback function for shiny.

### Details

At each constraint evaluation all primers that do not fulfill the current constraint are removed. Constraints that are specified in to.compute.constraints are computed on the fly.

### Value

The filtered primer data frame.

---

cbind.Primers	<i>cbind for Primers class.</i>
---------------	---------------------------------

---

**Description**

Ensures that the cbind result has the appropriate class.

**Usage**

```
## S3 method for class 'Primers'  
cbind(...)
```

**Arguments**

... Parameters for cbind function.

**Value**

Column binded Primers data frame.

**Examples**

```
data(Ippolito)  
primer.df <- cbind(primer.df, primer.df)
```

---

cbind.Templates	<i>cbind for Template class.</i>
-----------------	----------------------------------

---

**Description**

Ensures that the cbind result has the appropriate class.

**Usage**

```
## S3 method for class 'Templates'  
cbind(...)
```

**Arguments**

... Parameters for cbind function.

**Value**

Column binded Templates data frame.

**Examples**

```
data(Ippolito)  
template.df <- cbind(template.df, seq_len(nrow(template.df)))
```

---

cbind2,Primers,ANY-method

*S4 cbind for Primers.*

---

### Description

S4 cbind function for Primers.

S4 rbind function for Primers.

Slices a Primers data frame.

Stores data in a column of a Primers data frame.

### Usage

```
## S4 method for signature 'Primers,ANY'  
cbind2(x, y, ...)
```

```
## S4 method for signature 'Primers,ANY'  
rbind2(x, y, ...)
```

```
## S4 method for signature 'Primers,ANY'  
x[i, j, ..., drop = TRUE]
```

```
## S4 replacement method for signature 'Primers'  
x$name <- value
```

### Arguments

x	The Primers data frame.
y	Another data frame.
...	Other arguments to the slice operator.
i	The row index.
j	The column index.
drop	Simplify data frame?
name	The name of the column.
value	The values of the column.

### Value

Cbinded primer data frame.

Rbinded primer data frame.

Subset of primer data frame.

Primer data frame with replaced column.

**Examples**

```

data(Ippolito)
primer.df <- cbind2(primer.df, seq_len(nrow(primer.df)))
data(Ippolito)
primer.df <- primer.df[1:2,]
data(Ippolito)
primer.df$Forward[1] <- "ctagcgggaccg"

```

---

cbind2, Templates, ANY-method  
*S4 cbind for Templates.*

---

**Description**

S4 cbind function for Templates data frame.

S4 rbind function for templates.

Slicing of Templates data frame object.

Set a column in a Templates data frame.

**Usage**

```

## S4 method for signature 'Templates,ANY'
cbind2(x, y, ...)

## S4 method for signature 'Templates,ANY'
rbind2(x, y, ...)

## S4 method for signature 'Templates,ANY'
x[i, j, ..., drop = TRUE]

## S4 replacement method for signature 'Templates'
x$name <- value

```

**Arguments**

x	The Template data frame.
y	Another data frame.
...	Other arguments to the slice operator.
i	The row index.
j	The column index.
drop	Simplify data frame?
name	The name of the column.
value	The values of the column.

**Value**

Cbinded template data frame.

Rbinded template data frame.

Subsetted template data frame.

Templates with replaced column.

**Examples**

```
data(Ippolito)
template.df <- cbind2(template.df, seq_len(nrow(template.df)))
data(Ippolito)
template.df <- rbind2(template.df, template.df)
data(Ippolito)
template.df <- template.df[1:2,]
data(Ippolito)
template.df$ID[1] <- "newID"
```

---

check.3prime.hexamers *3' Hexamer Check.*

---

**Description**

Check whether the 3' hexamer of a primer is fully complementary to the corresponding region in the template.

**Usage**

```
check.3prime.hexamers(
  template.df,
  primer.df,
  mode.directionality = c("fw", "rev", "both")
)
```

**Arguments**

template.df      Template data frame.  
primer.df        Primer data frame.  
mode.directionality  
                  Primer directionality.

**Value**

Returns TRUE if the 3' hexamer of a primer is fully complementary to the corresponding template region and FALSE otherwise.

---

check.3prime.mismatches  
*3' Mismatch Check.*

---

**Description**

Check for mismatches at primer 3' ends.

**Usage**

```
check.3prime.mismatches(  
  template.df,  
  primer.df,  
  mode.directionality = c("fw", "rev", "both")  
)
```

**Arguments**

template.df      Template data frame.  
primer.df        Primer data frame.  
mode.directionality  
                  Primer directionality.

**Value**

Returns the distance of mismatches from the 3' terminal end of primers.

---

check.init.primer.length

*Primer Length Check.*

---

**Description**

Checks whether it is possible to construct primers of the desired length.

**Usage**

```
check.init.primer.length(  
  template.df,  
  allowed.region.definition = c("within", "any"),  
  primer.lengths,  
  mode.directionality = c("fw", "rev", "both")  
)
```

**Arguments**

template.df      Template data frame.  
allowed.region.definition  
                  Definition of allowed binding regions.  
primer.lengths    The desired lengths of the primers.  
mode.directionality  
                  The primer directionality.

**Value**

TRUE, if primers of the desired length can be constructed,

check.init.primer.length.single  
*Primer Length Check.*

---

**Description**

Checks whether it is possible to construct primers of the desired length.

**Usage**

```
check.init.primer.length.single(  
  allowed,  
  allowed.region.definition = c("within", "any"),  
  min.len  
)
```

**Arguments**

allowed	String containing the allowed binding sequence.
allowed.region.definition	Definition of allowed binding regions.
min.len	Minimal desired primer lengths.

**Value**

TRUE if primers of the desired length can be constructed, FALSE otherwise.

---

check.mutations      *Identification of Mismatch Mutations.*

---

**Description**

Identifies primers that induce mutations due to mismatch binding.

**Usage**

```
check.mutations(  
  primer.seq,  
  pos.start,  
  pos.end,  
  template.df,  
  covered.seqs,  
  ORF.data,  
  mode.directionality = c("fw", "rev"),  
  mutation.types = c("stop_codon", "substitution")  
)
```

**Arguments**

primer.seq	Primer sequence string.
pos.start	Binding position of primer (start).
pos.end	Binding position of primer (end).
template.df	Template data frame.
covered.seqs	Identifiers of covered templates.
ORF.data	Reading frame information of templates.
mode.directionality	Directionality of primers.
mutation.types	Character vector of the mutation types to be checked for.

**Details**

Checks for one primer and all covered templates whether any templates are bound with mismatches such that a forbidden mutation is induced. A boolean vector indicating which binding events induce a forbidden mutation is returned.

**Value**

TRUE if the primer.seq induces a mutation that is forbidden according to the provided mutation.types.

---

check.template.constraints

*Check Constraints on Templates*

---

**Description**

Transforms the comma-separated input strings to a boolean representation.

**Usage**

```
check.template.constraints(template.constraints)
```

**Arguments**

template.constraints	Strings with comma-separated values to be turned to logical.
----------------------	--

**Value**

List with boolean values

---

check.tool.function    *Check Functionality of Third-Party Tools.*

---

**Description**

Checks whether all required tools should work.

**Usage**

```
check.tool.function(frontend = FALSE)
```

**Arguments**

frontend            Whether tool functionality shall be checked for the frontend.

**Value**

TRUE for each functioning tool, FALSE for non-functioning tools.

---

check.tool.installation  
                          *Check Tool Installation*

---

**Description**

Checks whether all required tools are installed.

**Usage**

```
check.tool.installation(frontend = FALSE)
```

**Arguments**

frontend            Whether tool installation shall be checked for the frontend. If TRUE, dependencies that are required only by the frontend are considered additionally.

**Value**

TRUE for each installed tool, FALSE otherwise.

---

`check_constraints_comparison`*Batch Procedure for Evaluating Primer Sets.*

---

**Description**

Batch Procedure for Evaluating Primer Sets.

**Usage**

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

**Arguments**

`primer.data` A list of objects of class `Primers`.

`template.data` A list of objects of class `Templates` corresponding to `primer.data`.

`settings` An object of class `DesignSettings`.

`active.constraints`  
A character vector providing identifiers of constraints to be considered.

`to.compute.constraints`  
A character vector providing identifiers of constraints to be computed.

`for.shiny` A logical indicating whether the results are indicated for the Shiny app or not.

`updateProgress` A callback function to track progress in the Shiny app.

**Value**

A list with objects of class `Primers`.

**Examples**

```
## Not run:  
data(Comparison)  
eval.data <- check_constraints_comparison(primer.data[1:2], template.data[1:2], settings)  
  
## End(Not run)
```

---

check\_constraint\_settings\_validity

*Check the Validity of the Constraint Settings.*

---

**Description**

Checks whether the status and the active constraints match. Determines whether the constraints are allowed/known.

**Usage**

check\_constraint\_settings\_validity(object)

**Arguments**

object            An AbstractConstraintSettings object.

**Value**

TRUE if the constraint settings are valid, FALSE otherwise.

---

check\_correspondence    *Check of Primer and Template Correspondence.*

---

**Description**

Checks whether the primers relate to the correct templates.

**Usage**

check\_correspondence(primer.df, template.df)

**Arguments**

primer.df        An object of class Primers.

template.df     An object of class Templates.

**Value**

TRUE if the primers and templates seem to correspond, FALSE otherwise.

---

check\_cvf\_constraints *Evaluation of Coverage Constraints.*

---

### Description

Computes the biochemical properties specified in the settings object and determines whether the primers fulfill the required constraints.

### Usage

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

### Arguments

primer.df	A Primers object containing the primers to be checked.
template.df	A Templates object containing the template sequences corresponding to the primers.
settings	A DesignSettings object containing the coverage constraints to be checked and their settings.
active.constraints	Identifiers of constraints that are to be checked.
to.compute.constraints	Constraints that are to be computed.
for.shiny	Whether to format output for HTML.
updateProgress	Progress callback function for shiny.

### Value

A Primers object with with columns for each constraint in active.constraints.

### Note

Please note that some constraints can only be computed if additional software is installed, please see [DesignSettings](#) for an overview.

---

check_interval	<i>Check Constraint Intervals</i>
----------------	-----------------------------------

---

**Description**

Checks the validity of constraint intervals.

**Usage**

```
check_interval(constraints)
```

**Arguments**

constraints     A list with constraint settings.

**Value**

TRUE, if all constraints specify valid intervals, FALSE otherwise.

---

check_limits	<i>Validity Check for Limits.</i>
--------------	-----------------------------------

---

**Description**

Checks whether the constraint limits are at least as general as the constraint settings. This ensures that the relaxation works in the proper direction.

**Usage**

```
check_limits(constraint.settings, constraint.limits)
```

**Arguments**

constraint.settings  
                                  A list with the constraint settings.

constraint.limits  
                                  A list with the constraint relaxation limits.

**Value**

TRUE if the limits are at least as wide as the constraints, FALSE otherwise.

---

check_limit_value	<i>Check of limit correctness.</i>
-------------------	------------------------------------

---

**Description**

Checks whether a constraint limit is more general than the setting.

**Usage**

```
check_limit_value(setting, limit)
```

**Arguments**

setting	A single constraint setting.
limit	A single constraint limit.

**Value**

A vector containing TRUE if the limit is more general than the constraint setting and FALSE otherwise.

---

check_names	<i>Check Setting Names.</i>
-------------	-----------------------------

---

**Description**

Checks whether the specified settings have the correct names.

**Usage**

```
check_names(known.options, input.options)
```

**Arguments**

known.options	Allowed setting names.
input.options	Input setting names

**Value**

Mapping of input.options to known.options or NULL if invalid.

---

check\_report\_deps      *Check for Report Dependencies.*

---

### Description

Checks whether the dependencies for `rmarkdown::render()` are fulfilled.

### Usage

```
check_report_deps()
```

### Value

A logical vector giving the dependency availability status.

---

check\_restriction\_sites\_single  
*Identification of Sequence Restriction Sites.*

---

### Description

Checks the input sequences `seqs` for the presence of restriction sites. By removing the restriction sites from a primer set, it is possible to identify the coverage of the primers (e.g. using [check\\_constraints](#)) discounting for the impact of the mismatching bases caused by the insert.

### Usage

```
check_restriction_sites_single(
  primer.seqs,
  template.seqs,
  adapter.action,
  direction = c("fw", "rev"),
  selected = NULL,
  only.confident.calls = TRUE,
  updateProgress = NULL
)
```

### Arguments

<code>primer.seqs</code>	Nucleotide sequences of primers to be checked for restriction sites in terms of a <code>DNAStrngSet</code> object.
<code>template.seqs</code>	A <code>DNAStrngSet</code> object with nucleotide sequences containing the templates corresponding to <code>seqs</code> .
<code>adapter.action</code>	The action to be performed when adapter sequences are found. Either "warn" to issue warning about adapter sequences or "rm" to remove identified adapter sequences.
<code>selected</code>	Names of restriction sites that are to be checked. By default <code>selected</code> is <code>NULL</code> in which case all REBASE restriction sites are checked.

only.confident.calls Only output confident calls of restriction sites.  
 updateProgress A Shiny progress callback function.  
 The primer direction that is checked.

**Value**

A data frame with restriction sites, if any could be found.

**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>

---

check_setting	<i>Check Setting Validity.</i>
---------------	--------------------------------

---

**Description**

Checks whether the input settings are valid or not.

**Usage**

```
check_setting(known.options, options, mandatory.options = NULL)
```

**Arguments**

known.options Vector with names and classes of allowed options.  
 options Active options to be checked.  
 mandatory.options Fields that have to be present.

**Value**

TRUE if the setting is valid, FALSE otherwise.

---

check_settings_validity	<i>Validity Check for DesignSettings.</i>
-------------------------	---

---

**Description**

Validates whether a DesignSettings object has the correct structure.

**Usage**

```
check_settings_validity(object)
```

**Arguments**

object            A DesignSettings object to be checked for validity.

**Value**

TRUE if object is valid, FALSE otherwise.

---

combine.binding.events

*Combination of Binding Events.*

---

**Description**

Appends all binding events.

**Usage**

```
combine.binding.events(my.binding.fw, my.binding.rev, fw.m, rev.m)
```

**Arguments**

my.binding.fw    Forward binding events of individual primers.

my.binding.rev   Reverse binding events of individual primers.

fw.m             Forward binding events of paired primers.

rev.m            Reverse binding events of paired primers.

**Value**

IRanges of all binding events.

---

combine.strings

*Combination of OligoArrayAux Structure Sequences.*

---

**Description**

Combines the input strings.

**Usage**

```
combine.strings(s1, s2)
```

**Arguments**

s1                A character vector to be combined with s2.

s2                A character vector to be included into s1.

**Value**

A character vector.

---

comp	<i>Sequence complement</i>
------	----------------------------

---

**Description**

Complements the input sequence (re-write of seqinr comp function for gap support)

**Usage**

```
comp(seq, forceToLower = TRUE, ambiguous = FALSE)
```

**Arguments**

seq	Input char vector.
forceToLower	if TRUE the input is transformed to lower case.
ambiguous	if TRUE ambiguous IUPAC nucleotides are complemented.

**Value**

The complement of seq.

---

compare.constraints	<i>Constraint list comparison</i>
---------------------	-----------------------------------

---

**Description**

Determines whether two list with constraints are identical.

**Usage**

```
compare.constraints(A, B)
```

**Arguments**

A	First constraint list.
B	Second constraint list.

**Value**

TRUE if the constraints are identical, FALSE else.

comparison.cvg      *Comparison Coverage Stats.*

---

**Description**

Computes coverage stats for primer comparison.

**Usage**

```
comparison.cvg(primer.data, template.data)
```

**Arguments**

primer.data      List with primer data frames.  
template.data    List with template data frames.

**Value**

Coverage statistics for comparing primers.

---

comparison.stats.raw    *Computation of Raw Stats for Primer Comparison*

---

**Description**

Computes raw stats for primer comparison.

**Usage**

```
comparison.stats.raw(primer.data, template.data)
```

**Arguments**

primer.data      List with primer data frames.  
template.data    List with template data frames.

**Value**

Raw statistics for primer comparison.

---

complement.sequence     *Sequence complement*

---

**Description**

Computes the complement of the input sequence.

**Usage**

```
complement.sequence(seq)
```

**Arguments**

seq                    The input sequence strings.

**Value**

The complements of the input sequences.

---

compute.all.cross.dimers  
                          *Cross dimerization*

---

**Description**

Compute worst-case DeltaG data frame with all possible primer cross-dimers.

**Usage**

```
compute.all.cross.dimers(  
  primer.df,  
  primer_conc,  
  na_salt_conc,  
  mg_salt_conc,  
  k_salt_conc,  
  tris_salt_conc,  
  annealing.temp,  
  results = NULL,  
  check.idx = NULL,  
  for.shiny = FALSE,  
  no.structures = FALSE  
)
```

**Arguments**

primer.df	Input primers data frame.
primer_conc	Primer concentration.
na_salt_conc	Sodium ion concentration.
mg_salt_conc	Magnesium ion concentration.
k_salt_conc	Potassium ion concentration.
tris_salt_conc	Tris buffer concentration.
annealing.temp	The PCR annealing temperature.
results	(optional) Cross dimer data frame (unfiltered)
check.idx	Indices of primers for checking cross-dimerization.
for.shiny	Whether the table is intended for HTML display.
no.structures	Whether dimerization structures shall not be outputted.

**Value**

Worst-case cross dimers.

---

compute.all.cross.dimers.frontend  
*Cross Dimerization.*

---

**Description**

Computes all cross dimers in a user-formatted way.

**Usage**

```
compute.all.cross.dimers.frontend(
  primer.df,
  primer_conc,
  na_salt_conc,
  mg_salt_conc,
  k_salt_conc,
  tris_salt_conc,
  annealing.temp,
  for.shiny = FALSE,
  no.structures = FALSE
)
```

**Arguments**

primer.df	Input primer data frame.
primer_conc	Primer concentration.
na_salt_conc	Sodium ion concentration.
mg_salt_conc	Magnesium ion concentration.
k_salt_conc	Potassium ion concentration.

tris\_salt\_conc Tris buffer concentration.  
 annealing.temp The PCR annealing temperature.  
 for.shiny Whether to format the table for HTML output.  
 no\_structures Whether to compute structures of dimers.

**Value**

A formatted data frame with cross-dimerization infos

---

compute.all.cross.dimers.unfiltered  
*Cross dimerization*

---

**Description**

Compute DeltaG data frame for possible primer cross-dimers.

**Usage**

```

compute.all.cross.dimers.unfiltered(
  primer.df,
  primer_conc,
  na_salt_conc,
  mg_salt_conc,
  k_salt_conc,
  tris_salt_conc,
  annealing.temp,
  check.idx = NULL,
  for.shiny = FALSE,
  no_structures = FALSE
)

```

**Arguments**

primer.df Input primers data frame.  
 primer\_conc Primer concentration.  
 na\_salt\_conc Sodium ion concentration.  
 mg\_salt\_conc Magnesium ion concentration.  
 k\_salt\_conc Potassium ion concentration.  
 tris\_salt\_conc Tris buffer concentration.  
 annealing.temp The PCR annealing temperature.  
 check.idx Indices of primers for checking cross-dimerization.  
 for.shiny Whether to format for HTML output.  
 no\_structures Whether dimer structures shall not be determined. If TRUE, structures are not computed resulting in faster runtimes.

**Value**

All cross dimers.

```
compute.all.primer.subsets.ILP
```

*Computation of Primer Subsets*

---

### Description

Computes all optimal primer subsets and stores their plots.

### Usage

```
compute.all.primer.subsets.ILP(  
  primer.df,  
  template.df,  
  k,  
  groups,  
  cur.results.loc,  
  required.cvg = 1  
)
```

### Arguments

primer.df	Primer data frame.
template.df	Template data frame.
k	Subset size-increment.
groups	Identifiers of template groups in order to limit coverage to certain groups of template sequences.
cur.results.loc	Location for storing the results.
required.cvg	The required coverage ratio.

### Value

Write-out of results.

---

```
compute.all.self.dimers
```

*Self dimers*

---

### Description

Computes all possible self dimers for the primers in the input data frame.

**Usage**

```
compute.all.self.dimers(  
  primer.df,  
  primer_conc,  
  na_salt_conc,  
  mg_salt_conc,  
  k_salt_conc,  
  tris_salt_conc,  
  annealing.temp,  
  for.shiny = FALSE,  
  no.structures = FALSE  
)
```

**Arguments**

primer.df	Input primer data frame.
primer_conc	Primer concentration.
na_salt_conc	Sodium ion concentration.
mg_salt_conc	Magnesium ion concentration.
k_salt_conc	Potassium ion concentration.
tris_salt_conc	Tris buffer concentration.
annealing.temp	The PCR annealing temperature.
for.shiny	Whether the output is to be formatted for HTML.
no.structures	Whether dimerization structures shall be outputted.

**Value**

Data frame with thermodynamic information on all self dimers.

---

compute.all.self.dimers.frontend  
*Self Dimerization.*

---

**Description**

Computes all self dimers in a user-formatted way.

**Usage**

```
compute.all.self.dimers.frontend(  
  primer.df,  
  primer_conc,  
  na_salt_conc,  
  mg_salt_conc,  
  k_salt_conc,  
  tris_salt_conc,  
  annealing.temp,  
  for.shiny = FALSE,  
  no.structures = FALSE  
)
```

**Arguments**

primer.df	Input primer data frame.
primer_conc	Primer concentration.
na_salt_conc	Sodium ion concentration.
mg_salt_conc	Magnesium ion concentration.
k_salt_conc	Potassium ion concentration.
tris_salt_conc	Tris buffer concentration.
annealing.temp	The PCR annealing temperature.
for.shiny	Whether to format the table for HTML output.
no.structures	Whether dimerization structures shall be outputted.

**Value**

A formatted data frame with self-dimerization infos

---

compute.basic.details *Computation of Coverage Details*

---

**Description**

Determines binding properties of primers.

**Usage**

```
compute.basic.details(
  binding,
  mode = c("on_target", "off_target"),
  template.df,
  primers,
  mode.directionality = c("fw", "rev", "both"),
  allowed.mismatches,
  allowed.other.binding.ratio,
  allowed.region.definition = c("within", "any"),
  updateProgress = NULL
)
```

**Arguments**

binding	An IRanges object with primer binding information.
mode	Either on_target for on-target binding or off_target for off-target binding.
template.df	Template data frame.
primers	Primer data frame.
mode.directionality	Primer directionality.
allowed.mismatches	The number of allowed mismatches per binding event.

- `allowed.other.binding.ratio`  
Ratio of primers that are allowed to bind to non-allowed regions. If `allowed.other.binding.ratio` >0 primers are allowed to bind at any location within the templates. However, a warning is given if the ratio of primers binding to non-target regions exceeds the `allowed.other.binding.ratio`.
- `allowed.region.definition`  
Definition of the target binding sites used for evaluating the coverage. If `allowed.region.definition` is within, primers have to lie within the allowed binding region. If `allowed.region.definition` is any, primers have to overlap with the allowed binding region. The default is that primers have to bind within the target binding region.
- `updateProgress` Progress callback function for shiny.

**Value**

Primer data frame with information on the covered template sequences.

---

`compute.constraints`    *Computation of Constraints.*

---

**Description**

Computes the specified constraints for the input primers.

**Usage**

```
compute.constraints(
  primer.df,
  mode.directionality = c("fw", "rev", "both"),
  template.df,
  settings,
  active.constraints = c("primer_coverage", "primer_length", "primer_specificity",
    "gc_clamp", "gc_ratio", "no_runs", "no_repeats", "self_dimerization",
    "cross_dimerization", "melting_temp_range", "melting_temp_diff",
    "secondary_structure", "primer_efficiency", "annealing_DeltaG", "stop_codon",
    "terminal_mismatch_pos", "substitution", "hexamer_coverage", "coverage_model",
    "off_primer_efficiency", "off_annealing_DeltaG", "off_coverage_model"),
  no_structures = FALSE,
  for.shiny = FALSE,
  updateProgress = NULL
)
```

**Arguments**

- `primer.df`            Primer data frame.
- `mode.directionality`  
Primer directionality.
- `template.df`        Template data frame.
- `settings`            A `DesignSettings` object.
- `active.constraints`  
Strings giving the constraints that are to be computed.

no.structures Whether dimer structures shall be computed.  
 for.shiny Whether to format output for HTML.  
 updateProgress Progress callback function for shiny.

### Value

A data frame with columns for each constraint in active.constraints.

---

compute.covered.Ta *Annealing temperature*

---

### Description

Computes the annealing temperature using all binding events.

### Usage

```
compute.covered.Ta(
  primer.df,
  mode.directionality = c("fw", "rev", "both"),
  template.df,
  na_salt_conc,
  mg_salt_conc,
  k_salt_conc,
  tris_salt_conc,
  primer_conc
)
```

### Arguments

primer.df Primer data frame.  
 mode.directionality Primer directionality.  
 template.df Template data frame  
 na\_salt\_conc Sodium ion concentration.  
 mg\_salt\_conc Magnesium ion concentration.  
 k\_salt\_conc Potassium ion concentration.  
 tris\_salt\_conc Tris buffer concentration.  
 primer\_conc Primer concentration.

### Value

The recommended annealing temperature.

---

compute.dimer.matrix    *Dimerization matrix*

---

### Description

Computes a matrix indicating all dimerizing primers according to a DeltaG cutoff.

### Usage

```
compute.dimer.matrix(G, deltaG.cutoff = -7)
```

### Arguments

G                    Matrix with free energies of all considered primer pairs.  
deltaG.cutoff    Primers with free energies below the cutoff are considered dimerizing.

### Value

Binary matrix with dimerization events according to the deltaG.cutoff. Contains '1' if primers (i,j) dimerize and '0' else.

---

compute. efficiency    *Primer Efficiency.*

---

### Description

Computes the efficiency of primer binding events for Taq polymerase.

### Usage

```
compute. efficiency(  
  fw.primers,  
  fw.start,  
  fw.end,  
  covered,  
  taqEfficiency,  
  annealing.temp,  
  primer_conc,  
  sodium.eq.concentration,  
  mode.directionality = c("fw", "rev"),  
  seqs  
)
```

**Arguments**

fw.primers	Primer sequence strings.
fw.start	Binding position (start).
fw.end	Binding position (end).
covered	List of covered template indices per primer.
taqEfficiency	Whether the efficiency shall be computed using a mismatch-model developed for Taq polymerases. The default setting is TRUE. Set taqEfficiency to FALSE if you are using another polymerase than Taq.
annealing.temp	Annealing temperature for which to evaluate efficiency.
primer_conc	Primer concentration.
sodium.eq.concentration	The sodium-equivalent concentration of ions.
mode.directionality	Primer directionality.
seqs	Template sequence strings.

**Details**

This function uses DECIPHER's [CalculateEfficiencyPCR](#).

**Value**

The efficiencies of primer binding events.

**References**

Wright, Erik S., et al. "Exploiting extension bias in polymerase chain reaction to improve primer specificity in ensembles of nearly identical DNA templates." *Environmental microbiology* 16.5 (2014): 1354-1365.

---

compute.empiric.melting.temp

*Non-Thermodynamic Computation of Melting Temperatures.*

---

**Description**

Computes the melting temperature of primers from an empiric formula.

**Usage**

```
compute.empiric.melting.temp(primer.df)
```

**Arguments**

primer.df      A Primers object.

**Value**

A data frame with melting temperature information for the primers.

---

compute.gc.ratio	<i>GC ratio</i>
------------------	-----------------

---

**Description**

Computes the ratio of G/Cs in a sequence.

**Usage**

```
compute.gc.ratio(x)
```

**Arguments**

x	Input sequence.
---	-----------------

**Details**

In case of ambiguities, the mean GC ratio of all possible sequences is computed.

**Value**

The fraction of G/Cs in x.

---

compute.melting.temps	<i>Computation of Melting Temperatures.</i>
-----------------------	---

---

**Description**

Use nearest-neighbor thermodynamic computations to find melting temperatures.

**Usage**

```
compute.melting.temps(  
  primer.df,  
  primer_conc,  
  na_salt_conc,  
  mg_salt_conc,  
  k_salt_conc,  
  tris_salt_conc,  
  mode.directionality = c("fw", "rev", "both")  
)
```

**Arguments**

primer.df	Primer data frame.
primer_conc	Primer concentration.
na_salt_conc	Sodium ion concentration.
mg_salt_conc	Magnesium ion concentration.
k_salt_conc	Potassium ion concentration.

tris\_salt\_conc Tris buffer concentration.  
 mode.directionality  
                   Direction of primers

### Value

Data frame with melting temperature info for the input primers.

---

compute.melting.temps.thermo  
                                   *Computation of Thermodynamic Melting Temperatures.*

---

### Description

Use nearest-neighbor thermodynamic computations to find melting temperatures.

### Usage

```

compute.melting.temps.thermo(
  primer.df,
  primer_conc,
  na_salt_conc,
  mg_salt_conc,
  k_salt_conc,
  tris_salt_conc,
  mode.directionality = c("fw", "rev", "both")
)

```

### Arguments

primer.df       Primer data frame.  
 primer\_conc     Primer concentration.  
 na\_salt\_conc    Sodium ion concentration.  
 mg\_salt\_conc    Magnesium ion concentration.  
 k\_salt\_conc     Potassium ion concentration.  
 tris\_salt\_conc Tris buffer concentration.  
 mode.directionality  
                   Direction of primers

### Value

Data frame with melting temperature info for the input primers.

---

`compute.mismatch.table`*Mismatch overview table*

---

**Description**

Computes a table summarizing all of the mismatches caused by the primers in the input data frame.

**Usage**

```
compute.mismatch.table(  
  primer.df,  
  template.df,  
  mode.directionality = c("fw", "rev")  
)
```

**Arguments**

`primer.df`        Primer data frame.  
`template.df`     Template data.  
`mode.directionality`  
                  Direction of primers.

**Value**

: A data frame summarizing all mismatches of the input primers with the input templates.

---

`compute.primer efficiencies`*Primer Efficiency.*

---

**Description**

Computes the efficiency of primer binding events for Taq polymerase.

**Usage**

```
compute.primer efficiencies(  
  primer.df,  
  template.df,  
  annealing.temp,  
  taqEfficiency,  
  primer_conc,  
  na_salt_conc,  
  mg_salt_conc,  
  k_salt_conc,  
  tris_salt_conc,  
  mode = c("on_target", "off_target")  
)
```

**Arguments**

primer.df	Primer data frame.
template.df	Template data frame.
annealing.temp	Annealing temperature for which to evaluate efficiency.
taqEfficiency	Whether the efficiency shall be computed using a mismatch-model developed for Taq polymerases. The default setting is TRUE. Set taqEfficiency to FALSE if you are using another polymerase than Taq.
primer_conc	Primer concentration.
na_salt_conc	Sodium ion concentration.
mg_salt_conc	Magnesium ion concentration.
k_salt_conc	Potassium ion concentration.
tris_salt_conc	Tris ion concentration.
mode	Compute efficiencies for on-target coverage events (on_target) or off-target coverage events (off_target).

