

RNAlib-2.4.11

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
1.1	A Library for predicting and comparing RNA secondary structures . . . . .	1
1.2	License . . . . .	1
1.3	Contributors . . . . .	2
<b>2</b>	<b>Getting Started</b>	<b>3</b>
2.1	Installation and Configuration . . . . .	3
2.1.1	Installing the ViennaRNA Package . . . . .	3
2.1.1.1	Quick-start . . . . .	3
2.1.1.2	Installation without root privileges . . . . .	3
2.1.1.3	Notes for MacOS X users . . . . .	4
2.1.2	Configuring RNALib features . . . . .	4
2.1.2.1	Streaming SIMD Extension (SSE) support . . . . .	4
2.1.2.2	Scripting Interfaces . . . . .	4
2.1.2.3	Cluster Analysis . . . . .	5
2.1.2.4	Kinfold . . . . .	5
2.1.2.5	RNAforester . . . . .	5
2.1.2.6	Kinwalker . . . . .	5
2.1.2.7	Link Time Optimization (LTO) . . . . .	5
2.1.2.8	OpenMP support . . . . .	6
2.1.2.9	POSIX threads (pthread) support . . . . .	6
2.1.2.10	Stochastic backtracking using Boustrophedon scheme . . . . .	6
2.1.2.11	SVM Z-score filter in RNALfold . . . . .	6
2.1.2.12	GNU Scientific Library . . . . .	6

2.1.2.13	Disable C11/C++11 feature support . . . . .	7
2.1.2.14	Enable warnings for use of deprecated symbols . . . . .	7
2.1.2.15	Single precision partition function . . . . .	7
2.1.2.16	Help . . . . .	7
2.1.3	Linking against RNAlib . . . . .	7
2.1.3.1	Compiler and Linker flags . . . . .	8
2.1.3.2	The pkg-config tool . . . . .	9
2.2	HelloWorld . . . . .	9
2.3	HelloWorld (Perl/Python) . . . . .	11
2.3.1	Perl5 . . . . .	11
2.3.2	Python . . . . .	12
<b>3</b>	<b>Concepts and Algorithms</b>	<b>15</b>
3.1	RNA Structure . . . . .	16
3.1.1	RNA Structures . . . . .	16
3.1.2	Levels of Structure Abstraction . . . . .	16
3.1.2.1	Primary Structure . . . . .	16
3.1.2.2	Secondary Structure . . . . .	16
3.1.2.3	Tertiary Structure . . . . .	16
3.1.2.4	Quarternary Structure . . . . .	16
3.1.2.5	Pseudo-Knots . . . . .	16
3.2	Distance Measures . . . . .	16
3.2.1	Functions for Tree Edit Distances . . . . .	17
3.2.2	Functions for String Alignment . . . . .	18
3.2.3	Functions for Comparison of Base Pair Probabilities . . . . .	18
3.3	Free Energy of Secondary Structures . . . . .	18
3.3.1	Secondary Structure Loop Decomposition . . . . .	19
3.3.1.1	Free Energy Evaluation API . . . . .	20
3.3.2	Free Energy Parameters . . . . .	20
3.3.2.1	Free Energy Parameters Modification API . . . . .	20
3.3.3	Fine-tuning of the Energy Evaluation Model . . . . .	20

3.4	Secondary Structure Folding Grammar	20
3.4.1	Secondary Structure Folding Recurrences	21
3.4.2	Additional Structural Domains	21
3.4.2.1	Structured Domains	22
3.4.2.2	Unstructured Domains	22
3.4.2.3	Domain Extension API	22
3.4.3	Constraints on the Folding Grammar	23
3.4.3.1	Hard Constraints API	23
3.4.3.2	Soft Constraints API	23
3.5	RNA Secondary Structure Landscapes	23
3.5.1	The Neighborhood of a Secondary Structure	24
3.5.2	The Secondary Structure Landscape API	24
3.6	Minimum Free Energy Algorithm(s)	24
3.6.1	Zuker's Algorithm	24
3.6.2	MFE for circular RNAs	24
3.6.3	MFE Algorithm API	24
3.7	Partition Function and Equilibrium Probability Algorithm(s)	24
3.7.1	Equilibrium Ensemble Statistics	24
3.7.2	Partition Function and Equilibrium Probability API	25
3.8	Suboptimals and (other) Representative Structures	25
3.8.1	Suboptimal Secondary Structures	25
3.8.2	Sampling Secondary Structures from the Ensemble	25
3.8.3	Structure Enumeration and Sampling API	25
3.9	RNA-RNA Interaction	26
3.9.1	rip_intro	26
3.9.2	Concatenating RNA sequences	26
3.9.3	RNA-RNA interaction as a Stepwise Process	26
3.9.4	RNA-RNA Interaction API	26
3.10	Locally Stable Secondary Structures	26
3.10.1	local_intro	26

3.10.2	<a href="#">local_mfe</a>	26
3.10.3	<a href="#">local_pf</a>	26
3.10.4	<a href="#">Locally Stable Secondary Structure API</a>	26
3.11	<a href="#">Comparative Structure Prediction</a>	26
3.11.1	<a href="#">Incorporate Evolutionary Information</a>	26
3.11.2	<a href="#">Comparative Structure Prediction API</a>	27
3.12	<a href="#">Classified DP variations</a>	27
3.12.1	<a href="#">The Idea of Classified Dynamic Programming</a>	27
3.12.2	<a href="#">Distance Class Partitioning</a>	27
3.12.3	<a href="#">Density of States (DOS)</a>	27
3.12.4	<a href="#">Classified DP API</a>	27
3.13	<a href="#">RNA Sequence Design</a>	27
3.13.1	<a href="#">Generate Sequences that fold into particular Secondary Structures</a>	27
3.13.2	<a href="#">RNA Sequence Design API</a>	27
3.14	<a href="#">Experimental Structure Probing Data</a>	27
3.14.1	<a href="#">Guide the Structure Prediction using Experimental Data</a>	27
3.14.1.1	<a href="#">SHAPE reactivities</a>	27
3.14.2	<a href="#">Structure Probing Data API</a>	27
3.15	<a href="#">Ligand Binding</a>	28
3.15.1	<a href="#">Small Molecules and Proteins that bind to specific RNA Structures</a>	28
3.15.2	<a href="#">ligand_binding_api</a>	28
3.16	<a href="#">(Tertiary) Structure Motifs</a>	28
3.16.1	<a href="#">Incorporating Higher-Order (Tertiary) Structure Motifs</a>	28
3.16.2	<a href="#">RNA G-Quadruplexes</a>	28
3.16.3	<a href="#">(Tertiary) Structure Motif API</a>	28

<b>4</b>	<b>I/O Formats</b>	<b>29</b>
4.1	RNA Structure Notations	29
4.1.1	Representations of Secondary Structures	29
4.1.2	Dot-Bracket Notation (a.k.a. Dot-Parenthesis Notation)	29
4.1.3	Extended Dot-Bracket Notation	30
4.1.4	Washington University Secondary Structure (WUSS) notation	30
4.1.5	Tree Representations of Secondary Structures	31
4.1.6	Examples for Structure Parsing and Conversion	32
4.1.7	Structure Parsing and Conversion API	32
4.2	File Formats	34
4.2.1	File formats for Multiple Sequence Alignments (MSA)	34
4.2.1.1	ClustalW format	34
4.2.1.2	Stockholm 1.0 format	35
4.2.1.3	FASTA (Pearson) format	35
4.2.1.4	MAF format	36
4.2.2	File formats to manipulate the RNA folding grammar	37
4.2.2.1	Command Files	37
4.3	Plotting	40
4.3.1	Producing secondary structure graphs	40
4.3.2	Producing (colored) dot plots for base pair probabilities	41
4.3.3	Producing (colored) alignments	42
<b>5</b>	<b>Basic Data Structures</b>	<b>43</b>
5.1	Sequence and Structure Data	43
5.2	The 'Fold Compound'	43
5.3	Model Details	43

<b>6</b>	<b>API Features</b>	<b>45</b>
6.1	RNAlib API v3.0 . . . . .	45
6.1.1	Introduction . . . . .	45
6.1.2	What are the major changes? . . . . .	46
6.1.3	How to port your program to the new API . . . . .	46
6.1.4	Some Examples using RNAlib API v3.0 . . . . .	46
6.2	Callback Functions . . . . .	46
6.2.1	The purpose of Callback mechanisms . . . . .	46
6.2.2	List of available Callbacks . . . . .	47
6.3	Scripting Language interface(s) . . . . .	48
6.3.1	Introduction . . . . .	48
6.3.2	Function Renaming . . . . .	48
6.3.2.1	Global Variables . . . . .	48
6.3.3	Object oriented Interface for Data Structures . . . . .	48
6.3.4	Examples . . . . .	49
6.3.5	SWIG generated Wrapper notes . . . . .	49
<b>7</b>	<b>Additional Utilities</b>	<b>57</b>
<b>8</b>	<b>Examples</b>	<b>59</b>
8.1	C Examples . . . . .	59
8.1.1	Hello World Examples . . . . .	59
8.1.2	First Steps with the Fold Compound . . . . .	61
8.1.3	Writing Callback Functions . . . . .	62
8.1.4	Application of Soft Constraints . . . . .	63
8.1.5	Other Examples . . . . .	64
8.1.6	Deprecated Examples . . . . .	65
8.2	Perl5 Examples . . . . .	66
8.3	Python Examples . . . . .	67
<b>9</b>	<b>Changelog</b>	<b>69</b>



<b>10 Deprecated List</b>	<b>71</b>
<b>11 Bug List</b>	<b>83</b>
<b>12 Module Index</b>	<b>85</b>
12.1 The RNALib API . . . . .	85
<b>13 Data Structure Index</b>	<b>87</b>
13.1 Data Structures . . . . .	87
<b>14 File Index</b>	<b>89</b>
14.1 File List . . . . .	89
<b>15 Module Documentation</b>	<b>95</b>
15.1 Free Energy Evaluation . . . . .	95
15.1.1 Detailed Description . . . . .	95
15.1.2 Function Documentation . . . . .	98
15.1.2.1 vrna_eval_structure() . . . . .	98
15.1.2.2 vrna_eval_covar_structure() . . . . .	99
15.1.2.3 vrna_eval_structure_verbose() . . . . .	100
15.1.2.4 vrna_eval_structure_v() . . . . .	100
15.1.2.5 vrna_eval_structure_pt() . . . . .	101
15.1.2.6 vrna_eval_structure_pt_verbose() . . . . .	102
15.1.2.7 vrna_eval_structure_pt_v() . . . . .	102
15.1.2.8 vrna_eval_structure_simple() . . . . .	103
15.1.2.9 vrna_eval_circ_structure() . . . . .	104
15.1.2.10 vrna_eval_gquad_structure() . . . . .	104
15.1.2.11 vrna_eval_circ_gquad_structure() . . . . .	105
15.1.2.12 vrna_eval_structure_simple_verbose() . . . . .	106
15.1.2.13 vrna_eval_structure_simple_v() . . . . .	106
15.1.2.14 vrna_eval_circ_structure_v() . . . . .	107
15.1.2.15 vrna_eval_gquad_structure_v() . . . . .	108
15.1.2.16 vrna_eval_circ_gquad_structure_v() . . . . .	109

15.1.2.17	<code>vrna_eval_consensus_structure_simple()</code>	110
15.1.2.18	<code>vrna_eval_circ_consensus_structure()</code>	110
15.1.2.19	<code>vrna_eval_gquad_consensus_structure()</code>	111
15.1.2.20	<code>vrna_eval_circ_gquad_consensus_structure()</code>	112
15.1.2.21	<code>vrna_eval_consensus_structure_simple_verbose()</code>	113
15.1.2.22	<code>vrna_eval_consensus_structure_simple_v()</code>	114
15.1.2.23	<code>vrna_eval_circ_consensus_structure_v()</code>	114
15.1.2.24	<code>vrna_eval_gquad_consensus_structure_v()</code>	115
15.1.2.25	<code>vrna_eval_circ_gquad_consensus_structure_v()</code>	116
15.1.2.26	<code>vrna_eval_structure_pt_simple()</code>	117
15.1.2.27	<code>vrna_eval_structure_pt_simple_verbose()</code>	118
15.1.2.28	<code>vrna_eval_structure_pt_simple_v()</code>	118
15.1.2.29	<code>vrna_eval_consensus_structure_pt_simple()</code>	119
15.2	Energy Evaluation for Individual Loops	121
15.2.1	Detailed Description	121
15.2.2	Function Documentation	122
15.2.2.1	<code>vrna_eval_loop_pt()</code>	122
15.2.2.2	<code>vrna_eval_loop_pt_v()</code>	122
15.3	Energy Evaluation for Atomic Moves	124
15.3.1	Detailed Description	124
15.3.2	Function Documentation	124
15.3.2.1	<code>vrna_eval_move()</code>	124
15.3.2.2	<code>vrna_eval_move_pt()</code>	125
15.4	Deprecated Interface for Free Energy Evaluation	126
15.4.1	Detailed Description	126
15.4.2	Function Documentation	127
15.4.2.1	<code>energy_of_structure()</code>	127
15.4.2.2	<code>energy_of_struct_par()</code>	128
15.4.2.3	<code>energy_of_circ_structure()</code>	128
15.4.2.4	<code>energy_of_circ_struct_par()</code>	129

15.4.2.5	<a href="#">energy_of_structure_pt()</a>	129
15.4.2.6	<a href="#">energy_of_struct_pt_par()</a>	130
15.4.2.7	<a href="#">energy_of_move()</a>	131
15.4.2.8	<a href="#">energy_of_move_pt()</a>	132
15.4.2.9	<a href="#">loop_energy()</a>	132
15.4.2.10	<a href="#">energy_of_struct()</a>	133
15.4.2.11	<a href="#">energy_of_struct_pt()</a>	134
15.4.2.12	<a href="#">energy_of_circ_struct()</a>	134
15.4.2.13	<a href="#">E_Stem()</a>	135
15.4.2.14	<a href="#">exp_E_ExtLoop()</a>	136
15.4.2.15	<a href="#">exp_E_Stem()</a>	137
15.4.2.16	<a href="#">E_IntLoop()</a>	137
15.4.2.17	<a href="#">exp_E_IntLoop()</a>	139
15.5	<a href="#">The RNA Folding Grammar</a>	141
15.5.1	<a href="#">Detailed Description</a>	141
15.5.2	<a href="#">Data Structure Documentation</a>	141
15.5.2.1	<a href="#">struct vrna_gr_aux_s</a>	141
15.6	<a href="#">Fine-tuning of the Implemented Models</a>	142
15.6.1	<a href="#">Detailed Description</a>	142
15.6.2	<a href="#">Data Structure Documentation</a>	146
15.6.2.1	<a href="#">struct vrna_md_s</a>	146
15.6.3	<a href="#">Macro Definition Documentation</a>	149
15.6.3.1	<a href="#">VRNA_MODEL_DEFAULT_TEMPERATURE</a>	149
15.6.3.2	<a href="#">VRNA_MODEL_DEFAULT_PF_SCALE</a>	150
15.6.3.3	<a href="#">VRNA_MODEL_DEFAULT_BETA_SCALE</a>	150
15.6.3.4	<a href="#">VRNA_MODEL_DEFAULT_DANGLES</a>	150
15.6.3.5	<a href="#">VRNA_MODEL_DEFAULT_SPECIAL_HP</a>	150
15.6.3.6	<a href="#">VRNA_MODEL_DEFAULT_NO_LP</a>	151
15.6.3.7	<a href="#">VRNA_MODEL_DEFAULT_NO_GU</a>	151
15.6.3.8	<a href="#">VRNA_MODEL_DEFAULT_NO_GU_CLOSURE</a>	151

15.6.3.9	VRNA_MODEL_DEFAULT_CIRC . . . . .	151
15.6.3.10	VRNA_MODEL_DEFAULT_GQUAD . . . . .	152
15.6.3.11	VRNA_MODEL_DEFAULT_UNIQ_ML . . . . .	152
15.6.3.12	VRNA_MODEL_DEFAULT_ENERGY_SET . . . . .	152
15.6.3.13	VRNA_MODEL_DEFAULT_BACKTRACK . . . . .	152
15.6.3.14	VRNA_MODEL_DEFAULT_BACKTRACK_TYPE . . . . .	153
15.6.3.15	VRNA_MODEL_DEFAULT_COMPUTE_BPP . . . . .	153
15.6.3.16	VRNA_MODEL_DEFAULT_MAX_BP_SPAN . . . . .	153
15.6.3.17	VRNA_MODEL_DEFAULT_WINDOW_SIZE . . . . .	153
15.6.3.18	VRNA_MODEL_DEFAULT_LOG_ML . . . . .	154
15.6.3.19	VRNA_MODEL_DEFAULT_ALI_OLD_EN . . . . .	154
15.6.3.20	VRNA_MODEL_DEFAULT_ALI_RIBO . . . . .	154
15.6.3.21	VRNA_MODEL_DEFAULT_ALI_CV_FACT . . . . .	154
15.6.3.22	VRNA_MODEL_DEFAULT_ALI_NC_FACT . . . . .	155
15.6.4	Function Documentation . . . . .	155
15.6.4.1	vrna_md_set_default() . . . . .	155
15.6.4.2	vrna_md_update() . . . . .	155
15.6.4.3	vrna_md_copy() . . . . .	156
15.6.4.4	vrna_md_option_string() . . . . .	156
15.6.4.5	vrna_md_defaults_reset() . . . . .	156
15.6.4.6	vrna_md_defaults_temperature() . . . . .	157
15.6.4.7	vrna_md_defaults_temperature_get() . . . . .	157
15.6.4.8	vrna_md_defaults_betaScale() . . . . .	158
15.6.4.9	vrna_md_defaults_betaScale_get() . . . . .	158
15.6.4.10	vrna_md_defaults_dangles() . . . . .	159
15.6.4.11	vrna_md_defaults_dangles_get() . . . . .	159
15.6.4.12	vrna_md_defaults_special_hp() . . . . .	159
15.6.4.13	vrna_md_defaults_special_hp_get() . . . . .	160
15.6.4.14	vrna_md_defaults_noLP() . . . . .	160
15.6.4.15	vrna_md_defaults_noLP_get() . . . . .	160

15.6.4.16 vrna_md_defaults_noGU()	161
15.6.4.17 vrna_md_defaults_noGU_get()	161
15.6.4.18 vrna_md_defaults_noGUclosure()	162
15.6.4.19 vrna_md_defaults_noGUclosure_get()	162
15.6.4.20 vrna_md_defaults_logML()	162
15.6.4.21 vrna_md_defaults_logML_get()	163
15.6.4.22 vrna_md_defaults_circ()	163
15.6.4.23 vrna_md_defaults_circ_get()	163
15.6.4.24 vrna_md_defaults_gquad()	164
15.6.4.25 vrna_md_defaults_gquad_get()	164
15.6.4.26 vrna_md_defaults_uniq_ML()	165
15.6.4.27 vrna_md_defaults_uniq_ML_get()	165
15.6.4.28 vrna_md_defaults_energy_set()	165
15.6.4.29 vrna_md_defaults_energy_set_get()	166
15.6.4.30 vrna_md_defaults_backtrack()	166
15.6.4.31 vrna_md_defaults_backtrack_get()	166
15.6.4.32 vrna_md_defaults_backtrack_type()	167
15.6.4.33 vrna_md_defaults_backtrack_type_get()	167
15.6.4.34 vrna_md_defaults_compute_bpp()	168
15.6.4.35 vrna_md_defaults_compute_bpp_get()	168
15.6.4.36 vrna_md_defaults_max_bp_span()	168
15.6.4.37 vrna_md_defaults_max_bp_span_get()	169
15.6.4.38 vrna_md_defaults_min_loop_size()	169
15.6.4.39 vrna_md_defaults_min_loop_size_get()	169
15.6.4.40 vrna_md_defaults_window_size()	170
15.6.4.41 vrna_md_defaults_window_size_get()	170
15.6.4.42 vrna_md_defaults_oldAliEn()	171
15.6.4.43 vrna_md_defaults_oldAliEn_get()	171
15.6.4.44 vrna_md_defaults_ribo()	171
15.6.4.45 vrna_md_defaults_ribo_get()	172

15.6.4.46 vrna_md_defaults_cv_fact()	172
15.6.4.47 vrna_md_defaults_cv_fact_get()	172
15.6.4.48 vrna_md_defaults_nc_fact()	173
15.6.4.49 vrna_md_defaults_nc_fact_get()	173
15.6.4.50 vrna_md_defaults_sfact()	174
15.6.4.51 vrna_md_defaults_sfact_get()	174
15.6.4.52 set_model_details()	174
15.6.5 Variable Documentation	175
15.6.5.1 temperature	175
15.6.5.2 pf_scale	175
15.6.5.3 dangles	176
15.6.5.4 tetra_loop	176
15.6.5.5 noLonelyPairs	176
15.6.5.6 energy_set	176
15.6.5.7 do_backtrack	177
15.6.5.8 backtrack_type	177
15.6.5.9 nonstandards	177
15.6.5.10 max_bp_span	177
15.7 Energy Parameters	178
15.7.1 Detailed Description	178
15.7.2 Data Structure Documentation	179
15.7.2.1 struct vrna_param_s	179
15.7.2.2 struct vrna_exp_param_s	180
15.7.3 Typedef Documentation	181
15.7.3.1 paramT	181
15.7.3.2 pf_paramT	181
15.7.4 Function Documentation	181
15.7.4.1 vrna_params()	181
15.7.4.2 vrna_params_copy()	182
15.7.4.3 vrna_exp_params()	182

15.7.4.4	<code>vrna_exp_params_comparative()</code>	183
15.7.4.5	<code>vrna_exp_params_copy()</code>	183
15.7.4.6	<code>vrna_params_subst()</code>	184
15.7.4.7	<code>vrna_exp_params_subst()</code>	184
15.7.4.8	<code>vrna_exp_params_rescale()</code>	185
15.7.4.9	<code>vrna_params_reset()</code>	186
15.7.4.10	<code>vrna_exp_params_reset()</code>	187
15.7.4.11	<code>get_scaled_pf_parameters()</code>	187
15.7.4.12	<code>get_boltzmann_factors()</code>	188
15.7.4.13	<code>get_boltzmann_factor_copy()</code>	188
15.7.4.14	<code>get_scaled_alipf_parameters()</code>	189
15.7.4.15	<code>get_boltzmann_factors_ali()</code>	189
15.7.4.16	<code>scale_parameters()</code>	190
15.7.4.17	<code>get_scaled_parameters()</code>	190
15.8	Extending the Folding Grammar with Additional Domains	192
15.8.1	Detailed Description	192
15.9	Unstructured Domains	193
15.9.1	Detailed Description	193
15.9.2	Data Structure Documentation	195
15.9.2.1	<code>struct vrna_unstructured_domain_s</code>	195
15.9.3	Typedef Documentation	196
15.9.3.1	<code>vrna_callback_ud_energy</code>	196
15.9.3.2	<code>vrna_callback_ud_exp_energy</code>	196
15.9.3.3	<code>vrna_callback_ud_production</code>	197
15.9.3.4	<code>vrna_callback_ud_exp_production</code>	197
15.9.3.5	<code>vrna_callback_ud_probs_add</code>	198
15.9.3.6	<code>vrna_callback_ud_probs_get</code>	198
15.9.4	Function Documentation	198
15.9.4.1	<code>vrna_ud_motifs_centroid()</code>	198
15.9.4.2	<code>vrna_ud_motifs_MEA()</code>	199

15.9.4.3	<a href="#">vrna_ud_motifs_MFE()</a>	199
15.9.4.4	<a href="#">vrna_ud_add_motif()</a>	200
15.9.4.5	<a href="#">vrna_ud_remove()</a>	201
15.9.4.6	<a href="#">vrna_ud_set_data()</a>	201
15.9.4.7	<a href="#">vrna_ud_set_prod_rule_cb()</a>	202
15.9.4.8	<a href="#">vrna_ud_set_exp_prod_rule_cb()</a>	203
15.10	<a href="#">Structured Domains</a>	205
15.10.1	<a href="#">Detailed Description</a>	205
15.11	<a href="#">Constraining the RNA Folding Grammar</a>	206
15.11.1	<a href="#">Detailed Description</a>	206
15.11.2	<a href="#">Macro Definition Documentation</a>	209
15.11.2.1	<a href="#">VRNA_CONSTRAINT_FILE</a>	209
15.11.2.2	<a href="#">VRNA_CONSTRAINT_SOFT_MFE</a>	209
15.11.2.3	<a href="#">VRNA_CONSTRAINT_SOFT_PF</a>	210
15.11.2.4	<a href="#">VRNA_DECOMP_PAIR_HP</a>	210
15.11.2.5	<a href="#">VRNA_DECOMP_PAIR_IL</a>	211
15.11.2.6	<a href="#">VRNA_DECOMP_PAIR_ML</a>	211
15.11.2.7	<a href="#">VRNA_DECOMP_ML_ML_ML</a>	212
15.11.2.8	<a href="#">VRNA_DECOMP_ML_STEM</a>	213
15.11.2.9	<a href="#">VRNA_DECOMP_ML_ML</a>	213
15.11.2.10	<a href="#">VRNA_DECOMP_ML_UP</a>	214
15.11.2.11	<a href="#">VRNA_DECOMP_ML_ML_STEM</a>	214
15.11.2.12	<a href="#">VRNA_DECOMP_ML_COAXIAL</a>	215
15.11.2.13	<a href="#">VRNA_DECOMP_ML_COAXIAL_ENC</a>	215
15.11.2.14	<a href="#">VRNA_DECOMP_EXT_EXT</a>	216
15.11.2.15	<a href="#">VRNA_DECOMP_EXT_UP</a>	216
15.11.2.16	<a href="#">VRNA_DECOMP_EXT_STEM</a>	217
15.11.2.17	<a href="#">VRNA_DECOMP_EXT_EXT_EXT</a>	217
15.11.2.18	<a href="#">VRNA_DECOMP_EXT_STEM_EXT</a>	218
15.11.2.19	<a href="#">VRNA_DECOMP_EXT_EXT_STEM</a>	218



15.11.2.20VRNA_DECOMP_EXT_EXT_STEM1 . . . . .	219
15.11.3 Function Documentation . . . . .	219
15.11.3.1 vrna_constraints_add() . . . . .	219
15.11.3.2 vrna_message_constraint_options() . . . . .	220
15.11.3.3 vrna_message_constraint_options_all() . . . . .	221
15.12Hard Constraints . . . . .	222
15.12.1 Detailed Description . . . . .	222
15.12.2 Data Structure Documentation . . . . .	224
15.12.2.1 struct vrna_hc_s . . . . .	224
15.12.2.2 struct vrna_hc_up_s . . . . .	225
15.12.3 Macro Definition Documentation . . . . .	225
15.12.3.1 VRNA_CONSTRAINT_DB . . . . .	225
15.12.3.2 VRNA_CONSTRAINT_DB_ENFORCE_BP . . . . .	226
15.12.3.3 VRNA_CONSTRAINT_DB_PIPE . . . . .	226
15.12.3.4 VRNA_CONSTRAINT_DB_DOT . . . . .	226
15.12.3.5 VRNA_CONSTRAINT_DB_X . . . . .	227
15.12.3.6 VRNA_CONSTRAINT_DB_RND_BRACK . . . . .	227
15.12.3.7 VRNA_CONSTRAINT_DB_INTRAMOL . . . . .	227
15.12.3.8 VRNA_CONSTRAINT_DB_INTERMOL . . . . .	228
15.12.3.9 VRNA_CONSTRAINT_DB_GQUAD . . . . .	228
15.12.3.10VRNA_CONSTRAINT_DB_WUSS . . . . .	228
15.12.3.11VRNA_CONSTRAINT_DB_DEFAULT . . . . .	229
15.12.4 Typedef Documentation . . . . .	229
15.12.4.1 vrna_callback_hc_evaluate . . . . .	229
15.12.5 Function Documentation . . . . .	230
15.12.5.1 vrna_hc_init() . . . . .	230
15.12.5.2 vrna_hc_add_up() . . . . .	230
15.12.5.3 vrna_hc_add_up_batch() . . . . .	231
15.12.5.4 vrna_hc_add_bp() . . . . .	231
15.12.5.5 vrna_hc_add_bp_nonspecific() . . . . .	232

15.12.5.6 vrna_hc_free()	232
15.12.5.7 vrna_hc_add_from_db()	233
15.13 Soft Constraints	234
15.13.1 Detailed Description	234
15.13.2 Data Structure Documentation	235
15.13.2.1 struct vrna_sc_s	235
15.13.3 Typedef Documentation	236
15.13.3.1 vrna_callback_sc_energy	237
15.13.3.2 vrna_callback_sc_exp_energy	238
15.13.3.3 vrna_callback_sc_backtrack	239
15.13.4 Function Documentation	239
15.13.4.1 vrna_sc_init()	239
15.13.4.2 vrna_sc_set_bp()	240
15.13.4.3 vrna_sc_add_bp()	241
15.13.4.4 vrna_sc_set_up()	241
15.13.4.5 vrna_sc_add_up()	242
15.13.4.6 vrna_sc_remove()	243
15.13.4.7 vrna_sc_free()	243
15.13.4.8 vrna_sc_add_data()	243
15.13.4.9 vrna_sc_add_f()	244
15.13.4.10 vrna_sc_add_bt()	245
15.13.4.11 vrna_sc_add_exp_f()	245
15.14 The RNA Secondary Structure Landscape	247
15.14.1 Detailed Description	247
15.15 Minimum Free Energy (MFE) Algorithms	248
15.15.1 Detailed Description	248
15.16 Partition Function and Equilibrium Properties	249
15.16.1 Detailed Description	249
15.16.2 Function Documentation	250
15.16.2.1 vrna_pf_float_precision()	250