**Details**

This function uses DECIPHER's [CalculateEfficiencyPCR](#).

**Value**

A list with the efficiency of every primer binding event.

**Examples**

```
data(Ippolito)
p <- PCR(settings)
# Requires OligoArrayAux software:
## Not run:
eff.df <- compute.primerefficiencies(primer.df, template.df, 55,
  p$primer_concentration, p$Na_concentration,
  p$Mg_concentration, p$K_concentration, p$Tris_concentration)

## End(Not run)
```

---

```
compute.secondary.structures
```

*Secondary Structure Computations.*

---

**Description**

Computes the secondary structures of the input primers using ViennaRNA.

**Usage**

```
compute.secondary.structures(
  primer.df,
  mode.directionality = c("fw", "rev", "both"),
  annealing.temperature
)
```

**Arguments**

primer.df          Primer data frame.  
mode.directionality          Direction of primers.  
annealing.temperature          Temperatures at which to compute secondary structures for every primer

**Value**

Data frame with secondary structure information.

**References**

Lorenz, Ronny and Bernhart, Stephan H. and Höner zu Siederdisen, Christian and Tafer, Hakim and Flamm, Christoph and Stadler, Peter F. and Hofacker, Ivo L. ViennaRNA Package 2.0 Algorithms for Molecular Biology, 6:1 26, 2011, doi:10.1186/1748-7188-6-26

---

compute.sodium.equivalent.conc

*Sodium-equivalent Concentration*

---

**Description**

Computes the sodium-equivalent concentration for the input ion concentrations.

**Usage**

```
compute.sodium.equivalent.conc(  
  na_salt_conc,  
  mg_salt_conc,  
  k_salt_conc,  
  tris_salt_conc  
)
```

**Arguments**

na\_salt\_conc          Sodium ion concentration.  
mg\_salt\_conc          Magnesium ion concentration.  
k\_salt\_conc          Potassium ion concentration.  
tris\_salt\_conc          Tris buffer concentration.

**Value**

The sodium-equivalent concentration of the input ion concentrations.

## References

Record, M. Thomas. "Effects of Na<sup>+</sup> and Mg<sup>++</sup> ions on the helix–coil transition of DNA." *Biopolymers* 14.10 (1975): 2137-2158.

Owczarzy, Richard, et al. "Predicting stability of DNA duplexes in solutions containing magnesium and monovalent cations." *Biochemistry* 47.19 (2008): 5336-5353.

Peyret, Nicolas. *Prediction of nucleic acid hybridization: parameters and algorithms*. Detroit: Wayne State University, 2000.

---

compute.structure.vienna

*Computation of Secondary Structures with ViennaRNA.*

---

## Description

Computes secondary structures using ViennaRNA.

## Usage

```
compute.structure.vienna(  
  seqs,  
  annealing.temperature,  
  folding.constraints = NULL,  
  id = ""  
)
```

## Arguments

seqs	The input sequences for which structures shall be computed.
annealing.temperature	The temperature in degree Celsius at which to compute secondary structures.
folding.constraints	Character vector specifying the folding conditions for every input sequence. For example the constraint xxxxxx... would forbid folding in the first 6 bases and allow folding in the last 3 bases.
id	An identifier for storing the files
constraints	If provide

## Value

A data frame with secondary structures.

---

compute.Ta	<i>Annealing temperature</i>
------------	------------------------------

---

### Description

Computes the annealing temperature using all binding events.

### Usage

```
compute.Ta(  
  primer.df,  
  template.df,  
  mode.directionality = c("fw", "rev", "both"),  
  na_salt_conc,  
  mg_salt_conc,  
  k_salt_conc,  
  tris_salt_conc,  
  primer_conc  
)
```

### Arguments

primer.df	Primer data frame.
template.df	Template data frame
mode.directionality	Primer directionality.
na_salt_conc	Sodium ion concentration.
mg_salt_conc	Magnesium ion concentration.
k_salt_conc	Potassium ion concentration.
tris_salt_conc	Tris buffer concentration.
primer_conc	Primer concentration.

### Value

All annealing temperatures for given binding events.

### References

Rychlik, W. J. S. W., W. J. Spencer, and R. E. Rhoads. "Optimization of the annealing temperature for DNA amplification in vitro." *Nucleic acids research* 18.21 (1990): 6409-6412.

---

compute.template.secondary.structures  
*Template Secondary Structures*

---

### Description

Computes template secondary structures.

### Usage

```
compute.template.secondary.structures(  
  template.df,  
  annealing.temperature,  
  regions = NULL,  
  constraints = NULL  
)
```

### Arguments

template.df	Template data frame.
annealing.temperature	Temperature [C] at which to compute secondary structures.
regions	List containing the positional intervals for which the template secondary structure should be computed.
constraints	String giving secondary structure constraints. For example xxxxxx... would forbid folding in the first 6 bases of a template with length 9 and allow folding in its last 3 bases.

### Value

Data frame with info on template secondary structures.

---

compute.Tm.baldino     *Baldino Formula*

---

### Description

Computes the melting temperature using the formulation by Baldino.

### Usage

```
compute.Tm.baldino(  
  sequences,  
  na_salt_conc,  
  mg_salt_conc,  
  k_salt_conc,  
  tris_salt_conc,  
  primer_conc  
)
```

**Arguments**

sequences	Input sequence strings.
na_salt_conc	Sodium ion concentration.
mg_salt_conc	Magnesium ion concentration.
k_salt_conc	Potassium ion concentration.
tris_salt_conc	Tris buffer concentration.
primer_conc	Primer concentration.

**Value**

The melting temperature for the input sequences.

**References**

Rychlik, W. J. S. W., W. J. Spencer, and R. E. Rhoads. "Optimization of the annealing temperature for DNA amplification in vitro." *Nucleic acids research* 18.21 (1990): 6409-6412.

---

compute.Tm.sets	<i>Cross-Dimerization Filtering</i>
-----------------	-------------------------------------

---

**Description**

Removes cross-dimerizing primers from the input data.

**Usage**

```
compute.Tm.sets(
  primer.df,
  template.df,
  Tm.brackets,
  settings,
  mode.directionality = c("fw", "rev"),
  primer_conc,
  template_conc,
  na_salt_conc,
  mg_salt_conc,
  k_salt_conc,
  tris_salt_conc,
  allowed.mismatches,
  allowed.other.binding.ratio,
  allowed.stop.codons,
  allowed.region.definition,
  disallowed.mismatch.pos,
  opti.mode = FALSE,
  required.cvg = NULL,
  primers.fw = NULL,
  diagnostic.location = NULL,
  updateProgress = NULL
)
```

**Arguments**

<code>primer.df</code>	Primer data frame.
<code>template.df</code>	Template data frame.
<code>Tm.brackets</code>	Data frame with target primer melting temperatures.
<code>settings</code>	A DesignSettings object.
<code>mode.directionality</code>	Identifier of strand for which primers shall be designed.
<code>primer_conc</code>	Primer concentration.
<code>template_conc</code>	Template concentration.
<code>na_salt_conc</code>	Sodium ion concentration.
<code>mg_salt_conc</code>	Magnesium ion concentration.
<code>k_salt_conc</code>	Potassium ion concentration.
<code>tris_salt_conc</code>	Tris ion concentration.
<code>allowed.mismatches</code>	The number of mismatches primers are allowed to have with the templates.
<code>allowed.other.binding.ratio</code>	Ratio of primers allowed to bind to non-target regions.
<code>allowed.stop.codons</code>	Consider mismatch binding events that induce stop codons.
<code>allowed.region.definition</code>	Definition of the allowed region.
<code>disallowed.mismatch.pos</code>	The number of positions from the primer 3' end where mismatches should not be allowed. All primers binding templates with mismatches within <code>disallowed.mismatch.pos</code> from the 3' end are disregarded.
<code>opti.mode</code>	Compute optimization constraints and relax delta Tm if necessary.
<code>required.cvg</code>	Target coverage ratio.
<code>primers.fw</code>	Already designed primer sets for the target temperatures given in <code>Tm.brackets</code> . Used to determine cross-dimerization.
<code>diagnostic.location</code>	Directory for storing results.
<code>updateProgress</code>	Shiny progress callback function.
<code>primers.rev</code>	The primer data set to be filtered for cross-dimers.
<code>opti.constraints</code>	List with optimization constraint settings.
<code>annealing.temp</code>	The PCR annealing temperature.

**Value**

`primers.rev` with removed cross-dimerizing primers.  
Primer data frames for every target temperature.

---

`compute.unique.covered.idx`

*Unique Coverage Indices Computes the indices of templates that are covered uniquely covered by an individual primer.*

---

### Description

Unique Coverage Indices Computes the indices of templates that are covered uniquely covered by an individual primer.

### Usage

```
compute.unique.covered.idx(primer.df, template.df)
```

### Arguments

<code>primer.df</code>	Primer data frame.
<code>template.df</code>	Template data frame.

### Value

Index of templates uniquely covered per input primer.

---

`compute_annealing_temp`

*Annealing temperature.*

---

### Description

Identifies the optimal annealing temperature of a set of primers. If primers cover template sequences, the annealing temperature is computed using Rychlik's formula. Otherwise, the annealing temperature is determined using the rule of thumb based on the melting temperatures of the primers.

### Usage

```
compute_annealing_temp(  
  primer.df,  
  mode.directionality,  
  template.df,  
  na_salt_conc,  
  mg_salt_conc,  
  k_salt_conc,  
  tris_salt_conc,  
  primer_conc  
)
```

**Arguments**

<code>primer.df</code>	Primer data frame.
<code>mode.directionality</code>	Primer directionality.
<code>template.df</code>	Template data frame
<code>na_salt_conc</code>	Sodium ion concentration.
<code>mg_salt_conc</code>	Magensium ion concentration.
<code>k_salt_conc</code>	Potassium ion concentration.
<code>tris_salt_conc</code>	Tris buffer concentration.
<code>primer_conc</code>	Primer concentration.

**Value**

The optimal annealing temperature.

---

`condition`

*Condition Constructor*

---

**Description**

Constructs a condition for custom errors.

**Usage**

```
condition(subclass, message, call = sys.call(-1), ...)
```

**Arguments**

<code>subclass</code>	String giving the specific error.
<code>message</code>	String giving the user message.
<code>call</code>	Environment object.
<code>...</code>	Other arguments for the output structure.

**Value**

A condition structure.

---

consecutive.GC.count    *Consecutive GCs*

---

**Description**

Determines the maximum number of consecutive G/Cs

**Usage**

```
consecutive.GC.count(y, len)
```

**Arguments**

y	Positions where G/C occurs. Positions are numbered from 1 to 5 where 5 is the end of the primer.
len	Is the number of bases from the primer end considered.

**Value**

The maximal number of consecutive G/Cs.

---

constraints.to.df    *Conversion of Constraints List to Data Frame.*

---

**Description**

Converts the input constraints to a data frame representation.

**Usage**

```
constraints.to.df(  
  limit.constraints,  
  out.names,  
  format.type = c("backend", "shiny", "report")  
)
```

**Arguments**

limit.constraints	A list with constraints.
out.names	The desired column names.
format.type	The type of formatting to be performed on the table

**Value**

A data frame giving an overview of the constraints.

---

`constraints.xml.format`*Constraint XML Format.*

---

**Description**

Format constraint settings for XML output.

**Usage**

```
constraints.xml.format(constraints, set.name)
```

**Arguments**

<code>constraints</code>	List with constraint settings.
<code>set.name</code>	Identifier for the constraint settings.

**Value**

XML string containing the constraint settings.

---

`constraints_to_unit`    *Mapping of Constraints to Units.*

---

**Description**

Maps constraints to units for plotting.

**Usage**

```
constraints_to_unit(  
  constraint,  
  use.unit = TRUE,  
  format.type = c("backend", "HTML", "report")  
)
```

**Arguments**

<code>constraint</code>	The names of the constraints to convert to their plot identifiers (units).
<code>use.unit</code>	Whether constraint names should be annotated with their units.
<code>use.HTML</code>	Whether constraint units should be annotated with HTML units.

**Value**

A list of constraint names.

---

convert.from.iupac      *Conversion from IUPAC nucleotides*

---

**Description**

Convert sequences with IUPAC ambiguity codes to all possible sequences without ambiguities.

**Usage**

```
convert.from.iupac(seqs)
```

**Arguments**

seqs                    A vector of strings.

**Value**

A list containing the disambiguated input sequences.

---

convert.PCR.units      *Conversion of PCR Units*

---

**Description**

Converts frontend PCR concentration units to the units used for the backend.

**Usage**

```
convert.PCR.units(pcr.settings, to.mol = TRUE)
```

**Arguments**

pcr.settings          List with several PCR settings (concentrations).

to.mol                 If TRUE, convert to the molar concentration. If FALSE convert to the unit representation in the XML.

**Value**

List with concentrations for usage in the backend.

convert.temperature      *Conversion between Celsius and Kelvin*

---

**Description**

Converts the input from Kelvin to Celsius or from Celsius to Kelvin.

**Usage**

```
convert.temperature(temp, temp.scale = c("K", "C"))
```

**Arguments**

temp                      The input temperature.  
temp.scale                The desired unit of the output temperature.

**Details**

If temp.scale is 'K', T\_m is transformed from Celsius to Kelvin. If temp.scale is 'C', T\_m is transformed from Kelvin to Celsius. The default is to transform from Celsius to Kelvin.

**Value**

Transforms the input temperature to the specified temp.scale.

---

convert.to.iupac      *Merge sequences.*

---

**Description**

Merges the input sequences to one sequence containing IUPAC ambiguity codes.

**Usage**

```
convert.to.iupac(seqs)
```

**Arguments**

seqs                      Vector of strings.

**Value**

Consensus sequence of seqs.

---

con_select	<i>Quick Selection of Constraints.</i>
------------	--

---

**Description**

Select constraints that can be used according to third-party tools quickly.

**Usage**

```
con_select(active.constraints)
```

**Arguments**

```
active.constraints  
    Identifiers of constraints.
```

**Value**

The identifiers of constraints that can be computed.

---

copy.melt.config	<i>Copy MELTING Config File</i>
------------------	---------------------------------

---

**Description**

Copies modified MELTING tandem mismatch file to the MELTING data folder.

**Usage**

```
copy.melt.config(melt.bin = NULL)
```

**Value**

TRUE if the file is available in the MELTING folder, FALSE otherwise.

---

covered.primers.to.ID.string	<i>Conversion of Primer Indices to ID string</i>
------------------------------	--

---

**Description**

Converts the input coverage indices to a comma-separated string with the template IDs.

**Usage**

```
covered.primers.to.ID.string(covered.primers, primer.df)
```

**Arguments**

covered.primers      Identifiers of primers covering sequences.  
 primer.df          Primer data frame.

**Value**

String containing the covered template IDs.  
 A string containing the IDs of covering primers.

---

covered.seqs.to.ID.string      *Conversion of Template Coverage Indices to ID string*

---

**Description**

Converts the input coverage indices to a comma-separated string with the template IDs.

**Usage**

covered.seqs.to.ID.string(covered.seqs, template.df)

**Arguments**

covered.seqs      Indices of covered template sequences.  
 template.df      Template data frame.

**Value**

A string containing the covered template IDs.

---

covered.seqs.to.idx      *Conversion of Coverage Strings to Indices.*

---

**Description**

Converts the input coverage strings (comma separated template Identifiers) into indices.

**Usage**

covered.seqs.to.idx(covered.seqs, template.df)

**Arguments**

covered.seqs      Strings of covered sequences to be converted.  
 template.df      Template data frame containing the identifiers of templates

**Value**

Indices of covered templates.

---

`create.constraint.table`*Output a Constraint Overview Table*

---

**Description**

Outputs a table showing the values of constraints.

**Usage**

```
create.constraint.table(  
  constraints,  
  constraint.limits = NULL,  
  constraints.used.fw = NULL,  
  constraints.used.rev = NULL,  
  format.type = c("backend", "shiny", "report")  
)
```

**Arguments**

`constraints` List with constraint settings.  
`constraint.limits` List with constraint limits.  
`constraints.used.fw` Constraints used for forward primer design.  
`constraints.used.rev` Constraints used for reverse primer design.  
`format.type` The type of formatting to be performed on entries.

**Value**

Data frame with summary of constraints.

---

`create.constraint.XML` *XML Output of Constraints*

---

**Description**

Creates an XML summarizing all settings.

**Usage**

```
create.constraint.XML(  
  filtering.constraints,  
  c.f.lim,  
  cvg.constraints,  
  PCR.settings,  
  constraint.settings  
)
```

**Arguments**

- filtering.constraints      List with constraint settings for filtering.
- c.f.lim                    Relaxation limits for the filtering constraints.
- cvg.constraints            List with constraints for coverage computations.
- PCR.settings              Settings for the PCR.
- constraint.settings        Other settings of constraints (e.g. coverage).

**Value**

String in XML format containing all constraint settings.

---

<code>create.cvg.text</code>	<i>Coverage Info Text</i>
------------------------------	---------------------------

---

**Description**

Creates a string with information on the coverage

**Usage**

```
create.cvg.text(stats, selected.group = NULL, ident = NULL)
```

**Arguments**

- stats                      Data frame with coverage statistics.
- selected.group          Retrieve information for a subgroup of templates only.
- ident                      An identifier for the coverage.

**Value**

A string with information on the coverage.

---

<code>create.G.matrix</code>	<i>Create free energy matrix</i>
------------------------------	----------------------------------

---

**Description**

Creates a matrix giving the deltaG values of all primers.

**Usage**

```
create.G.matrix(primer.df, G.df, primer.df.2 = NULL)
```

**Arguments**

primer.df	Primer data frame.
G.df	Free energy data for the primers.
primer.df.2	Optional second primer data frame

**Value**

Matrix with the smallest free dimerization energy for every primer pair.

---

```
create.initial.primers.set
```

*Creation of an Initial Primer Set.*

---

**Description**

Creates an initial set of candidate primers for primer design.

**Usage**

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

**Arguments**

template.df	Template data frame.
primer.lengths	Vector containing the permissible primer lengths.
mode.directionality	Direction of primers to be created.
sample	Name of the template sample.
allowed.region.definition	Definition of the allowed binding region.
init.algo	Algorithm for initializing primers.
max.degen	Maximal allowed degeneration of created primers.
conservation	Required conservation of primers. The value of conservation should be in the range[0,1].
updateProgress	Shiny progress object.

**Value**

An initialized data frame of candidate primers.

---

create.k.mers                      *Creation of k-mers for multiple sequences.*

---

**Description**

Creation of k-mers for multiple sequences.

**Usage**

```
create.k.mers(seqs, k)
```

**Arguments**

k	The size of the k-mer.
seq	A character vector.

**Value**

A list with named character vectors, containing the k-mers.

---

create.kmer                      *Creation of k-mers of a single sequence.*

---

**Description**

Creation of k-mers of a single sequence.

**Usage**

```
create.kmer(seq, k)
```

**Arguments**

seq	A character vector.
k	The size of the k-mer.

**Value**

A names character vector, where the names are the relative positions of the k-mers and the values give the character vector of the k-mer.

---

create.options.table    *Creation of a Table for Constraint Options.*

---

**Description**

Creation of a Table for Constraint Options.

**Usage**

```
create.options.table(  
  other.settings,  
  format.type = c("backend", "shiny", "report")  
)
```

**Arguments**

other.settings    List with constraint options  
format.type        How the table shall be formatted.

**Value**

A data frame.

---

create.other.table    *Creation of a Table for Other Constraint Settings.*

---

**Description**

Creation of a Table for Other Constraint Settings.

**Usage**

```
create.other.table(other.settings, col.names, format.type)
```

**Arguments**

other.settings    List with other constraint settings.  
format.type        How the table shall be formatted.

**Value**

A data frame.

---

create.PCR.table      *Creation of a Table for PCR Conditions*

---

### Description

Creation of a Table for PCR Conditions

### Usage

```
create.PCR.table(other.settings, format.type = c("backend", "shiny", "report"))
```

### Arguments

other.settings    List with PCR settings.  
format.type        How the table shall be formatted.

### Value

A data frame.

---

create.primers.ranges    *Ranges for Initial Primers.*

---

### Description

Creates a data frame indicating primer starts and ends.

### Usage

```
create.primers.ranges(  
  end.position,  
  p.lens,  
  start.position,  
  step.size = 1,  
  groups = NULL  
)
```

### Arguments

end.position    End positions of primers.  
p.lens            Desired primer lengths.  
step.size        A numeric giving the steps with which start positions are cycled. Should be 1 for primer design (evaluate all positions) and higher values can be used for windowing.  
groups           Character vector with group annotation.  
start.position   The start positions of primers.

### Value

Data frame with ranges for initial primers.

---

create.primers.naive *Naive Initialization of Primers.*

---

### Description

Initialize primers by extracting substrings from all templates.

### Usage

```
create.primers.naive(  
  seqs,  
  seq.IDs,  
  seq.groups,  
  l.s,  
  e.s,  
  primer.lengths,  
  allowed.region.definition,  
  max.degen,  
  sample = "",  
  identifier = "",  
  updateProgress = NULL  
)
```

### Arguments

seqs	The template sequence strings.
seq.IDs	The identifiers of the templates.
seq.groups	The group identifiers of the templates.
l.s	The positions where the allowed region starts for each template.
e.s	The positions where the allowed region ends for each template.
primer.lengths	Vector of desired primer lengths.
allowed.region.definition	Definition of the allowed region.
max.degen	Maximum allowed degeneracy of primers.
sample	Template sample identifier.
updateProgress	Shiny progress object.

### Value

Data frame with initialized primer candidates.

---

create.primers.tree     *Tree-based Initialization of Primers.*

---

## Description

Creates a set of candidate primers using a tree-based algorithm.

## Usage

```
create.primers.tree(  
  seqs,  
  seq.IDs,  
  seq.groups,  
  start,  
  end,  
  primer.lengths,  
  allowed.region.definition,  
  max.degen,  
  conservation,  
  sample = "",  
  identifier = "",  
  updateProgress = NULL  
)
```

## Arguments

seqs	Template sequences.
seq.IDs	Identifiers of template sequences.
seq.groups	Group identifiers of template sequences.
start	For each template the start of the interval where primers should be created.
end	For each template the end of the interval where primers should be created.
primer.lengths	Vector of desired primer lengths.
allowed.region.definition	Definition of allowed regions.
max.degen	Maximal degeneracy of primers.
conservation	Required conservation of template regions considered for the generation of primers. Conservation identifies the top conserved percentile of possible primers.
sample	Sample name for the analysis.
identifier	Identifier (e.g. for directionality).
updateProgress	Shiny progress object.

## Details

First, primers are aligned and their sequence similarity is determined to compute a phylogenetic tree using hierarchical clustering. Next, the tree is traversed from leaves to top in order to identify groups of primers that can be merged (consensus) without exceeding the maximum degeneracy of primers.

**Value**

A vector with initialized primers.

---

create.Tm.brackets      *Creation of Melting Temperature Groups*

---

**Description**

Creates a data frame identifying target melting temperatures of individual primer sets.

**Usage**

```
create.Tm.brackets(primers, template.df, settings, target.temps = NULL)
```

**Arguments**

primers	An object of class Primers for which to create groups based on melting temperatures.
template.df	An object of class Templates corresponding to the primers.
settings	A DesignSettings objects.
target.temps	Pre-defined target melting temperatures to use instead of automatically determining groups from the primers.

**Value**

Data frame with target melting temperatures for individual primer sets.

---

create.uniform.leaders  
*Uniform Binding Ranges.*

---

**Description**

Creates uniform binding regions for all templates.

**Usage**

```
create.uniform.leaders(fw.interval, rev.interval, template.df, gap.char)
```

**Arguments**

fw.interval	Binding region for forward templates.
rev.interval	Binding region for reverse templates.
template.df	Template data frame.
gap.char	The character for gaps in alignments.

**Value**

Data frame with binding region information.

---

create\_fulfilled\_counts

*Creation of Fulfilled/Failed Constraint Counts.*

---

### Description

Creates counts of fulfilled/failed constraints.

### Usage

```
create_fulfilled_counts(primer.df, eval.cols = NULL)
```

### Arguments

primer.df	An evaluated Primers object.
eval.cols	Evaluation columns in primer.df to consider. By default (NULL) all evaluation columns are considered.

### Value

A data frame with the number of fulfilled/failed constraints for primer.df.

---

create\_report,list,list-method

*Creation of a PDF Report for Primer Comparison.*

---

### Description

Creates a PDF report for comparing multiple primers.

### Usage

```
## S4 method for signature 'list,list'
create_report(
  primers,
  templates,
  fname,
  settings,
  sample.name = NULL,
  used.settings = NULL
)
```

### Arguments

primers	A list with evaluated Primers objects.
templates	A list with Templates objects.
fname	A character vector giving the file to store the report in.
settings	A DesignSettings object.
sample.name	An identifier for your analysis.
used.settings	A named list (with fields "fw" and "rev") containing the forward/reverse used design settings.

**Value**

Creates a PDF file giving a report on the comparison of the input primers.

**Note**

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

---

create\_report, Primers, Templates-method  
*Creation of a PDF report.*

---

**Description**

Creates a PDF report for a set of primers.

**Usage**

```
## S4 method for signature 'Primers, Templates'
create_report(
  primers,
  templates,
  fname,
  settings,
  sample.name,
  used.settings,
  required.cvg = 1
)
```

**Arguments**

primers	An evaluated Primers object.
templates	A Templates object.
fname	A character vector giving the file to store the report in.
settings	A DesignSettings object.
sample.name	An identifier for your analysis.
used.settings	A named list (with fields "fw" and "rev") containing the forward/reverse used design settings.
required.cvg	The required coverage ratio.

**Value**

Creates a PDF file reporting on the input primers.

**Note**

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

Data

*Data Sets.***Description**

Ippolito IGHV primer data from Ippolito et al.

Tiller IGHV primer data from Tiller et al.

Comparison 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.

RefCoverage Experimental results of multiplex PCR.

**Usage**

```
data(Comparison)
```

```
data(Ippolito)
```

```
data(RefCoverage)
```

```
data(Tiller)
```

**Format**

For the RefCoverage data set, 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.

For the Ippolito data set, `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.

For the Comparison data set, `primer.data` and `template.data` are lists of `Primers` and `Templates` objects, respectively.

For the Tiller data set, `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`.

**References**

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

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 Check-points in the Mouse. PLoS One 7:e35497.

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

```
# Load the comparison 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)

# Load the data from Ippolito et al.
data(Ippolito)
primer.df
template.df
constraints(settings)

# Load experimental PCR results
data(RefCoverage)

# Load the data from Tiller et al.
data(Tiller)
tiller.primer.df
tiller.template.df
constraints(tiller.settings)
```

---

design\_primers.single *Design Primers for a Single Direction*

---

## Description

Designs primers for a single direction.

## Usage

```
design_primers.single(
  template.df,
  sample.name,
  mode.directionality = c("fw", "rev"),
  settings,
  timeout,
  opti.algo,
  allowed.region.definition,
  init.algo,
  max.degen,
  conservation,
```

```

target.temps,
required.cvg,
fw.primers = NULL,
cur.results.loc = NULL,
primer.df = NULL,
updateProgress = NULL
)

```

### Arguments

template.df	Template data frame with sequences for which primers shall be designed.
sample.name	Identifier for the templates.
mode.directionality	Template strands for which primers shall be designed. Primers can be designed either only for forward strands, only for reverse strands, or both strand directions.
settings	A DesignSettings object specifying the criteria for designing primers.
timeout	Timeout in seconds for the optimization with ILPs.
opti.algo	The algorithm to be used for solving the primer set covering problem.
allowed.region.definition	Definition of the target binding sites used for evaluating the coverage. If allowed.region.definition is "within", primers have to lie within the allowed binding region. If allowed.region.definition is "any", primers have to overlap with the allowed binding region. The default is that primers have to bind within the target binding region.
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.
max.degen	Maximal degeneracy of merged primers.
conservation	When using the tree-based primer initialization, consider only the conservation percentile of regions with the highest conservation.
target.temps	Target melting temperatures for optimized primer sets in Celsius. Only required when optimizing primers for both strand directions and one optimization was already performed.
required.cvg	The target ratio of covered template sequences. If the target ratio cannot be reached, the constraint settings are relaxed up to the relaxation limits.
fw.primers	List with optimized primer data frames corresponding to target.temps. Only required for optimizing both strand directions and only in the second optimization run in order to check for cross dimerization.
cur.results.loc	Directory for storing results of the primer design procedure.
primer.df	A data frame of evaluated primer candidates that can be optimized directly.
updateProgress	Shiny progress callback function.

### Value

A list containing the results of the primer design procedure:

opti: A Primers object representing the set of optimized primers.

**all\_results:** A list containing the optimal results for each sampled melting temperature range in terms of a Primers object in case that the `melting_temp_diff` constraint was active. Otherwise, `all_results` only has a single entry representing a primer set relating to an undefined melting temperature.

**used\_constraints:** A DesignSettings object with the (adjusted) analysis settings.

**filtered:** A Primers object containing the primer candidates that passed the filtering procedure and which gave rise to the final optimal set.

---

`detect.gap.columns`      *Identification of Gappy Columns in Alignments.*

---

### Description

Identification of Gappy Columns in Alignments.

### Usage

```
detect.gap.columns(bins, gap.cutoff = 0.95, gap.char = "-")
```

### Arguments

<code>bins</code>	A list of DNABin alignments.
<code>gap.cutoff</code>	The required percentage of gaps for consideration as a gap column.
<code>gap.char</code>	The gap character in the alignments.

### Value

A list with indices giving the gap columns for every alignment in bins.

---

`dimerization.table`      *Dimerization Table.*

---

### Description

Summarizes how often individual primers dimerize according to the `deltaG.cutoff`.

### Usage

```
dimerization.table(
  dimer.data,
  deltaG.cutoff,
  dimer.type = c("Self-Dimerization", "Cross-Dimerization")
)
```

### Arguments

<code>dimer.data</code>	Data frame with dimerization data.
<code>deltaG.cutoff</code>	Free energy cutoff for dimerization.
<code>dimer.type</code>	String identifying whether <code>dimer.data</code> refers to cross-dimers or self-dimers?