15.17 Global MFE Prediction	251
15.17.1 Detailed Description	251
15.17.2 Function Documentation	252
15.17.2.1 vrna_mfe()	252
15.17.2.2 vrna_mfe_dimer()	252
15.17.2.3 vrna_fold()	253
15.17.2.4 vrna_circfold()	254
15.17.2.5 vrna_alifold()	254
15.17.2.6 vrna_circalifold()	255
15.17.2.7 vrna_cofold()	256
15.18 Local (sliding window) MFE Prediction	258
15.18.1 Detailed Description	258
15.18.2 Typedef Documentation	259
15.18.2.1 vrna_mfe_window_callback	259
15.18.3 Function Documentation	260
15.18.3.1 vrna_mfe_window()	260
15.18.3.2 vrna_mfe_window_zscore()	260
15.18.3.3 vrna_Lfold()	261
15.18.3.4 vrna_Lfoldz()	262
15.19 Backtracking MFE structures	263
15.19.1 Detailed Description	263
15.19.2 Function Documentation	263
15.19.2.1 vrna_BT_hp_loop()	263
15.19.2.2 vrna_BT_mb_loop()	264
15.20 Global Partition Function and Equilibrium Probabilities	265
15.20.1 Detailed Description	265
15.20.2 Data Structure Documentation	266
15.20.2.1 struct vrna_dimer_pf_s	266
15.20.3 Function Documentation	267
15.20.3.1 vrna_mean_bp_distance_pr()	267

15.20.3.2 vrna_mean_bp_distance()	267
15.20.3.3 vrna_ensemble_defect()	268
15.20.3.4 vrna_stack_prob()	268
15.20.3.5 vrna_pf_dimer_probs()	269
15.20.3.6 vrna_pr_structure()	269
15.20.3.7 vrna_pf()	270
15.20.3.8 vrna_pf_dimer()	271
15.20.3.9 vrna_pf_fold()	272
15.20.3.10 vrna_pf_circfold()	272
15.20.3.11 vrna_pf_alifold()	273
15.20.3.12 vrna_pf_circalifold()	274
15.20.3.13 vrna_plist_from_probs()	275
15.20.3.14 vrna_pf_co_fold()	275
15.21 Local (sliding window) Partition Function and Equilibrium Probabilities	277
15.21.1 Detailed Description	277
15.21.2 Macro Definition Documentation	278
15.21.2.1 VRNA_PROBS_WINDOW_BPP	278
15.21.2.2 VRNA_PROBS_WINDOW_UP	279
15.21.2.3 VRNA_PROBS_WINDOW_STACKP	279
15.21.2.4 VRNA_PROBS_WINDOW_UP_SPLIT	279
15.21.2.5 VRNA_PROBS_WINDOW_PF	280
15.21.3 Typedef Documentation	280
15.21.3.1 vrna_probs_window_callback	280
15.21.4 Function Documentation	281
15.21.4.1 vrna_probs_window()	281
15.21.4.2 vrna_pfl_fold()	282
15.21.4.3 vrna_pfl_fold_cb()	283
15.21.4.4 vrna_pfl_fold_up()	283
15.21.4.5 vrna_pfl_fold_up_cb()	284
15.22 Suboptimals and Representative Structures	287

15.22.1 Detailed Description . . . . .	287
15.23 Suboptimal Structures sensu Stiegler et al. 1984 / Zuker et al. 1989 . . . . .	288
15.23.1 Detailed Description . . . . .	288
15.23.2 Function Documentation . . . . .	288
15.23.2.1 vrna_subopt_zuker() . . . . .	288
15.23.2.2 zukersubopt() . . . . .	289
15.23.2.3 zukersubopt_par() . . . . .	289
15.24 Suboptimal Structures within an Energy Band around the MFE . . . . .	290
15.24.1 Detailed Description . . . . .	290
15.24.2 Typedef Documentation . . . . .	290
15.24.2.1 vrna_subopt_callback . . . . .	290
15.24.3 Function Documentation . . . . .	291
15.24.3.1 vrna_subopt() . . . . .	291
15.24.3.2 vrna_subopt_cb() . . . . .	292
15.24.3.3 subopt() . . . . .	293
15.24.3.4 subopt_circ() . . . . .	293
15.25 Random Structure Samples from the Ensemble . . . . .	295
15.25.1 Detailed Description . . . . .	295
15.25.2 Function Documentation . . . . .	295
15.25.2.1 vrna_pbacktrack5() . . . . .	295
15.25.2.2 vrna_pbacktrack_nr() . . . . .	296
15.25.2.3 vrna_pbacktrack() . . . . .	297
15.25.2.4 pbacktrack() . . . . .	297
15.25.2.5 pbacktrack_circ() . . . . .	298
15.25.3 Variable Documentation . . . . .	298
15.25.3.1 st_back . . . . .	298
15.26 Compute the Structure with Maximum Expected Accuracy (MEA) . . . . .	300
15.26.1 Detailed Description . . . . .	300
15.26.2 Function Documentation . . . . .	300
15.26.2.1 MEA() . . . . .	300

15.27 Compute the Centroid Structure . . . . .	301
15.27.1 Detailed Description . . . . .	301
15.27.2 Function Documentation . . . . .	301
15.27.2.1 vrna_centroid() . . . . .	301
15.27.2.2 vrna_centroid_from_plist() . . . . .	302
15.27.2.3 vrna_centroid_from_probs() . . . . .	302
15.28 RNA-RNA Interaction . . . . .	304
15.28.1 Detailed Description . . . . .	304
15.29 Classified Dynamic Programming Variants . . . . .	305
15.29.1 Detailed Description . . . . .	305
15.30 Distance Based Partitioning of the Secondary Structure Space . . . . .	306
15.30.1 Detailed Description . . . . .	306
15.31 Computing MFE representatives of a Distance Based Partitioning . . . . .	307
15.31.1 Detailed Description . . . . .	307
15.31.2 Data Structure Documentation . . . . .	308
15.31.2.1 struct vrna_sol_TwoD_t . . . . .	308
15.31.2.2 struct TwoDfold_vars . . . . .	308
15.31.3 Typedef Documentation . . . . .	309
15.31.3.1 vrna_sol_TwoD_t . . . . .	309
15.31.3.2 TwoDfold_vars . . . . .	310
15.31.4 Function Documentation . . . . .	310
15.31.4.1 vrna_mfe_TwoD() . . . . .	310
15.31.4.2 vrna_backtrack5_TwoD() . . . . .	311
15.31.4.3 get_TwoDfold_variables() . . . . .	311
15.31.4.4 destroy_TwoDfold_variables() . . . . .	312
15.31.4.5 TwoDfoldList() . . . . .	312
15.31.4.6 TwoDfold_backtrack_f5() . . . . .	313
15.32 Computing Partition Functions of a Distance Based Partitioning . . . . .	315
15.32.1 Detailed Description . . . . .	315
15.32.2 Data Structure Documentation . . . . .	315

15.32.2.1 struct vrna_sol_TwoD_pf_t . . . . .	315
15.32.3 Typedef Documentation . . . . .	316
15.32.3.1 vrna_sol_TwoD_pf_t . . . . .	316
15.32.4 Function Documentation . . . . .	316
15.32.4.1 vrna_pf_TwoD() . . . . .	316
15.33 Stochastic Backtracking of Structures from Distance Based Partitioning . . . . .	318
15.33.1 Detailed Description . . . . .	318
15.33.2 Function Documentation . . . . .	318
15.33.2.1 vrna_pbacktrack_TwoD() . . . . .	318
15.33.2.2 vrna_pbacktrack5_TwoD() . . . . .	319
15.34 Compute the Density of States . . . . .	321
15.34.1 Detailed Description . . . . .	321
15.34.2 Variable Documentation . . . . .	321
15.34.2.1 density_of_states . . . . .	321
15.35 Inverse Folding (Design) . . . . .	322
15.35.1 Detailed Description . . . . .	322
15.35.2 Function Documentation . . . . .	322
15.35.2.1 inverse_fold() . . . . .	322
15.35.2.2 inverse_pf_fold() . . . . .	323
15.35.3 Variable Documentation . . . . .	323
15.35.3.1 final_cost . . . . .	323
15.35.3.2 give_up . . . . .	324
15.35.3.3 inv_verbose . . . . .	324
15.36 Neighborhood Relation and Move Sets for Secondary Structures . . . . .	325
15.36.1 Detailed Description . . . . .	325
15.36.2 Data Structure Documentation . . . . .	327
15.36.2.1 struct vrna_move_s . . . . .	327
15.36.3 Macro Definition Documentation . . . . .	328
15.36.3.1 VRNA_MOVESET_INSERTION . . . . .	328
15.36.3.2 VRNA_MOVESET_DELETION . . . . .	328

15.36.3.3 VRNA_MOVESET_SHIFT . . . . .	329
15.36.3.4 VRNA_MOVESET_NO_LP . . . . .	329
15.36.3.5 VRNA_MOVESET_DEFAULT . . . . .	329
15.36.4 Function Documentation . . . . .	329
15.36.4.1 vrna_move_list_free() . . . . .	329
15.36.4.2 vrna_move_apply() . . . . .	329
15.36.4.3 vrna_loopidx_update() . . . . .	330
15.36.4.4 vrna_neighbors() . . . . .	330
15.36.4.5 vrna_neighbors_successive() . . . . .	331
15.37 Refolding Paths of Secondary Structures . . . . .	333
15.37.1 Detailed Description . . . . .	333
15.37.2 Macro Definition Documentation . . . . .	333
15.37.2.1 VRNA_PATH_STEEPEST_DESCENT . . . . .	334
15.37.2.2 VRNA_PATH_RANDOM . . . . .	334
15.37.2.3 VRNA_PATH_NO_TRANSITION_OUTPUT . . . . .	334
15.37.2.4 VRNA_PATH_DEFAULT . . . . .	334
15.37.3 Function Documentation . . . . .	335
15.37.3.1 vrna_path() . . . . .	335
15.37.3.2 vrna_path_gradient() . . . . .	336
15.37.3.3 vrna_path_random() . . . . .	337
15.38 Experimental Structure Probing Data . . . . .	338
15.38.1 Detailed Description . . . . .	338
15.39 SHAPE Reactivity Data . . . . .	339
15.39.1 Detailed Description . . . . .	339
15.39.2 Function Documentation . . . . .	339
15.39.2.1 vrna_sc_add_SHAPE_deigan() . . . . .	340
15.39.2.2 vrna_sc_add_SHAPE_deigan_ali() . . . . .	340
15.39.2.3 vrna_sc_add_SHAPE_zarringhalam() . . . . .	341
15.39.2.4 vrna_sc_SHAPE_to_pr() . . . . .	342
15.40 Generate Soft Constraints from Data . . . . .	343



15.40.1 Detailed Description . . . . .	343
15.40.2 Macro Definition Documentation . . . . .	344
15.40.2.1 VRNA_OBJECTIVE_FUNCTION_QUADRATIC . . . . .	344
15.40.2.2 VRNA_OBJECTIVE_FUNCTION_ABSOLUTE . . . . .	344
15.40.2.3 VRNA_MINIMIZER_CONJUGATE_FR . . . . .	344
15.40.2.4 VRNA_MINIMIZER_CONJUGATE_PR . . . . .	344
15.40.2.5 VRNA_MINIMIZER_VECTOR_BFGS . . . . .	345
15.40.2.6 VRNA_MINIMIZER_VECTOR_BFGS2 . . . . .	345
15.40.2.7 VRNA_MINIMIZER_STEEPEST_DESCENT . . . . .	345
15.40.3 Typedef Documentation . . . . .	345
15.40.3.1 progress_callback . . . . .	345
15.40.4 Function Documentation . . . . .	346
15.40.4.1 vrna_sc_minimize_pertubation() . . . . .	346
15.41 Ligands Binding to RNA Structures . . . . .	348
15.41.1 Detailed Description . . . . .	348
15.42 Ligands Binding to Unstructured Domains . . . . .	349
15.43 Incorporating Ligands Binding to Specific Sequence/Structure Motifs using Soft Constraints . . . . .	350
15.43.1 Detailed Description . . . . .	350
15.43.2 Function Documentation . . . . .	351
15.43.2.1 vrna_sc_add_hi_motif() . . . . .	351
15.44 Complex Structured Modules . . . . .	352
15.44.1 Detailed Description . . . . .	352
15.45 G-Quadruplexes . . . . .	353
15.45.1 Detailed Description . . . . .	353
15.45.2 Function Documentation . . . . .	353
15.45.2.1 get_gquad_matrix() . . . . .	353
15.45.2.2 parse_gquad() . . . . .	354
15.45.2.3 backtrack_GQuad_IntLoop() . . . . .	354
15.45.2.4 backtrack_GQuad_IntLoop_L() . . . . .	355
15.46 Utilities . . . . .	356

15.46.1 Detailed Description . . . . .	356
15.46.2 Macro Definition Documentation . . . . .	358
15.46.2.1 VRNA_INPUT_FASTA_HEADER . . . . .	358
15.46.2.2 VRNA_INPUT_CONSTRAINT . . . . .	358
15.46.3 Function Documentation . . . . .	359
15.46.3.1 vrna_alloc() . . . . .	359
15.46.3.2 vrna_realloc() . . . . .	359
15.46.3.3 vrna_urn() . . . . .	359
15.46.3.4 vrna_int_urn() . . . . .	360
15.46.3.5 vrna_time_stamp() . . . . .	360
15.46.3.6 get_input_line() . . . . .	361
15.46.3.7 vrna_idx_row_wise() . . . . .	361
15.46.3.8 vrna_idx_col_wise() . . . . .	362
15.46.4 Variable Documentation . . . . .	362
15.46.4.1 xsubi . . . . .	363
15.47 Exterior Loops . . . . .	364
15.47.1 Detailed Description . . . . .	364
15.47.2 Typedef Documentation . . . . .	364
15.47.2.1 vrna_mx_pf_aux_el_t . . . . .	365
15.47.3 Function Documentation . . . . .	365
15.47.3.1 vrna_E_ext_stem() . . . . .	365
15.47.3.2 vrna_E_ext_loop() . . . . .	366
15.47.3.3 vrna_exp_E_ext_stem() . . . . .	366
15.48 Hairpin Loops . . . . .	368
15.48.1 Detailed Description . . . . .	368
15.48.2 Function Documentation . . . . .	368
15.48.2.1 vrna_E_hp_loop() . . . . .	369
15.48.2.2 vrna_E_ext_hp_loop() . . . . .	369
15.48.2.3 vrna_eval_hp_loop() . . . . .	370
15.48.2.4 E_Hairpin() . . . . .	370

15.48.2.5 exp_E_Hairpin()	371
15.48.2.6 vrna_exp_E_hp_loop()	372
15.49 Internal Loops	373
15.49.1 Detailed Description	373
15.49.2 Function Documentation	373
15.49.2.1 vrna_eval_int_loop()	373
15.50 Multibranch Loops	374
15.50.1 Detailed Description	374
15.50.2 Typedef Documentation	374
15.50.2.1 vrna_mx_pf_aux_ml_t	375
15.50.3 Function Documentation	375
15.50.3.1 vrna_E_mb_loop_stack()	375
15.51 Deprecated Interface for Global MFE Prediction	376
15.51.1 Detailed Description	376
15.51.2 Function Documentation	377
15.51.2.1 alifold()	377
15.51.2.2 cofold()	378
15.51.2.3 cofold_par()	378
15.51.2.4 free_co_arrays()	379
15.51.2.5 update_cofold_params()	379
15.51.2.6 update_cofold_params_par()	379
15.51.2.7 export_cofold_arrays_gq()	380
15.51.2.8 export_cofold_arrays()	380
15.51.2.9 get_monomere_mfes()	381
15.51.2.10 initialize_cofold()	382
15.51.2.11 fold_par()	382
15.51.2.12 fold()	383
15.51.2.13 circfold()	384
15.51.2.14 free_arrays()	384
15.51.2.15 update_fold_params()	385

15.51.2.16	<code>update_fold_params_par()</code>	385
15.51.2.17	<code>export_fold_arrays()</code>	385
15.51.2.18	<code>export_fold_arrays_par()</code>	386
15.51.2.19	<code>export_circfold_arrays()</code>	386
15.51.2.20	<code>export_circfold_arrays_par()</code>	386
15.51.2.21	<code>LoopEnergy()</code>	387
15.51.2.22	<code>HairpinE()</code>	387
15.51.2.23	<code>initialize_fold()</code>	387
15.51.2.24	<code>circalifold()</code>	388
15.51.2.25	<code>free_alifold_arrays()</code>	388
15.52	Deprecated Interface for Local (Sliding Window) MFE Prediction	389
15.52.1	Detailed Description	389
15.52.2	Function Documentation	389
15.52.2.1	<code>Lfold()</code>	389
15.52.2.2	<code>Lfoldz()</code>	389
15.53	Deprecated Interface for Global Partition Function Computation	390
15.53.1	Detailed Description	390
15.53.2	Function Documentation	391
15.53.2.1	<code>alipf_fold_par()</code>	391
15.53.2.2	<code>pf_fold_par()</code>	392
15.53.2.3	<code>pf_fold()</code>	393
15.53.2.4	<code>pf_circ_fold()</code>	394
15.53.2.5	<code>free_pf_arrays()</code>	395
15.53.2.6	<code>update_pf_params()</code>	395
15.53.2.7	<code>update_pf_params_par()</code>	396
15.53.2.8	<code>export_bppm()</code>	396
15.53.2.9	<code>get_pf_arrays()</code>	396
15.53.2.10	<code>mean_bp_distance()</code>	397
15.53.2.11	<code>mean_bp_distance_pr()</code>	397
15.53.2.12	<code>stackProb()</code>	398

15.53.2.13	nit_pf_fold()	398
15.53.2.14	co_pf_fold()	399
15.53.2.15	co_pf_fold_par()	399
15.53.2.16	compute_probabilities()	400
15.53.2.17	nit_co_pf_fold()	401
15.53.2.18	export_co_bppm()	401
15.53.2.19	free_co_pf_arrays()	401
15.53.2.20	update_co_pf_params()	401
15.53.2.21	update_co_pf_params_par()	402
15.53.2.22	assign_plist_from_db()	402
15.53.2.23	assign_plist_from_pr()	403
15.53.2.24	alipf_fold()	403
15.53.2.25	alipf_circ_fold()	404
15.53.2.26	export_alipf_bppm()	404
15.53.2.27	free_alipf_arrays()	405
15.53.2.28	alipf_backtrack()	405
15.53.2.29	get_alipf_arrays()	406
15.54	Deprecated Interface for Local (Sliding Window) Partition Function Computation	408
15.54.1	Detailed Description	408
15.54.2	Function Documentation	408
15.54.2.1	update_pf_paramsLP()	408
15.54.2.2	pfl_fold()	408
15.54.2.3	putoutpU_prob()	409
15.54.2.4	putoutpU_prob_bin()	410
15.55	Partition Function for Two Hybridized Sequences	411
15.55.1	Detailed Description	411
15.55.2	Function Documentation	412
15.55.2.1	vrna_pf_co_fold()	412
15.55.2.2	vrna_pf_dimer_concentrations()	413
15.56	Partition Function for two Hybridized Sequences as a Stepwise Process	414

15.56.1 Detailed Description . . . . .	414
15.56.2 Function Documentation . . . . .	414
15.56.2.1 pf_unstru() . . . . .	414
15.56.2.2 pf_interact() . . . . .	415
15.57 Reading/Writing Energy Parameter Sets from/to File . . . . .	417
15.57.1 Detailed Description . . . . .	417
15.57.2 Function Documentation . . . . .	417
15.57.2.1 last_parameter_file() . . . . .	417
15.57.2.2 read_parameter_file() . . . . .	417
15.57.2.3 write_parameter_file() . . . . .	418
15.58 Converting Energy Parameter Files . . . . .	419
15.58.1 Detailed Description . . . . .	419
15.58.2 Macro Definition Documentation . . . . .	420
15.58.2.1 VRNA_CONVERT_OUTPUT_ALL . . . . .	420
15.58.2.2 VRNA_CONVERT_OUTPUT_HP . . . . .	420
15.58.2.3 VRNA_CONVERT_OUTPUT_STACK . . . . .	420
15.58.2.4 VRNA_CONVERT_OUTPUT_MM_HP . . . . .	420
15.58.2.5 VRNA_CONVERT_OUTPUT_MM_INT . . . . .	420
15.58.2.6 VRNA_CONVERT_OUTPUT_MM_INT_1N . . . . .	421
15.58.2.7 VRNA_CONVERT_OUTPUT_MM_INT_23 . . . . .	421
15.58.2.8 VRNA_CONVERT_OUTPUT_MM_MULTI . . . . .	421
15.58.2.9 VRNA_CONVERT_OUTPUT_MM_EXT . . . . .	421
15.58.2.10 VRNA_CONVERT_OUTPUT_DANGLE5 . . . . .	421
15.58.2.11 VRNA_CONVERT_OUTPUT_DANGLE3 . . . . .	421
15.58.2.12 VRNA_CONVERT_OUTPUT_INT_11 . . . . .	422
15.58.2.13 VRNA_CONVERT_OUTPUT_INT_21 . . . . .	422
15.58.2.14 VRNA_CONVERT_OUTPUT_INT_22 . . . . .	422
15.58.2.15 VRNA_CONVERT_OUTPUT_BULGE . . . . .	422
15.58.2.16 VRNA_CONVERT_OUTPUT_INT . . . . .	422
15.58.2.17 VRNA_CONVERT_OUTPUT_ML . . . . .	422

15.58.2.18	VRNA_CONVERT_OUTPUT_MISC . . . . .	423
15.58.2.19	VRNA_CONVERT_OUTPUT_SPECIAL_HP . . . . .	423
15.58.2.20	VRNA_CONVERT_OUTPUT_VANILLA . . . . .	423
15.58.2.21	VRNA_CONVERT_OUTPUT_NINIO . . . . .	423
15.58.2.22	VRNA_CONVERT_OUTPUT_DUMP . . . . .	423
15.58.3	Function Documentation . . . . .	424
15.58.3.1	convert_parameter_file() . . . . .	424
15.59	Direct Refolding Paths between two Secondary Structures . . . . .	425
15.59.1	Detailed Description . . . . .	425
15.59.2	Data Structure Documentation . . . . .	426
15.59.2.1	struct vrna_path_s . . . . .	426
15.59.3	Typedef Documentation . . . . .	426
15.59.3.1	path_t . . . . .	426
15.59.4	Function Documentation . . . . .	426
15.59.4.1	vrna_path_findpath_saddle() . . . . .	426
15.59.4.2	vrna_path_findpath_saddle_ub() . . . . .	427
15.59.4.3	vrna_path_findpath() . . . . .	428
15.59.4.4	vrna_path_findpath_ub() . . . . .	429
15.59.4.5	find_saddle() . . . . .	430
15.59.4.6	free_path() . . . . .	430
15.59.4.7	get_path() . . . . .	430
15.60	Utilities to deal with Nucleotide Alphabets . . . . .	433
15.60.1	Detailed Description . . . . .	433
15.60.2	Data Structure Documentation . . . . .	434
15.60.2.1	struct vrna_sequence_s . . . . .	434
15.60.3	Enumeration Type Documentation . . . . .	434
15.60.3.1	vrna_seq_type_e . . . . .	434
15.60.4	Function Documentation . . . . .	434
15.60.4.1	vrna_ptypes() . . . . .	434
15.60.4.2	vrna_nucleotide_encode() . . . . .	435

15.60.4.3 vrna_nucleotide_decode()	435
15.61 (Nucleic Acid Sequence) String Utilities	437
15.61.1 Detailed Description	437
15.61.2 Macro Definition Documentation	438
15.61.2.1 FILENAME_MAX_LENGTH	438
15.61.2.2 FILENAME_ID_LENGTH	438
15.61.3 Function Documentation	438
15.61.3.1 vrna_strdup_printf()	438
15.61.3.2 vrna_strdup_vprintf()	439
15.61.3.3 vrna_strcat_printf()	439
15.61.3.4 vrna_strcat_vprintf()	440
15.61.3.5 vrna_strsplit()	441
15.61.3.6 vrna_random_string()	442
15.61.3.7 vrna_hamming_distance()	442
15.61.3.8 vrna_hamming_distance_bound()	442
15.61.3.9 vrna_seq_toRNA()	443
15.61.3.10 vrna_seq_toupper()	443
15.61.3.11 vrna_seq_ungapped()	444
15.61.3.12 vrna_cut_point_insert()	444
15.61.3.13 vrna_cut_point_remove()	444
15.62 Secondary Structure Utilities	446
15.62.1 Detailed Description	446
15.62.2 Function Documentation	446
15.62.2.1 vrna_bp_distance()	447
15.62.2.2 vrna_refBPcnt_matrix()	447
15.62.2.3 vrna_refBPdist_matrix()	447
15.62.2.4 vrna_db_from_bp_stack()	448
15.63 Dot-Bracket Notation of Secondary Structures	450
15.63.1 Detailed Description	450
15.63.2 Macro Definition Documentation	450



15.63.2.1 VRNA_BRACKETS_ALPHA . . . . .	451
15.63.2.2 VRNA_BRACKETS_RND . . . . .	451
15.63.2.3 VRNA_BRACKETS_CLY . . . . .	451
15.63.2.4 VRNA_BRACKETS_ANG . . . . .	451
15.63.2.5 VRNA_BRACKETS_SQR . . . . .	452
15.63.2.6 VRNA_BRACKETS_DEFAULT . . . . .	452
15.63.3 Function Documentation . . . . .	452
15.63.3.1 vrna_db_pack() . . . . .	452
15.63.3.2 vrna_db_unpack() . . . . .	453
15.63.3.3 vrna_db_flatten() . . . . .	453
15.63.3.4 vrna_db_flatten_to() . . . . .	454
15.63.3.5 vrna_db_from_ptable() . . . . .	455
15.63.3.6 vrna_db_from_WUSS() . . . . .	455
15.63.3.7 vrna_db_from_plist() . . . . .	456
15.63.3.8 vrna_db_to_element_string() . . . . .	456
15.64 Pair Table Representation of Secondary Structures . . . . .	457
15.64.1 Detailed Description . . . . .	457
15.64.2 Function Documentation . . . . .	457
15.64.2.1 vrna_ptable() . . . . .	457
15.64.2.2 vrna_ptable_from_string() . . . . .	458
15.64.2.3 vrna_pt_pk_get() . . . . .	458
15.64.2.4 vrna_ptable_copy() . . . . .	459
15.64.2.5 vrna_pt_snoop_get() . . . . .	459
15.65 Pair List Representation of Secondary Structures . . . . .	460
15.65.1 Detailed Description . . . . .	460
15.65.2 Data Structure Documentation . . . . .	460
15.65.2.1 struct vrna_elem_prob_s . . . . .	460
15.65.3 Function Documentation . . . . .	461
15.65.3.1 vrna_plist() . . . . .	461
15.66 Helix List Representation of Secondary Structures . . . . .	462

15.66.1 Detailed Description . . . . .	462
15.66.2 Data Structure Documentation . . . . .	462
15.66.2.1 struct vrna_hx_s . . . . .	462
15.66.3 Function Documentation . . . . .	462
15.66.3.1 vrna_hx_from_ptable() . . . . .	462
15.67 Tree Representation of Secondary Structures . . . . .	464
15.67.1 Detailed Description . . . . .	464
15.67.2 Macro Definition Documentation . . . . .	464
15.67.2.1 VRNA_STRUCTURE_TREE_HIT . . . . .	464
15.67.2.2 VRNA_STRUCTURE_TREE_SHAPIRO_SHORT . . . . .	465
15.67.2.3 VRNA_STRUCTURE_TREE_SHAPIRO . . . . .	465
15.67.2.4 VRNA_STRUCTURE_TREE_SHAPIRO_EXT . . . . .	465
15.67.2.5 VRNA_STRUCTURE_TREE_SHAPIRO_WEIGHT . . . . .	465
15.67.2.6 VRNA_STRUCTURE_TREE_EXPANDED . . . . .	466
15.67.3 Function Documentation . . . . .	466
15.67.3.1 vrna_db_to_tree_string() . . . . .	466
15.67.3.2 vrna_tree_string_unweight() . . . . .	467
15.67.3.3 vrna_tree_string_to_db() . . . . .	467
15.68 Deprecated Interface for Secondary Structure Utilities . . . . .	470
15.68.1 Detailed Description . . . . .	470
15.68.2 Function Documentation . . . . .	471
15.68.2.1 b2HIT() . . . . .	471
15.68.2.2 b2C() . . . . .	472
15.68.2.3 b2Shapiro() . . . . .	472
15.68.2.4 add_root() . . . . .	473
15.68.2.5 expand_Shapiro() . . . . .	473
15.68.2.6 expand_Full() . . . . .	473
15.68.2.7 unexpand_Full() . . . . .	474
15.68.2.8 unweight() . . . . .	474
15.68.2.9 unexpand_aligned_F() . . . . .	475

15.68.2.10	<code>parse_structure()</code>	475
15.68.2.11	<code>pack_structure()</code>	475
15.68.2.12	<code>unpack_structure()</code>	476
15.68.2.13	<code>make_pair_table()</code>	476
15.68.2.14	<code>copy_pair_table()</code>	477
15.68.2.15	<code>alimake_pair_table()</code>	477
15.68.2.16	<code>make_pair_table_snoop()</code>	478
15.68.2.17	<code>bp_distance()</code>	478
15.68.2.18	<code>make_referenceBP_array()</code>	478
15.68.2.19	<code>compute_BPdifferences()</code>	479
15.68.2.20	<code>parenthesis_structure()</code>	479
15.68.2.21	<code>parenthesis_zuker()</code>	480
15.68.2.22	<code>ppm_to_structure()</code>	480
15.68.2.23	<code>ppm_symbol()</code>	480
15.69	Multiple Sequence Alignment Utilities	481
15.69.1	Detailed Description	481
15.69.2	Data Structure Documentation	482
15.69.2.1	<code>struct vrna_pinfo_s</code>	482
15.69.3	Macro Definition Documentation	483
15.69.3.1	<code>VRNA_MEASURE_SHANNON_ENTROPY</code>	483
15.69.4	Function Documentation	483
15.69.4.1	<code>vrna_aln_mpi()</code>	483
15.69.4.2	<code>vrna_aln_pinfo()</code>	483
15.69.4.3	<code>vrna_aln_slice()</code>	485
15.69.4.4	<code>vrna_aln_free()</code>	485
15.69.4.5	<code>vrna_aln_uppercase()</code>	487
15.69.4.6	<code>vrna_aln_toRNA()</code>	487
15.69.4.7	<code>vrna_aln_copy()</code>	488
15.69.4.8	<code>vrna_aln_conservation_struct()</code>	488
15.69.4.9	<code>vrna_aln_conservation_col()</code>	489