**Value**

Data frame with dimer counts.

---

dir.copy                      *Copy Directories.*

---

**Description**

Copies a directory to another location.

**Usage**

```
dir.copy(src.dir, dest.dir, overwrite)
```

**Arguments**

src.dir	The directory to be copied.
dest.dir	The target directory.
overwrite	Overwrite existing files in dest.dir.

**Value**

TRUE if copying was successful, FALSE otherwise.

---

disambiguate.primers    *Disambiguation of Primers.*

---

**Description**

Disambiguates ambiguous primer sequences into all possible sequences.

**Usage**

```
disambiguate.primers(p.df)
```

**Arguments**

p.df	Primer data frame.
------	--------------------

**Value**

Data frame with disambiguated primers.

---

estimate.cvg	<i>Estimation of Primer Coverage.</i>
--------------	---------------------------------------

---

**Description**

Estimates the possible coverage of primers using probes of size k and only considering perfect matches without consideration of ambiguities.

**Usage**

```
estimate.cvg(lex.df, k = 18, mode.directionality, sample = "")
```

**Arguments**

k	A numeric giving the size of the primers.
mode.directionality	Estimation of coverage for forward/reverse/both?
sample	An optional identifier for the sample.
seqs	A character vector of sequences to evaluate coverage for.

**Value**

A list with entries fw and rev giving data frames for forward/reverse binding.

---

estimate.cvg.dir	<i>Estimation of Primer Coverage.</i>
------------------	---------------------------------------

---

**Description**

Estimates the possible coverage of primers using probes of size k and only considering perfect matches without consideration of ambiguities.

**Usage**

```
estimate.cvg.dir(seqs, k, id = "")
```

**Arguments**

seqs	A character vector of sequences to evaluate coverage for.
k	A numeric giving the size of the primers.
id	An optional identifier for the primers.

**Value**

A data frame with binding information.

---

```
eval.comparison.primers
```

*Evaluation of Primers for Comparison*

---

### Description

Evaluate multiple primer sets according to the input constraint settings.

### Usage

```
eval.comparison.primers(primer.data, constraint.settings)
```

### Arguments

```
primer.data    List with primer data frames.
constraint.settings
                List with constraint.settings.
```

### Value

List with evaluated primer data frames.

---

```
eval.constraints
```

*Evaluation of Constraints'*

---

### Description

Evaluates whether the given primer data frame fulfills the required conditions.

### Usage

```
eval.constraints(
  constraint.df,
  constraint.settings,
  active.constraints,
  mode.directionality = c("fw", "rev", "both"),
  primer.df
)
```

### Arguments

```
constraint.df  Primer data frame with computed constraints.
constraint.settings
                List with allowed values pers constraint.
active.constraints
                Names of constraints to be evaluated.
mode.directionality
                Directionality of primers
primer.df      Primer data frame corresponding to constraint.df.
```

**Details**

Constraint values should be contained in `constraint.df`. For each constraint in `active.constraints`, a boolean column with the name `EVAL_<constraint_name>` is generated, which indicates whether a primer in a given rows fulfills a constraint or not.

**Value**

Augments the `constraint.df` data frame with evaluation columns.

---

evaluate.basic.cvg	<i>Evaluation of Primer Coverage.</i>
--------------------	---------------------------------------

---

**Description**

Evaluates the coverage of a set of primers.

**Usage**

```
evaluate.basic.cvg(
  template.df,
  primers,
  mode.directionality = c("fw", "rev", "both"),
  allowed.mismatches,
  allowed.other.binding.ratio,
  allowed.region.definition = c("within", "any"),
  updateProgress = NULL
)
```

**Arguments**

<code>template.df</code>	Template data frame.
<code>primers</code>	Primer data frame.
<code>mode.directionality</code>	Primer directionality.
<code>allowed.mismatches</code>	The number of allowed mismatches per binding event.
<code>allowed.other.binding.ratio</code>	Ratio of primers that are allowed to bind to non-allowed regions. If <code>allowed.other.binding.ratio &gt; 0</code> primers are allowed to bind at any location within the templates. However, a warning is given if the ratio of primers binding to non-target regions exceeds the <code>allowed.other.binding.ratio</code> .
<code>allowed.region.definition</code>	Definition of the target binding sites used for evaluating the coverage. If <code>allowed.region.definition</code> is <code>within</code> , primers have to lie within the allowed binding region. If <code>allowed.region.definition</code> is <code>any</code> , primers have to overlap with the allowed binding region. The default is that primers have to bind within the target binding region.
<code>updateProgress</code>	Progress callback function for shiny.

**Value**

Primer data frame with information on the covered template sequences.

evaluate.constrained.cvg

*Evaluation of Primer Coverage.*

---

### Description

Evaluates the coverage of a set of primers.

### Usage

```
evaluate.constrained.cvg(  
  template.df,  
  primer.df,  
  cvg.df,  
  mode.directionality = c("fw", "rev", "both"),  
  settings,  
  updateProgress = NULL  
)
```

### Arguments

template.df	Template data frame.
primer.df	Primer data frame.
cvg.df	Data frame with basic coverage entries.
mode.directionality	Primer directionality.
settings	A DesignSettings object.
updateProgress	Progress callback function for shiny.

### Value

Primer data frame with information on the covered template sequences.

---

evaluate.cvg

*Evaluation of Coverage.*

---

### Description

Evaluates primer coverage.

### Usage

```
evaluate.cvg(  
  template.seqs,  
  primers,  
  mode.directionality = c("fw", "rev"),  
  allowed.mismatches,  
  updateProgress = NULL  
)
```

**Arguments**

template.seqs    Template sequences as a DNASTringSet.  
primers            Primer sequences as a DNASTringSet.  
mode.directionality    Directionality of primres  
allowed.mismatches    Allowed number of mismatches between a primer and a template.  
updateProgress    Progress function for shiny

**Value**

IRanges object with primer coverage information.

---

evaluate.diff.primer.cvg  
*Evaluation of Coverage.*

---

**Description**

Re-evaluates the coverage of primers under exclusion of certain templates.

**Usage**

```
evaluate.diff.primer.cvg(primers, excluded.seqs, template.df)
```

**Arguments**

primers            Primer data frame.  
excluded.seqs    Identifiers of templates to be excluded.  
template.df      Template data frame

**Details**

This function requires that primers was already annotated with primer coverage before.

**Value**

Primer data frame with updated coverage under the exclusion of excluded.seqs.

---

```
evaluate.fw.rev.combinations
```

*Evaluation of Set Combinations*

---

### Description

Evaluates the combinations of forward and reverse primer sets.

### Usage

```
evaluate.fw.rev.combinations(opti.fw, opti.rev, compatible.df, template.df)
```

### Arguments

<code>opti.fw</code>	List with forward optimal primer sets.
<code>opti.rev</code>	List with reverse optimal primer sets.
<code>compatible.df</code>	Data frame containing the indices of temperature-compatible forward and reverse primers sets.
<code>template.df</code>	Template data frame for which primers were designed.
<code>opti.rev.indices</code>	Indices for accessing <code>opti.rev</code> .

### Value

List with information on the combinations of forward and reverse primers as well as the combined data frames themselves.

---

```
evaluate.GC.clamp
```

*GC clamp*

---

### Description

Determines the number of consecutive G/Cs at the 3' end.

### Usage

```
evaluate.GC.clamp(y)
```

### Arguments

<code>y</code>	Primer sequence from 5' to 3'.
----------------	--------------------------------

### Value

The length of the GC clamp.

---

evaluate.primer.cvg     *Evaluation of Primer Coverage.*

---

### Description

Evaluates the coverage of a set of primers.

### Usage

```
evaluate.primer.cvg(  
  template.df,  
  primers,  
  mode.directionality = c("fw", "rev", "both"),  
  settings,  
  updateProgress = NULL  
)
```

### Arguments

template.df     Template data frame.  
primers           Primer data frame.  
mode.directionality  
                 Primer directionality.  
settings         A DesignSettings object.  
updateProgress   Progress callback function for shiny.

### Value

Primer data frame with information on the covered template sequences.

---

evaluate.template.constraints  
                                 *Evaluation of Template Constraints.*

---

### Description

Evaluates the input template constraints.

### Usage

```
evaluate.template.constraints(  
  constraint.values,  
  constraint.settings,  
  active.constraints,  
  mode.directionality = c("fw", "rev", "both")  
)
```

**Arguments**

`constraint.values`  
 Data frame with template constraints  
`constraint.settings`  
 List specifying the allowed values for constraint evaluation.  
`active.constraints`  
 Strings specifying the constraints to check.  
`mode.directionality`  
 Direction of primers.

**Value**

List indicating which template constraints were fulfilled or not (TRUE/FALSE).

---

`exclude.cols`      *Exclusion of Columns*

---

**Description**

Removes columns from a data frame.

**Usage**

```
exclude.cols(excl.col, template.df)
```

**Arguments**

`excl.col`      Names of columns in `template.df` to be removed.  
`template.df`    Data frame for which columns in `excl.col` should be removed.

**Value**

`template.df` with removed columns as specified in `excl.col`.

---

`filter.by.constraints`    *Filter By Constraints*

---

**Description**

Remove primers that do not fulfill the current constraints (evaluate all primers).

**Usage**

```

filter.by.constraints(
  filtered.df,
  constraint.df,
  current.constraints,
  active.constraints,
  mode.directionality = c("fw", "rev", "both"),
  template.df
)

```

**Arguments**

filtered.df	Primer data frame.
constraint.df	Data frame with constraint values.
current.constraints	List with constraint settings.
active.constraints	Strings giving the names of active constraints.
mode.directionality	Direction of primers
template.df	Template data frame.

**Value**

A list containing the filtered primer data frame, as well as a data frame of the excluded primers and the used filtering settings.

---

filter.comparison.primers  
*Filter Multiple Primer Sets.*

---

**Description**

Filters multiple primer sets at once.

**Usage**

```
filter.comparison.primers(  
  primers,  
  templates,  
  active.constraints,  
  settings,  
  updateProgress = NULL  
)
```

**Arguments**

primers	List with primer data frames.
templates	List with template data frames.
active.constraints	Strings giving the constraints that are to be checked.
settings	List with settings.
updateProgress	Progress callback function for shiny.

**Value**

A list with filtered primer data frames.

---

`filter.primers.candidates`*Filtering of Primer Candidates*

---

**Description**

Filters primer candidates according to length and duplications.

**Usage**

```
filter.primers.candidates(primer.candidates, min.len)
```

**Arguments**

<code>primer.candidates</code>	Alignment of candidate primers.
<code>min.len</code>	Minimal required length of primers.

**Value**

Filtered alignment of candidate primers.

---

`filter.primers.set.opti`*Filtering of Primers*

---

**Description**

Filters a primer set during the optimization procedure.

**Usage**

```
filter.primers.set.opti(  
  primer.df,  
  sample,  
  template.df,  
  settings,  
  mode.directionality,  
  required.cvg,  
  results.loc,  
  target.temps  
)
```

**Arguments**

<code>primer.df</code>	Primer data frame.
<code>sample</code>	Name of the current template sample.
<code>template.df</code>	Template data frame.
<code>settings</code>	List with settings for the constraints to be used for filtering.
<code>mode.directionality</code>	Primer direction.
<code>required.cvg</code>	Required ratio of covered templates. If <code>required.cvg</code> is set to 0, the constraints are not relaxed.
<code>results.loc</code>	Path to a directory where the results should be written.
<code>target.temps</code>	Target melting temperature of the primers in Celsius. This argument is only required if we try to match the melting temperatures of another primer set, e.g. when first optimizing forward and then optimizing reverse primers.

**Value**

The filtered primer data frame with respect to `required.cvg`.

---

<code>filterLimits</code>	<i>Getter for Filtering Constraint Limits.</i>
---------------------------	--

---

**Description**

Gets the limits on the constraints that are used for the filtering procedure when designing primers using the `Input_Constraint_Boundaries` slot of the provided `DesignSettings` object `x`.

**Usage**

```
filterLimits(x)

## S4 method for signature 'DesignSettings'
filterLimits(x)
```

**Arguments**

<code>x</code>	A <code>DesignSettings</code> object.
----------------	---------------------------------------

**Value**

Gets the list of filtering limits.

---

<code>filters</code>	<i>Getter for Filtering Constraints.</i>
----------------------	--

---

**Description**

Gets the constraints on the physicochemical properties that are used for the filtering procedure when designing primers using the `Input_Constraints` slot of the provided `DesignSettings` object `x`.

**Usage**

```
filters(x)

## S4 method for signature 'DesignSettings'
filters(x)
```

**Arguments**

`x`                    A `DesignSettings` object.

**Value**

Gets the list of filtering constraints.

---

<code>filter_primers.by.Tm.delta</code>	<i>Filter by Melting Temperature Difference</i>
---	---

---

**Description**

Filters primers by melting temperature differences.

**Usage**

```
filter_primers.by.Tm.delta(target.temp, selected.primers, max.Tm.delta)
```

**Arguments**

`target.temp`        Target melting temperature in Celsius.  
`selected.primers`    Current candidate primer data frame.  
`max.Tm.delta`        Maximum allowed difference of primer melting temperatures to target temperature.

**Value**

Filtered primer data frame.

---

`fix_constraint_boundaries`*Correction of Constraint Boundaries.*

---

**Description**

Fixes the constraint boundaries if they are more narrow than the current settings.

**Usage**

```
fix_constraint_boundaries(constraints, constraint.limits, fix.limit = TRUE)
```

**Arguments**

`constraints` A list with constraint settings.

`constraint.limits`  
A list with constraint limits.

`fix.limit` Whether the constraint limits should be adjusted. If FALSE, the constraint settings are adjusted.

**Value**

The corrected constraint limits.

---

`format.constraints` *Format Constraint Names.*

---

**Description**

Formats constraint names for frontend output.

**Usage**

```
## S3 method for class 'constraints'  
format(constraints)
```

**Arguments**

`constraints` The character vector of constraints to transform.

**Value**

A character vector with formatted constraint names.

---

format.seq.ali	<i>Format mismatches</i>
----------------	--------------------------

---

**Description**

Formats a sequence for highlighting mismatches in an alignment.

**Usage**

```
## S3 method for class 'seq.ali'  
format(seq, pos, format.type)
```

**Arguments**

seq	The input sequence.
pos	The mismatch positions to be formatted.
format.type	Vector of giving the style (bold/italics) for each pos.

**Value**

The input sequence with highlighted mismatch positions.

---

format.seqs.tex	<i>Format a Sequence for LateX output.</i>
-----------------	--

---

**Description**

Formats a sequence for LateX report output.

**Usage**

```
## S3 method for class 'seqs.tex'  
format(seqs)
```

**Arguments**

seqs	Character vector of sequences.
------	--------------------------------

**Value**

Formatted sequences.

---

`get.3prime.mismatch.pos`*Identification of 3' Mismatches.*

---

**Description**

Computes the lastmost position of a 3' mismatches of a primer with a template.

**Usage**

```
get.3prime.mismatch.pos(primers, mismatches)
```

**Arguments**

primers	Primer sequence strings.
mismatches	Comma-separated strings containing the primer mismatch positions.

**Value**

The closest position of a mismatch relative to the 3' end of the primer. Here, 1 indicates the terminal position, 2 the penultimate position, and so on. No mismatch is indicated by an infinite value.

---

`get.analysis.mode`      *Direction of Primers.*

---

**Description**

Identifies the directionality of the input primers.

**Usage**

```
get.analysis.mode(primers)
```

**Arguments**

primers	A primer data frame.
---------	----------------------

**Value**

both if both, forward and reverse primers exist in primers. Otherwise, if either only forward primers or reverse primers exist, returns fw or rev, respectively.

---

`get.consensus.seq`      *Computation of Consensus.*

---

**Description**

Computes the consensus of the sequences in the input alignment.

**Usage**

```
get.consensus.seq(ali)
```

**Arguments**

`ali`                    An alignment object.

**Value**

A consensus sequence without gap characters.

---

`get.constraint.value.idx`  
*Retrive Constraint Indices.*

---

**Description**

Gets the index of the required constraint columns in the primer data frame.

**Usage**

```
get.constraint.value.idx(active.constraints, constraint.df)
```

**Arguments**

`active.constraints`      The names of the constraints for which to find the indices in `constraint.df`.  
`constraint.df`      The primer data frame where the `active.constraints` should be found.

**Value**

Indices of `active.constraints` in `constraint.df`.

---

get.constraint.values *Get the Values of a Constraint.*

---

**Description**

Get the Values of a Constraint.

**Usage**

```
get.constraint.values(con.name, cur.candidates, mode.directionality)
```

**Arguments**

con.name	The name of the constraint.
cur.candidates	The Primers data frame where the values should be retrieved.
mode.directionality	The direction for which values should be retrieved.

**Value**

The constraint values corresponding to con.name for the primers cur.candidates.

---

get.coverage.matrix *Coverage Matrix*

---

**Description**

Constructs a coverage matrix where rows indicate templates and columns indicate primers.

**Usage**

```
get.coverage.matrix(primer.df, template.df, constraints = NULL)
```

**Arguments**

primer.df	Primer data frame.
template.df	Template data frame.
constraints	A character vector of coverage constraints to be used as entries for the coverage matrix instead of the 0/1 encoding. At its default setting (NULL), the 0/1 encoding is used.

**Details**

Entry (i,j) in the matrix is equal to 1 if primer j covers template i and otherwise 0.

**Value**

The binary coverage matrix.

---

get.covered.templates *Covered Templates*

---

### Description

Get the indices of covered templates.

### Usage

```
get.covered.templates(Tm.set, template.df)
```

### Arguments

Tm.set	Primer data frame.
template.df	Template data set.

### Value

Index of templates that are covered by the primers in Tm.set.

---

get.cross.dimers *Cross dimers*

---

### Description

Computes all possible primer cross-dimers.

### Usage

```
get.cross.dimers(
  primers.1,
  primers.2,
  ions,
  annealing.temp,
  check.idx = NULL,
  no.structures = FALSE,
  mode = c("symmetric", "asymmetric")
)
```

### Arguments

primers.1	Input primers.
primers.2	Input primers.
ions	Sodium-equivalent ionic concentration.
annealing.temp	The PCR annealing temperature.
check.idx	indices of primers for checking cross-dimerization
no.structures	Whether to compute structures of dimers.
mode	'symmetric', if primers.1 and primers.2 carry the same information (i.e. fw-fw, rev-rev, fw-rev), 'asymmetric' else.

**Value**

Data frame with potential cross dimers.

---

```
get.cvg.constraint.settings
```

*Gather all Coverage Constraints.*

---

**Description**

Constructor for coverage constraint settings.

**Usage**

```
get.cvg.constraint.settings(  
  allowed.stop.codons,  
  allowed.efficiency,  
  disallowed.mismatch.pos,  
  allowed.anneal.deltaG,  
  allowed.substitutions,  
  allowed.coverage.model  
)
```

**Arguments**

`allowed.stop.codons`  
Whether mismatch binding events inducing stop codons in the amino acid sequence are allowed.

`allowed.efficiency`  
Min/max for primer efficiency.

`disallowed.mismatch.pos`  
The positions from the 3' terminal end of primers where mismatches shall be prevented.

`allowed.anneal.deltaG`  
Maximal allowed free energy of template-primer annealing.

`allowed.substitutions`  
Whether mismatch binding events inducing substitutions in the amino acid sequence are allowed.

**Value**

List with all coverage constraint settings.

---

get.cvg.gain	<i>Computation of Coverage Gain.</i>
--------------	--------------------------------------

---

**Description**

Computes the coverage gain from covered.seqs.

**Usage**

```
get.cvg.gain(
  covered.seqs,
  template.df,
  missing.df,
  candidate.df,
  con.names,
  constraint.limits,
  feasible.only = FALSE
)
```

**Arguments**

covered.seqs	List with covered sequences.
template.df	A Templates data frame.
missing.df	A Templates data frame containing only the templates that still need to be covered.
candidate.df	A Primers data frame containing candidate primers.
con.names	The constraint to evaluate the coverage gain for upon being relaxed.
constraint.limits	A list with constraint limits.
feasible.only	Whether only feasible coverage gains are to be outtputed. Here, <i>feasible</i> relates to coverage gains that can be obtained directly with the next relaxation.

**Value**

The number of covered sequences to be gained.

---

get.delta.G	<i>Change in Free Energy.</i>
-------------	-------------------------------

---

**Description**

Computes the change in free energy.

**Usage**

```
get.delta.G(delta.H, delta.S, temp = 37)
```

**Arguments**

delta.H	Change in enthalpy in cal/mol.
delta.S	Change in entropy cal/mol*K.
temp	Temperature in Celsius for which to compute free energy change.

**Value**

The change in free energy in kcal/mol.

---

get.dimer.data      *Retrieval of dimerization energies.*

---

**Description**

Uses OligoArrayAux to compute dimerization candidates.

**Usage**

```
get.dimer.data(s1, s2, annealing.temp, ions, no.structures)
```

**Arguments**

s1	Nucleotide character vectors (5' to 3')
s2	Nucleotide character vectors (5' to 3')
annealing.temp	The PCR annealing temperature in Celsius.
ions	The sodium-equivalent ions used in the PCR.
no.structures	Whether to compute structures of dimers.

**Value**

A data frame containing free energies in the field DeltaG and the dimerization structure in Structure.

---

get.duplex.energies      *Determination of the Free Binding Energy.*

---

**Description**

Computest the free energy of annealing between primers and templates. If the mode is set to "on\_target", the free energies of binding events in the allowed region are computed, while if the mode is set to "off\_target", the free energies of off-target events are computed.

**Usage**

```
get.duplex.energies(
  primer.df,
  template.df,
  annealing.temp,
  settings,
  mode = c("on_target", "off_target")
)
```

**Arguments**

primer.df	A Primers object.
template.df	A Templates object.
annealing.temp	The vector of optimal annealing temperatures of the primers.
settings	A DesignSettings object.
mode	If the mode is set to "on_target", the free energies of binding events in the allowed region are computed, while if the mode is set to "off_target", the free energies of off-target events are computed.

**Value**

A list of lists containing the numeric free energies of the annealing events for every primer.

---

get.eval.cols	<i>Retrieval of Evaluation Columns.</i>
---------------	---

---

**Description**

Retrieves the evaluation columns by intersecting the already evaluated constraints in primer.data as well as the constraints specified in input constraint settings.

**Usage**

```
get.eval.cols(primer.data, constraint.settings)
```

**Arguments**

primer.data	A list with Primers objects.
constraint.settings	A list with constraint settings.

**Value**

A character vector with EVAL-columns.

---

get.extension	<i>Identification of File extension.</i>
---------------	--

---

**Description**

Identifies the file extension of x.

**Usage**

```
get.extension(x)
```

**Arguments**

x	A string for a filename.
---	--------------------------

**Value**

The extension of x.

---

get.ILP.vars	<i>Retrieval of ILP Decisions</i>
--------------	-----------------------------------

---

**Description**

Retrieves ILP decision variables.

**Usage**

```
get.ILP.vars(ILP, original.dim = NULL)
```

**Arguments**

ILP	A solved ILP instance.
original.dim	Dimension of ILP before using presolve.

**Details**

The original dimension of the ILP is required to determine the correct decisions when presolve has been active and dimensions of the ILP might have changed.

**Value**

The ILP decision variables.

---

get.init.file.name	<i>File Name for Initialized Primers.</i>
--------------------	---

---

**Description**

Constructs a filename for initialized primers.

**Usage**

```
get.init.file.name(
  cur.results.loc,
  GROUP,
  primer.lengths,
  mode.directionality,
  allowed.region.definition,
  init.algo,
  max.degen,
  conservation
)
```

**Arguments**

<code>cur.results.loc</code>	Directory where the file should be stored.
<code>GROUP</code>	Sample name of templates.
<code>primer.lengths</code>	Interval of desired primer lengths.
<code>mode.directionality</code>	Directionality of the primers
<code>allowed.region.definition</code>	Definition of the allowed region.
<code>init.algo</code>	Initialization algorithm identifier.
<code>max.degen</code>	Maximum degeneracy of primers.
<code>conservation</code>	Required ratio of primer conservation.

**Value**

A filename for the initialized primers.

---

`get.leader.exon.regions`  
*Assign Binding Regions*

---

**Description**

Augments a template data frame with individual binding regions.

**Usage**

```
get.leader.exon.regions(lex.seqs, uni.leaders)
```

**Arguments**

<code>lex.seqs</code>	Data frame with template sequences.
<code>uni.leaders</code>	Data frame with individual allowed binding regions.

**Value**

Template data frame with annotated binding regions.

---

```
get.leader.exon.regions.single
```

*Individual Binding Annotation*

---

**Description**

Annotate individual binding regions.

**Usage**

```
get.leader.exon.regions.single(
  l.seq,
  lex.seq,
  direction = c("fw", "rev"),
  gap.char
)
```

**Arguments**

l.seq	Data frame with individual binding regions.
lex.seq	Template data frame.
direction	The primer direction for which the binding info is valid.
gap.char	The character for gaps in alignments.

**Value**

Template data frame with annotated binding regions.

---

```
get.matches
```

*Identification of Sequence Matches.*

---

**Description**

Identifies matches between two strings provided by OligoArrayAux.

**Usage**

```
get.matches(s1, s2)
```

**Arguments**

s1	The aligned nucleotide sequence character vector.
s2	The aligned, matching substring of s1.

**Value**

A match vector (M for matches, X for mismatches).

get.melting.temp.diff *Computation of Maximal Melting Temperature Differences.*

---

**Description**

Computation of Maximal Melting Temperature Differences.

**Usage**

```
get.melting.temp.diff(Tm.fw, Tm.rev)
```

**Arguments**

Tm.fw	The melting temperatures of forward primers.
Tm.rev	The melting temperatures of reverse primers.

**Value**

The worst-case melting temperature difference, for every primer.

---

get.merge.idx *Indices for merging sequences*

---

**Description**

Identifies the indices of similar input sequences to be merged.

**Usage**

```
get.merge.idx(seqs, max.degeneracy)
```

**Arguments**

seqs	The input sequence strings.
max.degeneracy	The maximal allowed degeneracy of a merged seq.

**Value**

A list of lists containing the indices of seqs to be merged. For example `[[1,2,3]]` would indicate to merge primers 1, 2, and 3.

---

get.missing.df	<i>Uncovered Templates.</i>
----------------	-----------------------------

---

**Description**

Computes a data frame containing the templates that are not yet covered.

**Usage**

```
get.missing.df(  
  filtered.df,  
  template.df,  
  Tm.brackets,  
  settings,  
  mode.directionality  
)
```

**Arguments**

filtered.df	An object of class Primers.
template.df	An object of class Templates.

**Value**

A Templates data frame containing the missing templates.

---

get.ORFs	<i>Identification of ORFs.</i>
----------	--------------------------------

---

**Description**

Given a template data frame, identify the exon reading frames in the sequences.

**Usage**

```
get.ORFs(template.df)
```

**Arguments**

template.df	template data frame.
-------------	----------------------

**Value**

Returns a data frame containing the shift of the ORF (either 0,1, or 2) for every sequence, as well as a comment in case of problems.

---

```
get.other.constraint.settings
```

*Gather all Other Constraints (for Shiny frontend).*

---

### Description

Constructor for other constraint settings (non-PCR, non-filtering, non-optimization).

### Usage

```
get.other.constraint.settings(  
  allowed_mismatches,  
  allowed_other_binding_ratio,  
  allowed_region_definition  
)
```

### Arguments

allowed\_mismatches

Allowed mismatches for primers binding events.

allowed\_other\_binding\_ratio

Ratio of primers allowed to bind to non-target regions.

allowed\_region\_definition

The definition of the allowed region.

### Value

List with all other constraint settings.

---

```
get.PCR.settings
```

*Gather all PCR settings.*

---

### Description

Gathers all PCR settings (e.g. for XML output).

### Usage

```
get.PCR.settings(  
  use_taq_polymerase,  
  annealing_temp,  
  Na_concentration,  
  Mg_concentration,  
  K_concentration,  
  Tris_concentration,  
  primer_concentration,  
  template_concentration,  
  nbr_cycles  
)
```

**Arguments**

annealing\_temp Annealing temperature in Celsius.  
Na\_concentration Sodium ion concentration.  
Mg\_concentration Magnesium ion concentration.  
K\_concentration Potassium ion concentration.  
Tris\_concentration Tris buffer concentration.  
primer\_concentration Primer concentration.  
template\_concentration Template concentration.

**Value**

List with all PCR settings.

---

get.plot.height      *Plot Extent*

---

**Description**

Returns the extent of a plot.

**Usage**

```
get.plot.height(N, px.per.n = 50, min.size = 300, max.size = 1500000)
```

**Arguments**

N                    Number of observations to plot.  
px.per.n            Pixels required per observations.  
min.size            Minimal extent of plot in pixels.  
max.size            Maximal extent of plot in pixels.

**Value**

The extent of the plot.

---

`get.primer.binding.idx`*Retrieval of Allowed Binding Indices.*

---

**Description**

Retrieves the indices of allowed binding events in binding for the primer with index `x` and type `primer.type`.

**Usage**

```
get.primer.binding.idx(  
  binding,  
  primer.type = c("fw", "rev", "both"),  
  x,  
  allowed.other.binding.ratio  
)
```

**Arguments**

<code>binding</code>	IRanges binding information.
<code>primer.type</code>	Direction of primer.
<code>x</code>	Index of primer in the primer data frame.
<code>allowed.other.binding.ratio</code>	The ratio of allowed off-target binding events.

**Value**

Indices in binding for primer with index `codex` that are allowed.

---

`get.primer.identifier.string`*Primer Identifier Creation.*

---

**Description**

Creates identifiers for generated primers.

**Usage**

```
get.primer.identifier.string(  
  sample,  
  seq.IDs,  
  seq.identifier,  
  all.starts,  
  all.ends,  
  identifier,  
  seq.primers  
)
```

**Arguments**

<code>sample</code>	Sample name of the templates.
<code>seq.IDs</code>	Identifiers of the templates.
<code>all.starts</code>	Primer positions (start).
<code>all.ends</code>	Primer positions (end).
<code>identifier</code>	Direction keyword.
<code>seq.primers</code>	The primer sequences as strings.
<code>seq.identifiers</code>	The index of the seq.

**Value**

Identifiers for each primer.

---

`get.redundant.cols`     *Identification of Redudant Primers.*

---

**Description**

Identifies primers that are redundant.