15.69.4.10vrna_aln_consensus_sequence()	490
15.69.4.11vrna_aln_consensus_mis()	490
15.70Deprecated Interface for Multiple Sequence Alignment Utilities	491
15.70.1 Detailed Description	491
15.70.2 Typedef Documentation	491
15.70.2.1 pair_info	491
15.70.3 Function Documentation	491
15.70.3.1 get_mpi()	492
15.70.3.2 encode_aln_sequence()	492
15.70.3.3 alloc_sequence_arrays()	493
15.70.3.4 free_sequence_arrays()	493
15.71Files and I/O	495
15.71.1 Detailed Description	495
15.71.2 Function Documentation	496
15.71.2.1 vrna_read_line()	496
15.71.2.2 vrna_filename_sanitize()	496
15.71.2.3 vrna_file_exists()	497
15.72Nucleic Acid Sequences and Structures	498
15.72.1 Detailed Description	498
15.72.2 Macro Definition Documentation	499
15.72.2.1 VRNA_OPTION_MULTILINE	499
15.72.2.2 VRNA_CONSTRAINT_MULTILINE	499
15.72.3 Function Documentation	499
15.72.3.1 vrna_file_helixlist()	499
15.72.3.2 vrna_file_connect()	500
15.72.3.3 vrna_file_bpseq()	501
15.72.3.4 vrna_file_json()	501
15.72.3.5 vrna_file_fasta_read_record()	501
15.72.3.6 vrna_extract_record_rest_structure()	503
15.72.3.7 vrna_file_SHAPE_read()	504

15.72.3.8 vrna_extract_record_rest_constraint()	504
15.72.3.9 read_record()	505
15.73 Multiple Sequence Alignments	506
15.73.1 Detailed Description	506
15.73.2 Macro Definition Documentation	507
15.73.2.1 VRNA_FILE_FORMAT_MSA_CLUSTAL	507
15.73.2.2 VRNA_FILE_FORMAT_MSA_STOCKHOLM	507
15.73.2.3 VRNA_FILE_FORMAT_MSA_FASTA	508
15.73.2.4 VRNA_FILE_FORMAT_MSA_MAF	508
15.73.2.5 VRNA_FILE_FORMAT_MSA_MIS	508
15.73.2.6 VRNA_FILE_FORMAT_MSA_DEFAULT	509
15.73.2.7 VRNA_FILE_FORMAT_MSA_NOCHECK	509
15.73.2.8 VRNA_FILE_FORMAT_MSA_UNKNOWN	509
15.73.2.9 VRNA_FILE_FORMAT_MSA_APPEND	510
15.73.2.10 VRNA_FILE_FORMAT_MSA_QUIET	510
15.73.2.11 VRNA_FILE_FORMAT_MSA_SILENT	510
15.73.3 Function Documentation	510
15.73.3.1 vrna_file_msa_read()	511
15.73.3.2 vrna_file_msa_read_record()	512
15.73.3.3 vrna_file_msa_detect_format()	513
15.73.3.4 vrna_file_msa_write()	514
15.74 Command Files	516
15.74.1 Detailed Description	516
15.74.2 Macro Definition Documentation	517
15.74.2.1 VRNA_CMD_PARSE_HC	517
15.74.2.2 VRNA_CMD_PARSE_SC	517
15.74.2.3 VRNA_CMD_PARSE_UD	517
15.74.2.4 VRNA_CMD_PARSE_SD	518
15.74.2.5 VRNA_CMD_PARSE_DEFAULTS	518
15.74.3 Function Documentation	518

15.74.3.1 vrna_file_commands_read()	518
15.74.3.2 vrna_file_commands_apply()	519
15.74.3.3 vrna_commands_apply()	520
15.74.3.4 vrna_commands_free()	520
15.75 Plotting	521
15.75.1 Detailed Description	521
15.75.2 Data Structure Documentation	522
15.75.2.1 struct COORDINATE	522
15.75.2.2 struct vrna_dotplot_auxdata_t	523
15.75.3 Macro Definition Documentation	523
15.75.3.1 VRNA_PLOT_TYPE_SIMPLE	523
15.75.3.2 VRNA_PLOT_TYPE_NAVIEW	523
15.75.3.3 VRNA_PLOT_TYPE_CIRCULAR	524
15.75.4 Function Documentation	524
15.75.4.1 vrna_file_PS_aln()	524
15.75.4.2 vrna_file_PS_aln_sub()	524
15.75.4.3 aliPS_color_aln()	525
15.75.4.4 simple_xy_coordinates()	525
15.75.4.5 simple_circplot_coordinates()	526
15.75.4.6 PS_dot_plot_list()	526
15.75.4.7 PS_dot_plot()	527
15.75.4.8 vrna_file_PS_rnaplot()	527
15.75.4.9 vrna_file_PS_rnaplot_a()	528
15.75.4.10 gmlRNA()	528
15.75.4.11 tssv_rna_plot()	529
15.75.4.12 svg_rna_plot()	530
15.75.4.13 rna_plot()	530
15.75.4.14 PS_rna_plot()	530
15.75.4.15 PS_rna_plot_a()	531
15.75.4.16 PS_rna_plot_a_gquad()	531

15.75.5 Variable Documentation . . . . .	531
15.75.5.1 rna_plot_type . . . . .	532
15.76 Annotation . . . . .	533
15.76.1 Detailed Description . . . . .	533
15.77 Search Algorithms . . . . .	534
15.77.1 Detailed Description . . . . .	534
15.77.2 Function Documentation . . . . .	534
15.77.2.1 vrna_search_BMH_num() . . . . .	534
15.77.2.2 vrna_search_BMH() . . . . .	535
15.77.2.3 vrna_search_BM_BCT_num() . . . . .	536
15.77.2.4 vrna_search_BM_BCT() . . . . .	536
15.78 Combinatorics Algorithms . . . . .	538
15.78.1 Detailed Description . . . . .	538
15.78.2 Function Documentation . . . . .	538
15.78.2.1 vrna_enumerate_necklaces() . . . . .	538
15.78.2.2 vrna_rotational_symmetry_num() . . . . .	539
15.78.2.3 vrna_rotational_symmetry_pos_num() . . . . .	540
15.78.2.4 vrna_rotational_symmetry() . . . . .	540
15.78.2.5 vrna_rotational_symmetry_pos() . . . . .	541
15.78.2.6 vrna_rotational_symmetry_db() . . . . .	542
15.78.2.7 vrna_rotational_symmetry_db_pos() . . . . .	542
15.79 (Abstract) Data Structures . . . . .	544
15.79.1 Detailed Description . . . . .	544
15.79.2 Data Structure Documentation . . . . .	546
15.79.2.1 struct vrna_basepair_s . . . . .	546
15.79.2.2 struct vrna_cpair_s . . . . .	546
15.79.2.3 struct vrna_color_s . . . . .	546
15.79.2.4 struct vrna_data_linear_s . . . . .	546
15.79.2.5 struct vrna_sect_s . . . . .	546
15.79.2.6 struct vrna_bp_stack_s . . . . .	546

15.79.2.7 struct pu_contrib . . . . .	546
15.79.2.8 struct interact . . . . .	547
15.79.2.9 struct pu_out . . . . .	547
15.79.2.10 struct constrain . . . . .	547
15.79.2.11 struct duplexT . . . . .	548
15.79.2.12 struct node . . . . .	548
15.79.2.13 struct snoopT . . . . .	548
15.79.2.14 struct dupVar . . . . .	548
15.79.3 Typedef Documentation . . . . .	548
15.79.3.1 PAIR . . . . .	548
15.79.3.2 plist . . . . .	548
15.79.3.3 cpair . . . . .	549
15.79.3.4 sect . . . . .	549
15.79.3.5 bondT . . . . .	549
15.79.4 Function Documentation . . . . .	549
15.79.4.1 vrna_C11_features() . . . . .	550
15.80 Messages . . . . .	551
15.80.1 Detailed Description . . . . .	551
15.80.2 Function Documentation . . . . .	551
15.80.2.1 vrna_message_error() . . . . .	551
15.80.2.2 vrna_message_verror() . . . . .	552
15.80.2.3 vrna_message_warning() . . . . .	552
15.80.2.4 vrna_message_vwarning() . . . . .	553
15.80.2.5 vrna_message_info() . . . . .	553
15.80.2.6 vrna_message_vinfo() . . . . .	554
15.80.2.7 vrna_message_input_seq_simple() . . . . .	554
15.80.2.8 vrna_message_input_seq() . . . . .	555
15.81 Unit Conversion . . . . .	557
15.81.1 Detailed Description . . . . .	557
15.81.2 Enumeration Type Documentation . . . . .	557



15.81.2.1 vrna_unit_energy_e . . . . .	557
15.81.2.2 vrna_unit_temperature_e . . . . .	558
15.81.3 Function Documentation . . . . .	559
15.81.3.1 vrna_convert_energy() . . . . .	559
15.81.3.2 vrna_convert_temperature() . . . . .	559
15.82The Fold Compound . . . . .	561
15.82.1 Detailed Description . . . . .	561
15.82.2 Data Structure Documentation . . . . .	562
15.82.2.1 struct vrna_fc_s . . . . .	562
15.82.3 Macro Definition Documentation . . . . .	570
15.82.3.1 VRNA_STATUS_MFE_PRE . . . . .	570
15.82.3.2 VRNA_STATUS_MFE_POST . . . . .	571
15.82.3.3 VRNA_STATUS_PF_PRE . . . . .	571
15.82.3.4 VRNA_STATUS_PF_POST . . . . .	571
15.82.3.5 VRNA_OPTION_MFE . . . . .	571
15.82.3.6 VRNA_OPTION_PF . . . . .	572
15.82.3.7 VRNA_OPTION_EVAL_ONLY . . . . .	572
15.82.4 Typedef Documentation . . . . .	572
15.82.4.1 vrna_callback_free_auxdata . . . . .	572
15.82.4.2 vrna_callback_recursion_status . . . . .	573
15.82.5 Enumeration Type Documentation . . . . .	573
15.82.5.1 vrna_fc_type_e . . . . .	573
15.82.6 Function Documentation . . . . .	574
15.82.6.1 vrna_fold_compound() . . . . .	574
15.82.6.2 vrna_fold_compound_comparative() . . . . .	575
15.82.6.3 vrna_fold_compound_free() . . . . .	576
15.82.6.4 vrna_fold_compound_add_auxdata() . . . . .	576
15.82.6.5 vrna_fold_compound_add_callback() . . . . .	577
15.83The Dynamic Programming Matrices . . . . .	578
15.83.1 Detailed Description . . . . .	578

15.83.2 Data Structure Documentation . . . . .	579
15.83.2.1 struct vrna_mx_mfe_s . . . . .	579
15.83.2.2 struct vrna_mx_pf_s . . . . .	579
15.83.3 Enumeration Type Documentation . . . . .	580
15.83.3.1 vrna_mx_type_e . . . . .	580
15.83.4 Function Documentation . . . . .	581
15.83.4.1 vrna_mx_add() . . . . .	581
15.83.4.2 vrna_mx_mfe_free() . . . . .	581
15.83.4.3 vrna_mx_pf_free() . . . . .	582
15.84 Hash Tables . . . . .	583
15.84.1 Detailed Description . . . . .	583
15.84.2 Data Structure Documentation . . . . .	584
15.84.2.1 struct vrna_ht_entry_db_t . . . . .	584
15.84.3 Typedef Documentation . . . . .	584
15.84.3.1 vrna_hash_table_t . . . . .	585
15.84.3.2 vrna_callback_ht_compare_entries . . . . .	585
15.84.3.3 vrna_callback_ht_hash_function . . . . .	585
15.84.3.4 vrna_callback_ht_free_entry . . . . .	586
15.84.4 Function Documentation . . . . .	586
15.84.4.1 vrna_ht_init() . . . . .	586
15.84.4.2 vrna_ht_size() . . . . .	587
15.84.4.3 vrna_ht_collisions() . . . . .	587
15.84.4.4 vrna_ht_get() . . . . .	589
15.84.4.5 vrna_ht_insert() . . . . .	589
15.84.4.6 vrna_ht_remove() . . . . .	590
15.84.4.7 vrna_ht_clear() . . . . .	590
15.84.4.8 vrna_ht_free() . . . . .	591
15.84.4.9 vrna_ht_db_comp() . . . . .	591
15.84.4.10 vrna_ht_db_hash_func() . . . . .	592
15.84.4.11 vrna_ht_db_free_entry() . . . . .	592
15.85 Buffers . . . . .	594
15.85.1 Detailed Description . . . . .	594
15.85.2 Typedef Documentation . . . . .	595
15.85.2.1 vrna_callback_stream_output . . . . .	595
15.85.3 Function Documentation . . . . .	595
15.85.3.1 vrna_cstr() . . . . .	595
15.85.3.2 vrna_cstr_free() . . . . .	596
15.85.3.3 vrna_cstr_close() . . . . .	596
15.85.3.4 vrna_cstr_fflush() . . . . .	596
15.85.3.5 vrna_ostream_init() . . . . .	597
15.85.3.6 vrna_ostream_free() . . . . .	598
15.85.3.7 vrna_ostream_request() . . . . .	598
15.85.3.8 vrna_ostream_provide() . . . . .	598

<b>16 Data Structure Documentation</b>	<b>601</b>
16.1 <code>_struct_en</code> Struct Reference	601
16.1.1 Detailed Description	601
16.2 <code>LIST</code> Struct Reference	601
16.3 <code>LST_BUCKET</code> Struct Reference	601
16.4 <code>Postorder_list</code> Struct Reference	602
16.4.1 Detailed Description	602
16.5 <code>swString</code> Struct Reference	602
16.5.1 Detailed Description	602
16.6 <code>Tree</code> Struct Reference	602
16.6.1 Detailed Description	602
16.7 <code>TwoDpfold_vars</code> Struct Reference	603
16.7.1 Detailed Description	603
16.8 <code>vrna_dimer_conc_s</code> Struct Reference	604
16.8.1 Detailed Description	604
16.9 <code>vrna_hc_bp_storage_t</code> Struct Reference	604
16.9.1 Detailed Description	604
16.10 <code>vrna_sc_bp_storage_t</code> Struct Reference	604
16.10.1 Detailed Description	604
16.11 <code>vrna_sc_motif_s</code> Struct Reference	605
16.12 <code>vrna_structured_domains_s</code> Struct Reference	605
16.13 <code>vrna_subopt_sol_s</code> Struct Reference	605
16.13.1 Detailed Description	605
16.14 <code>vrna_unstructured_domain_motif_s</code> Struct Reference	605

<b>17 File Documentation</b>	<b>607</b>
17.1 ViennaRNA/2Dfold.h File Reference	607
17.1.1 Detailed Description	608
17.2 ViennaRNA/2Dpfold.h File Reference	608
17.2.1 Detailed Description	609
17.2.2 Function Documentation	609
17.2.2.1 get_TwoDpfold_variables()	609
17.2.2.2 destroy_TwoDpfold_variables()	609
17.2.2.3 TwoDpfoldList()	610
17.2.2.4 TwoDpfold_pbacktrack()	610
17.2.2.5 TwoDpfold_pbacktrack5()	611
17.3 ViennaRNA/alifold.h File Reference	612
17.3.1 Detailed Description	613
17.3.2 Function Documentation	613
17.3.2.1 energy_of_alistruct()	613
17.3.2.2 update_alifold_params()	614
17.3.3 Variable Documentation	614
17.3.3.1 cv_fact	614
17.3.3.2 nc_fact	614
17.4 ViennaRNA/aln_util.h File Reference	615
17.4.1 Detailed Description	615
17.5 ViennaRNA/alphabet.h File Reference	615
17.5.1 Detailed Description	615
17.6 ViennaRNA/boltzmann_sampling.h File Reference	616
17.6.1 Detailed Description	616
17.7 ViennaRNA/centroid.h File Reference	616
17.7.1 Detailed Description	617
17.7.2 Function Documentation	617
17.7.2.1 get_centroid_struct_pl()	617
17.7.2.2 get_centroid_struct_pr()	617

17.8 ViennaRNA/char_stream.h File Reference . . . . .	617
17.8.1 Detailed Description . . . . .	617
17.9 ViennaRNA/datastructures/char_stream.h File Reference . . . . .	618
17.9.1 Detailed Description . . . . .	618
17.10ViennaRNA/cofold.h File Reference . . . . .	618
17.10.1 Detailed Description . . . . .	619
17.11ViennaRNA/combinatorics.h File Reference . . . . .	619
17.11.1 Detailed Description . . . . .	620
17.12ViennaRNA/commands.h File Reference . . . . .	620
17.12.1 Detailed Description . . . . .	620
17.13ViennaRNA/concentrations.h File Reference . . . . .	621
17.13.1 Detailed Description . . . . .	621
17.13.2 Function Documentation . . . . .	621
17.13.2.1 get_concentrations() . . . . .	622
17.14ViennaRNA/constraints.h File Reference . . . . .	622
17.14.1 Detailed Description . . . . .	622
17.15ViennaRNA/constraints/hard.h File Reference . . . . .	623
17.15.1 Detailed Description . . . . .	625
17.15.2 Macro Definition Documentation . . . . .	625
17.15.2.1 VRNA_CONSTRAINT_NO_HEADER . . . . .	625
17.15.2.2 VRNA_CONSTRAINT_DB_ANG_BRACK . . . . .	625
17.15.3 Enumeration Type Documentation . . . . .	626
17.15.3.1 vrna_hc_type_e . . . . .	626
17.15.4 Function Documentation . . . . .	626
17.15.4.1 vrna_hc_add_data() . . . . .	626
17.15.4.2 print_tty_constraint() . . . . .	627
17.15.4.3 print_tty_constraint_full() . . . . .	627
17.15.4.4 constrain_ptypes() . . . . .	627
17.16ViennaRNA/constraints/ligand.h File Reference . . . . .	628
17.16.1 Detailed Description . . . . .	628

17.17ViennaRNA/constraints/SHAPE.h File Reference . . . . .	628
17.17.1 Detailed Description . . . . .	629
17.17.2 Function Documentation . . . . .	629
17.17.2.1 vrna_sc_SHAPE_parse_method() . . . . .	629
17.18ViennaRNA/constraints/soft.h File Reference . . . . .	630
17.18.1 Detailed Description . . . . .	631
17.18.2 Enumeration Type Documentation . . . . .	631
17.18.2.1 vrna_sc_type_e . . . . .	631
17.19ViennaRNA/constraints_hard.h File Reference . . . . .	632
17.19.1 Detailed Description . . . . .	632
17.20ViennaRNA/constraints_ligand.h File Reference . . . . .	632
17.20.1 Detailed Description . . . . .	632
17.21ViennaRNA/constraints_SHAPE.h File Reference . . . . .	632
17.21.1 Detailed Description . . . . .	632
17.22ViennaRNA/constraints_soft.h File Reference . . . . .	633
17.22.1 Detailed Description . . . . .	633
17.23ViennaRNA/convert_epars.h File Reference . . . . .	633
17.23.1 Detailed Description . . . . .	633
17.24ViennaRNA/data_structures.h File Reference . . . . .	633
17.24.1 Detailed Description . . . . .	633
17.25ViennaRNA/datastructures/hash_tables.h File Reference . . . . .	634
17.25.1 Detailed Description . . . . .	635
17.26ViennaRNA/dist_vars.h File Reference . . . . .	635
17.26.1 Detailed Description . . . . .	635
17.26.2 Variable Documentation . . . . .	635
17.26.2.1 edit_backtrack . . . . .	635
17.26.2.2 cost_matrix . . . . .	636
17.27ViennaRNA/dp_matrices.h File Reference . . . . .	636
17.27.1 Detailed Description . . . . .	637
17.28ViennaRNA/duplex.h File Reference . . . . .	637

17.28.1 Detailed Description . . . . .	637
17.29ViennaRNA/edit_cost.h File Reference . . . . .	637
17.29.1 Detailed Description . . . . .	637
17.30ViennaRNA/energy_const.h File Reference . . . . .	637
17.30.1 Detailed Description . . . . .	637
17.31ViennaRNA/energy_par.h File Reference . . . . .	638
17.31.1 Detailed Description . . . . .	638
17.32ViennaRNA/equilibrium_probs.h File Reference . . . . .	638
17.32.1 Detailed Description . . . . .	639
17.33ViennaRNA/eval.h File Reference . . . . .	639
17.33.1 Detailed Description . . . . .	642
17.34ViennaRNA/exterior_loops.h File Reference . . . . .	642
17.34.1 Detailed Description . . . . .	642
17.35ViennaRNA/file_formats.h File Reference . . . . .	642
17.35.1 Detailed Description . . . . .	642
17.36ViennaRNA/io/file_formats.h File Reference . . . . .	643
17.36.1 Detailed Description . . . . .	643
17.37ViennaRNA/file_formats_msa.h File Reference . . . . .	644
17.37.1 Detailed Description . . . . .	644
17.38ViennaRNA/io/file_formats_msa.h File Reference . . . . .	644
17.38.1 Detailed Description . . . . .	645
17.39ViennaRNA/file_utils.h File Reference . . . . .	645
17.39.1 Detailed Description . . . . .	645
17.40ViennaRNA/findpath.h File Reference . . . . .	645
17.40.1 Detailed Description . . . . .	646
17.41ViennaRNA/fold.h File Reference . . . . .	646
17.41.1 Detailed Description . . . . .	647
17.42ViennaRNA/fold_compound.h File Reference . . . . .	647
17.42.1 Detailed Description . . . . .	649
17.43ViennaRNA/fold_vars.h File Reference . . . . .	649

17.43.1 Detailed Description . . . . .	649
17.43.2 Variable Documentation . . . . .	649
17.43.2.1 RibosumFile . . . . .	649
17.43.2.2 james_rule . . . . .	650
17.43.2.3 logML . . . . .	650
17.43.2.4 cut_point . . . . .	650
17.43.2.5 base_pair . . . . .	650
17.43.2.6 pr . . . . .	650
17.43.2.7 iindx . . . . .	651
17.44ViennaRNA/gquad.h File Reference . . . . .	651
17.44.1 Detailed Description . . . . .	651
17.45ViennaRNA/grammar.h File Reference . . . . .	651
17.45.1 Detailed Description . . . . .	652
17.46ViennaRNA/hairpin_loops.h File Reference . . . . .	652
17.46.1 Detailed Description . . . . .	652
17.47ViennaRNA/interior_loops.h File Reference . . . . .	652
17.47.1 Detailed Description . . . . .	652
17.48ViennaRNA/inverse.h File Reference . . . . .	652
17.48.1 Detailed Description . . . . .	653
17.49ViennaRNA/Lfold.h File Reference . . . . .	653
17.49.1 Detailed Description . . . . .	653
17.50ViennaRNA/loop_energies.h File Reference . . . . .	653
17.50.1 Detailed Description . . . . .	653
17.51ViennaRNA/loops/all.h File Reference . . . . .	654
17.51.1 Detailed Description . . . . .	654
17.52ViennaRNA/loops/external.h File Reference . . . . .	654
17.52.1 Detailed Description . . . . .	655
17.53ViennaRNA/loops/hairpin.h File Reference . . . . .	655
17.53.1 Detailed Description . . . . .	656
17.54ViennaRNA/loops/internal.h File Reference . . . . .	656



17.54.1 Detailed Description . . . . .	656
17.55ViennaRNA/loops/multibranch.h File Reference . . . . .	657
17.55.1 Detailed Description . . . . .	657
17.56ViennaRNA/LPfold.h File Reference . . . . .	658
17.56.1 Detailed Description . . . . .	658
17.56.2 Function Documentation . . . . .	658
17.56.2.1 init_pf_foldLP() . . . . .	658
17.57ViennaRNA/MEA.h File Reference . . . . .	658
17.57.1 Detailed Description . . . . .	659
17.58ViennaRNA/mfe.h File Reference . . . . .	659
17.58.1 Detailed Description . . . . .	660
17.59ViennaRNA/mfe_window.h File Reference . . . . .	660
17.59.1 Detailed Description . . . . .	661
17.60ViennaRNA/mm.h File Reference . . . . .	661
17.60.1 Detailed Description . . . . .	661
17.60.2 Function Documentation . . . . .	661
17.60.2.1 vrna_maximum_matching() . . . . .	661
17.60.2.2 vrna_maximum_matching_simple() . . . . .	661
17.61ViennaRNA/model.h File Reference . . . . .	662
17.61.1 Detailed Description . . . . .	666
17.62ViennaRNA/multibranch_loops.h File Reference . . . . .	666
17.62.1 Detailed Description . . . . .	666
17.63ViennaRNA/naview.h File Reference . . . . .	666
17.63.1 Detailed Description . . . . .	666
17.64ViennaRNA/plotting/naview.h File Reference . . . . .	667
17.65ViennaRNA/neighbor.h File Reference . . . . .	667
17.65.1 Detailed Description . . . . .	668
17.66ViennaRNA/params.h File Reference . . . . .	668
17.66.1 Detailed Description . . . . .	668
17.67ViennaRNA/params/1.8.4_epars.h File Reference . . . . .	668

17.67.1 Detailed Description . . . . .	668
17.68ViennaRNA/params/1.8.4_intloops.h File Reference . . . . .	669
17.68.1 Detailed Description . . . . .	669
17.69ViennaRNA/params/basic.h File Reference . . . . .	669
17.69.1 Detailed Description . . . . .	670
17.70ViennaRNA/constraints/basic.h File Reference . . . . .	671
17.70.1 Detailed Description . . . . .	672
17.71ViennaRNA/utils/basic.h File Reference . . . . .	672
17.71.1 Detailed Description . . . . .	674
17.71.2 Function Documentation . . . . .	674
17.71.2.1 get_line() . . . . .	674
17.71.2.2 print_tty_input_seq() . . . . .	675
17.71.2.3 print_tty_input_seq_str() . . . . .	675
17.71.2.4 warn_user() . . . . .	675
17.71.2.5 nrerror() . . . . .	676
17.71.2.6 space() . . . . .	676
17.71.2.7 xrealloc() . . . . .	676
17.71.2.8 init_rand() . . . . .	676
17.71.2.9 urn() . . . . .	677
17.71.2.10nt_urn() . . . . .	677
17.71.2.11filecopy() . . . . .	677
17.71.2.12time_stamp() . . . . .	677
17.72ViennaRNA/datastructures/basic.h File Reference . . . . .	678
17.72.1 Detailed Description . . . . .	679
17.73ViennaRNA/params/constants.h File Reference . . . . .	679
17.73.1 Detailed Description . . . . .	680
17.73.2 Macro Definition Documentation . . . . .	680
17.73.2.1 GASCONST . . . . .	680
17.73.2.2 K0 . . . . .	680
17.73.2.3 INF . . . . .	680

17.73.2.4 FORBIDDEN . . . . .	680
17.73.2.5 BONUS . . . . .	681
17.73.2.6 NBPAIRS . . . . .	681
17.73.2.7 TURN . . . . .	681
17.73.2.8 MAXLOOP . . . . .	681
17.74ViennaRNA/params/convert.h File Reference . . . . .	681
17.74.1 Detailed Description . . . . .	682
17.75ViennaRNA/params/io.h File Reference . . . . .	682
17.75.1 Detailed Description . . . . .	682
17.76ViennaRNA/part_func.h File Reference . . . . .	682
17.76.1 Detailed Description . . . . .	684
17.76.2 Function Documentation . . . . .	684
17.76.2.1 centroid() . . . . .	684
17.76.2.2 get_centroid_struct_gquad_pr() . . . . .	685
17.76.2.3 mean_bp_dist() . . . . .	685
17.76.2.4 expLoopEnergy() . . . . .	685
17.76.2.5 expHairpinEnergy() . . . . .	685
17.77ViennaRNA/part_func_co.h File Reference . . . . .	686
17.77.1 Detailed Description . . . . .	686
17.77.2 Function Documentation . . . . .	686
17.77.2.1 get_plist() . . . . .	687
17.78ViennaRNA/part_func_up.h File Reference . . . . .	687
17.78.1 Detailed Description . . . . .	687
17.79ViennaRNA/part_func_window.h File Reference . . . . .	687
17.79.1 Detailed Description . . . . .	689
17.80ViennaRNA/perturbation_fold.h File Reference . . . . .	689
17.80.1 Detailed Description . . . . .	690
17.81ViennaRNA/plot_aln.h File Reference . . . . .	690
17.81.1 Detailed Description . . . . .	690
17.82ViennaRNA/plot_layouts.h File Reference . . . . .	690

17.82.1 Detailed Description . . . . .	690
17.83ViennaRNA/plot_structure.h File Reference . . . . .	690
17.83.1 Detailed Description . . . . .	691
17.84ViennaRNA/plot_utils.h File Reference . . . . .	691
17.84.1 Detailed Description . . . . .	691
17.85ViennaRNA/plotting/alignments.h File Reference . . . . .	691
17.85.1 Detailed Description . . . . .	691
17.86ViennaRNA/utls/alignments.h File Reference . . . . .	692
17.86.1 Detailed Description . . . . .	693
17.87ViennaRNA/plotting/layouts.h File Reference . . . . .	693
17.87.1 Detailed Description . . . . .	694
17.88ViennaRNA/plotting/probabilities.h File Reference . . . . .	694
17.88.1 Detailed Description . . . . .	695
17.89ViennaRNA/plotting/structures.h File Reference . . . . .	695
17.89.1 Detailed Description . . . . .	696
17.90ViennaRNA/utls/structures.h File Reference . . . . .	696
17.90.1 Detailed Description . . . . .	699
17.91ViennaRNA/profiledist.h File Reference . . . . .	699
17.91.1 Function Documentation . . . . .	699
17.91.1.1 profile_edit_distance() . . . . .	699
17.91.1.2 Make_bp_profile_bppm() . . . . .	700
17.91.1.3 free_profile() . . . . .	700
17.91.1.4 Make_bp_profile() . . . . .	700
17.92ViennaRNA/PS_dot.h File Reference . . . . .	701
17.92.1 Detailed Description . . . . .	701
17.93ViennaRNA/read_epars.h File Reference . . . . .	701
17.93.1 Detailed Description . . . . .	701
17.94ViennaRNA/ribo.h File Reference . . . . .	701
17.94.1 Detailed Description . . . . .	702
17.95ViennaRNA/RNAstruct.h File Reference . . . . .	702