**Usage**

```
get.redundant.cols(cvg.matrix)
```

**Arguments**

<code>cvg.matrix</code>	Binary matrix of coverage events.
-------------------------	-----------------------------------

**Details**

Redundant primers do not reduce the coverage when removed.

**Value**

TRUE for redundant primers, FALSE otherwise.

get.relative.binding.pos

*Retrieval of Relative Binding Positions.*

---

### **Description**

Retrieves primer binding position relative to allowed regions of either forward or reverse primers, as specified by direction.

### **Usage**

```
get.relative.binding.pos(allowed, primer.pos, direction, covered.seqs.idx)
```

### **Arguments**

allowed	Positions where binding is allowed in the templates.
primer.pos	Binding position of primer (absolute).
direction	Direction (either fw/rev).
covered.seqs.idx	Indices of covered templates.

### **Value**

Numeric of relative binding position to allowed region.

---

get.run.names

*Getter for Run Names.*

---

### **Description**

Retrieves the run names of the input data.

### **Usage**

```
get.run.names(primer.data)
```

### **Arguments**

primer.data	A list with Primers or Templates.
-------------	-----------------------------------

### **Value**

A vector with identifiers for every set.

---

get.self.dimers            *Self dimerization*

---

**Description**

Computes possible self-dimers.

**Usage**

```
get.self.dimers(
  primers.1,
  primers.2,
  ions,
  annealing.temp,
  no.structures = FALSE
)
```

**Arguments**

primers.1            Input primers  
 primers.2            (Copy/reverse) of the input primers  
 ions                 Sodium-equivalent ionic concentration.  
 annealing.temp     The annealing temperature.  
 no.structures       Whether the dimerization structure shall be computed.

**Value**

Possible self-dimer conformations.

---

get.sets.from.decisions  
                                  *Optimal Sets from Decision Variables*

---

**Description**

Determines primer sets from decision variables from ILP.

**Usage**

```
get.sets.from.decisions(ILP.df, Tm.sets)
```

**Arguments**

ILP.df                Data frame with ILP optimization results.  
 Tm.sets              List with primer data frames for every target melting temperature.

**Value**

A list with optimal primer data sets for every target temperature.

get.static.tool.info    *Retrieval of Tool Information.*

---

**Description**

Constructs a data frame containing information about the tools.

**Usage**

```
get.static.tool.info()
```

**Value**

A data frame with information about the required tools.

---

get.tree.seqs            *Determine Tree Consensus Sequences*

---

**Description**

Creates all possible consensus sequences from a phylogenetic tree.

**Usage**

```
get.tree.seqs(tree, max.degen, primer.candidates)
```

**Arguments**

tree	The phylogenetic tree.
max.degen	The maximal degeneration of consensus primers.
primer.candidates	Alignment of primers.

**Details**

Ambiguous sequences are only generated with a degeneracy of at most max.degen. The tree is iterated from leaves to the top, i.e., starting from least degeneracy to most degeneracy. Merges only take place when the degeneracy of the resulting sequence would be at most max.degen. Gaps are removed from the alignments.

**Value**

Data frame with consensus primers extracted from the tree.

---

get.unlist.idx      *Index for Unlisting.*

---

**Description**

Determines indices for unlisting.

**Usage**

```
get.unlist.idx(primer.start, primer.data.idx)
```

**Arguments**

primer.start      Numeric vector.  
primer.data.idx      Selection indices.

**Value**

Indices.

---

get\_constraint\_deviation\_data  
*Retrieve data for Constraint Deviations.*

---

**Description**

Retrieve data for Constraint Deviations.

**Usage**

```
get_constraint_deviation_data(constraint.df, constraint.settings)
```

**Arguments**

constraint.df      An evaluated object of class Primers.  
constraint.settings      A list with settings for the constraints that are to be evaluated.

**Value**

A data frame providing primer-specific information on deviations of primer properties from the desired properties.

---

`get_covered.vanilla`     *Determination of the Covered Sequences.*

---

### **Description**

Determines the covered template sequences given by `template.df` that are covered by the primers given by `primers`.

### **Usage**

```
get_covered.vanilla(primers, template.df, mode.directionality = NULL)
```

### **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.

`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. If `mode.directionality` is not provided the direction is determined by the input primers.

### **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**

The IDs of all covered templates.

---

`get_cvg_stats,list-method`  
                                   *Coverage Statistics for Multiple Primer Sets.*

---

### **Description**

Retrieve statistics on covered templates for multiple primer sets.

**Usage**

```
## S4 method for signature 'list'
get_cvgs_stats(
  primers,
  templates,
  for.viewing = FALSE,
  total.percentages = FALSE,
  allowed.mismatches = Inf,
  cvg.definition = c("constrained", "basic")
)
```

**Arguments**

primers	A list with objects of class Primers containing primers with evaluated coverage.
templates	A list with objects of class Templates containing templates with evaluated coverage.
for.viewing	Whether the table should be formatted for viewing rather than processing.
total.percentages	Whether group coverage percentages should relate to all template sequences or just those templates belonging to a specific group.
allowed.mismatches	The maximal allowed number of mismatches. By default, the number of mismatches is not restricted.
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 with coverage statistics.

---

```
get_cvgs_stats,Primers-method
```

*Coverage Statistics of a Primer Set.*

---

**Description**

Retrieve statistics on the templates that are covered by a primer set.

**Usage**

```
## S4 method for signature 'Primers'
get_cvgs_stats(
  primers,
  templates,
  for.viewing = FALSE,
  total.percentages = FALSE,
  allowed.mismatches = Inf,
  cvg.definition = c("constrained", "basic")
)
```

**Arguments**

for.viewing	Whether the table should be formatted for viewing rather than processing.
total.percentages	Whether group coverage percentages should relate to all template sequences or just those templates belonging to a specific group.
allowed.mismatches	The maximal allowed number of mismatches. By default, the number of mismatches is not restricted.
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".
primer.df	An object of class Primers containing primers with evaluated coverage.
template.df	An object of class Templates containing templates with evaluated coverage.

**Value**

Data frame with coverage statistics.

---

get\_max\_set\_coverage *Determination of Maximal Coverage.*

---

**Description**

Determines the maximal coverage ratio of a set of primers for primer subsets valid for a certain temperature range. a certain melting temperature range.

**Usage**

```
get_max_set_coverage(
  primer.df,
  template.df,
  Tm.brackets,
  settings,
  mode.directionality,
  max.only = TRUE
)
```

**Arguments**

primer.df	An object of class Primers.
template.df	An object of class Templates.
Tm.brackets	A data frame with temperature information.
settings	A DesignSettings object.
mode.directionality	The direction of the primers.
max.only	Whether only the maximum coverage shall be returned. If max.only is FALSE, the coverage ratios of all melting temperature sets according to Tm.brackets are returned.

**Value**

The maximal coverage ratio of a primer set if `max.only` is TRUE or the coverages of all melting temperature sets if `max.only` is FALSE.

---

get\_plot\_primer\_data *Data for Primer Plot.*

---

**Description**

Constructs a data frame containing information about primer binding events.

**Usage**

```
get_plot_primer_data(
  primer.df,
  template.df,
  identifier = NULL,
  relation = c("fw", "rev")
)
```

**Arguments**

<code>primer.df</code>	An object of class <code>Primers</code> containing primers with evaluated primer coverage.
<code>template.df</code>	An object of class <code>Templates</code> with template sequences corresponding to <code>primer.df</code> .
<code>identifier</code>	Identifiers of primers that are to be considered. If <code>identifier</code> is set to <code>NULL</code> (the default), all primers are considered.
<code>relation</code>	Compute binding positions relative to forward ( <code>fw</code> ) or reverse ( <code>rev</code> ) binding regions. The default is <code>"fw"</code> .

**Value**

Data frame with primer binding data.

---

get\_primer\_cvg\_mm\_plot\_df  
*Data for Mismatch Primer Coverage Plot.*

---

**Description**

Ensures that there's an entry for every possible mismatch setting.

**Usage**

```
get_primer_cvg_mm_plot_df(primer.df, template.df)
```

**Arguments**

<code>primer.df</code>	A <code>Primers</code> object.
<code>template.df</code>	A <code>Templates</code> object.

**Value**

A data frame for plotting mismatch primer coverage.

---

get\_report\_fname      *Creation of a Filename for Reports.*

---

**Description**

Creates the filename for reports.

**Usage**

```
get_report_fname(report.name, sample.name)
```

**Arguments**

report.name      The identifier for the report type.  
sample.name      The identifier of the sample that was analyzed.

**Value**

A character vector.

---

get\_template\_cvg\_data      *Retrieval of Template Coverage Data.*

---

**Description**

Determines the coverage of the templates for individual allowed mismatch settings and coverage definitions.

**Usage**

```
get_template_cvg_data(primer.df, template.df)
```

**Arguments**

primer.df      A Primers object.  
template.df      A Templates object.

**Value**

Computes a data frame providing the coverage of the templates for the basic as well as expected (constrained) coverage.

---

hclust.tree	<i>Hierarchical Clustering.</i>
-------------	---------------------------------

---

**Description**

Performs hierarchical clustering on aligned primer sequences.

**Usage**

```
hclust.tree(primer.candidates)
```

**Arguments**

primer.candidates  
Alignment of primer candidates.

**Details**

The clustering is performed to identify similar groups of primer candidates that can be merged to form degenerate primers.

**Value**

Phylogeny of the input primer.candidates.

---

highlight.mismatch	<i>Highlight mismatches</i>
--------------------	-----------------------------

---

**Description**

Collects information on the mutations present in the input and highlights the mutations in the sequence.

**Usage**

```
highlight.mismatch(seq, mm.seq)
```

**Arguments**

seq                    character vector of the original sequence  
mm.seq                character vector of the mutated sequence

**Value**

A list highlighting the mutations and additional information (mutation type, number, etc.)

`html.format.structure` *Formats a Dimerization Structure for HTML.*

---

**Description**

Formats a Dimerization Structure for HTML.

**Usage**

```
html.format.structure(structures)
```

**Arguments**

`structures`      A character vector of dimerization structures.

**Value**

HTML-formatted character vectors.

---

`I.cvg`              *Primer Coverage.*

---

**Description**

Determines the indices of covered templates for every primer.

**Usage**

```
I.cvg(cvg.matrix)
```

**Arguments**

`cvg.matrix`      Binary matrix of covering events.

**Value**

A list with covered templates for every primer.

---

ILPConstrained	<i>Construct Coverage ILP.</i>
----------------	--------------------------------

---

**Description**

Constructs an ILP modeling the primer set cover problem.

**Usage**

```
ILPConstrained(D, cvg.matrix, time.limit = NULL, presolve.active = FALSE)
```

**Arguments**

D	Binary dimerization matrix.
cvg.matrix	Binary coverage matrix.
time.limit	Time limit for ILP optimization in seconds.
presolve.active	Whether the ILP presolver should be used. This is set to FALSE by default, since presolving may lead to inferior solutions. However, for large problems presolving might be useful.

**Value**

An instance of the set cover ILP.

---

initialize.primers.set	<i>Creation of Initial Primers</i>
------------------------	------------------------------------

---

**Description**

Creates a set of candidate primers.

**Usage**

```
initialize.primers.set(
  template.df,
  sample.name,
  primer.lengths,
  allowed.region.definition,
  mode.directionality,
  init.algo,
  max.degen,
  conservation,
  cur.results.loc
)
```

**Arguments**

<code>template.df</code>	Template data frame.
<code>sample.name</code>	Name of the template sample.
<code>primer.lengths</code>	Interval of minimal and maximal desired primer length.
<code>allowed.region.definition</code>	Definition of the allowed binding region.
<code>mode.directionality</code>	Direction of primers to be created.
<code>init.algo</code>	Algorithm for initializing primers.
<code>max.degen</code>	Maximal allowed degeneration of created primers.
<code>conservation</code>	Required conservation of primers. The value of conservation should be in the range[0,1].
<code>cur.results.loc</code>	Location for writing the primers as csv.

**Value**

An initial primer data frame.

---

Input

*Input Functionalities.*

---

**Description**

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

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

`read_settings` Loads primer analysis settings from an XML file.

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

**Primers** 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**

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

```
read_templates(
  fname,
  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.char = "-",
    run = NULL
  )

Primers(...)

read_primers(
  fname,
  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
)

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

```

### Arguments

...	A data frame fulfilling the structural requirements for initializing a Templates or Primers object.
fname	Character vector providing either a single or multiple paths to FASTA or CSV files.
hdr.structure	A character vector describing the information contained in the FASTA headers. In case that the headers of <code>fasta.file</code> contain template group information, please include the keyword "GROUP" in <code>hdr.structure</code> . If the number of elements provided via <code>hdr.structure</code> 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 <code>fw.region</code> 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 <code>rev.region</code> is set to the last 30 bases of the templates.
gap.char	The character in the input file representing gaps. Gaps are automatically removed upon input and the default character is "-".

<code>run</code>	An identifier for the set of template sequences. By default, <code>run</code> is NULL and its value is set via <code>template.file</code> .
<code>fw.id</code>	For FASTA input, the identifier for forward primers in the FASTA headers.
<code>rev.id</code>	For FASTA input, the identifier for reverse primers in the FASTA headers.
<code>merge.ambig</code>	Indicates whether similar primers should be merged ("merge") using IUPAC ambiguity codes or whether primers should be disambiguated ("unmerge"). By default <code>merge.ambig</code> is set to "none", leaving primers as they are.
<code>max.degen</code>	A scalar numeric providing the maximum allowed degeneracy for merging primers if <code>merge.ambig</code> is set to "merge". Degeneracy is defined by the number of disambiguated sequences that are represented by a degenerate primer.
<code>template.df</code>	An object of class <code>Templates</code> . If <code>template.df</code> is provided for <code>read_primers</code> then the primers are checked for restriction sites upon input; otherwise they are not checked.
<code>adapter.action</code>	The action to be performed when <code>template.df</code> 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".
<code>sample.name</code>	An identifier for the input primers.
<code>updateProgress</code>	A Shiny progress callback function. This is NULL by default such that no progress is tracked.
<code>filename</code>	Path to a valid XML file containing the primer analysis settings. By default, <code>filename</code> is set to all settings that are shipped with <code>openPrimeR</code> and the lexicographically first file is loaded.
<code>frontend</code>	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

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.

When loading a FASTA file with `read_templates`, 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 a CSV file is loaded with `read_templates`, 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.

When loading primers via `read_primers`, the input arguments `fw.id`, `rev.id`, `merge.ambig`, and `max.degen` are only used for loading primers from a FASTA file. In this case, please ensure that `fw.id` and `rev.id` are set according to the keywords indicating the primer directionalities in the FASTA file. When loading primers from a CSV file, the format of the file should adhere to the structure defined by the `Primers` class.

When loading a settings file with `read_settings`, 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

The `Templates` constructor returns a `Templates` object, an instance of a data frame.

`read_templates` returns a single object of class `Templates` if a single filename was provided or a list of such objects if multiple file names were provided.

The `Primers` constructor returns an object of class `Primers`.

`read_primers` returns a single object of class `Primers` if a single input file is provided or a list of such objects if multiple files are provided.

`read_settings` returns an object of class `DesignSettings`.

## 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 in an object of class `Primers` 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.

### Examples

```
# Load a set of templates:
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")
# 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 multiple 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)

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

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)

# Select available settings
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)

```

---

insert\_str

*String Insertion.*


---

## Description

Inserts a string into another string at the specified position.

## Usage

```
insert_str(target, insert, index)
```

**Arguments**

target	The string to be modified.
insert	The string to be inserted.
index	The position where the insertion should take place.

**Value**

A string where insert is inserted into target at position index.

---

interleave	<i>Interleave strings Combines the input vectors in an interleaved fashion.</i>
------------	---

---

**Description**

Interleave strings Combines the input vectors in an interleaved fashion.

**Usage**

```
interleave(v1, v2)
```

**Arguments**

v1	Input string.
v2	Input string.

**Value**

The interleaved combination of v1 and v2.

---

J.cvg	<i>Template Coverage.</i>
-------	---------------------------

---

**Description**

Determines the indices of covering primers for every template.

**Usage**

```
J.cvg(cvg.matrix)
```

**Arguments**

cvg.matrix	Binary matrix of covering events.
------------	-----------------------------------

**Value**

A list with covering primers for every template.

---

joule.to.cal	<i>Conversion from J to cal</i>
--------------	---------------------------------

---

**Description**

Converts the input from Joule to calories.

**Usage**

```
joule.to.cal(val.J)
```

**Arguments**

val.J            Numeric Joule value.

**Value**

The value corresponding to val.J in calories.

---

listToXml	<i>List to XML</i>
-----------	--------------------

---

**Description**

Convert List to XML.

**Usage**

```
listToXml(item, tag)
```

**Arguments**

item  
tag            xml tag

**Details**

Can convert list or other object to an xml object using xmlNode.

**Value**

xmlNode

**Author(s)**

David LeBauer, Carl Davidson, Rob Kooper

merge.ambig.primers     *Merge similar primers*

---

### Description

Merges similar primers contained in the input primer data frame.

### Usage

```
## S3 method for class 'ambig.primers'  
merge(primer.df, mode.directionality = c("fw", "rev", "both"), max.degeneracy)
```

### Arguments

primer.df            Primer data frame.  
mode.directionality            Analysis direction.  
max.degeneracy    Maximal degeneracy of merged primers.

### Value

A primer data frame where similar primers are merged into one entry.

---

merge.binding.information  
                          *Merge of Forward/Reverse Binding Information.*

---

### Description

Determines binding events of individual and pairs of primers.

### Usage

```
## S3 method for class 'binding.information'  
merge(  
  primers,  
  fw.binding.filtered,  
  rev.binding.filtered,  
  mode.directionality = c("fw", "rev", "both"),  
  idx.fw,  
  idx.rev  
)
```

**Arguments**

primers	The primer data frame.
mode.directionality	Primer directionality.
idx.fw	Index of fw primers.
idx.rev	Index of rev primers.
fw.binding	IRanges object with binding events of fw primers.
rev.binding	IRanges object with binding events of rev primers.

**Value**

IRanges with correct binding events.

---

merge.primer.entries *Merge similar primers*

---

**Description**

Merges the entries of similar entries in the input primer data frame, given a list with merge indices.

**Usage**

```
## S3 method for class 'primer.entries'  
merge(opti.result, merge.idx, mode.directionality = c("fw", "rev", "both"))
```

**Arguments**

opti.result	Input primer data frame.
merge.idx	List of lists with merge indices (get.merge.idx).
mode.directionality	Direction of primers.

**Value**

A primer data frame where entries of similar primers are merged.

merge.primer.entries.single  
*Merge input sequences*

---

**Description**

Merges the input sequences given a list with merge indices.

**Usage**

```
## S3 method for class 'primer.entries.single'  
merge(seqs, merge.idx)
```

**Arguments**

seqs	The input sequences.
merge.idx	List of list with merge indices.

**Value**

Merged input sequences according to the input merge indices.

---

merge.select      *Select merge indices*

---

**Description**

Greedly identifies the smallest number of possible sequences merges that can be performed.

**Usage**

```
## S3 method for class 'select'  
merge(merge.idx)
```

**Arguments**

merge.idx	list of lists containing the indices of possible merges
-----------	---

**Value**

The smallest number of possible merge operations as an index list.

---

`merge.template.decisions`*Merge Template Decisions.*

---

**Description**

Merges the results for multiple template evaluations.

**Usage**

```
## S3 method for class 'template.decisions'  
merge(eval.t)
```

**Arguments**

`eval.t` List with evaluated template constraints

**Value**

List with merged boolean decisions.

---

`mismatch.info`*Information about Mismatches.*

---

**Description**

Computes information about mismatch binding events.

**Usage**

```
mismatch.info(primer, seqs)
```

**Arguments**

`primer` Primer character vector.

`seqs` Template binding sequences of primers as a XStringsView object.

**Value**

List with positions and number of mismatches of the primer in the seqs. The list contains the field `mm.pos` containing a list with the positions of the mismatches and the field `Nbr` containing a numeric vector with the number of mismatches per template binding event.

---

`mismatch.mutation.check`*Identification of Mutations Induced by Mismatch Binding Events.*

---

**Description**

Identifies whether mutations are induced by mismatch binding events.

**Usage**

```
mismatch.mutation.check(  
  primer.df,  
  template.df,  
  mutation.types = c("stop_codon", "substitution")  
)
```

**Arguments**

`primer.df`      A Primers object.  
`template.df`    A Template object.  
`mutation.types` Character vector of the mutation types to be checked for.

**Details**

Checks for one primer and all covered templates whether any templates are bound with mismatches such that mismatches are induced. A numeric vector indicating which binding events induce a forbidden mismatch according to `mutation.types` is returned such that 1 indicates forbidden events and 0 allowed events.

**Value**

A list containing data frames where an entry of 1 is present if the `primer.seq` induces a mutation that is forbidden according to the provided `mutation.types`, otherwise 0.

---

`mismatch.string.to.list`*Conversion of Mismatch Postions String to List.*

---

**Description**

Conversion of Mismatch Postions String to List.

**Usage**

```
mismatch.string.to.list(mismatches)
```

**Arguments**

`mismatches`    A character vector where parenthesis give mismatches for a template binding event.

**Value**

A list with the mismatches for every template for every primer.

---

modify.col.rep	<i>Modification of Column Names.</i>
----------------	--------------------------------------

---

**Description**

Modifies column names for frontend output.

**Usage**

```
modify.col.rep(template.df, for.shiny = TRUE)
```

**Arguments**

template.df	The data frame whose column names are to be modified.
for.shiny	Whether formatting should be for shiny.

**Value**

template.df with modified column names.

---

my.disambiguate	<i>Disambiguation of Sequences.</i>
-----------------	-------------------------------------

---

**Description**

Disambiguates the input sequences, but does not disambiguate highly degenerate sequences.

**Usage**

```
my.disambiguate(template.seqs, gap.char = "-", degen.cutoff = 2^10)
```

**Arguments**

template.seqs	A DNASTringSet object with sequences to disambiguate.
gap.char	The character indicating gaps in alignments.
degen.cutoff	The maximal degeneration of sequences to be disambiguated.

**Value**

A DNASTringSetList object with disambiguated sequences.

`my.error`*Custom Error*

---

**Description**

Creates an error with a custom class.

**Usage**

```
my.error(subclass, message, call = sys.call(-1), ...)
```

**Arguments**

<code>subclass</code>	String giving the specific type of error.
<code>message</code>	Message to be displayed to the user.
<code>call</code>	Environment where the error occurred.
<code>...</code>	Other arguments to be passed to the condition function.

**Value**

Generates a custom error.

---

`my.read.fasta`*Read FASTA File.*

---

**Description**

Reads the input FASTA file.

**Usage**

```
my.read.fasta(fasta.file, NTs)
```

**Arguments**

<code>fasta.file</code>	The path to a FASTA file.
<code>NTs</code>	The allowed set of nucleotides.

**Value**

List with vectors of chars.

---

my.warning	<i>Custom Warning.</i>
------------	------------------------

---

**Description**

Creates a warning with a custom class.

**Usage**

```
my.warning(subclass, message, call = sys.call(-1), ...)
```

**Arguments**

subclass	String giving the specific type of error.
message	Message to be displayed to the user.
call	Environment where the error occurred.
...	Other arguments to the condition function.

**Value**

Generates a custom warning.

---

my_ggsave	<i>Wrapper for the ggplot2::ggsave function.</i>
-----------	--

---

**Description**

Saves a plot using ggplot2's ggsave function.

**Usage**

```
my_ggsave(filename, plot = ggplot2::last_plot(), ...)
```

**Arguments**

filename	The filename to store the plot.
plot	The ggplot object.
...	Further arguments to the ggplot2 ggsave function.

**Value**

Stores p in fname.

---

my_rbind	<i>Smartbind preserving classes.</i>
----------	--------------------------------------

---

**Description**

Rbind allowing for column mismatch, retains the classes of the data frames. Motivation: smartbind/rbind.fill only keep the data.frame class but not additional classes.

**Usage**

```
my_rbind(...)
```

**Arguments**

... Data frames.

**Value**

A data frame resulting from row binding of ...

---

nbr.of.repeats	<i>Number of Repeats</i>
----------------	--------------------------

---

**Description**

Computes the number of dinucleotide repeats in the input sequences.

**Usage**

```
nbr.of.repeats(x)
```

**Arguments**

x Input sequence strings.

**Value**

The maximal number of dinucleotide repeats in x.

---

nbr.of.runs	<i>Number of Runs</i>
-------------	-----------------------

---

**Description**

Computes the longest run of a single character in the input sequence.

**Usage**

```
nbr.of.runs(x)
```

**Arguments**

x                   Primer character sequences.

**Value**

The longest repeat of a single character in x.

---

opti	<i>Getter for Optimization Constraints.</i>
------	---

---

**Description**

Gets the constraints on the physicochemical properties that are applied just before the optimization procedure using the Input\_Constraints slot of the provided DesignSettings object x.

**Usage**

```
opti(x)
```

```
## S4 method for signature 'DesignSettings'  
opti(x)
```

**Arguments**

x                   A DesignSettings object.

**Value**

Gets the list of optimization constraints.

---

optiLimits	<i>Getter for Optimization Constraint Limits.</i>
------------	---

---

**Description**

Gets the limits for the constraints that are applied just before the optimization procedure using the Input\_Constraint\_Boundaries slot of the provided DesignSettings object x.

**Usage**

```
optiLimits(x)

## S4 method for signature 'DesignSettings'
optiLimits(x)
```

**Arguments**

x                    A DesignSettings object.

**Value**

Gets the optimization constraint limits.

---

optimize.ILP	<i>Solver for ILP Set Cover</i>
--------------	---------------------------------

---

**Description**

Solves the primer set cover problem using an ILP formulation.

**Usage**

```
optimize.ILP(
  primer.df,
  template.df,
  settings,
  primer_conc,
  template_conc,
  na_salt_conc,
  mg_salt_conc,
  k_salt_conc,
  tris_salt_conc,
  allowed.mismatches,
  allowed.other.binding.ratio,
  allowed.stop.codons,
  allowed.region.definition,
  disallowed.mismatch.pos,
  target.temps,
  required.cvg,
  fw.primers = NULL,
```

```

    diagnostic.location = NULL,
    timeout = Inf,
    updateProgress = NULL
  )

```

### Arguments

<code>primer.df</code>	Primer data frame to be optimized.
<code>template.df</code>	Template data frame with sequences.
<code>settings</code>	A <code>DesignSettings</code> object.
<code>primer_conc</code>	Primer concentration.
<code>template_conc</code>	Template concentration.
<code>na_salt_conc</code>	Sodium ion concentration.
<code>mg_salt_conc</code>	Magnesium ion concentration.
<code>k_salt_conc</code>	Potassium ion concentration.
<code>tris_salt_conc</code>	Tris ion concentration.
<code>allowed.mismatches</code>	The number of mismatches primers are allowed to have with the templates.
<code>allowed.other.binding.ratio</code>	Ratio of primers allowed to bind to non-target regions.
<code>allowed.stop.codons</code>	Consider mismatch binding events that induce stop codons.
<code>allowed.region.definition</code>	Definition of the target binding sites used for evaluating the coverage. If <code>allowed.region.definition</code> is <code>within</code> , primers have to lie within the allowed binding region. If <code>allowed.region.definition</code> is <code>any</code> , primers have to overlap with the allowed binding region. The default is that primers have to bind within the target binding region.
<code>disallowed.mismatch.pos</code>	The number of positions from the primer 3' end where mismatches should not be allowed. All primers binding templates with mismatches within <code>disallowed.mismatch.pos</code> from the 3' end are disregarded.
<code>target.temps</code>	Target melting temperatures for primer sets in Celsius.
<code>required.cvg</code>	Target coverage ratio of the templates by the primers.
<code>fw.primers</code>	List with optimized primer data frames corresponding to <code>target.temps</code> . Only required for optimizing both strand directions and only in the second optimization run in order to check for cross dimerization.
<code>diagnostic.location</code>	Directory for storing results.
<code>timeout</code>	Timeout in seconds for the optimization with ILPs.
<code>updateProgress</code>	Shiny progress callback function.

### Value

List with optimization results.

---

optimize.primer.cvg     *Greedy Optimization*

---

### Description

Greedy approach for solving the primer set coverage problem.

### Usage

```
optimize.primer.cvg(  
  primers,  
  template.df,  
  mode.directionality,  
  cur.opti.constraints,  
  target.temp,  
  allowed.mismatches,  
  opti.limits,  
  primer_conc,  
  na_salt_conc,  
  mg_salt_conc,  
  k_salt_conc,  
  tris_salt_conc,  
  updateProgress = NULL  
)
```

### Arguments

primers	Primer data frame to be optimized.
template.df	Template data frame.
mode.directionality	Primer direction.
cur.opti.constraints	List with optimization constraint settings.
target.temp	Target annealing temperature of the optimized primer set in Celsius.
allowed.mismatches	The number of mismatches primers are allowed to have with the templates.
opti.limits	List with optimization limits.
primer_conc	Primer concentration.
na_salt_conc	Sodium ion concentration.
mg_salt_conc	Magnesium ion concentration.
k_salt_conc	Potassium ion concentration.
tris_salt_conc	Tris ion concentration.
updateProgress	Shiny progress callback function.
template_conc	Template concentration.

### Value

List with optimization data.

---

```
optimize.template.binding.regions.dir
```

*Optimization of Binding Regions*

---

**Description**

Optimizes the template binding regions.

**Usage**

```
optimize.template.binding.regions.dir(  
  template.df,  
  annealing.temperature = NULL,  
  primer.lengths,  
  mode.directionality = c("fw", "rev", "both")  
)
```

**Arguments**

`template.df`      Template data frame.  
`annealing.temperature`  
                    Temperature at which to compute secondary structures.  
`primer.lengths`    Target length of primers that are to be used.  
`mode.directionality`  
                    Direction of primers.

**Value**

List with intervals indicating improved primer binding regions.

---

```
optimize.template.binding.regions.single
```

*Optimization of Template Binding*

---

**Description**

Optimizes template binding regions according to secondary structures.

**Usage**

```
optimize.template.binding.regions.single(  
  template.df,  
  annealing.temperature,  
  primer.lengths,  
  mode.directionality = c("fw", "rev")  
)
```

**Arguments**

`template.df`      Template data frame.  
`annealing.temperature`  
                     Temperature at which to compute secondary structures.  
`primer.lengths`    Target length of primers that are to be used.  
`mode.directionality`  
                     Direction of primers.

**Value**

List with new binding intervals for every template.