17.95.1 Detailed Description . . . . .	703
17.96ViennaRNA/search/BoyerMoore.h File Reference . . . . .	703
17.96.1 Detailed Description . . . . .	703
17.97ViennaRNA/sequence.h File Reference . . . . .	703
17.97.1 Detailed Description . . . . .	704
17.98ViennaRNA/stream_output.h File Reference . . . . .	704
17.98.1 Detailed Description . . . . .	704
17.99ViennaRNA/datastructures/stream_output.h File Reference . . . . .	704
17.99.1 Detailed Description . . . . .	705
17.100ViennaRNA/string_utils.h File Reference . . . . .	705
17.100.1 Detailed Description . . . . .	705
17.101ViennaRNA/stringdist.h File Reference . . . . .	705
17.101.1 Detailed Description . . . . .	706
17.101.2 Function Documentation . . . . .	706
17.101.2.1 Make_swString() . . . . .	706
17.101.2.2 string_edit_distance() . . . . .	706
17.102ViennaRNA/structure_utils.h File Reference . . . . .	707
17.102.1 Detailed Description . . . . .	707
17.103ViennaRNA/structured_domains.h File Reference . . . . .	707
17.103.1 Detailed Description . . . . .	707
17.104ViennaRNA/subopt.h File Reference . . . . .	707
17.104.1 Detailed Description . . . . .	709
17.104.2 Typedef Documentation . . . . .	709
17.104.2.1 SOLUTION . . . . .	709
17.105ViennaRNA/svm_utils.h File Reference . . . . .	709
17.105.1 Detailed Description . . . . .	709
17.106ViennaRNA/treedist.h File Reference . . . . .	709
17.106.1 Detailed Description . . . . .	710
17.106.2 Function Documentation . . . . .	710
17.106.2.1 make_tree() . . . . .	710

17.106.2.2	tree_edit_distance()	710
17.106.2.3	tree_tree()	711
17.107	ViennaRNA/units.h File Reference	711
17.107.1	Detailed Description	711
17.108	ViennaRNA/unstructured_domains.h File Reference	712
17.108.1	Detailed Description	713
17.108.2	Function Documentation	713
17.108.2.1	vrna_ud_set_prob_cb()	713
17.109	ViennaRNA/utls.h File Reference	714
17.109.1	Detailed Description	714
17.110	ViennaRNA/io/utls.h File Reference	714
17.110.1	Detailed Description	714
17.111	ViennaRNA/plotting/utls.h File Reference	715
17.111.1	Detailed Description	715
17.112	ViennaRNA/utls/strings.h File Reference	715
17.112.1	Detailed Description	716
17.112.2	Function Documentation	716
17.112.2.1	str_uppercase()	717
17.112.2.2	str_DNA2RNA()	717
17.112.2.3	random_string()	717
17.112.2.4	hamming()	717
17.112.2.5	hamming_bound()	718
17.113	ViennaRNA/walk.h File Reference	718
17.113.1	Detailed Description	718
<b>Bibliography</b>		<b>720</b>
<b>Index</b>		<b>721</b>

# Chapter 1

## Main Page

### 1.1 A Library for predicting and comparing RNA secondary structures

The core of the ViennaRNA Package ([13], [11]) is formed by a collection of routines for the prediction and comparison of RNA secondary structures. These routines can be accessed through stand-alone programs, such as RNAfold, RNAdistance etc., which should be sufficient for most users. For those who wish to develop their own programs we provide a library which can be linked to your own code.

This document describes the library and will be primarily useful to programmers. However, it also contains details about the implementation that may be of interest to advanced users. The stand-alone programs are described in separate man pages. The latest version of the package including source code and html versions of the documentation can be found at

<http://www.tbi.univie.ac.at/RNA>

#### Date

1994-2018

#### Authors

Ivo Hofacker, Peter Stadler, Ronny Lorenz, and so many more

### 1.2 License

#### Disclaimer and Copyright

The programs, library and source code of the Vienna RNA Package are free software. They are distributed in the hope that they will be useful but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Permission is granted for research, educational, and commercial use and modification so long as 1) the package and any derived works are not redistributed for any fee, other than media costs, 2) proper credit is given to the authors and the Institute for Theoretical Chemistry of the University of Vienna.

If you want to include this software in a commercial product, please contact the authors.

## 1.3 Contributors

Over the past decades since the `ViennaRNA Package` first sprang to life as part of Ivo Hofackers PhD project, several different authors contributed more and more algorithm implementations. In 2008, Ronny Lorenz took over the extensive task to harmonize and simplify the already existing implementations for the sake of easier feature addition. This eventually lead to version 2.0 of the `ViennaRNA Package`. Since then, he (re-)implemented a large portion of the currently existing library features, such as the new, generalized constraints framework, RNA folding grammar domain extensions, and the major part of the scripting language interface. Below is a list of most people who contributed larger parts of the implementations:

- Juraj Michalik (non-redundant Boltzmann sampling)
- Gregor Entzian (neighbor, walk)
- Mario Koestl (worked on SWIG interface and related unit testing)
- Dominik Luntzer (perturbation fold)
- Stefan Badelt (cofold evaluation, RNAdesign.pl, cofold findpath extensions)
- Stefan Hammer (parts of SWIG interface and corresponding unit tests)
- Ronny Lorenz (circfold, version 2.0, generic constraints, grammar extensions, and much more)
- Hakim Tafer (RNAplex, RNAsnoop)
- Ulrike Mueckstein (RNAup)
- Stephan Bernhart (cofold, plfold, unpaired probabilities, alifold, and so many more)
- Ivo Hofacker, Peter Stadler, and Christoph Flamm (almost every implementation up to version 1.8.5)

We also want to thank the following people:

- Sebastian Bonhoeffer's implementation of partition function folding served as a precursor to our `part_func.c`
- Manfred Tacker hacked constrained folding into `fold.c` for the first time
- Martin Fekete made the first attempts at "alignment folding"
- Andrea Tanzer and Martin Raden (Mann) for not stopping to report bugs found through comprehensive usage of our applications and `RNAlib`
- Thanks also to everyone else who helped testing and finding bugs, especially Christoph Flamm, Martijn Huynen, Baerbel Krakhofer, and many more



## Chapter 2

# Getting Started

- [Installation and Configuration](#) describes how to install and configure `RNAlib` for your requirements
- [HelloWorld](#) presents some small example programs to get a first impression on how to use this library
- [HelloWorld \(Perl/Python\)](#) contains small examples that show how to use `RNAlib` even without C/C++ programming skills from within your favorite scripting language

## 2.1 Installation and Configuration

A documentation on how to configure the different features of `RNAlib`, how to install the ViennaRNA Package, and finally, how to link you own programs against `RNAlib`.

### 2.1.1 Installing the ViennaRNA Package

For best portability the ViennaRNA package uses the GNU autoconf and automake tools. The instructions below are for installing the ViennaRNA package from source. However, pre-compiled binaries for various Linux distributions, as well as for Windows users are available from Download section of the [main ViennaRNA homepage](#).

#### 2.1.1.1 Quick-start

Usually you'll just unpack, configure and make. To do this type:

```
tar -zxvf ViennaRNA-2.4.11.tar.gz
cd ViennaRNA-2.4.11
./configure
make
sudo make install
```

#### 2.1.1.2 Installation without root privileges

If you do not have root privileges on your computer, you might want to install the ViennaRNA Package to a location where you actually have write access to. To do so, you can set the installation prefix of the `./configure` script like so:

```
./configure --prefix=/home/username/ViennaRNA
make install
```

This will install the entire ViennaRNA Package into a new directory `ViennaRNA` directly into the users `username` home directory.

### 2.1.1.3 Notes for MacOS X users

Although users will find `/usr/bin/gcc` and `/usr/bin/g++` executables in their directory tree, these programs are not at all what they pretend to be. Instead of including the GNU programs, Apple decided to install clang/llvm in disguise. Unfortunately, the default version of clang/llvm does not support OpenMP (yet), but only complains at a late stage of the build process when this support is required. Therefore, it seems necessary to deactivate OpenMP support by passing the option `--disable-openmp` to the `./configure` script.

Additionally, since MacOS X 10.5 the perl and python installation distributed with MacOS X always include so called universal-binaries (a.k.a. fat-binaries), i.e. binaries for multiple architecture types. In order to compile and link the programs, library, and scripting language interfaces of the ViennaRNA Package for multiple architectures, we've added a new configure switch that sets up the required changes automatically:

```
./configure --enable-universal-binary
```

#### Note

Note, that with link time optimization turned on, MacOS X's default compiler (llvm/clang) generates an intermediary binary format that can not easily be combined into a multi-architecture library. Therefore, the `--enable-universal-binary` switch turns off link time optimization!

## 2.1.2 Configuring RNAlib features

The ViennaRNA Package includes additional executable programs such as RNAforester, Kinfold, and Kinwalker. Furthermore, we include several features in our C-library that may be activated by default, or have to be explicitly turned on at configure-time. Below we list a selection of the available configure options that affect the features included in all executable programs, the RNAlib C-library, and the corresponding scripting language interface(s).

### 2.1.2.1 Streaming SIMD Extension (SSE) support

Since version 2.3.5 our sources contain code that implements a faster multibranch loop decomposition in global MFE predictions, as used e.g. in RNAfold. This implementation makes use of modern processors capability to execute particular instructions on multiple data simultaneously (SIMD - single instruction multiple data, thanks to W. B. Langdon for providing the modified code). Consequently, the time required to assess the minimum of all multibranch loop decompositions is reduced up to about one half compared to the runtime of the original implementation. This feature is enabled by default since version 2.4.11 and a dispatcher ensures that the correct implementation will be selected at runtime. If for any reason you want to disable this feature at compile-time use the following configure flag:

```
./configure --disable-simd
```

### 2.1.2.2 Scripting Interfaces

The ViennaRNA Package comes with scripting language interfaces for Perl 5, Python 2, and Python 3 (provided by swig), that allow one to use the implemented algorithms directly without the need of calling an executable program. The interfaces are build by default whenever the autoconf tool-chain detects the required build tools on your system. You may, however, explicitly turn off particular scripting language interface support at configure-time, for instance for Perl 5 and Python 2, before the actual installation.

Example:

```
./configure --without-perl --without-python
```

Disabling the scripting language support all-together can be accomplished using the following switch:

```
./configure --without-swig
```

### 2.1.2.3 Cluster Analysis

The programs AnalyseSeqs and AnalyseDists offer some cluster analysis tools (split decomposition, statistical geometry, neighbor joining, Ward's method) for sequences and distance data. To also build these programs add

```
--with-cluster
```

to your configure options.

### 2.1.2.4 Kinfold

The Kinfold program can be used to simulate the folding dynamics of an RNA molecule, and is compiled by default. Use the

```
--without-kinfold
```

option to skip compilation and installation of Kinfold.

### 2.1.2.5 RNAforester

The RNAforester program is used for comparing secondary structures using tree alignment. Similar to Kinfold, use the

```
--without-forester
```

option to skip compilation and installation of RNAforester.

### 2.1.2.6 Kinwalker

The Kinwalker algorithm performs co-transcriptional folding of RNAs, starting at a user specified structure (default↵: open chain) and ending at the minimum free energy structure. Compilation and installation of this program is deactivated by default. Use the

```
--with-kinwalker
```

option to enable building and installation of Kinwalker.

### 2.1.2.7 Link Time Optimization (LTO)

To increase the performance of our implementations, the ViennaRNA Package tries to make use of the Link Time Optimization (LTO) feature of modern C-compilers. If you are experiencing any troubles at make-time or run-time, or the configure script for some reason detects that your compiler supports this feature although it doesn't, you can deactivate it using the flag

```
./configure --disable-lto
```

Note, that GCC before version 5 is known to produce unreliable LTO code, especially in combination with SSE (see config\_sse). We therefore recommend using a more recent compiler (GCC 5 or above) or to turn off one of the two features, LTO or SSE optimized code.

### 2.1.2.8 OpenMP support

To enable concurrent computation of our implementations and in some cases parallelization of the algorithms we make use of the OpenMP API. This interface is well understood by most modern compilers. However, in some cases it might be necessary to deactivate OpenMP support and therefore transform *RNAlib* into a C-library that is not entirely *thread-safe*. To do so, add the following configure option

```
./configure --disable-openmp
```

### 2.1.2.9 POSIX threads (pthread) support

To enable concurrent computation of multiple input data in RNAfold, and for our implementation of the concurrent unordered insert, ordered output flush data structure `vrna_ostream_t` we make use of POSIX threads. This should be supported on all modern platforms and usually does not pose any problems. Unfortunately, we use a threadpool implementation that is not compatible with Microsoft Windows yet. Thus, POSIX thread support can not be activated for Windows builds until we have fixed this problem. If you want to compile RNAfold and RNAlib without POSIX threads support for any other reasons, add the following configure option

```
./configure --disable-pthreads
```

### 2.1.2.10 Stochastic backtracking using Boustrophedon scheme

Stochastic backtracking for single RNA sequences, e.g. available through the RNAsubopt program, received a major speedup by implementing a Boustrophedon scheme (see this article for details). If for some reason you want to deactivate this feature, you can do that by adding the following switch to the configure script:

```
./configure --disable-boustrophedon
```

### 2.1.2.11 SVM Z-score filter in RNAfold

By default, RNAfold that comes with the ViennaRNA Package allows for z-score filtering of its predicted results using a support vector machine (SVM). However, the library we use to implement this feature (`libsvm`) is statically linked to our own RNAlib. If this introduces any problems for your own third-party programs that link against RNAlib, you can safely switch off the z-scoring implementation using

```
./configure --without-svm
```

### 2.1.2.12 GNU Scientific Library

The new program RNApvmmin computes a pseudo-energy perturbation vector that aims to minimize the discrepancy of predicted, and observed pairing probabilities. For that purpose it implements several methods to solve the optimization problem. Many of them are provided by the GNU Scientific Library, which is why the RNApvmmin program, and the RNAlib C-library are required to be linked against `libgsl`. If this introduces any problems in your own third-party programs that link against RNAlib, you can turn off a larger portion of available minimizers in RNApvmmin and linking against `libgsl` all-together, using the switch

```
./configure --without-gsl
```

### 2.1.2.13 Disable C11/C++11 feature support

By default, we use C11/C++11 features in our implementations. This mainly accounts for unnamed unions/structs within *RNAlib*. The configure script automatically detects whether or not your compiler understands these features. In case you are using an older compiler, these features will be deactivated by setting a specific pre-processor directive. If for some reason you want to deactivate C11/C++11 features despite the capabilities of your compiler, use the following configure option:

```
./configure --disable-c11
```

### 2.1.2.14 Enable warnings for use of deprecated symbols

Since version 2.2 we are in the process of transforming the API of our *RNAlib*. Hence, several symbols are marked as *deprecated* whenever they have been replaced by the new API. By default, deprecation warnings at compile time are deactivated. If you want to get your terminal spammed by tons of deprecation warnings, enable them using:

```
./configure --enable-warn-deprecated
```

### 2.1.2.15 Single precision partition function

Calculation of partition functions (via `RNAfold -p`) uses double precision floats by default, to avoid overflow errors on longer sequences. If your machine has little memory and you don't plan to fold sequences over 1000 bases in length you can compile the package to do the computations in single precision by running

```
./configure --enable-floatpf
```

#### Note

Using this option is discouraged and not necessary on most modern computers.