---

Output	<i>Output Functionalities.</i>
--------	--------------------------------

---

**Description**

`write_primers`    Writes a set of primers to disk, either as a FASTA or CSV file.  
`write_settings`   Stores primer analysis settings to a file in XML format.  
`write_templates` Stores a set of templates as a FASTA or CSV file.  
`create_report`    Creates a PDF report for analyzed primer sets.  
`create_coverage_xls` 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**

```

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

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

create_coverage_xls(primer.df, template.df, fname, settings)

create_report(
  primers,
  templates,
  fname,
  settings,
  sample.name = NULL,
  used.settings = NULL,
  ...
)

write_settings(settings, fname)
  
```

**Arguments**

<code>template.df</code>	An object of class <code>Templates</code> .
<code>fname</code>	The path to the output file.
<code>ftype</code>	A character vector giving the type of the file. This can either be "FASTA" or "CSV" (default: "FASTA").
<code>primer.df</code>	An object of class <code>Primers</code> .
<code>settings</code>	A <code>DesignSettings</code> object to be stored to disk.
<code>primers</code>	To create a report for a single primer set, please provide an evaluated <code>Primers</code> object. For creating a report comparing multiple primer sets, please provide a list of <code>Primers</code> objects.
<code>templates</code>	If <code>primers</code> is a <code>Primers</code> object, <code>templates</code> should be a <code>Templates</code> object. If <code>primers</code> is a list of <code>Primers</code> objects, <code>templates</code> should be a list of <code>Templates</code> objects of the same length as <code>primers</code> .
<code>sample.name</code>	An identifier for your analysis. By default ( <code>NULL</code> ), the sample identifier is selected from the <code>Run</code> column of the input <code>templates</code> .
<code>used.settings</code>	A named list (with fields <code>fw</code> and <code>rev</code> ) containing the relaxed settings for designing forward/reverse primers. By default ( <code>NULL</code> ), the relaxed settings are not shown in the report.
<code>...</code>	<code>required.cvg</code> (optional, default: 1), the desired coverage ratio if <code>primers</code> is a single primer set.

**Value**

`write_templates` stores `templates` to `fname`.

`write_primers` stores `primers` to disk.

`create_coverage_xls` stores information on the primer coverage in a spreadsheet.

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

`write_settings` returns the status from closing the connection to the output file.

**Note**

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

**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")

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")

data(Ippolito)
filename <- tempfile("cvg_overview", fileext = ".xls")
# Store coverage of a single primer in an XLS file:
my.primers <- primer.df[3,]
cvd <- unique(unlist(strsplit(my.primers$Covered_Seqs, split = ",")))
m <- match(cvd, template.df$Identifier)
my.templates <- template.df[m,]
create_coverage_xls(my.primers, my.templates, filename, settings)

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)
set.sizes <- sapply(primer.data, nrow)
sel.sets <- order(set.sizes)[1:2]
out.file.comp <- tempfile("comparison_report", fileext = ".pdf")
create_report(primer.data[sel.sets], template.data[sel.sets], out.file.comp, settings)

# Store settings to disk
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)

```

---

pair\_primers

*Pairing of Forward and Reverse Primers.*


---

## Description

Pairs forward and reverse primers such that coverage is maximized for every pair.

## Usage

```
pair_primers(primer.df, template.df)
```

## Arguments

`primer.df`      An object of class `Primers`.

## Value

An object of class `Primers` containing the paired primers.

---

parse.constraints	<i>Parse XML Constraint Data.</i>
-------------------	-----------------------------------

---

**Description**

Parses the constraint settings contained in an XML object.

**Usage**

```
parse.constraints(xml_data)
```

**Arguments**

xml_data	XML object from a parsed XML file.
----------	------------------------------------

**Value**

List with constraint settings.

---

parse.header	<i>Parse FASTA headers</i>
--------------	----------------------------

---

**Description**

Parses the headers of a FASTA file.

**Usage**

```
parse.header(hdr, delim, hdr.str, id.column)
```

**Arguments**

hdr	The headers (> entries) of a FASTA File.
delim	The delimiter used to separate distinct fields in the headers. For example,   for headers such as > E.coli   GeneX   ...
hdr.str	Names of the fields appearing in the header.
id.column	Field in the header to be used used as an identifier for the sequences.

**Value**

Data frame with structured information from the headers.

parse.IMG.T.gene.groups

*Parser for IMG.T Groups.*

---

### **Description**

Parses IMG.T group information contained in FASTA headers.

### **Usage**

parse.IMG.T.gene.groups(IDs)

### **Arguments**

IDs                      Group information strings to be parsed.

### **Value**

Data frame with structured group information.

---

parse.oligo.results

*Parser for OligoArrayAux Dimerization Data.*

---

### **Description**

Parses the free energies and structures of OligoArrayAux.

### **Usage**

parse.oligo.results(deltaG.file, struct.file)

### **Arguments**

deltaG.file            A path to a file with OligoArrayAux energies.

struct.file            A path to a file with OligoArrayAux structures.

### **Value**

A data frame with structures and free energies.

---

plot.all.cvg.info      *Plots Coverage Information*

---

### Description

Visualizes all coverage-related data.

### Usage

```
## S3 method for class 'all.cvg.info'
plot(
  sample,
  results.loc,
  primers,
  template.df,
  mode.directionality,
  identifier = c("filtering", "optimized"),
  primer_conc,
  na_salt_conc,
  mg_salt_conc,
  k_salt_conc,
  tris_salt_conc,
  settings,
  required.cvg,
  used.settings = NULL
)
```

### Arguments

sample	Primer design run identifier.
results.loc	Location where the filtering results are stored.
primers	Primer data frame.
template.df	Template data frame.
mode.directionality	Design direction.
identifier	Identifies whether filtering or optimization info should be displayed.
primer_conc	Primer concentration.
na_salt_conc	Sodium ion concentration.
mg_salt_conc	Magnesium ion concentration.
k_salt_conc	Potassium ion concentration.
tris_salt_conc	Tris ion concentration.
settings	The DesignSettings object.
required.cvg	The required coverage.
used.settings	A list containing DesignSettings objects for the 'fw' and 'rev' optimization.

### Value

Writes plots to files.

---

```
plot.all.filtering.stats
```

*Plot Filtering Stats.*

---

### Description

Plots filtering statistics.

### Usage

```
## S3 method for class 'all.filtering.stats'  
plot(  
  results.loc,  
  sample,  
  excluded.df,  
  filtered.stats,  
  stats.relax,  
  template.df  
)
```

### Arguments

results.loc	Location where the filtering results are stored.
sample	Primer design run identifier.
excluded.df	Data frame with excluded primers.
filtered.stats	Filtering statistics data frame.
stats.relax	Filtering statistics after relaxation.
template.df	Template data frame.

### Value

Write-out of filtering results.

---

```
plot.Delta.DeltaG
```

*Delta DeltaG Plot*

---

### Description

Plots the difference between the free energy of constrained and unconstrained foldings.

### Usage

```
## S3 method for class 'Delta.DeltaG'  
plot(constrained.foldings, stratify = FALSE)
```

**Arguments**

constrained.foldings      Data frame with info from constrained foldings.  
 stratify                  Stratify according to template groups?

**Value**

Plot of Delta DeltaG.

---

plot.dimer.dist              *Plot Dimer DeltaG*

---

**Description**

Plot the distribution of dimerization free energies.

**Usage**

```
## S3 method for class 'dimer.dist'
plot(dimer.data, deltaG.cutoff)
```

**Arguments**

dimer.data              Data frame with dimerization information.  
 deltaG.cutoff      Free energy cutoff for dimerization.

**Value**

A plot of dimerization free energies.

---

plot.excluded.hist          *Plot of Excluded Primers*

---

**Description**

Plots histogram of excluded primers.

**Usage**

```
## S3 method for class 'excluded.hist'
plot(excluded.df, filtered.stats, template.df)
```

**Arguments**

excluded.df              Data frame with excluded primers.  
 filtered.stats      Data frame with statistics of the filtering procedure.  
 template.df              Template data frame.

**Value**

A plot of excluded primers.

plot.filtering.runtime

*Plot Filtering Runtimes*

---

**Description**

Plots the runtimes of individual evaluation steps in the filtering procedure.

**Usage**

```
## S3 method for class 'filtering.runtime'  
plot(filtered.stats)
```

**Arguments**

filtered.stats    Stats from filtering.

**Value**

A plot of the runtime for each filtering step.

---

plot.filtering.stats    *Plot of Overall Filtering Stats.*

---

**Description**

Plots the number of primers remaining after each filtering step.

**Usage**

```
## S3 method for class 'filtering.stats'  
plot(stats, stats.relax = NULL)
```

**Arguments**

stats                    Statistics on the filtering procedure  
stats.relax             Statistic on the filtering procedure after relaxation.

**Value**

A plot for the number of primers after filtering.

---

```
plot.filtering.stats.cvg
```

*Plot of Filtering Stats for Coverage.*

---

### Description

Plots the remaining coverage after each filtering step.

### Usage

```
## S3 method for class 'filtering.stats.cvg'
plot(stats, stats.relax = NULL)
```

### Arguments

`stats`                 Statistics of the filtering procedure.  
`stats.relax`         Statistic of the relaxation procedure.

### Value

A plot showing the possible coverage after each filtering step.

---

Plots

*Plotting Functions.*

---

### Description

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

`plot_penalty_vs_set_size` 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`.

`plot_primer_subsets` Visualizes the coverage of optimized primer subsets.

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

`plot_template_cvg` Creates a bar plot visualizing the covered templates.

`plot_primer_cvg` Shows which groups of templates are covered by individual primers.

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

`plot_constraint_fulfillment` Visualizes which primers pass the constraints and which primers break the constraints

`plot_cvg_constraints` Plots the distribution of the coverage constraint values.

`plot_constraint_deviation` Plots the deviation of primer properties from the target ranges.

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

`plot_conservation` 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 = "-")

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

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

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

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

plot_constraint_deviation(
  primers,
  settings,
  active.constraints = names(constraints(settings)),
  ...
)

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

plot_penalty_vs_set_size(
  primer.data,
  settings,
```

```

    active.constraints = names(constraints(settings)),
    alpha = 0
  )

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

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

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

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

```

### Arguments

<code>entropy.df</code>	A data frame with entropies. Each row gives the entropies of a group of related template sequences for all columns of the alignment.
<code>alignments</code>	A list with DNABin alignment objects corresponding to the groups (rows) in the alignment.
<code>template.df</code>	An object of class <code>Templates</code> containing the template sequences.
<code>gap.char</code>	The gap char in the alignments. By default, <code>gap.char</code> is set to "-".
<code>primers</code>	Either a single <code>Primers</code> object with evaluated primer coverage or a list containing such <code>Primers</code> objects.
<code>templates</code>	If <code>primers</code> is a <code>Primers</code> object, <code>templates</code> should be a <code>Templates</code> object. If <code>primers</code> is a list, then <code>templates</code> should be a list of <code>Templates</code> objects.
<code>direction</code>	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.
<code>group</code>	Optional identifiers of template groups for which binding events should be determined. By default, <code>group</code> is set to <code>NULL</code> such that all templates are considered.
<code>relation</code>	Whether binding positions are computed relative to forward ("fw") or reverse ("rev") binding regions. The default is "fw".
<code>region.names</code>	Character vector of length 2 providing the names of the binding and amplification region.
<code>...</code>	Optional arguments <code>groups</code> (a character vector of groups to be plotted when <code>primers</code> is a single primer set), <code>highlight.set</code> (the identifier of a primer set to be highlighted when <code>primers</code> is a list), <code>ncol</code> (a numeric indicating the number of facet columns if <code>primers</code> is a list), <code>deviation.per.primer</code> (a boolean indicating whether constraint deviations should be plotted per primer rather than per constraint if <code>primers</code> is a list)
<code>settings</code>	An object of class <code>DesignSettings</code> containing the constraints to be considered.
<code>active.constraints</code>	A character vector containing the identifiers to be considered for plotting. By default, <code>active.constraints</code> is <code>NULL</code> such that all computed constraints found in <code>settings</code> are plotted.

<code>plot.p.vals</code>	An optional logical argument indicating whether p-values computed via <a href="#">primer_significance</a> should be annotated in the plot. The default is FALSE.
<code>primer.data</code>	List with objects of class <code>Primers</code> containing the primer sets that are to be compared.
<code>template.data</code>	List with objects of class <code>Templates</code> containing the templates corresponding to <code>primer.data</code> .
<code>show.labels</code>	Whether the identifiers of the primer sets should be annotated in the plot. The default is TRUE.
<code>highlight.set</code>	A character vector providing the identifiers of primer sets to highlight. By default, <code>highlight.set</code> is NULL such that no highlighting takes place.
<code>alpha</code>	A numeric in the range [0,1] defining the trade-off between the maximal deviation of a constraint (large alpha) and all constraint deviations (large alpha). By default, alpha is set to 0 such that the absolute deviation across all constraints is considered.
<code>primer.subsets</code>	A list with optimal primer subsets, each of class <code>Primers</code> . The k-th list entry should correspond to an object of class <code>Primers</code> representing the primer subset of size k whose coverage ratio is the largest among all possible subsets of size k.
<code>required.cvg</code>	The required coverage ratio. The default is 100%; this value is plotted as a horizontal line.
<code>primer.df</code>	An object of class <code>Primers</code> containing primers with evaluated primer coverage.
<code>identifier</code>	Identifiers of primers that are to be considered. If <code>identifier</code> is set to NULL (the default), all primers are considered.
<code>per.mismatch</code>	A logical specifying whether the visualization should be stratified according to the allowed number of mismatches. By default, <code>per.mismatch</code> is set to FALSE such that the overall coverage is plotted.

### Details

The deviations for `plot_constraint_deviation` 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  $s \leq p \leq e$ , we output 0.

The `primer.subsets` argument for `plot_primer_subsets` 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_conseration` returns a plot showing the degree of sequence conservation in the templates.

`plot_primer_binding_regions` returns a plot of the primer binding regions.

`plot_constraint` returns a plot showing the distribution of primer properties.

`plot_constraint_fulfillment` returns a plot indicating the constraints that are fulfilled by the input primers.

`plot_cvg_constraints` returns a plot showing the distribution of the coverage constraint values.

`plot_constraint_deviation` returns a plot showing the deviations of the primer properties from the target constraints.

`plot_cvgs_vs_set_size` returns a plot of coverage vs set size.

`plot_penalty_vs_set_size` returns a plot of constraint penalties vs primer set sizes.

`plot_primer_subsets` plots the coverages of the primer subsets provided via `primer.subsets`.

`plot_primer` plots the primer binding sites in the templates.

`plot_template_cvgs` creates a plot showing the number of covered template sequences.

`plot_primer_cvgs` creates a plot showing the coverage of individual primers.

## Note

Computing the conservation scores for using `plot_conservation` requires the MAFFT software for multiple alignments (<http://mafft.cbrc.jp/alignment/software/>).

## Examples

```
data(Ippolito)
# Select binding regions for every group of templates and plot:
template.df <- select_regions_by_conservation(template.df, win.len = 30)
if (length(template.df) != 0) {
  p1 <- 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)
if (length(template.df) != 0) {
  p2 <- plot_conservation(attr(template.df, "entropies"),
                        attr(template.df, "alignments"), template.df)
}

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

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

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

```

# Plot coverage constraints of a single primer set
data(Ippolito)
p <- plot_cvg_constraints(primer.df, settings)
# Plot coverage constraints for multiple primer sets
data(Comparison)
p.cmp <- plot_cvg_constraints(primer.data[1:2], settings)

# Deviations for a single primer set
data(Ippolito)
p.dev <- plot_constraint_deviation(primer.df, settings)
# Deviations for multiple primer sets
data(Comparison)
p.dev.cmp <- plot_constraint_deviation(primer.data, settings)

# Plot coverage vs primer set size
data(Comparison)
p <- plot_cvg_vs_set_size(primer.data, template.data)

# Plot penalties vs number of primers
data(Comparison)
p <- plot_penalty_vs_set_size(primer.data, settings)

# Plot the coverage of optimal primer subsets
data(Ippolito)
primer.subsets <- subset_primer_set(primer.df, template.df, k = 3)
p <- plot_primer_subsets(primer.subsets, template.df)

# Plot of individual primer binding positions
data(Ippolito)
p <- plot_primer(primer.df[1,], template.df[1:30,])

# Visualize the template coverage of a single primer set
data(Ippolito)
p.cvg <- plot_template_cvg(primer.df, template.df)
# Stratify by allowed mismatches:
p.mm.cvg <- plot_template_cvg(primer.df, template.df, per.mismatch = TRUE)
# Compare the coverage of multiple primer sets
data(Comparison)
p.cmp.cvg <- plot_template_cvg(primer.data[1:2], template.data[1:2])
# Stratify by allowed mismatches:
p.cmp.cvg.mm <- plot_template_cvg(primer.data[1:2], template.data[1:2],
                                per.mismatch = TRUE)

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

```

---

plot\_constraint,list-method

*Boxplot for Comparing Constraints.*

---

**Description**

Creates a boxplot visualizing the physicochemical properties of multiple primer sets.

**Usage**

```
## S4 method for signature 'list'
plot_constraint(
  primers,
  settings,
  active.constraints,
  highlight.set = NULL,
  nfacets = NULL
)
```

**Arguments**

primers	List with evaluated objects of class Primers. Each list element corresponds to a single primer set.
settings	A DesignSettings object containing the constraints to be plotted.
active.constraints	The names of the constraints to be plotted.
highlight.set	Identifiers of primer sets to be highlighted.
nfacets	A numeric providing the number of facet columns to show. By default nfacets is NULL such that the number of facet columns is chosen automatically.
constraint.settings	List with settings for each constraint.

**Value**

Boxplot comparing the values of the properties specified by constraints.

---

plot\_constraint,Primers-method

*Histogram of Constraint Values.*

---

**Description**

Plots a histogram of constraint values.

**Usage**

```
## S4 method for signature 'Primers'
plot_constraint(primers, settings, active.constraints)
```

**Arguments**

primers	An evaluated object of class Primers.
settings	A DesignSettings object containing the settings for the constraints to be plotted.
active.constraints	Identifiers of constraints to be plotted. provided settings are used to visualize the desired ranges of constraints. If active.constraints is not provided, the plotting method will automatically try to plot all constraints defined in settings.

**Value**

A histogram of constraint values for the properties specified by constraints.

---

plot\_constraint.histogram  
*Histogram of Constraints.*

---

**Description**

Plots a histogram of constraint values.

**Usage**

```
plot_constraint.histogram(
  primer.df,
  con.cols,
  con.identifier,
  boundaries = NULL,
  x.limits = NULL
)
```

**Arguments**

primer.df	Primer data frame, not necessarily a Primers object.
con.cols	Constraint identifiers in primer.df to plot.
con.identifier	Name of the constraint to plot.
boundaries	List with constraint settings.
x.limits	Interval limiting the extent of the x-axis.

**Value**

A constraint histogram plot.

---

`plot_constraint.histogram.nbr.mismatches`  
*Histogram of Number of Mismatches.*

---

**Description**

Plots a histogram of mismatches.

**Usage**

`plot_constraint.histogram.nbr.mismatches(primer.df, allowed.mismatches)`

**Arguments**

`primer.df`           Primer data frame.  
`allowed.mismatches`  
                      Number of allowed mismatches.

**Value**

A plot of the number of primer mismatches.

---

`plot_constraint.histogram.primer efficiencies`  
*Histogram of Efficiencies*

---

**Description**

Plots a histogram of primer efficiencies.

**Usage**

`plot_constraint.histogram.primer efficiencies(primer.df, opti.constraints)`

**Arguments**

`primer.df`           Primer data frame.  
`opti.constraints`  
                      List with constraint settings.

**Value**

A plot of primer efficiencies.

---

`plot_constraint_deviation,list-method`

*Plot of Constraint Deviations for Multiple Primer Sets.*

---

### **Description**

Plots a box plot of the absolute mean deviation of each primer for comparing multiple primer sets.

### **Usage**

```
## S4 method for signature 'list'
plot_constraint_deviation(
  primers,
  settings,
  active.constraints,
  deviation.per.primer = FALSE
)
```

### **Arguments**

`settings` A DesignSettings object containing the target ranges for the primer properties.  
`active.constraints` Constraint identifiers to be plotted.  
`deviation.per.primer` Whether to show the deviation per primer or per constraint.  
`constraint.df` An evaluated object of class Primers.

### **Value**

A boxplot of deviations

---

`plot_constraint_deviation,Primers-method`

*Plot of Constraint Deviations for a Single Primer Set.*

---

### **Description**

Plots a box plot of deviations of primer properties from the target ranges.

### **Usage**

```
## S4 method for signature 'Primers'
plot_constraint_deviation(primers, settings, active.constraints)
```

### **Arguments**

`primers` An evaluated object of class Primers.  
`settings` A DesignSettings object containing the target ranges for the primer properties.  
`active.constraints` Constraint identifiers to be plotted.

**Value**

A boxplot of deviations

---

plot\_constraint\_fulfillment,list-method  
*Comparison of Evaluation Results.*

---

**Description**

Plots the percentage of primers fulfilling the specified constraints for multiple primer sets.

**Usage**

```
## S4 method for signature 'list'  
plot_constraint_fulfillment(  
  primers,  
  settings,  
  active.constraints,  
  plot.p.vals = FALSE,  
  ncol = 2,  
  highlight.set = NULL  
)
```

**Arguments**

primers	A list of Primers objects.
settings	A DesignSettings object.
active.constraints	The identifiers of constraints to be plotted for fulfillment.
plot.p.vals	Whether p-values from Fisher's exact test should be annotated for every primer set.
ncol	The number of columns for facet wrap.
highlight.set	Identifiers of primer sets to be highlighted.

**Value**

Plot indicating the ratio of primers fulfilling the constraints specified in constraint.settings for each primer set in primers.

---

plot\_constraint\_fulfillment, Primers-method  
*Overview of Constraint Fulfillment.*

---

### Description

Plots an overview of which primers passed the filtering constraints and which primers did not.

### Usage

```
## S4 method for signature 'Primers'
plot_constraint_fulfillment(
  primers,
  settings,
  active.constraints,
  plot.p.vals = TRUE
)
```

### Arguments

primers	A Primers object.
settings	A DesignSettings object.
active.constraints	The identifiers of constraints to be plotted for fulfillment.
plot.p.vals	Show p-value from Fisher's exact test for the significance of primer constraint fulfillment in comparison to reference primer sets.

### Value

A data frame with statistics on fulfilled constraints.

---

plot\_cvg\_constraints,list-method  
*Plot for Comparing Primer Coverage Constraints.*

---

### Description

Plot for Comparing Primer Coverage Constraints.

### Usage

```
## S4 method for signature 'list'
plot_cvg_constraints(
  primers,
  settings,
  active.constraints,
  highlight.set = NULL
)
```

**Arguments**

primers	List with objects of class Primers.
settings	A DesignSettings object.
active.constraints	Names of the coverage constraints to be plotted.
highlight.set	Primer sets to highlight in the plot.

**Value**

Plot of primer coverage constraints for multiple sets.

---

plot\_cvg\_constraints,Primers-method  
*Histogram of Coverage Constraints.*

---

**Description**

Plots a histogram of coverage constraint values.

**Usage**

```
## S4 method for signature 'Primers'  
plot_cvg_constraints(primers, settings, active.constraints)
```

**Arguments**

primers	A Primers object.
settings	A DesignSettings object.
active.constraints	Names of coverage constraints to be plotted.

**Value**

A plot of coverage constraints.

---

plot\_primer.comparison.box  
*Boxplot for Primer Comparison*

---

**Description**

Constructs a box plot showing constraint values for each primer set.

**Usage**

```
plot_primer.comparison.box(
  primer.data,
  con.identifier,
  con.cols,
  boundaries,
  y.limits = NULL,
  show.points = TRUE,
  highlight.set = NULL,
  nfacets = NULL
)
```

**Arguments**

primer.data	List with primer data frames.
con.identifier	Identifier of constraint to be plotted.
con.cols	Column names with the constraint values in the primer data frames.
boundaries	List with constraint settings.
y.limits	Limits for the extent of the y-axis.
show.points	If TRUE (the default), individual data points are visualized in the boxplot, otherwise they are not shown.
highlight.set	The identifier of a primer set to highlight in the plot.
nfacets	A numeric providing the number of facet columns to show. By default nfacets is NULL such that the number of facet columns is chosen automatically.

**Value**

A boxplot for primer comparison.

---

```
plot_primer.comparison.mismatches
Plot Primer Mismatches
```

---

**Description**

Plots primer mismatches for every set.

**Usage**

```
plot_primer.comparison.mismatches(
  primer.data,
  template.data,
  allowed.mismatches,
  highlight.set = NULL
)
```

**Arguments**

primer.data List with primer data frames.  
 template.data List with template data frames.  
 allowed.mismatches Allowed mismatches.  
 highlight.set Primer sets to highlight in the plot.

**Value**

Plot of mismatches for comparison.

---

plot\_primer\_binding\_regions,list,list-method

*Plot of Primer Binding Regions for Multiple Sets.*

---

**Description**

Plots the primer binding regions for every primer set.

**Usage**

```
## S4 method for signature 'list,list'
plot_primer_binding_regions(
  primers,
  templates,
  direction = c("both", "fw", "rev"),
  group = NULL,
  relation = c("fw", "rev"),
  region.names = c("Binding region", "Amplification region"),
  highlight.set = NULL
)
```

**Arguments**

primers List with primer data frames.  
 templates List with template data frames.  
 direction Direction of primers.  
 group Template groups to plot. This defaults to plotting all groups.  
 relation Plot binding region relative to forward binding region or reverse?  
 region.names Names for the primer binding region and the amplified region.  
 highlight.set Primer sets to highlight in the plot.

**Value**

A plot for primer binding region comparison.

---

plot\_primer\_binding\_regions,Primers,Templates-method

*Plot of Primer Binding Regions for a Single Primer Set.*

---

### Description

Plots the primer binding regions in the templates.

### Usage

```
## S4 method for signature 'Primers,Templates'
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	An object of class Primers with annotated primer coverage.
templates	An object of class Templates providing the template sequences corresponding to primers.
direction	Primer direction
group	The template groups for which binding events should be determined. By default, group is set to NULL such that all templates are considered.
relation	A character vector specifying whether binding region data shall be plotted relative to the forward (fw) or reverse (rev) target binding regions.
region.names	Names for the primer binding region and the amplified region.

### Value

A histogram of primer binding regions.

---

plot\_primer\_cvg,list,list-method

*Plot Multiple Primer Coverages.*

---

### Description

Plots the coverage of individual primers for multiple sets.

### Usage

```
## S4 method for signature 'list,list'
plot_primer_cvg(primers, templates, per.mismatch = FALSE)
```

**Arguments**

primers	List with Primers objects.
templates	List with Templates objects.
per.mismatch	Whether the coverage should be broken down for individual settings of allowed mismatches.

**Value**

A bar plot showing the coverage of individual primers.

---

plot\_primer\_cvg,Primers,Templates-method  
*Plot Individual Primer Coverage.*

---

**Description**

Shows which templates are covered by individual primers.

**Usage**

```
## S4 method for signature 'Primers,Templates'
plot_primer_cvg(primers, templates, per.mismatch = FALSE, groups = NULL)
```

**Arguments**

per.mismatch	Whether the coverage should be broken down for individual settings of allowed mismatches.
p.df	Primer data frame.
template.df	Template data frame.
excluded.seqs	Identifiers of templates that should not be considered.

**Value**

A bar plot showing the coverage of individual primers.

---

plot\_primer\_cvg\_mismatches  
*Plot of Individual Primer Coverage and Mismatches.*

---

**Description**

Plots the coverage of individual primers for different mismatch settings.

**Usage**

```
plot_primer_cvg_mismatches(
  primer.df,
  template.df,
  groups = NULL,
  nfacets = NULL
)
```

**Arguments**

primer.df	A Primers object.
template.df	A Templates object.
groups	Optional identifiers of template groups to be considered. If not provided, all template groups are considered.
nfacets	A numeric providing the number of facet columns to use. By default, nfacets is set to NULL such that a suitable number of columns is chosen automatically.

**Value**

A bar plot showing the coverage of individual primers for different mismatch settings.

---

plot\_primer\_cvg\_unstratified

*Plot Individual Primer Coverage.*

---

**Description**

Plots the coverage of individual primers.

**Usage**

```
plot_primer_cvg_unstratified(p.df, template.df, groups = NULL)
```

**Arguments**

p.df	Primer data frame.
template.df	Template data frame.
groups	Optional identifiers of template groups to be considered. If not provided, all template groups are considered.

**Value**

A bar plot showing the coverage of individual primers.

---

```
plot_template_cvg,list,list-method
```

*Templates Coverage for Multiple Primer Sets.*

---

### Description

Plots the coverage of multiple primer sets.

### Usage

```
## S4 method for signature 'list,list'
plot_template_cvg(primers, templates, per.mismatch, highlight.set = NULL)
```

### Arguments

primers	List with primer data frames.
templates	List with template data frames.
highlight.set	Primer sets to be highlighted.
colors	Color for every primer set.

### Value

A plot for comparing primer coverage.

---

```
plot_template_cvg,Primers,Templates-method
```

*Bar Plot of Template Coverage.*

---

### Description

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

### Usage

```
## S4 method for signature 'Primers,Templates'
plot_template_cvg(primers, templates, per.mismatch, groups = NULL)
```

### Arguments

primers	A Primers object with evaluated primer coverage.
templates	A Templates object containing the template sequences.
per.mismatch	Whether to stratify by mismatches.
groups	Identifiers of template groups for which plot should be created. By default, groups is set to NULL such that all templates are considered. according to the number of mismatches between primer-template pairs.

### Value

A plot showing the number of covered template sequences.

---

plot\_template\_cvg\_comparison\_mismatch  
*Templates Coverage for Multiple Primer Sets.*

---

**Description**

Plots the coverage of multiple primer sets.

**Usage**

```
plot_template_cvg_comparison_mismatch(primers, templates, highlight.set = NULL)
```

**Arguments**

primers            List with primer data frames.  
 templates        List with template data frames.  
 highlight.set    Primer sets to be highlighted.

**Value**

A plot for comparing primer coverage.

---

plot\_template\_cvg\_comparison\_unstratified  
*Templates Coverage for Multiple Primer Sets.*

---

**Description**

Plots the coverage of multiple primer sets.

**Usage**

```
plot_template_cvg_comparison_unstratified(  
  primers,  
  templates,  
  highlight.set = NULL  
)
```

**Arguments**

primers            List with primer data frames.  
 templates        List with template data frames.  
 highlight.set    Primer sets to be highlighted.

**Value**

A plot for comparing primer coverage.

---

`plot_template_cvg_mismatches`*Bar Plot of Template Coverage for Mismatches.*

---

**Description**

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

**Usage**

```
plot_template_cvg_mismatches(  
  primer.df,  
  template.df,  
  groups = NULL,  
  nfacets = 2  
)
```

**Arguments**

<code>primer.df</code>	A Primers object.
<code>template.df</code>	A Templates object.
<code>groups</code>	Identifiers of template groups for which plot should be created. By default, <code>groups</code> is set to <code>NULL</code> such that all templates are considered.
<code>nfacets</code>	The number of facets columns to plot. By default, <code>nfacets</code> is set to 2.

**Value**

A plot showing the number of covered template sequences.

---

`plot_template_cvg_unstratified`*Bar Plot of Template Coverage.*

---

**Description**

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

**Usage**

```
plot_template_cvg_unstratified(primers, templates, groups = NULL)
```

**Arguments**

<code>primers</code>	A Primers object with evaluated primer coverage.
<code>templates</code>	A Templates object containing the template sequences.
<code>groups</code>	Identifiers of template groups for which plot should be created. By default, <code>groups</code> is set to <code>NULL</code> such that all templates are considered. according to the number of mismatches between primer-template pairs.

**Value**

A plot showing the number of covered template sequences.

---

plot\_template\_structure

*Plot of Template Folding Energies.*

---

**Description**

Plots the DeltaDeltaG of template folding, which is the difference between the free energy change of the unconstrained folding and the free energy change of the constrained folding.

**Usage**

```
plot_template_structure(fold.df)
```

**Arguments**

fold.df            A data frame with free energies for the template regions.

**Value**

A plot of DeltaDeltaG.

---

pos.to.range

*Conversion of Positions to Ranges.*

---

**Description**

Converts two numeric values to a range.

**Usage**

```
pos.to.range(pos1, pos2)
```

**Arguments**

pos1            The first value.

pos2            The second value.

**Value**

A character vector range.

---

predict_coverage	<i>Prediction of Primer Coverage.</i>
------------------	---------------------------------------

---

**Description**

Predicts primer coverage using a logistic regression model. Converts coverage probabilities to expected false positive rate for a given probability.

**Usage**

```
predict_coverage(  
  primer.df,  
  template.df,  
  settings,  
  mode = c("on_target", "off_target"),  
  updateProgress = NULL  
)
```

**Arguments**

primer.df	A Primers data frame.
template.df	A Templates data frame.
settings	A DesignSettings object.
mode	Whether on-target or off-target events shall be considered.

**Value**

The predictions for primer coverage

---

prefilter.primers.candidates	<i>Identification of Short Primers.</i>
------------------------------	---

---

**Description**

Identify initial primers that are too short.

**Usage**

```
prefilter.primers.candidates(primer.candidates, min.len)
```

**Arguments**

primer.candidates	Primer alignment.
min.len	Minimal primer length.

**Value**

The index of proposed primers that are shorter than min.len.

prepare.constraint.plot

*Preparation of Comparison Plot for Evaluation.*

---

### Description

Preparation of Comparison Plot for Evaluation.

### Usage

```
prepare.constraint.plot(primer.data, constraint.settings, plot.p.vals = FALSE)
```

### Arguments

primer.data	List with objects of class Primers. Each list entry corresponds to a single primer set.
constraint.settings	List with settings for each constraint. If NULL (the default), use the available evaluation results in primer.data.
plot.p.vals	Whether p-values from Fisher's exact test should be annotated for every primer set.

### Value

Plot indicating the ratio of primers fulfilling the constraints specified in constraint.settings for each primer set in primer.data.

---

prepare.dimer.seqs

*Preparation of Input for Dimerization.*

---

### Description

Preparation of Input for Dimerization.

### Usage

```
prepare.dimer.seqs(s1, s2)
```

### Arguments

s1	Nucleotide character vectors (5' to 3')
s2	Nucleotide character vectors (5' to 3')

### Value

A list with two fields containing character vectors.

---

prepare_mm_plot	<i>Data Preparation for Mismatch Plot.</i>
-----------------	--

---

**Description**

Data Preparation for Mismatch Plot.

**Usage**

```
prepare_mm_plot(primer.df, template.df, mode = c("on_target", "off_target"))
```

**Arguments**

primer.df	A Primers object.
template.df	A Templates object.
mode	Whether to compute for on-target or off-target events.

**Value**

A data frame with binding information for every primer.

---

prepare_template_cvg_mm_data	<i>Preparation of Data for Plotting Mismatch Template Coverage.</i>
------------------------------	---

---

**Description**

Creates a data frame for plotting a bar plot for the covered templates per allowed mismatches.

**Usage**

```
prepare_template_cvg_mm_data(primer.df, template.df, allowed.mismatches = NULL)
```

**Arguments**

primer.df	A Primers object.
template.df	A Templates object.
allowed.mismatches	An optional numeric specifying the number of mismatches to be considered at most for plotting. If not provided, the maximal number of mismatches found for the input primer set is used.

**Value**

A data frame for creating a plot.

---

```
primer.binding.regions.data
```

*Primer Binding Region Data*

---

### Description

Collects all data concerning primer binding regions.

### Usage

```
primer.binding.regions.data(
  primer.df,
  template.df,
  direction = c("both", "fw", "rev"),
  group = NULL,
  relation = c("fw", "rev")
)
```

### Arguments

primer.df	Primer data frame.
template.df	Template data frame.
direction	Primer direction
group	The groups for which binding data shall be retrieved.
relation	Binding region data relative to forward/reverse binding region?

### Value

Data frame with primer binding data.

---

```
primer.coverage.for.groups
```

*Determination of Primer Coverage for Groups.*

---

### Description

Modifies a primer data frame to retain only coverage events relating to the selected groups of templates.

### Usage

```
primer.coverage.for.groups(primer.df, template.df, groups)
```

### Arguments

primer.df	Primer data frame.
template.df	Template data frame.
groups	Template groups for which coverage should be determined.

**Value**

primer.df with coverage considered only for groups.

---

```
primer.set.parameter.stats
```

*Primer Set Statistics*

---

**Description**

Creates an overview of all primer set constraint values.

**Usage**

```
primer.set.parameter.stats(primer.df, mode.directionality, lex.seq)
```

**Arguments**

primer.df	Primer data frame.
mode.directionality	Direction of primers.
lex.seq	Template data frame.

**Value**

A data frame with statistics.

---

```
PrimerDesign
```

*Primer Design Functionalities.*

---

**Description**

`design_primers` 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.

`get_initial_primers` 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**

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

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
)

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

<code>template.df</code>	A Templates object containing the template sequences with annotated primer target binding regions.
<code>mode.directionality</code>	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".
<code>primer.length</code>	A scalar numeric providing the target length of the designed primers. The default length of generated primers is set to 18.
<code>primer.estimate</code>	Whether the number of required primers shall be estimated. By default (FALSE), the number of required primers is not estimated.
<code>required.cvg</code>	The desired ratio of covered template sequences. If the target coverage ratio cannot be reached, the constraint settings are relaxed according to the constraint limits in order to reach the target coverage. The default <code>required.cvg</code> is set to 1, indicating that 100% of the templates are to be covered.
<code>sample</code>	Character vector providing an identifier for the templates.
<code>primer.lengths</code>	Numeric vector of length 2 providing the minimal and maximal allowed lengths for generated primers.
<code>allowed.region.definition</code>	A character vector providing the definition of region where primers are to be constructed. If <code>allowed.region.definition</code> is "within", constructed primers lie within the allowed binding region. If <code>allowed.region.definition</code> is "any", primers overlap with the allowed binding region. The default is "within".

<code>init.algo</code>	The algorithm to be used for initializing primers. If <code>init.algo</code> is "naive", then primers are constructed from substrings of the input template sequences. If <code>init.algo</code> is "tree", phylogenetic trees are used to form degenerate primers whose degeneracy is bounded by <code>max.degen</code> . This option requires an installation of MAFFT (see notes). The default <code>init.algo</code> is "naive".
<code>max.degen</code>	The maximal degeneracy of primer candidates. This setting is particularly relevant when <code>init.algo</code> is set to "tree". The default setting is 16, which means that at most 4 maximally degenerate positions are allowed per primer.
<code>conservation</code>	Restrict the percentile of considered regions according to their conservation. Only applicable for the tree-based primer initialization. At the default of 1, all available binding regions are considered.
<code>updateProgress</code>	Shiny progress callback function. The default is NULL such that no progress is logged.
<code>settings</code>	A <code>DesignSettings</code> object specifying the constraint settings for designing primers.
<code>opti.algo</code>	The algorithm to be used for solving the primer set covering problem. If <code>opti.algo</code> is "Greedy" a greedy algorithm is used to solve the set cover problem. If <code>opti.algo</code> is "ILP" an integer linear programming formulation is used. The default <code>opti.algo</code> is "Greedy".
<code>timeout</code>	Timeout in seconds. Only applicable when <code>opti.algo</code> is "ILP". The default is <code>Inf</code> , which does not limit the runtime.
<code>sample.name</code>	An identifier for the primer design task. The default setting is NULL, which means that the run identifier provided in <code>template.df</code> is used.
<code>cur.results.loc</code>	Directory for storing the results of the primer design procedure. The default setting is NULL such that no output is stored.
<code>primer.df</code>	An optional <code>Primers</code> object. If an evaluated <code>primer.df</code> is provided, the primer design procedure only optimizes <code>primer.df</code> and does not perform the initialization and filtering steps. The default is NULL such that primers are initialized and filtered from scratch.

## Details

`classify_design_problem` determines the difficulty of a primer design task 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 compared 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.

The primer design algorithm used by `design_primers` 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.

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`.

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.

## Value

`classify_design_problem` returns 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`.

`get_initial_primers` returns a data frame with candidate primers for optimization.

`design_primers` returns 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.

## Note

Some constraints can only be computed if additional software is installed, please see the documentation of [DesignSettings](#) for more information. The usage of `init.algo = "tree"` requires an installation of the multiple alignment program MAFFT (<http://mafft.cbrc.jp/alignment/software/>).

## 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)
```

```

# Define PCR settings and primer criteria
data(Ippolito)
# design only with minimal set of constraints
constraints(settings)$primer_length <- c("min" = 18, "max" = 18)
constraints(settings) <- constraints(settings)[c("primer_length", "primer_coverage")]
# 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)

```

---

PrimerEval

*Primer Evaluation.*


---

## Description

- `check_constraints` Determines whether a set of primers fulfills the constraints on the properties of the primers.
- `check_restriction_sites` 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.
- `filter_primers` Filters a primer set according to the specified constraints such that all primers that do not fulfill the constraints are removed from the primer set.
- `primer_significance` Uses Fisher's exact test to determine the significance of a primer set according to its ratio of fulfilled constraints.
- `subset_primer_set` Determines subsets of the input primer set that are optimal with regard to the number of covered template sequences.

## Usage

```

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

check_constraints(
  primer.df,
  template.df,
  settings,

```

```

    active.constraints = names(constraints(settings)),
    to.compute.constraints = active.constraints,
    for.shiny = FALSE,
    updateProgress = NULL
  )

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

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

  primer_significance(primer.df, set.name = NULL, active.constraints = 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.  |
| 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 <i>confident</i> 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         | Progress callback function for shiny. The default is NULL meaning that no progress is monitored via the Shiny interface.  |
| settings               | A DesignSettings object containing the constraints that are to be considered.   |
| 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 <code>active.constraints</code> , all constraints specified via <code>active.constraints</code> are evaluated for fulfillment, but only the constraints in <code>to.compute.constraints</code> are newly calculated.
<code>for.shiny</code>	Whether the output of the function shall be formatted as HTML. The default setting is <code>FALSE</code> .
<code>k</code>	The spacing between generated primer subset sizes. By default, <code>k</code> is set to 1 such that all primer subsets are constructed.
<code>groups</code>	The identifiers of template groups according to which coverage should be determined. By default, <code>groups</code> is set to <code>NULL</code> such that all all covered templates are considered.
<code>identifier</code>	An identifier for storing the primer set. By default, <code>identifier</code> is set to <code>NULL</code> .
<code>cur.results.loc</code>	Directory for storing the results. By default, <code>cur.results.loc</code> is set to <code>NULL</code> , which means that no results are stored.
<code>set.name</code>	An identifier for the input primers. If <code>NULL</code> , the run identifier is used.

## Details

When the optional argument `active.constraints` is supplied to `check_constraints`, 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.

`subset_primer_set` determines optimal subsets of the input primer set 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.

`primer_significance` computes the significance 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

`check_restriction_sites` returns a data frame with possible restriction sites found in the primers.

`check_constraints` returns a `Primers` object that is augmented with columns providing the results for the evaluated constraints. 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.

`filter_primers` returns a `Primers` object containing only those primers fulfilling all specified constraints.

`subset_primer_set` returns a list with optimal primer subsets, each of class `Primers`.

`primer_significance` returns a numeric providing 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

**Note**

Please note that some constraint computations may require the installation of additional programs; for more information please view the documentation of [DesignSettings](#).

**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>

**Examples**

```
data(Ippolito)
# Check the first primer for restriction sites with respect to the first 10 templates
site.df <- check_restriction_sites(primer.df[1,], template.df[1:10])

data(Ippolito)
settings.xml <- system.file("extdata", "settings",
                           "C_Taq_PCR_high_stringency.xml", package = "openPrimeR")
settings <- read_settings(settings.xml)
# Check GC clamp and number of runs for all primers:
constraint.df <- check_constraints(primer.df, template.df,
                                  settings, active.constraints = c("gc_clamp", "no_runs"))
# Summarize the evaluation results
summary(constraint.df)

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"))

# Determine optimal primer subsets
data(Ippolito)
primer.subsets <- subset_primer_set(primer.df, template.df, k = 3)

# Determine the significance of a primer set
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
```

---

rbind.primer.data      *Rbind for Primer Data Frames.*

---

**Description**

Merges all primer data frames in primer.data into one data frame.

**Usage**

```
## S3 method for class 'primer.data'
rbind(primer.data)
```

**Arguments**

primer.data      List with primer data frames.

**Value**

A data frame containing all data in primer.data.

---

rbind.Primers      *rbind for Primers class.*

---

**Description**

Ensures that the rbind result has the appropriate class.

**Usage**

```
## S3 method for class 'Primers'  
rbind(...)
```

**Arguments**

...              Parameters for rbind function.

**Value**

Row-binded Primers data frame.

**Examples**

```
data(Ippolito)  
primer.df <- rbind(primer.df, primer.df)
```

---

rbind.Templates      *rbind for Template class.*

---

**Description**

Ensures that the rbind result has the appropriate class.

**Usage**

```
## S3 method for class 'Templates'  
rbind(...)
```

**Arguments**

...              Parameters for rbind function.

**Value**

Row-binded Templates data frame.

**Examples**

```
data(Ippolito)
template.df <- rbind(template.df, template.df)
```

---

read.leaders	<i>Read Individual Binding Regions</i>
--------------	--

---

**Description**

Reads individual binding regions into a data frame.

**Usage**

```
read.leaders(
  fasta.file,
  direction = c("fw", "rev"),
  rm.keywords = NULL,
  gap.character
)
```

**Arguments**

fasta.file	Path to a FASTA file with binding regions.
direction	String identifying whether the FASTA file contains information pertaining to the binding region of forward or reverse primers.
rm.keywords	A vector of keywords that are used to remove templates whose headers contain any of the keywords.
gap.character	The character for indicating gaps in sequences.

**Value**

A data frame with individual binding regions.

---

read.secondary.structure.raw	<i>Read a Secondary Structure</i>
------------------------------	-----------------------------------

---

**Description**

Reads the secondary structure output of ViennaRNA.

**Usage**

```
read.secondary.structure.raw(fw.out)
```

**Arguments**

fw.out	Path to a ViennaRNA secondary structure output file.
--------	--

**Value**

Data frame with information on secondary structures.

---

read.sequences      *Read Sequences.*

---

**Description**

Reads an input FASTA file.

**Usage**

```
read.sequences(fasta.file, gap.character)
```

**Arguments**

fasta.file	The path to a FASTA file.
The	character indicating gaps in sequences.

**Value**

A data frame with sequences.

---

read\_primers.internal      *Internal Function for Reading Primers*

---

**Description**

Reads the given primer sequences into a data frame.

**Usage**

```
read_primers.internal(
  primer.seqs,
  headers,
  fw.id,
  rev.id,
  merge.ambig = c("none", "merge", "unmerge"),
  max.degen,
  sample.name
)
```

**Arguments**

primer.seqs	Primer sequences.
headers	Headers of the primer FASTA file.
fw.id	Identifier of forward primers in the headers.
rev.id	Identifier of reverse primers in the headers.
merge.ambig	Should ambiguous primers be merged?
max.degen	Maximum allowed degeneracy

**Value**

A data frame with primer sequences.

---

read\_primers\_csv      *Read Primer CSV File.*

---

**Description**

Reads a primer data frame stored in a CSV file.

**Usage**

```
read_primers_csv(file)
```

**Arguments**

file                    The path to a csv file containing the primer data.

**Value**

A Primers object, an instance of a data frame.

---

read\_primers\_multiple      *Input of Multiple Primer Sets.*

---

**Description**

Reads multiple CSV files representing stored objects of class Primers.

**Usage**

```
read_primers_multiple(  
  filenames,  
  fw.id,  
  rev.id,  
  merge.ambig,  
  max.degen,  
  template.df,  
  adapter.action,  
  sample.name,  
  updateProgress  
)
```

**Arguments**

filenames                The paths to multiple primer CSV/FASTA files.

**Value**

A list containing objects of class Primers.

---

read\_templates\_csv     *Read Template CSV File*

---

**Description**

Reads an input template CSV file.

**Usage**

```
read_templates_csv(fname)
```

**Arguments**

fname                    The filename of the input template CSV file.

**Value**

A template data frame.

---

read\_templates\_fasta     *Input of Template Sequences.*

---

**Description**

Read template sequences from a FASTA file.

**Usage**

```
read_templates_fasta(
  fasta.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**

fasta.file            Path to a FASTA file containing the template sequences.

hdr.structure        Names 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*.

delim                Delimiter for the information in the FASTA headers.

id.column            Field in the header to be used as the identifier.

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.
rev.region	The positional interval from the template 3' end specifying the binding sites for reverse primers.
gap.character	The character in the input file representing gaps. Gaps are automatically removed upon input.
run	An identifier for the template sequences.

**Value**

An object of class `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")
```

---

`read_templates_multiple`

*Input of Multiple Templates.*

---

**Description**

Reads multiple template CSV/FASTA files.

**Usage**

```
read_templates_multiple(
  filenames,
  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**

filenames	Names of FASTA/CSV files containing template data.
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.
delim	Delimiter for the information in the FASTA headers.
id.column	Field in the header to be used as the identifier.
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.
rev.region	The positional interval from the template 3' end specifying the binding sites for reverse primers.
gap.character	The character in the input file representing gaps. Gaps are automatically removed upon input.
run	An identifier for the template sequences.

**Value**

A list containing objects of class Templates.

---

read\_templates\_single *Input of a Single Template File.*

---

**Description**

Read template sequences from a FASTA or CSV file.

**Usage**

```
read_templates_single(  
  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**

<code>template.file</code>	Path to a FASTA or CSV file containing the template sequences.
<code>hdr.structure</code>	A character vector describing the information contained in the FASTA headers. In case that the headers of <code>fasta.file</code> contain template group information, please include the keyword "GROUP" in <code>hdr.structure</code> .
<code>delim</code>	Delimiter for the information in the FASTA headers.
<code>id.column</code>	Field in the header to be used as the identifier.
<code>rm.keywords</code>	A vector of keywords that are used to remove templates whose headers contain any of the keywords.
<code>remove.duplicates</code>	Whether duplicate sequence shall be removed.
<code>fw.region</code>	The positional interval from the template 5' end specifying the binding sites for forward primers.
<code>rev.region</code>	The positional interval from the template 3' end specifying the binding sites for reverse primers.
<code>gap.character</code>	The character in the input file representing gaps. Gaps are automatically removed upon input.
<code>run</code>	An identifier for the template sequences.

**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. When supplying a CSV file with template sequences, the data are loaded without any modification and 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`.

---

`relax.constraints`      *Relaxation of Constraints*

---

**Description**

Relax constraints trying to reach the target coverage ratio.

**Usage**

```
relax.constraints(
  settings,
  filtered.df,
  excluded.df,
  stat.df,
  template.df,
  mode.directionality = c("fw", "rev"),
```

```

    required.cvg,
    target.temps = NULL,
    results.loc = NULL
)

```

### Arguments

settings	A DesignSettings object.
filtered.df	Data set of primers that fulfilled all constraints of the filtering procedure.
excluded.df	Data frame with excluded primers from the first filtering round.
stat.df	Data frame with statistics from filtering.
template.df	Template data frame.
mode.directionality	Primer direction.
required.cvg	Required ratio of covered templates.
target.temps	Target melting temperature values.
results.loc	The location where intermediary results should be stored.

### Value

A list containing information about the relaxation as well as the filtered primers.

---

```
relax.opti.constraints
```

*Relaxation of Optimization Constraints*

---

### Description

Relax optimization constraints.

### Usage

```

relax.opti.constraints(
  cur.opti.constraints,
  initial.opti.limits,
  initial.opti.constraints
)

```

### Arguments

cur.opti.constraints	List with optimization constraint settings.
initial.opti.limits	Initial optimization limits.
initial.opti.constraints	Initial optimization constraints.

### Value

Relaxed optimization constraints.

remove.redundant.cols *Removal of Redundant Primers.*

---

**Description**

Removes redundant primers from an optimal solution.

**Usage**

```
remove.redundant.cols(S, cvg.matrix)
```

**Arguments**

S	Indices of primers that are selected in an optimal solution.
cvg.matrix	Binary matrix of coverage events.

**Details**

An optimal solution can contain primers with redundant coverage when using presolve or greedy optimization.

**Value**

Updated indices of selected primers S such that indices representing primers with redundant coverage are removed.

---

remove.seqs.by.keyword  
*Removal of Partial Sequences.*

---

**Description**

Removes partial template sequences.

**Usage**

```
remove.seqs.by.keyword(template.df, keyword = "partial")
```

**Arguments**

template.df	Template data frame.
Header	keywords indicating templates that should be excluded.

**Value**

Template data frame with partial sequences removed.

---

`rename.constraint.options`*Renaming of Constraint Options.*

---

**Description**

Renames the input list with constraint options.

**Usage**

```
rename.constraint.options(constraint.options)
```

**Arguments**

`constraint.options`

A list with constraint options.

**Value**

A list with renamed constraint options.

---

`render_report`*Renders an rmarkdown file using Pandoc.*

---

**Description**

Creates a PDF report using rmarkdown and Pandoc by passing the specified params to the markdown file given by `report_template` and storing the PDF in `out.file`.

**Usage**

```
render_report(params, report_template, out.file)
```

**Arguments**

`params` A list with parameters for the R markdown parser.

`report_template`

A character vector giving the basename of the Rmarkdown template to use for report creation.

`out.file`

The filename of the report PDF to be created.

**Value**

Creates a PDF in `out.file` if successful.

---

reorder.primers.table    *Reorder Primers*

---

**Description**

Reorders a primer set according to the IDs of primers.

**Usage**

```
## S3 method for class 'primer.table'  
reorder(filtered.primers, primer.ID.order)
```

**Arguments**

filtered.primers  
                  Primer data frame.  
primer.ID.order  
                  new ordering of IDs in the data frame.

**Value**

Reordered primer data frame.

---

restriction\_ali        *Identification of Badly Fitting Regions.*

---

**Description**

Identify regions in the templates where the primers are not very complementary. These regions indicate possible restriction enzyme adapters.

**Usage**

```
restriction_ali(primer.seqs, template.seqs, search.hits)
```

**Arguments**

primer.seqs    Primer sequences.  
template.seqs    Template sequences.  
search.hits    Template substrings that agree well with the input primers.

**Value**

A list with putative restriction sites for every primer.

---

restriction_hits	<i>Identification of Restriction Sites.</i>
------------------	---

---

**Description**

Identifies restriction sites in a list with putative restriction sites provided by `bad.regions` using a data frame of restriction sites given by `DB`.

**Usage**

```
restriction_hits(bad.regions, DB)
```

**Arguments**

<code>DB</code>	A data frame with restriction enzyme sites.
<code>bad.region</code>	<code>IRanges</code> with possible adapter sites.

**Value**

A boolean data frame indicating the presence of adapters for all considered restriction sites.

---

restriction_match	<i>Identification of Sequence Matches.</i>
-------------------	--

---

**Description**

Determines the most similar template sequence for every input primer sequence. Used to identify regions for alignment for the identification of restriction sites.

**Usage**

```
restriction_match(primer.seqs, template.seqs)
```

**Arguments**

<code>primer.seqs</code>	Primer sequences.
<code>template.seqs</code>	Template sequences.

**Value**

A vector with the template regions matching the `primer.seqs` best.

```
retrieve.leader.region
```

*Retrieval of Binding Regions*

---

### Description

Retrieves information about individual binding regions.

### Usage

```
retrieve.leader.region(  
  template.df,  
  direction = c("fw", "rev"),  
  start,  
  end,  
  gap.char,  
  init.from.leader  
)
```

### Arguments

template.df	Template data frame.
direction	Identify forward and reverse.
start	Start positions.
end	End positions.
gap.char	The character for gaps in alignments.
init.from.leader	Whether the binding regions are initialized from leader sequences.

### Value

Data frame with information on allowed binding regions.

---

```
rev.comp.sequence      Reverse complement of a sequence
```

---

### Description

Computes the reverse complement of the input sequences.

### Usage

```
## S3 method for class 'comp.sequence'  
rev(seq)
```

### Arguments

seq	the input strings
-----	-------------------

**Value**

The reverse complement of the input sequences.

---

rev.sequence	<i>Reversion of a sequence</i>
--------------	--------------------------------

---

**Description**

Reverses the input sequences.

**Usage**

```
## S3 method for class 'sequence'
rev(seq)
```

**Arguments**

seq                    the input sequence.

**Value**

The input sequence in reverse order.

---

runTutorial	<i>The openPrimeR Tutorial.</i>
-------------	---------------------------------

---

**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**

```
## Not run:
# Open the tutorial
if (interactive()) {
  runTutorial()
}

## End(Not run)
```

---

sanitize_path	<i>Sanitization of Filename.</i>
---------------	----------------------------------

---

**Description**

Ensures that a filename is valid for the file system.

**Usage**

```
sanitize_path(path, suffix = "", sub.char = "_")
```

**Arguments**

path	The path to the file to be sanitized, without file extension.
suffix	The suffix (e.g. ".png") of a file.
sub_char	The character used to replacing disallowed chars.

**Value**

The sanitized filename

---

score.conservaion	<i>Conservation Scores</i>
-------------------	----------------------------

---

**Description**

Scores the conservation of alignment regions.

**Usage**

```
score.conservaion(primer.range, ali.entropy)
```

**Arguments**

primer.range	A data frame with starts/ends of primers.
ali.entropy	Entropies corresponding to the alignment

**Value**

Entropies indicating conservation (similarity) of regions.

**Description**

`score_degen` Determines the degeneration score of a sequence.

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

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

**Usage**

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

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

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

**Arguments**

`template.df` A Templates object providing the set of templates.

`gap.char` The gap character in the sequences. The default is "-".

`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.

`seq` A list of vectors containing individual characters of a nucleotide sequence.

`primer.df` A Primers object containing the primers.

`settings` A DesignSettings object containing the analysis settings.

`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

`score_degen` computes the degeneration of an ambiguous sequence by considering the number of unambiguous sequences that are represented by the the ambiguous sequence. Let a sequence  $S$  of length  $n$  be represented by a collection of sets such that

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

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  $i$ .

`score_primers` determines the penalty of a primer 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 list containing Entropies and Alignments. Entropies is 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 is a list of DNABin objects, where each object gives the alignment corresponding to one group of template sequences.

`score_degen` finds the number of unambiguous sequences that are represented by `seq`.

`score_primers` returns a data frame containing scores for individual primers.

## Note

`score_conservation` 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)
# Compute degeneration for sequences with differing number of ambiguous bases
seq <- strsplit(c("ctggaattacggtacc", "taggaaccggrtaagc", "rtaasrygtar"), split = "")
degen <- score_degen(seq)

# Score the primers
data(Ippolito)
primer.scores <- score_primers(primer.df, settings)
```

---

```
select.allowed.binding.events
```

*Selection of Binding Events*

---

**Description**

Selects primer binding events that are within the allowed binding regions.

**Usage**

```
select.allowed.binding.events(  
  bound.fw,  
  bound.to.allowed.region.fw,  
  allowed.other.binding.ratio  
)
```

**Arguments**

`bound.fw` Indices of covered templates of a single primer.  
`bound.to.allowed.region.fw` Corresponding allowed binding regions.  
`allowed.other.binding.ratio` The ratio of other binding events. If this is different from 0, disallowed binding events will also be reported.

**Value**

The indices of the allowed binding events.

---

```
select.best.ILP
```

*Selection of Best ILP*

---

**Description**

Selects the best solution from multiple solved ILP instances.

**Usage**

```
select.best.ILP(ILP.df)
```

**Arguments**

`ILP.df` Data frame with ILP result properties.

**Value**

The index of the best solution.

```
select.best.opti.result
```

*Selection of Best Greedy Result*

---

### Description

Selects best greedy primer data set.

### Usage

```
select.best.opti.result(opti.results, template.df)
```

### Arguments

`opti.results` List with primer data frames for different target melting temperatures.  
`template.df` Template data frame.

### Value

The index of the best primer data set.

---

```
select.best.primers.idx
```

*Greedy Choice*

---

### Description

Selects the currently best primer for Greedy primer selection.

### Usage

```
select.best.primers.idx(  
  result,  
  primers,  
  deltaG.cutoff,  
  target.temp,  
  primer_conc,  
  na_salt_conc,  
  mg_salt_conc,  
  k_salt_conc,  
  tris_salt_conc  
)
```

**Arguments**

result	Data frame of current optimized primer data set that is to be augmented.
primers	Data frame of candidate primers for addition to result.
deltaG.cutoff	Free energy cutoff for cross-dimerization.
target.temp	Target annealing temperature in Celsius.
primer_conc	Primer concentration.
na_salt_conc	Sodium ion concentration.
mg_salt_conc	Magnesium ion concentration.
k_salt_conc	Potassium ion concentration.
tris_salt_conc	Tris ion concentration.

**Value**

The index of a suitable primer according to Greedy selection.

---

select.best.primer.set

*Selection of Best Primer Set.*

---

**Description**

Selects the best primer set according to coverage and melting temperature differences among primers in the set.

**Usage**

```
select.best.primer.set(stats)
```

**Arguments**

stats	Statistics of the primer sets to be evaluated.
-------	--

**Value**

The index of the best primer set.

---

`select.binding.events` *Selection of Individual Binding Events*

---

**Description**

Selects only binding events of interest.

**Usage**

```
select.binding.events(fw.binding.filtered, p.idx)
```

**Arguments**

<code>fw.binding.filtered</code>	IRanges with binding events.
<code>p.idx</code>	Index of binding events to keep.

**Value**

An IRanges object containing only the selected binding events.

---

`select.constraints` *Selection of Constraints.*

---

**Description**

Selects constraints that can be computed according to installed third-party software. This function is only used for initializing the 'constraint\_order' option.

**Usage**

```
select.constraints(active.constraints)
```

**Arguments**

<code>active.constraints</code>	A vector whose names give the constraints to be checked.
---------------------------------	--

**Value**

A vector of useable constraint identifiers.

---

select.min.cross.idx    *Selection of cross dimer index*

---

**Description**

Select the index with the smallest DeltaG value.

**Usage**

```
select.min.cross.idx(deltaG, primers)
```

**Arguments**

deltaG	Data frame with thermodynamic info.
primers	The corresponding primers.

**Value**

The indices with smallest DeltaG for every primer.

---

select.primer.region.by.conservation  
*Selection by Conservation*

---

**Description**

Selects primer regions for initialization of primers according to their conservation scores.

**Usage**

```
select.primer.region.by.conservation(
  primer.range,
  ali.entropy,
  conservation,
  bin,
  gap.char = "-"
)
```

**Arguments**

primer.range	Data frame with primer starts/stops.
ali.entropy	Entropy values for the alignment.
conservation	Desired ratio of primer conservation. Only regions with a conservation of at least conservation are considered for the initialization of primers.
bin	DNABin alignment of templates.
gap.char	The character for alignment gaps.

**Details**

The conservation scores are computed using the entropies computed from the alignment of the template sequence regions of interests.

**Value**

Updated primer regions according to the desired conservation.

---

select.primers.by.cvg *Greedy Optimization.*

---

**Description**

Greedy approach for solving the primer set coverage problem.

**Usage**

```
select.primers.by.cvg(
  primers,
  settings,
  template.df,
  mode.directionality = c("fw", "rev"),
  required.cvg = 1,
  allowed.mismatches,
  primer_conc,
  na_salt_conc,
  mg_salt_conc,
  k_salt_conc,
  tris_salt_conc,
  template_conc,
  allowed.other.binding.ratio,
  allowed.stop.codons,
  allowed.region.definition = c("within", "any"),
  disallowed.mismatch.pos,
  target.temps = NULL,
  fw.primers = NULL,
  updateProgress = NULL
)
```

**Arguments**

primers	Primer data frame to be optimized.
settings	A DesignSettings object.
template.df	Template data frame.
mode.directionality	Primer direction.
required.cvg	Target coverage ratio.
allowed.mismatches	The number of mismatches primers are allowed to have with the templates.

primer_conc	Primer concentration.
na_salt_conc	Sodium ion concentration.
mg_salt_conc	Magnesium ion concentration.
k_salt_conc	Potassium ion concentration.
tris_salt_conc	Tris ion concentration.
template_conc	Template data frame.
allowed.other.binding.ratio	Ratio of primers allowed to bind to non-target regions.
allowed.stop.codons	Consider mismatch binding events that induce stop codons.
allowed.region.definition	Definition of the target binding sites used for evaluating the coverage. If allowed.region.definition is within, primers have to lie within the allowed binding region. If allowed.region.definition is any, primers have to overlap with the allowed binding region. The default is that primers have to bind within the target binding region.
disallowed.mismatch.pos	The number of positions from the primer 3' end where mismatches should not be allowed. All primers binding templates with mismatches within disallowed.mismatch.pos from the 3' end are disregarded.
target.temps	Target melting temperatures for optimized sets in Celsius.
fw.primers	List with already optimized primer data frames corresponding to target.temps.
updateProgress	Shiny progress callback function.

**Value**

List with optimization data.

---

select_best_binding	<i>Selection of Best (smallest number of mismatches) Binding Event per Template Coverage Event.</i>
---------------------	---

---

**Description**

Selection of Best (smallest number of mismatches) Binding Event per Template Coverage Event.

**Usage**

```
select_best_binding(binding, fw.mm.info)
```

**Arguments**

binding	Binding information.
fw.mm.info	Info about mismatches.

**Value**

A list with entries 'fw' and 'rev' giving the best indices of primers.

---

selenium.installed     *Determination if Selenium is installed.*

---

### Description

Checks whether selenium module for python is installed on the system.

### Usage

```
selenium.installed()
```

### Value

TRUE is selenium for python is available, FALSE otherwise.

---

set.new.constraint.value  
*Update Constraint Settings.*

---

### Description

Updates the constraint settings with new values. Sets to the maximal observed values within the limits or less if relax.speed is less than 1.

### Usage

```
set.new.constraint.value(values, relax.speed, cur.limits, cur.setting)
```

### Arguments

values	Observed constraint values considered for the update.
relax.speed	The speed at which the constraints should be relaxed. This value should be in the interval [0,1].
cur.limits	List with current relaxation limits.
cur.setting	List with current constraint settings.

### Value

Relaxed constraint settings according to the given values and cur.limits.

---

set.new.limits	<i>Relaxation of Constraint Limits.</i>
----------------	---

---

### Description

Relaxes the constraint limits by moving according to the difference between the `initial.limits` and `initial.constraints`.

### Usage

```
set.new.limits(
  cur.limits,
  initial.limits,
  initial.constraints,
  con.name = NULL
)
```

### Arguments

<code>cur.limits</code>	List with current constraint settings.
<code>initial.limits</code>	List with initial coonstraint limits.
<code>initial.constraints</code>	List with initial constraint settings before relaxing.
<code>con.name</code>	The constraint for which the settings are to be changed.

### Value

A list with relaxed constraint limits.

---

Settings	<i>Settings Functionalities.</i>
----------	----------------------------------

---

### Description

**DesignSettings** 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.

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

`constraints<-` Sets the active constraints of the provided `DesignSettings` object.

`cvg_constraints` Gets the coverage constraints of the provided `DesignSettings` object.

`cvg_constraints<-` Sets the coverage constraints of the provided `DesignSettings` object.

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

`conOptions<-` Sets the constraint settings of the provided `DesignSettings` object.

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

`constraintLimits<-` Sets the constraint limits of the provided `DesignSettings` object.

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

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

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

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

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

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

`parallel_setup` Registers the specified number of cores with the parallel backend.

### Usage

```
constraints(x)

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

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

cvg_constraints(x)

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

PCR(x)

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

conOptions(x)

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

constraintLimits(x)

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

constraints(x) <- value

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

```

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

cvg_constraints(x) <- value

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

constraintLimits(x) <- value

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

PCR(x) <- value

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

conOptions(x) <- value

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

parallel_setup(cores = NULL)

```

### Arguments

x	A <code>DesignSettings</code> object.
value	An object to be used in one of the setters. For <code>constraints&lt;-</code> and <code>constraintLimits&lt;-</code> , a list with constraint settings or boundaries. 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 <code>min</code> and <code>max</code> . For <code>conOptions&lt;-</code> , a list with constraint options. The permissible fields of the list and their types are documented in the <a href="#">ConstraintOptions</a> class. For <code>cvg_constraints&lt;-</code> , 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 <code>min</code> and <code>max</code> . The permissible constraint identifiers are documented in the <a href="#">CoverageConstraints</a> class. For <code>PCR&lt;-</code> , a named list providing PCR conditions. The permissible fields of the list and their types are documented in the <a href="#">PCR_Conditions</a> class.
cores	A numeric providing the number of cores to use. The default is <code>NULL</code> such that half the number of available cores are used.

### Details

Note that for the `DesignSettings` class, 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`.

For an overview of permissible constraints, please consider the [ConstraintSettings](#) documentation.

**Value**

The `ConstraintSettings` constructor defines a new `ConstraintSettings` object.

The `CoverageConstraints` constructor initializes a new `CoverageConstraints` object.

The `ConstraintOptions` constructor returns a new `ConstraintOptions` object.

The `PCR_Conditions` constructor defines a new `PCR_Conditions` object.

The `DesignSettings` constructor defines a `DesignSettings` object.

`constraints` gets a list with the active constraint settings.

`cvg_constraints` returns the list of active coverage constraints.

`PCR` gets the list of PCR conditions defined in the provided `DesignSettings` object.

`conOptions` returns a list with constraint options.

`constraintLimits` gets the list of constraint limits.

`constraints<-` sets the list of constraints in a `DesignSettings` object.

`cvg_constraints<-` sets the list of coverage constraints in the provided `DesignSettings` object.

`constraintLimits<-` sets the list of constraint limits in the provided `DesignSettings` object.

`PCR<-` sets the constraint options in the provided `DesignSettings` object.

`conOptions<-` sets the specified list of constraint options in the provided `DesignSettings` object.

`parallel_setup` returns NULL.

**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.

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

`settings` For `ConstraintSettings`, a named list containing the settings for 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).

For PCR\_Conditions, a named list with PCR conditions. 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.

**primer\_concentration:** The molar concentration of the PCR primers.

**template\_concentration:** The molar concentration of the PCR templates.

For CoverageConstraints, a 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 maximum 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).

For `ConstraintOptions`, a 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.

### **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 temperature 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 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**

The following external programs are required for constraint computations:

**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`

The following external programs are required for computing the coverage 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**

[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.

**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"]

# 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

# 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

# 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

# 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

# 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

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

# 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

# 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
```

```
# 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
# Use two cores for parallel processing:
parallel_setup(2)
```

---

shannon.entropy      *Shannon Entropy*

---

### Description

Computation of Shannon entropy for an alignment.

### Usage

```
shannon.entropy(ali)
```

### Arguments

ali                    An alignment of primer sequences.

### Value

The Shannon entropy for the alignment.

---

solve.ILP              *Solve an ILP*

---

### Description

Constructs and solves an ILP and outputs a list with the results.

### Usage

```
## S3 method for class 'ILP'
solve(
  cur.D,
  cur.G,
  cur.settings,
  cur.cvg.matrix,
  time.limit,
  required.cvg,
  primer.df,
  template.df
)
```

**Arguments**

cur.D	Binary dimerization matrix.
cur.G	Free energy matrix for cross-dimerization.
cur.settings	Current DesignSettings object.
cur.cvg.matrix	Binary coverage matrix.
time.limit	Time limit for solving the ILP in seconds.
required.cvg	The target coverage of the designed primer set.
primer.df	A Primers object.
template.df	A Templates object.
deltaG.cutoff	Cutoff for dimerization free energy.
deltaG.limit	Relaxation limit for free energy cutoff.

**Value**

List with ILP solution data.

---

split\_str\_by\_index     *Split a sequence*

---

**Description**

Splits a sequence at a specified positions

**Usage**

```
split_str_by_index(target, index)
```

**Arguments**

target	The target string.
index	The position for the split.

**Value**

List with splitted strings

---

stats_plot_data	<i>Combination of Filtering Stats.</i>
-----------------	--

---

**Description**

Summarizes filtering/relaxation statistics for plotting.

**Usage**

```
stats_plot_data(stats, stats.relax)
```

**Arguments**

stats	Statistics of the filtering procedure.
stats.relax	Statistic of the relaxation procedure.

**Value**

A data frame combinin filtering/relaxation stats.

---

store.filtering.sets	<i>Writes Filtering Data Sets to Disk.</i>
----------------------	--

---

**Description**

Writes Filtering Data Sets to Disk.

**Usage**

```
store.filtering.sets(
  filtered.df,
  excluded.df,
  results.loc,
  tag = "",
  stat.df = NULL,
  settings = NULL
)
```

**Arguments**

filtered.df	A filtered Primers set.
excluded.df	A set of Primers that were excluded.
results.loc	The location where to store the data.
tag	A tag for the output files.
stat.df	Data frame with statistics of the filtering procedure.
settings	A DesignSettings object.

**Value**

No return value, writes output to disk.

---

string.list.format     *Format a String List.*

---

**Description**

Formats a string list, summarizing values with percentages.

**Usage**

```
string.list.format(values, order.mode = c("percentage", "value"))
```

**Arguments**

values	The string list to format.
order.mode	How the result should be ordered. For "percentage", the strings are ordered by their percentages, while for "value", the strings are ordered by their values.

**Value**

A formatted string with percentage annotations.

---

string.list.format.total  
*Format Strings*

---

**Description**

Changes the representation of the comma-separated string input.

**Usage**

```
string.list.format.total(values)
```

**Arguments**

values	A comma-separated string with values.
--------	---------------------------------------

**Value**

A percentage-formatted representation of the input string.

---

string.to.IQR	<i>Conversion of Comma-Separated String to IQR String</i>
---------------	---

---

**Description**

Conversion of Comma-Separated String to IQR String

**Usage**

```
string.to.IQR(string.values)
```

**Arguments**

string.values    A vector of comma-separated numeric strings.

**Value**

The IQR corresponding to the input string.

---

subset.ILP	<i>Subset ILP Constructor</i>
------------	-------------------------------

---

**Description**

Constructs an ILP for selecting optimal primer subsets.

**Usage**

```
## S3 method for class 'ILP'
subset(primer.df, template.df, k)
```

**Arguments**

primer.df        Primer data frame to be subsetted.  
 template.df     Template data frame.  
 k                Required number of primers to be selected.

**Details**

Here, "optimal" refers to a subset of a certain size that maximizes the coverage.

**Value**

An ILP for choosing the primer subset of size k with the largest coverage.

---

 TemplatesFunctions      *Template Functionalities.*


---

**Description**

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

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

`update_template_cvg` Annotates the template coverage.

`select_regions_by_conservation` Computes Shannon entropy for the defined binding regions and determines the most conserved regions.

**Usage**

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

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

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

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

**Arguments**

<code>template.df</code>	An object of class <code>Templates</code> .
<code>primer.df</code>	An object of class <code>Primers</code> containing primers with annotated coverage that are to be used to update the template coverage in <code>template.df</code> .
<code>mode.directionality</code>	The directionality of primers/templates.
<code>region.fw</code>	Interval of new binding regions relative to the forward binding region defined in <code>template.df</code> .
<code>region.rev</code>	Interval of new binding regions relative to the reverse binding region defined in <code>template.df</code> .
<code>fw</code>	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 <code>fw</code> is missing, only <code>rev</code> is considered.

<code>rev</code>	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 <code>rev</code> is missing, only <code>fw</code> is considered.
<code>optimize.region</code>	If <code>TRUE</code> , the binding regions specified via <code>fw</code> and <code>rev</code> are adjusted such that binding regions that may form secondary structures are avoided. This feature requires ViennaRNA (see notes). If <code>FALSE</code> (the default), the input binding regions are not modified.
<code>primer.length</code>	A numeric scalar providing the probe length that is used for adjusting the primer binding regions when <code>optimize.region</code> is <code>TRUE</code> .
<code>gap.char</code>	The character in the input file representing gaps.
<code>win.len</code>	The extent of the desired primer binding region. This should be smaller than the <code>allowed.region</code> . The default is 30.
<code>by.group</code>	Shall the determination of binding regions be stratified according to the groups defined in <code>template.df</code> . By default, this is set to <code>TRUE</code> .

### Details

When modifying binding regions with `adjust_binding_regions`, new binding intervals can be specified via `fw` and `rev` for forward and reverse primers, respectively. The new regions should be provided relative to the existing definition of binding regions in `template.df`. For specifying the new binding regions, position 0 refers to 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 relate to positions outside the defined binding region.

Binding regions are defined using `assign_binding_regions`, where 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. Setting the forward interval to (1,30) indicates that the first 30 bases should be used for forward primers and specifying the reverse interval to (1,30) indicates that the last 30 bases should be used for reverse primer binding.

If `optimize.region` is `TRUE`, the input binding region is adjusted such that regions forming secondary structures are avoided.

### Value

`update_template_cvg` returns an object of class `Templates` with updated coverage columns.

`adjust_binding_regions` returns a `Templates` object with updated binding regions.

`assign_binding_regions` returns an object of class `Templates` with newly assigned binding regions.

`select_regions_by_conservation` returns 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**

assign\_binding\_regions requires the program ViennaRNA (<https://www.tbi.univie.ac.at/RNA/>) for adjusting the binding regions when optimize.region is set to TRUE.

select\_regions\_by\_conservation requires the MAFFT software for multiple alignments (<http://mafft.cbrc.jp/alignme>)

**Examples**

```
# Annotate the coverage of the templates
data(Ippolito)
template.df <- update_template_cvg(template.df, primer.df)
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))
data(Ippolito)
# Assignment of individual binding regions
l.fasta.file <- system.file("extdata", "IMGT_data", "templates",
  "Homo_sapiens_IGH_functional_leader.fasta", package = "openPrimerR")
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)
data(Ippolito)
new.template.df <- select_regions_by_conservation(template.df)
```

---

ungap\_sequence

*Ungapping of Sequences.*


---

**Description**

Removes gaps from the input sequences.

**Usage**

```
ungap_sequence(seqs, gap.char = "-")
```

**Arguments**

seqs	The input character vector with sequences
gap.char	The character used to represent gaps.

**Value**

seqs with gaps removed.

---

unify.leaders	<i>Unification of Leaders</i>
---------------	-------------------------------

---

**Description**

Unifies individual binding regions for forward and reverse primers.

**Usage**

```
unify.leaders(l.seq.fw, l.seq.rev, lex.seq, gap.char)
```

**Arguments**

l.seq.fw	Data frame with binding information for forward primers.
l.seq.rev	Data frame with binding information for reverse primers.
lex.seq	Template data frame.
gap.char	The character for indicating alignment gaps.

**Value**

Template data frame with annotated binding regions.

---

update.binding.ranges.by.conservation	<i>Updates Binding Region in the Alignment by conservation.</i>
---------------------------------------	---

---

**Description**

Updates Binding Region in the Alignment by conservation.

**Usage**

```
## S3 method for class 'binding.ranges.by.conservation'
update(
  template.df,
  bins,
  entropy.df,
  gap.char = "-",
  win.len = 30,
  direction = c("fw", "rev")
)
```

**Arguments**

template.df	A Templates object.
bins	A list with DNAbin alignments, one for each group of template sequences.
entropy.df	A data frame with entropy information.
gap.char	The gap character for alignments.
win.len	The desired length of the new binding region.
direction	The direction for which the binding range shall be adjusted.

**Value**

A Templates object with modified binding regions.

---

update.binding.regions

*Update of Binding Regions.*

---

**Description**

Updates the binding regions in the templates by providing new intervals for forward and reverse binding regions.

**Usage**

```
## S3 method for class 'binding.regions'  
update(template.df, opti.regions)
```

**Arguments**

template.df	An object of class Templates.
opti.regions	List with new binding intervals. The list can contain the components fw and rev providing numeric vectors of length 2 providing the start and end of the binding regions in the templates, for forward and reverse binding regions, respectively.

**Value**

A Templates object with updated binding regions.

---

update.constraint.values

*Update of Primer Constraints.*

---

**Description**

Updates the input primer data frame with the computed constraint values.

**Usage**

```
## S3 method for class 'constraint.values'  
update(constraint.df, constraint.values)
```

**Arguments**

constraint.df	Primer data frame.
constraint.values	Data frame with computed constraint values.

**Value**

A primer data frame with updated columns.

---

update.cvg.data      *Update Coverage Information.*

---

### Description

Updates the coverage-related columns in the input primer data frame. Does not modify the entries of template-specific coverage columns such as primer efficiency (comma-separated values).

### Usage

```
## S3 method for class 'cvg.data'
update(
  filtered.df,
  sel,
  template.df,
  mode = c("on_target", "off_target"),
  active.constraints
)
```

### Arguments

filtered.df	Primer data frame.
sel	List with indices of covered templates to be retained, one list with template indices to keep per primer.
template.df	Template data frame.
mode	Either on_target to filter on-target binding events or off_target to filter off-target binding events. The corresponding sel argument should be different.
active.constraints	The active coverage constraints.

### Details

Removes all coverage events of templates whose index is not in sel.

### Value

A primer data frame with updated coverage information.

---

update.individual.binding.region  
*Adjustment of Existing Binding Regions for one Direction.*

---

### Description

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

**Usage**

```
## S3 method for class 'individual.binding.region'  
update(min, max, template.df, mode.directionality)
```

**Arguments**

min	Position where binding should start.
max	End position of binding.
mode.directionality	Directionality of primers.
Template	data frame.

**Details**

Position 0 indicates the first position after the existing binding region. Hence, negative positions adjust the binding region towards the existing binding regions and non-negative positions extend the existing binding region definition away from the existing target region.

**Value**

Template data frame with updated binding regions.

---

update.opti.results    *Augmentation of Optimized Primer Data.*

---

**Description**

Adds melting\_temp\_diff and cross\_dimerization info to optimized sets.

**Usage**

```
## S3 method for class 'opti.results'  
update(primer.df, settings, template.df)
```

**Arguments**

primer.df	A primer data frame.
settings	A DesignSettings object.

**Value**

An updated primer data frame.

---

update\_primer\_binding\_regions  
*Update of Primer Binding Regions.*

---

### Description

Updates the relative primer binding sites in the templates when the template binding regions have changed since the last coverage computation.

### Usage

```
update_primer_binding_regions(primer.df, template.df, old.template.df)
```

### Arguments

primer.df            A Primers data frame.  
 template.df        Templates with the new binding regions.  
 old.template.df    Templates with the old binding regions.

### Value

A Primers object with updated relative binding positions.

---

update\_primer\_cvg        *Updates the Primer Coverage.*

---

### Description

Updates the most important columns in a primer data frame according to the selected coverage definition. Only coverage events with less or equal than the allowed number of mismatches according to the selected coverage definition will be retained.

### Usage

```
update_primer_cvg(  
  primer.df,  
  template.df,  
  allowed.mismatches,  
  cvg.definition = c("constrained", "basic")  
)
```

### Arguments

primer.df            A Primers object.  
 template.df        A Templates object.  
 allowed.mismatches  
                     A numeric giving the maximal number of allowed.mismatches.  
 cvg.definition    The definition of coverage to be used, either "constrained" or "basic".

**Value**

A primer data frame with modified coverage information.

---

validate\_primers      *Validates a Primers Object.*

---

**Description**

Checks whether a Primers object is valid or not.

**Usage**

```
validate_primers(object)
```

**Arguments**

object              An input data frame to be checked for being a primer data frame.

**Value**

TRUE, if the object is valid, FALSE otherwise.

---

validate\_templates      *Validates a Template Object.*

---

**Description**

Checks whether a Templates object is valid or not.

**Usage**

```
validate_templates(object)
```

**Arguments**

object              An input data frame to be checked for being a template data frame.

**Value**

TRUE, if the object is valid, FALSE otherwise.

---

view.cvg.primers      *View the Evaluated Primers.*

---

### Description

Creates a formatted primers table.

### Usage

```
view.cvg.primers(  
  primer.df,  
  template.df,  
  mode.directionality,  
  view.cvg.individual = c("active", "inactive"),  
  for.shiny = TRUE  
)
```

### Arguments

primer.df      A Primers object.  
template.df    A Templates object.  
mode.directionality  
                The direction of the primers.  
view.cvg.individual  
                Whether information on individual coverage events should be retained.  
for.shiny      Whether the table is intended for Shiny (HTML) or not.

### Value

A formatted primer table.

---

view.dimer.df      *Formatted dimerization data.*

---

### Description

Format a dimerization data frame for frontend output.

### Usage

```
view.dimer.df(dimers, type = c("Self", "Cross"))
```

### Arguments

dimers          Dimerization data frame.  
type            Type of dimerization.

### Value

A data frame whose columns are formatted in a user-readable way.

---

view.input.primers      *View the Input Primers.*

---

**Description**

Creates a formatted primers table.

**Usage**

```
view.input.primers(primer.df, mode.directionality, for.shiny = TRUE)
```

**Arguments**

primer.df      A Primers object.  
mode.directionality      The direction of the primers.  
for.shiny      Whether output is intended for Shiny.

**Value**

A formatted primer table.

---

view.primers      *View the Evaluated Primers.*

---

**Description**

Creates a formatted primers table.

**Usage**

```
view.primers(primer.df, template.df)
```

**Arguments**

primer.df      A Primers object.  
template.df      A Templates object.

**Value**

A formatted primer table.

`view.primers.report` *View the Evaluated Primers in the Report.*

---

### Description

Creates a formatted primers table for the report PDF.

### Usage

```
view.primers.report(primer.df, template.df)
```

### Arguments

`primer.df`      A Primers object.  
`template.df`    A Templates object.

### Value

A formatted primer table.

---

`visualize.all.results` *Visualization of Design Results.*

---

### Description

Visualizes all results from designing primers.

### Usage

```
visualize.all.results(  
  sample,  
  filtering.results.loc,  
  opti.results.loc,  
  primer_conc,  
  na_salt_conc,  
  mg_salt_conc,  
  k_salt_conc,  
  tris_salt_conc,  
  settings,  
  mode.directionality,  
  used.settings,  
  required.cvg  
)
```

**Arguments**

sample	Identifier of the design run.
filtering.results.loc	Location of filtering results.
opti.results.loc	Location of optimization results.
primer_conc	Primer concentration.
na_salt_conc	Sodium ion concentration.
mg_salt_conc	Magnesium ion concentration.
k_salt_conc	Potassium ion concentration.
tris_salt_conc	Tris ion concentration.
settings	The DesignSettings object.
mode.directionality	Strand direction for which primers were designed.
used.settings	A list with the used settings for optimization (fields "fw" and "rev").
required.cvg	The required coverage.
template_conc	Template concentration.

**Value**

Writes visualizations to files in.

---

visualize.filtering.results

*Visualization of Filtering Results.*

---

**Description**

Visualizes the filtering results.

**Usage**

```
visualize.filtering.results(
  sample,
  results.loc,
  mode.directionality,
  excluded.df,
  template.df,
  filtered.df,
  filtered.stats,
  stats.relax,
  primer_conc,
  na_salt_conc,
  mg_salt_conc,
  k_salt_conc,
  tris_salt_conc,
  settings,
  required.cvg
)
```

**Arguments**

sample	Primer design run identifier.
results.loc	Location where the filtering results are stored.
mode.directionality	Design direction.
excluded.df	Data frame with excluded primers.
template.df	Template data frame.
filtered.df	Primer data frame containing the primers that passed the constraints.
stats.relax	Filtering statistics after relaxation.
primer_conc	Primer concentration.
na_salt_conc	Sodium ion concentration.
mg_salt_conc	Magnesium ion concentration.
k_salt_conc	Potassium ion concentration.
tris_salt_conc	Tris ion concentration.
settings	A DesignSettings object.
required.cvg	The required coverage.
template_conc	Template concentration.

**Value**

Write-out of filtering results.

---

were.constraints.relaxed

*Check for Relaxation*

---

**Description**

Determines whether constraints were relaxed or not.

**Usage**

```
were.constraints.relaxed(used.constraints, input.constraints)
```

**Arguments**

used.constraints	The constraints that were used during the optimization.
input.constraints	The user-specified constraints.

**Value**

If `input.constraints` was relaxed TRUE is returned, otherwise FALSE.

---

write.out.primer.info *Write Out Optimization Data*

---

### Description

Writes out all data relating to the optimization of primers.

### Usage

```
write.out.primer.info(
  opti.results.loc,
  optimal.primers.data,
  mode.directionality,
  settings,
  sample.name,
  template.df,
  max.degen
)
```

### Arguments

opti.results.loc	Folder where optimization data reside.
optimal.primers.data	List with optimization results.
mode.directionality	Direction of primers.
settings	Settings used in the optimization procedure. List containing fw, rev settings.
sample.name	Name of template sample.
template.df	Template data frame.
max.degen	Maximal degeneracy of primers.

### Value

Write-out of primer information to opti.results.loc.

---

xmlToChar	<i>Conversion of XML to Character.</i>
-----------	--

---

### Description

Converts an XML object to a character string.

### Usage

```
xmlToChar(xml)
```

**Arguments**

`xml`                    An xml object to be converted to character.

**Value**

A character vector.

# Index

- \* **Creation**
  - compute.Tm.sets, [65](#)
- \* **Cross-dimerization**
  - compute.Tm.sets, [65](#)
- \* **FALSE**
  - check.init.primer.length, [33](#)
- \* **Melting**
  - compute.Tm.sets, [65](#)
- \* **Sets.**
  - compute.Tm.sets, [65](#)
- \* **Settings**
  - Settings, [225](#)
- \* **Stratifies**
  - compute.Tm.sets, [65](#)
- \* **Temperature**
  - compute.Tm.sets, [65](#)
- \* **according**
  - compute.Tm.sets, [65](#)
- \* **and**
  - compute.Tm.sets, [65](#)
- \* **a**
  - compute.Tm.sets, [65](#)
- \* **between**
  - compute.Tm.sets, [65](#)
- \* **case**
  - compute.Tm.sets, [65](#)
- \* **checked**
  - compute.Tm.sets, [65](#)
- \* **checks**
  - compute.Tm.sets, [65](#)
- \* **constraints.**
  - compute.Tm.sets, [65](#)
- \* **datasets**
  - Data, [86](#)
- \* **data**
  - compute.Tm.sets, [65](#)
- \* **forward**
  - compute.Tm.sets, [65](#)
- \* **frames**
  - compute.Tm.sets, [65](#)
- \* **here,**
  - compute.Tm.sets, [65](#)
- \* **internal**
  - AbstractConstraintSettings-class, [10](#)
  - add.coverage.constraints, [11](#)
  - add.dimerization.constraints, [11](#)
  - add.uniform.leaders.to.seqs, [12](#)
  - adjust.ORF.start, [12](#)
  - align.seqs, [13](#)
  - align.structures, [13](#)
  - ancestor\_of, [16](#)
  - annealing.temp.rule.of.thumb, [17](#)
  - annotate.binding.events, [17](#)
  - apply.constraint, [18](#)
  - apply.constraint.list, [18](#)
  - assign\_binding\_regions.character, [19](#)
  - assign\_binding\_regions.numeric, [20](#)
  - augment.primer.cvg, [21](#)
  - batchify, [21](#)
  - batchify.simple, [22](#)
  - batchify.temp, [22](#)
  - build.gain.df, [23](#)
  - build.ILP.df, [23](#)
  - build.tool.overview, [24](#)
  - build\_leader\_df, [25](#)
  - call.melt, [25](#)
  - call.melt.single, [26](#)
  - cascaded.filter, [27](#)
  - cascaded.filter.quick, [28](#)
  - cbind.Primers, [29](#)
  - cbind.Templates, [29](#)
  - cbind2,Primers,ANY-method, [30](#)
  - cbind2,Templates,ANY-method, [31](#)
  - check.3prime.hexamers, [32](#)
  - check.3prime.mismatches, [32](#)
  - check.init.primer.length, [33](#)
  - check.init.primer.length.single, [34](#)
  - check.mutations, [34](#)
  - check.template.constraints, [35](#)
  - check.tool.function, [36](#)
  - check.tool.installation, [36](#)
  - check\_constraint\_settings\_validity, [38](#)

- check\_constraints\_comparison, 37
- check\_correspondence, 38
- check\_cvq\_constraints, 39
- check\_interval, 40
- check\_limit\_value, 41
- check\_limits, 40
- check\_names, 41
- check\_report\_deps, 42
- check\_restriction\_sites\_single, 42
- check\_setting, 43
- check\_settings\_validity, 43
- combine.binding.events, 44
- combine.strings, 44
- comp, 45
- compare.constraints, 45
- comparison.cvq, 46
- comparison.stats.raw, 46
- complement.sequence, 47
- compute.all.cross.dimers, 47
- compute.all.cross.dimers.frontend, 48
- compute.all.cross.dimers.unfiltered, 49
- compute.all.primer.subsets.ILP, 50
- compute.all.self.dimers, 50
- compute.all.self.dimers.frontend, 51
- compute.basic.details, 52
- compute.constraints, 53
- compute.covered.Ta, 54
- compute.dimer.matrix, 55
- compute.entropy, 55
- compute.entropy.melting.temp, 56
- compute.gc.ratio, 57
- compute.melting.temps, 57
- compute.melting.temps.thermo, 58
- compute.mismatch.table, 59
- compute.primer.energies, 59
- compute.secondary.structures, 60
- compute.sodium.equivalent.conc, 61
- compute.structure.vienna, 62
- compute.Ta, 63
- compute.template.secondary.structures, 64
- compute.Tm.baldino, 64
- compute.Tm.sets, 65
- compute.unique.covered.idx, 67
- compute\_annealing\_temp, 67
- con\_select, 73
- condition, 68
- consecutive.GC.count, 69
- constraints.to.df, 69
- constraints.xml.format, 70
- constraints\_to\_unit, 70
- convert.from.iupac, 71
- convert.PCR.units, 71
- convert.temperature, 72
- convert.to.iupac, 72
- copy.melt.config, 73
- covered.primers.to.ID.string, 73
- covered.seqs.to.ID.string, 74
- covered.seqs.to.idx, 74
- create.constraint.table, 75
- create.constraint.XML, 75
- create.cvq.text, 76
- create.G.matrix, 76
- create.initial.primer.set, 77
- create.k.mers, 78
- create.kmer, 78
- create.options.table, 79
- create.other.table, 79
- create.PCR.table, 80
- create.primer.ranges, 80
- create.primers.naive, 81
- create.primers.tree, 82
- create.Tm.brackets, 83
- create.uniform.leaders, 83
- create\_fulfilled\_counts, 84
- create\_report,list,list-method, 84
- create\_report,Primers,Templates-method, 85
- design.primers.single, 87
- detect.gap.columns, 89
- dimerization.table, 89
- dir.copy, 90
- disambiguate.primers, 90
- estimate.cvq, 91
- estimate.cvq.dir, 91
- eval.comparison.primers, 92
- eval.constraints, 92
- evaluate.basic.cvq, 93
- evaluate.constrained.cvq, 94
- evaluate.cvq, 94
- evaluate.diff.primer.cvq, 95
- evaluate.fw.rev.combinations, 96
- evaluate.GC.clamp, 96
- evaluate.primer.cvq, 97
- evaluate.template.constraints, 97
- exclude.cols, 98
- filter.by.constraints, 98
- filter.comparison.primers, 99
- filter.primer.candidates, 100
- filter.primer.set.opti, 100
- filter\_primers.by.Tm.delta, 102

- filterLimits, 101
- filters, 102
- fix\_constraint\_boundaries, 103
- format.constraints, 103
- format.seq.ali, 104
- format.seqs.tex, 104
- get.3prime.mismatch.pos, 105
- get.analysis.mode, 105
- get.consensus.seq, 106
- get.constraint.value.idx, 106
- get.constraint.values, 107
- get.coverage.matrix, 107
- get.covered.templates, 108
- get.cross.dimers, 108
- get.cvg.constraint.settings, 109
- get.cvg.gain, 110
- get.delta.G, 110
- get.dimer.data, 111
- get.duplex.energies, 111
- get.eval.cols, 112
- get.extension, 112
- get.ILP.vars, 113
- get.init.file.name, 113
- get.leader.exon.regions, 114
- get.leader.exon.regions.single, 115
- get.matches, 115
- get.melting.temp.diff, 116
- get.merge.idx, 116
- get.missing.df, 117
- get.ORFs, 117
- get.other.constraint.settings, 118
- get.PCR.settings, 118
- get.plot.height, 119
- get.primer.binding.idx, 120
- get.primer.identifier.string, 120
- get.redundant.cols, 121
- get.relative.binding.pos, 122
- get.run.names, 122
- get.self.dimers, 123
- get.sets.from.decisions, 123
- get.static.tool.info, 124
- get.tree.seqs, 124
- get.unlist.idx, 125
- get\_constraint\_deviation\_data, 125
- get\_covered.vanilla, 126
- get\_cvg\_stats,list-method, 126
- get\_cvg\_stats,Primers-method, 127
- get\_max\_set\_coverage, 128
- get\_plot\_primer\_data, 129
- get\_primer\_cvg\_mm\_plot\_df, 129
- get\_report\_fname, 130
- get\_template\_cvg\_data, 130
- hclust.tree, 131
- highlight.mismatch, 131
- html.format.structure, 132
- I.cvg, 132
- ILPConstrained, 133
- initialize.primer.set, 133
- insert\_str, 139
- interleave, 140
- J.cvg, 140
- joule.to.cal, 141
- listToXml, 141
- merge.ambig.primers, 142
- merge.binding.information, 142
- merge.primer.entries, 143
- merge.primer.entries.single, 144
- merge.select, 144
- merge.template.decisions, 145
- mismatch.info, 145
- mismatch.mutation.check, 146
- mismatch.string.to.list, 146
- modify.col.rep, 147
- my.disambiguate, 147
- my.error, 148
- my.read.fasta, 148
- my.warning, 149
- my\_ggsave, 149
- my\_rbind, 150
- nbr.of.repeats, 150
- nbr.of.runs, 151
- opti, 151
- optiLimits, 152
- optimize.ILP, 152
- optimize.primer.cvg, 154
- optimize.template.binding.regions.dir, 155
- optimize.template.binding.regions.single, 155
- pair\_primers, 158
- parse.constraints, 159
- parse.header, 159
- parse.IMG.t.gene.groups, 160
- parse.oligo.results, 160
- plot.all.cvg.info, 161
- plot.all.filtering.stats, 162
- plot.Delta.DeltaG, 162
- plot.dimer.dist, 163
- plot.excluded.hist, 163
- plot.filtering.runtime, 164
- plot.filtering.stats, 164
- plot.filtering.stats.cvg, 165
- plot\_constraint,list-method, 170

- plot\_constraint, Primers-method, 171
- plot\_constraint.histogram, 172
- plot\_constraint.histogram.nbr.mismatches, 173
- plot\_constraint.histogram.primers efficienciest, 173
- plot\_constraint\_deviation, list-method, 174
- plot\_constraint\_deviation, Primers-method, 174
- plot\_constraint\_fulfillment, list-method, 175
- plot\_constraint\_fulfillment, Primers-method, 176
- plot\_cvg\_constraints, list-method, 176
- plot\_cvg\_constraints, Primers-method, 177
- plot\_primer.comparison.box, 177
- plot\_primer.comparison.mismatches, 178
- plot\_primer\_binding\_regions, list, list-method, 179
- plot\_primer\_binding\_regions, Primers, Templates-method, 180
- plot\_primer\_cvg, list, list-method, 180
- plot\_primer\_cvg, Primers, Templates-method, 181
- plot\_primer\_cvg\_mismatches, 181
- plot\_primer\_cvg\_unstratified, 182
- plot\_template\_cvg, list, list-method, 183
- plot\_template\_cvg, Primers, Templates-method, 183
- plot\_template\_cvg\_comparison\_mismatch, 184
- plot\_template\_cvg\_comparison\_unstratified, 184
- plot\_template\_cvg\_mismatches, 185
- plot\_template\_cvg\_unstratified, 185
- plot\_template\_structure, 186
- pos.to.range, 186
- predict\_coverage, 187
- prefilter.primers.candidates, 187
- prepare.constraint.plot, 188
- prepare.dimer.seqs, 188
- prepare\_mm\_plot, 189
- prepare\_template\_cvg\_mm\_data, 189
- primer.binding.regions.data, 190
- primer.coverage.for.groups, 190
- primer.set.parameter.stats, 191
- rbind.primers.data, 198
- rbind.Primers, 199
- rbind.Templates, 199
- read.leaders, 200
- read.secondary.structure.raw, 200
- read.sequences, 201
- read\_primers.internal, 201
- read\_primers\_csv, 202
- read\_primers\_multiple, 202
- read\_templates\_csv, 203
- read\_templates\_fasta, 203
- read\_templates\_multiple, 204
- read\_templates\_single, 205
- relax.constraints, 206
- relax.opti.constraints, 207
- remove.redundant.cols, 208
- remove.seqs.by.keyword, 208
- rename.constraint.options, 209
- render\_report, 209
- reorder.primers.table, 210
- restriction\_ali, 210
- restriction\_hits, 211
- restriction\_match, 211
- retrieve.leader.region, 212
- rev.comp.sequence, 212
- rev.sequence, 213
- sanitize\_path, 214
- score.conservations, 214
- select.allowed.binding.events, 217
- select.best.ILP, 217
- select.best.opti.result, 218
- select.best.primers.idx, 218
- select.best.primers.set, 219
- select.binding.events, 220
- select.constraints, 220
- select.min.cross.idx, 221
- select.primers.region.by.conservations, 221
- select.primers.by.cvg, 222
- select\_best\_binding, 223
- selenium.installed, 224
- set.new.constraint.value, 224
- set.new.limits, 225
- shannon.entropy, 234
- solve.ILP, 234
- split\_str\_by\_index, 235
- stats\_plot\_data, 236
- store.filtering.sets, 236
- string.list.format, 237
- string.list.format.total, 237

- string.to.IQR, 238
- subset.ILP, 238
- ungap\_sequence, 241
- unify.leaders, 242
- update.binding.ranges.by.conserva-  
tion, 242
- update.binding.regions, 243
- update.constraint.values, 243
- update.cvg.data, 244
- update.individual.binding.region,  
244
- update.opti.results, 245
- update\_primer\_binding\_regions, 246
- update\_primer\_cvg, 246
- validate\_primers, 247
- validate\_templates, 247
- view.cvg.primers, 248
- view.dimer.df, 248
- view.input.primers, 249
- view.primers, 249
- view.primers.report, 250
- visualize.all.results, 250
- visualize.filtering.results, 251
- were.constraints.relaxed, 252
- write.out.primers.info, 253
- xmlToChar, 253
- \* **in**  
compute.Tm.sets, 65
- \* **is**  
compute.Tm.sets, 65
- \* **list(primers.fw)**  
compute.Tm.sets, 65
- \* **list**  
compute.Tm.sets, 65
- \* **melting**  
compute.Tm.sets, 65
- \* **of**  
compute.Tm.sets, 65
- \* **optimization**  
compute.Tm.sets, 65
- \* **otherwise.**  
check.init.primers.length, 33
- \* **primers**  
compute.Tm.sets, 65
- \* **primer**  
compute.Tm.sets, 65
- \* **provided.**  
compute.Tm.sets, 65
- \* **reverse**  
compute.Tm.sets, 65
- \* **run**  
compute.Tm.sets, 65
- \* **second**  
compute.Tm.sets, 65
- \* **settings functions**  
Settings, 225
- \* **temperature-dependent**  
compute.Tm.sets, 65
- \* **temperatures**  
compute.Tm.sets, 65
- \* **templates**  
Plots, 165
- \* **their**  
compute.Tm.sets, 65
- \* **this**  
compute.Tm.sets, 65
- \* **to**  
compute.Tm.sets, 65
- \* **with**  
compute.Tm.sets, 65
- [,Primers,ANY-method  
(cbind2,Primers,ANY-method), 30
- [,Primers-method  
(cbind2,Primers,ANY-method), 30
- [,Templates,ANY-method  
(cbind2,Templates,ANY-method),  
31
- [,Templates-method  
(cbind2,Templates,ANY-method),  
31
- \$<-,Primers-method  
(cbind2,Primers,ANY-method), 30
- \$<-,Templates-method  
(cbind2,Templates,ANY-method),  
31
- AbstractConstraintSettings  
(AbstractConstraintSettings-class),  
10
- AbstractConstraintSettings-class, 10
- add.coverage.constraints, 11
- add.dimerization.constraints, 11
- add.uniform.leaders.to.seqs, 12
- adjust.ORF.start, 12
- adjust\_binding\_regions  
(TemplatesFunctions), 239
- align.seqs, 13
- align.structures, 13
- AnalysisStats, 14
- ancestor\_of, 16
- annealing.temp.rule.of.thumb, 17
- annotate.binding.events, 17
- apply.constraint, 18
- apply.constraint.list, 18
- assign\_binding\_regions, 137

- assign\_binding\_regions
  - (TemplatesFunctions), 239
- assign\_binding\_regions.character, 19
- assign\_binding\_regions.numeric, 20
- augment.primers.cvg, 21
  
- batchify, 21
- batchify.simple, 22
- batchify.temp, 22
- build.gain.df, 23
- build.ILP.df, 23
- build.tool.overview, 24
- build\_leader\_df, 25
  
- CalculateEfficiencyPCR, 56, 60
- call.melt, 25
- call.melt.single, 26
- cascaded.filter, 27
- cascaded.filter.quick, 28
- cbind.Primers, 29
- cbind.Templates, 29
- cbind2, Primers, ANY-method, 30
- cbind2, Templates, ANY-method, 31
- check.3prime.hexamers, 32
- check.3prime.mismatches, 32
- check.init.primers.length, 33
- check.init.primers.length.single, 34
- check.mutations, 34
- check.template.constraints, 35
- check.tool.function, 36
- check.tool.installation, 36
- check\_constraint\_settings\_validity, 38
- check\_constraints, 9, 42, 137, 138
- check\_constraints (PrimerEval), 195
- check\_constraints\_comparison, 37
- check\_correspondence, 38
- check\_cvg\_constraints, 39
- check\_interval, 40
- check\_limit\_value, 41
- check\_limits, 40
- check\_names, 41
- check\_report\_deps, 42
- check\_restriction\_sites (PrimerEval), 195
- check\_restriction\_sites\_single, 42
- check\_setting, 43
- check\_settings\_validity, 43
- classify\_design\_problem (PrimerDesign), 191
- combine.binding.events, 44
- combine.strings, 44
- comp, 45
- compare.constraints, 45
- comparison.cvg, 46
- comparison.stats.raw, 46
- complement.sequence, 47
- compute.all.cross.dimers, 47
- compute.all.cross.dimers.frontend, 48
- compute.all.cross.dimers.unfiltered, 49
- compute.all.primers.subsets.ILP, 50
- compute.all.self.dimers, 50
- compute.all.self.dimers.frontend, 51
- compute.basic.details, 52
- compute.constraints, 53
- compute.covered.Ta, 54
- compute.dimer.matrix, 55
- compute.efficiency, 55
- compute.empiric.melting.temp, 56
- compute.gc.ratio, 57
- compute.melting.temps, 57
- compute.melting.temps.thermo, 58
- compute.mismatch.table, 59
- compute.primers.efficiencys, 59
- compute.secondary.structures, 60
- compute.sodium.equivalent.conc, 61
- compute.structure.vienna, 62
- compute.Ta, 63
- compute.template.secondary.structures, 64
- compute.Tm.baldino, 64
- compute.Tm.sets, 65
- compute.unique.covered.idx, 67
- compute\_annealing\_temp, 67
- con\_select, 73
- condition, 68
- conOptions, 9, 232
- conOptions (Settings), 225
- conOptions, DesignSettings-method (Settings), 225
- conOptions<- (Settings), 225
- conOptions<-, DesignSettings-method (Settings), 225
- consecutive.GC.count, 69
- constraintLimits, 9, 232
- constraintLimits (Settings), 225
- constraintLimits, DesignSettings-method (Settings), 225
- constraintLimits<- (Settings), 225
- constraintLimits<-, DesignSettings-method (Settings), 225
- ConstraintOptions, 227, 230
- ConstraintOptions (Settings), 225
- ConstraintOptions-class (Settings), 225
- constraints, 9, 232

- constraints (Settings), 225
- constraints, AbstractConstraintSettings-method (Settings), 225
- constraints, DesignSettings-method (Settings), 225
- constraints.to.df, 69
- constraints.xml.format, 70
- constraints<- (Settings), 225
- constraints<-, AbstractConstraintSettings, list-method (Settings), 225
- constraints<-, DesignSettings, list-method (Settings), 225
- constraints\_to\_unit, 70
- ConstraintSettings, 197, 227–229
- ConstraintSettings (Settings), 225
- ConstraintSettings-class (Settings), 225
- convert.from.iupac, 71
- convert.PCR.units, 71
- convert.temperature, 72
- convert.to.iupac, 72
- copy.melt.config, 73
- CoverageConstraints, 156, 227, 228, 230
- CoverageConstraints (Settings), 225
- CoverageConstraints-class (Settings), 225
- covered.primers.to.ID.string, 73
- covered.seqs.to.ID.string, 74
- covered.seqs.to.idx, 74
- create.constraint.table, 75
- create.constraint.XML, 75
- create.cvg.text, 76
- create.G.matrix, 76
- create.initial.primer.set, 77
- create.k.mers, 78
- create.kmer, 78
- create.options.table, 79
- create.other.table, 79
- create.PCR.table, 80
- create.primer.ranges, 80
- create.primers.naive, 81
- create.primers.tree, 82
- create.Tm.brackets, 83
- create.uniform.leaders, 83
- create\_coverage\_xls (Output), 156
- create\_fulfilled\_counts, 84
- create\_report, 9
- create\_report (Output), 156
- create\_report, list, list-method, 84
- create\_report, Primers, Templates-method, 85
- cvg\_constraints, 9, 232
- cvg\_constraints (Settings), 225
- cvg\_constraints, DesignSettings-method (Settings), 225
- cvg\_constraints<- (Settings), 225
- cvg\_constraints<-, DesignSettings-method (Settings), 225
- Data, 86
- design\_primers, 9, 225
- design\_primers (PrimerDesign), 191
- design\_primers.single, 87
- DesignSettings, 9, 39, 194, 198
- DesignSettings (Settings), 225
- DesignSettings-class (Settings), 225
- detect.gap.columns, 89
- dimerization.table, 89
- dir.copy, 90
- disambiguate.primers, 90
- estimate.cvg, 91
- estimate.cvg.dir, 91
- eval.comparison.primers, 92
- eval.constraints, 92
- evaluate.basic.cvg, 93
- evaluate.constrained.cvg, 94
- evaluate.cvg, 94
- evaluate.diff.primer.cvg, 95
- evaluate.fw.rev.combinations, 96
- evaluate.GC.clamp, 96
- evaluate.primer.cvg, 97
- evaluate.template.constraints, 97
- exclude.cols, 98
- feature.matrix (Data), 86
- filter.by.constraints, 98
- filter.comparison.primers, 99
- filter.primer.candidates, 100
- filter.primer.set.opti, 100
- filter\_primers, 225
- filter\_primers (PrimerEval), 195
- filter\_primers.by.Tm.delta, 102
- filterLimits, 101
- filterLimits, DesignSettings-method (filterLimits), 101
- filters, 102
- filters, DesignSettings-method (filters), 102
- fix\_constraint\_boundaries, 103
- format.constraints, 103
- format.seq.ali, 104
- format.seqs.tex, 104
- get.3prime.mismatch.pos, 105
- get.analysis.mode, 105

- get.consensus.seq, 106
- get.constraint.value.idx, 106
- get.constraint.values, 107
- get.coverage.matrix, 107
- get.covered.templates, 108
- get.cross.dimers, 108
- get.cvg.constraint.settings, 109
- get.cvg.gain, 110
- get.delta.G, 110
- get.dimer.data, 111
- get.duplex.energies, 111
- get.eval.cols, 112
- get.extension, 112
- get.ILP.vars, 113
- get.init.file.name, 113
- get.leader.exon.regions, 114
- get.leader.exon.regions.single, 115
- get.matches, 115
- get.melting.temp.diff, 116
- get.merge.idx, 116
- get.missing.df, 117
- get.ORFs, 117
- get.other.constraint.settings, 118
- get.PCR.settings, 118
- get.plot.height, 119
- get.primer.binding.idx, 120
- get.primer.identifier.string, 120
- get.redundant.cols, 121
- get.relative.binding.pos, 122
- get.run.names, 122
- get.self.dimers, 123
- get.sets.from.decisions, 123
- get.static.tool.info, 124
- get.tree.seqs, 124
- get.unlist.idx, 125
- get\_comparison\_table (AnalysisStats), 14
- get\_constraint\_deviation\_data, 125
- get\_covered.vanilla, 126
- get\_cvg\_ratio (AnalysisStats), 14
- get\_cvg\_stats, 9
- get\_cvg\_stats (AnalysisStats), 14
- get\_cvg\_stats, list-method, 126
- get\_cvg\_stats, Primers-method, 127
- get\_cvg\_stats\_primer (AnalysisStats), 14
- get\_initial\_primers (PrimerDesign), 191
- get\_max\_set\_coverage, 128
- get\_plot\_primer\_data, 129
- get\_primer\_cvg\_mm\_plot\_df, 129
- get\_report\_fname, 130
- get\_template\_cvg\_data, 130
  
- hclust.tree, 131
- highlight.mismatch, 131
  
- html.format.structure, 132
  
- I.cvg, 132
- ILPConstrained, 133
- initialize.primer.set, 133
- Input, 134
- insert\_str, 139
- interleave, 140
  
- J.cvg, 140
- joule.to.cal, 141
  
- listToXml, 141
  
- merge.ambig.primers, 142
- merge.binding.information, 142
- merge.primer.entries, 143
- merge.primer.entries.single, 144
- merge.select, 144
- merge.template.decisions, 145
- mismatch.info, 145
- mismatch.mutation.check, 146
- mismatch.string.to.list, 146
- modify.col.rep, 147
- my.disambiguate, 147
- my.error, 148
- my.read.fasta, 148
- my.warning, 149
- my\_ggsave, 149
- my\_rbind, 150
  
- nbr.of.repeats, 150
- nbr.of.runs, 151
  
- openPrimeR (openPrimeR-package), 9
- openPrimeR-package, 9
- opti, 151
- opti, DesignSettings-method (opti), 151
- optiLimits, 152
- optiLimits, DesignSettings-method (optiLimits), 152
- optimize.ILP, 152
- optimize.primer.cvg, 154
- optimize.template.binding.regions.dir, 155
- optimize.template.binding.regions.single, 155
- Output, 156
  
- pair\_primers, 158
- parallel\_setup (Settings), 225
- parse.constraints, 159
- parse.header, 159
- parse.IMGT.gene.groups, 160

- parse.oligo.results, 160
- PCR, 9, 232
- PCR (Settings), 225
- PCR, DesignSettings-method (Settings), 225
- PCR<- (Settings), 225
- PCR<- , DesignSettings-method (Settings), 225
- PCR\_Conditions, 227, 228
- PCR\_Conditions (Settings), 225
- PCR\_Conditions-class (Settings), 225
- plot.all.cvg.info, 161
- plot.all.filtering.stats, 162
- plot.Delta.DeltaG, 162
- plot.dimer.dist, 163
- plot.excluded.hist, 163
- plot.filtering.runtime, 164
- plot.filtering.stats, 164
- plot.filtering.stats.cvg, 165
- plot\_conservation (Plots), 165
- plot\_constraint (Plots), 165
- plot\_constraint, list-method, 170
- plot\_constraint, Primers-method, 171
- plot\_constraint.histogram, 172
- plot\_constraint.histogram.nbr.mismatches, 173
- plot\_constraint.histogram.primers.efficiencies, 173
- plot\_constraint\_deviation, 9
- plot\_constraint\_deviation (Plots), 165
- plot\_constraint\_deviation, list-method, 174
- plot\_constraint\_deviation, Primers-method, 174
- plot\_constraint\_fulfillment (Plots), 165
- plot\_constraint\_fulfillment, list-method, 175
- plot\_constraint\_fulfillment, Primers-method, 176
- plot\_cvg\_constraints (Plots), 165
- plot\_cvg\_constraints, list-method, 176
- plot\_cvg\_constraints, Primers-method, 177
- plot\_cvg\_vs\_set\_size (Plots), 165
- plot\_penalty\_vs\_set\_size (Plots), 165
- plot\_primer (Plots), 165
- plot\_primer.comparison.box, 177
- plot\_primer.comparison.mismatches, 178
- plot\_primer\_binding\_regions (Plots), 165
- plot\_primer\_binding\_regions, list, list-method, 179
- plot\_primer\_binding\_regions, Primers, Templates-method, 180
- plot\_primer\_cvg (Plots), 165
- plot\_primer\_cvg, list, list-method, 180
- plot\_primer\_cvg, Primers, Templates-method, 181
- plot\_primer\_cvg\_mismatches, 181
- plot\_primer\_cvg\_unstratified, 182
- plot\_primer\_subsets (Plots), 165
- plot\_template\_cvg (Plots), 165
- plot\_template\_cvg, list, list-method, 183
- plot\_template\_cvg, Primers, Templates-method, 183
- plot\_template\_cvg\_comparison\_mismatch, 184
- plot\_template\_cvg\_comparison\_unstratified, 184
- plot\_template\_cvg\_mismatches, 185
- plot\_template\_cvg\_unstratified, 185
- plot\_template\_structure, 186
- Plots, 165
- pos.to.range, 186
- predict\_coverage, 187
- prefilter.primers.candidates, 187
- prepare.constraint.plot, 188
- prepare.dimer.seqs, 188
- prepare\_mm\_plot, 189
- prepare\_template\_cvg\_mm\_data, 189
- primers.binding.regions.data, 190
- primers.coverage.for.groups, 190
- primers.data (Data), 86
- primers.df (Data), 86
- primers.set.parameter.stats, 191
- primers\_significance, 168
- primers\_significance (PrimerEval), 195
- PrimerDesign, 191
- PrimerEval, 195
- Primers, 137
- Primers (Input), 134
- Primers-class (Input), 134
- rbind.primers.data, 198
- rbind.Primers, 199
- rbind.Templates, 199
- rbind2, Primers, ANY-method (cbind2, Primers, ANY-method), 30
- rbind2, Templates, ANY-method (cbind2, Templates, ANY-method), 31
- read.leaders, 200
- read.secondary.structure.raw, 200
- read.sequences, 201
- read.primers, 9, 137

- read\_primers (Input), 134
- read\_primers.internal, 201
- read\_primers\_csv, 202
- read\_primers\_multiple, 202
- read\_settings, 9, 232
- read\_settings (Input), 134
- read\_templates, 9
- read\_templates (Input), 134
- read\_templates\_csv, 203
- read\_templates\_fasta, 203
- read\_templates\_multiple, 204
- read\_templates\_single, 205
- ref.data (Data), 86
- RefCoverage (Data), 86
- relax.constraints, 206
- relax.opti.constraints, 207
- remove.redundant.cols, 208
- remove.seqs.by.keyword, 208
- rename.constraint.options, 209
- render\_report, 209
- reorder.primers.table, 210
- restriction\_ali, 210
- restriction\_hits, 211
- restriction\_match, 211
- retrieve.leader.region, 212
- rev.comp.sequence, 212
- rev.sequence, 213
- runTutorial, 213
- sanitize\_path, 214
- score.conservation, 214
- score\_conservation (Scoring), 215
- score\_degen (Scoring), 215
- score\_primers, 165
- score\_primers (Scoring), 215
- Scoring, 215
- select.allowed.binding.events, 217
- select.best.ILP, 217
- select.best.opti.result, 218
- select.best.primers.idx, 218
- select.best.primers.set, 219
- select.binding.events, 220
- select.constraints, 220
- select.min.cross.idx, 221
- select.primers.region.by.conservation, 221
- select.primers.by.cvg, 222
- select\_best\_binding, 223
- select\_regions\_by\_conservation (TemplatesFunctions), 239
- selenium.installed, 224
- set.new.constraint.value, 224
- set.new.limits, 225
- Settings, 225
- settings (Data), 86
- shannon.entropy, 234
- solve.ILP, 234
- split\_str\_by\_index, 235
- stats\_plot\_data, 236
- store.filtering.sets, 236
- string.list.format, 237
- string.list.format.total, 237
- string.to.IQR, 238
- subset.ILP, 238
- subset\_primer\_set, 168
- subset\_primer\_set (PrimerEval), 195
- template.data (Data), 86
- template.df (Data), 86
- Templates, 9, 137, 206
- Templates (Input), 134
- Templates-class (Input), 134
- TemplatesFunctions, 239
- Tiller (Data), 86
- tiller.primers.df (Data), 86
- tiller.settings (Data), 86
- tiller.template.df (Data), 86
- ungap\_sequence, 241
- unify.leaders, 242
- update.binding.ranges.by.conservation, 242
- update.binding.regions, 243
- update.constraint.values, 243
- update.cvg.data, 244
- update.individual.binding.region, 244
- update.opti.results, 245
- update\_primer\_binding\_regions, 246
- update\_primer\_cvg, 246
- update\_template\_cvg (TemplatesFunctions), 239
- validate\_primers, 247
- validate\_templates, 247
- view.cvg.primers, 248
- view.dimer.df, 248
- view.input.primers, 249
- view.primers, 249
- view.primers.report, 250
- visualize.all.results, 250
- visualize.filtering.results, 251
- were.constraints.relaxed, 252
- write.out.primers.info, 253
- write\_primers (Output), 156
- write\_settings, 137, 232

`write_settings` (Output), [156](#)  
`write_templates`, [137](#), [206](#)  
`write_templates` (Output), [156](#)  
`xmlToChar`, [253](#)