### 2.1.2.16 Help

For a complete list of all `./configure` options and important environment variables, type

```
./configure --help
```

For more general information on the build process see the `INSTALL` file.

## 2.1.3 Linking against RNAlib

In order to use our implemented algorithms you simply need to link your program to our *RNAlib* C-library that usually comes along with the ViennaRNA Package installation. If you've installed the ViennaRNA Package as a pre-build binary package, you probably need the corresponding development package, e.g. *viennarna-devel*, or *viennarna-dev*. The only thing that is left is to include the ViennaRNA header files into your source code, e.g.:

```
#include <ViennaRNA/mfe.h>
```

and start using our fast and efficient algorithm implementations.

#### See also

In the `mp_example` and [Some Examples using RNAlib API v3.0](#) sections, we list a small set of example code that usually is a good starting point for your application.

### 2.1.3.1 Compiler and Linker flags

Of course, simply adding the ViennaRNA header files into your source code is usually not enough. You probably need to tell your compiler where to find the header files, and sometimes add additional pre-processor directives. Whenever your installation of *RNAlib* was build with default settings and the header files were installed into their default location, a simple

```
-I/usr/include
```

pre-processor/compile flag should suffice. It can even be omitted in this case, since your compiler should search this directory by default anyway. You only need to change the path from */usr/include* to the correct location whenever the header files have been installed into a non-standard directory.

On the other hand, if you've compiled *RNAlib* with some non-default settings then you probably need to define some additional pre-processor macros:

- *VRNA\_DISABLE\_C11\_FEATURES* ... Disable C11/C++11 features.

#### Warning

Add this directive to your pre-processor/compile flags only if *RNAlib* was build with the *--disable-c11* configure option.

#### See also

[Disable C11/C++11 feature support](#) and [vrna\\_C11\\_features\(\)](#)

- *VRNA\_WARN\_DEPRECATED* ... Enable warnings for using deprecated symbols.

#### Note

Adding this directive enables compiler warnings whenever you use symbols in *RNAlib* that are marked *deprecated*.

#### See also

[Enable warnings for use of deprecated symbols](#) and [Deprecated List](#)

- *USE\_FLOAT\_PF* ... Use single precision floating point operations instead of double precision in partition function computations.

#### Warning

Define this macro only if *RNAlib* was build with the *--enable-floatpf* configure option!

#### See also

[Single precision partition function](#)

Simply add the corresponding definition(s) to your pre-processor/compile flags, for instance:

```
-DVRNA_DISABLE_C11_FEATURES
```

Finally, linking against *RNAlib* is achieved by adding the following linker flag

```
-L/usr/lib -lRNA -fopenmp
```

Again, the path to the library, */usr/lib*, may be omitted if this path is searched for libraries by default. The second flag tells the linker to include *libRNA.a*, and the remaining two flags activate [Link Time Optimization \(LTO\)](#) and [OpenMP support](#) support, respectively.

**Note**

Depending on your linker, the last two flags may differ.

Depending on your configure time decisions, you can drop one or both of the last flags.

In case you've compiled *RNAlib* with LTO support (See [Link Time Optimization \(LTO\)](#)) and you are using the same compiler for your third-party project that links against our library, you may add the

```
-flto
```

flag to enable Link Time Optimization.

**2.1.3.2 The pkg-config tool**

Instead of hard-coding the required compiler and linker flags, you can also let the *pkg-config* tool automatically determine the required flags. This tool is usually packaged for any Linux distribution and should be available for MacOS X and MinGW as well. We ship a file *RNAlib2.pc* which is installed along with the static *libRNA.a* C-library and populated with all required compiler and linker flags that correspond to your configure time decisions.

The compiler flags required for properly building your code that uses *RNAlib* can be easily obtained via

```
pkg-config --cflags RNAlib2
```

You get the corresponding linker flags using

```
pkg-config --libs RNAlib2
```

With this widely accepted standard it is also very easy to integrate *RNAlib* in your *autotools* project, just have a look at the *PKG\_CHECK\_MODULES* macro.

**2.2 HelloWorld**

Below, you'll find some more or less simple C programs showing first steps into using *RNAlib*. A complete list of example C programs can be found in the [C Examples](#) section.

**Simple MFE prediction for a given sequence**

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <ViennaRNA/fold.h>
#include <ViennaRNA/utils/basic.h>

int
main()
{
    /* The RNA sequence */
    char *seq = "GAGUAGUGGAACCAGGCUAUGUUUGUGACUCGCAGACUAACA";

    /* allocate memory for MFE structure (length + 1) */
    char *structure = (char *)vrna_alloc(sizeof(char) * (strlen(seq) + 1));

    /* predict Minum Free Energy and corresponding secondary structure */
    float mfe = vrna_fold(seq, structure);

    /* print sequence, structure and MFE */
    printf("%s\n%s [ %6.2f ]\n", seq, structure, mfe);

    /* cleanup memory */
    free(structure);

    return 0;
}
```

**See also**

examples/helloworld\_mfe.c in the source code tarball

## Simple MFE prediction for a multiple sequence alignment

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <ViennaRNA/alifold.h>
#include <ViennaRNA/Utils/basic.h>
#include <ViennaRNA/Utils/alignments.h>

int
main()
{
    /* The RNA sequence alignment */
    const char *sequences[] = {
        "CUGCCUCACAACGUAUUGUGCCUCAGUUAACCGUAGAUGUAGUGAGGGU",
        "CUGCCUCACAACAUUUGUGCCUCAGUUAACCAUAGAUGUAGUGAGGGU",
        "---CUCGACACCACU---GCCUCGGUUAACCAUCGGUGCAGUGCGGGU",
        NULL /* indicates end of alignment */
    };

    /* compute the consensus sequence */
    char *cons = consensus(sequences);

    /* allocate memory for MFE consensus structure (length + 1) */
    char *structure = (char *)vrna_alloc(sizeof(char) * (strlen(sequences[0]) + 1));

    /* predict Minimum Free Energy and corresponding secondary structure */
    float mfe = vrna_alifold(sequences, structure);

    /* print consensus sequence, structure and MFE */
    printf("%s\n%s [ %.2f ]\n", cons, structure, mfe);

    /* cleanup memory */
    free(cons);
    free(structure);

    return 0;
}
```

### See also

examples/helloworld\_mfe\_comparative.c in the source code tarball

## Simple Base Pair Probability computation

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <ViennaRNA/fold.h>
#include <ViennaRNA/part_func.h>
#include <ViennaRNA/Utils/basic.h>

int
main()
{
    /* The RNA sequence */
    char *seq = "GAGUAGUGGAACAGGCUAUGUUGUGACUCGCAGACUAACA";

    /* allocate memory for pairing propensity string (length + 1) */
    char *propensity = (char *)vrna_alloc(sizeof(char) * (strlen(seq) + 1));

    /* pointers for storing and navigating through base pair probabilities */
    vrna_ep_t *ptr, *pair_probabilities = NULL;

    float en = vrna_pf_fold(seq, propensity, &pair_probabilities);

    /* print sequence, pairing propensity string and ensemble free energy */
    printf("%s\n%s [ %.2f ]\n", seq, propensity, en);

    /* print all base pairs with probability above 50% */
    for (ptr = pair_probabilities; ptr->i != 0; ptr++)
        if (ptr->p > 0.5)
            printf("p(%d, %d) = %g\n", ptr->i, ptr->j, ptr->p);

    /* cleanup memory */
    free(pair_probabilities);
    free(propensity);

    return 0;
}
```



**See also**

examples/helloworld\_probabilities.c in the source code tarball

**Deviating from the Default Model**

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <ViennaRNA/model.h>
#include <ViennaRNA/fold_compound.h>
#include <ViennaRNA/utils/basic.h>
#include <ViennaRNA/utils/strings.h>
#include <ViennaRNA/mfe.h>

int
main()
{
    /* initialize random number generator */
    vrna_init_rand();

    /* Generate a random sequence of 50 nucleotides */
    char *seq = vrna_random_string(50, "ACGU");

    /* allocate memory for MFE structure (length + 1) */
    char *structure = (char *)vrna_alloc(sizeof(char) * (strlen(seq) + 1));

    /* create a new model details structure to store the Model Settings */
    vrna_md_t md;

    /* ALWAYS set default model settings first! */
    vrna_md_set_default(&md);

    /* change temperature and activate G-Quadruplex prediction */
    md.temperature = 25.0; /* 25 Deg Celcius */
    md.gquad = 1; /* Turn-on G-Quadruples support */

    /* create a fold compound */
    vrna_fold_compound_t *fc = vrna_fold_compound(seq, &md,
        VRNA_OPTION_DEFAULT);

    /* predict Minum Free Energy and corresponding secondary structure */
    float mfe = vrna_mfe(fc, structure);

    /* print sequence, structure and MFE */
    printf("%s\n%s [ %6.2f ]\n", seq, structure, mfe);

    /* cleanup memory */
    free(structure);
    vrna_fold_compound_free(fc);

    return 0;
}
```

**See also**

examples/fold\_compound\_md.c in the source code tarball

**2.3 HelloWorld (Perl/Python)****2.3.1 Perl5****Simple MFE prediction for a given sequence**

```
use RNA;

# The RNA sequence
my $seq = "GAGUAGUGGAACCAGGCUAUGUUUGACUCGCAGACUAACA";

# compute minimum free energy (MFE) and corresponding structure
my ($ss, $mfe) = RNA::fold($seq);

# print output
printf "%s\n%s [ %6.2f ]\n", $seq, $ss, $mfe;
```

## Simple MFE prediction for a multiple sequence alignment

```
use RNA;

# The RNA sequence alignment
my @sequences = (
    "CUGCCUCACAACGUUUUGUGCCUCAGUUACCCGUAGAUGUAGUGAGGGU",
    "CUGCCUCACAACAUUUUGUGCCUCAGUUACUCAUAGAUGUAGUGAGGGU",
    "---CUCGACACCACU---GCCUCGGUUAACCAUCGGUGCAGUGCGGGU"
);

# compute the consensus sequence
my $cons = RNA::consensus(\@sequences);

# predict Minimum Free Energy and corresponding secondary structure
my ($ss, $mfe) = RNA::alifold(\@sequences);

# print output
printf "%s\n%s [ %6.2f ]\n", $cons, $ss, $mfe;
```

## Deviating from the Default Model

```
use RNA;

# The RNA sequence
my $seq = "GAGUAGUGGAACCAGGCUAUGUUUGUGACUCGCAGACUAACA";

# create a new model details structure
my $md = new RNA::md();

# change temperature and dangle model
$md->{temperature} = 20.0; # 20 Deg Celcius
$md->{dangles}      = 1;    # Dangle Model 1

# create a fold compound
my $fc = new RNA::fold_compound($seq, $md);

# predict Minimum Free Energy and corresponding secondary structure
my ($ss, $mfe) = $fc->mfe();

# print sequence, structure and MFE
printf "%s\n%s [ %6.2f ]\n", $seq, $ss, $mfe;
```

## 2.3.2 Python

### Simple MFE prediction for a given sequence

```
import RNA

# The RNA sequence
seq = "GAGUAGUGGAACCAGGCUAUGUUUGUGACUCGCAGACUAACA"

# compute minimum free energy (MFE) and corresponding structure
(ss, mfe) = RNA.fold(seq)

# print output
print "%s\n%s [ %6.2f ]" % (seq, ss, mfe)
```

### Simple MFE prediction for a multiple sequence alignment

```
import RNA

# The RNA sequence alignment
sequences = [
    "CUGCCUCACAACGUUUUGUGCCUCAGUUACCCGUAGAUGUAGUGAGGGU",
    "CUGCCUCACAACAUUUUGUGCCUCAGUUACUCAUAGAUGUAGUGAGGGU",
    "---CUCGACACCACU---GCCUCGGUUAACCAUCGGUGCAGUGCGGGU"
]

# compute the consensus sequence
cons = RNA.consensus(sequences)

# predict Minimum Free Energy and corresponding secondary structure
(ss, mfe) = RNA.alifold(sequences);

# print output
print "%s\n%s [ %6.2f ]" % (cons, ss, mfe)
```

## Deviating from the Default Model

```
import RNA

# The RNA sequence
seq = "GAGUAGUGGAACCGGCUAUGUUUGUGACUCGCAGACUAACA"

# create a new model details structure
md = RNA.md()

# change temperature and dangle model
md.temperature = 20.0 # 20 Deg Celcius
md.dangles     = 1    # Dangle Model 1

# create a fold compound
fc = RNA.fold_compound(seq, md)

# predict Minnum Free Energy and corresponding secondary structure
(ss, mfe) = fc.mfe()

# print sequence, structure and MFE
print "%s\n%s [ %6.2f ]\n" % (seq, ss, mfe)
```



## Chapter 3

# Concepts and Algorithms

This is an overview of the concepts and algorithms for which implementations can be found in this library.

Almost all of them rely on the physics based Nearest Neighbor Model for RNA secondary structure prediction.

- [RNA Structure](#) gives an introduction into the different layers of abstraction for RNA structures
- [Distance Measures](#) introduces different metrics to allow for the comparison of secondary structures
- [Free Energy of Secondary Structures](#) shows how the stability of a secondary structure can be quantified in terms of free energy
- [Secondary Structure Folding Grammar](#) explains the basic recursive decomposition scheme that is applied in secondary structure prediction
- [RNA Secondary Structure Landscapes](#) describes how transition paths between secondary structures span a landscape like graph
- [Minimum Free Energy Algorithm\(s\)](#) compute the most stable conformation in thermodynamic equilibrium
- [Partition Function and Equilibrium Probability Algorithm\(s\)](#) enable one to apply statistical mechanics to derive equilibrium probabilities of structure features
- [Suboptimals and \(other\) Representative Structures](#) allow for alternative description and enumeration of the structure ensemble
- [RNA-RNA Interaction](#) introduces how to model the interaction between RNA molecules
- [Locally Stable Secondary Structures](#) offer insights into structuredness of long sequences and entire genomes
- [Comparative Structure Prediction](#) augment structure prediction with evolutionary conservation of homologous sequences
- [Classified DP variations](#) perform an *a priori* partitioning of the structure ensemble and compute various properties for the resulting classes.
- [RNA Sequence Design](#) constitutes the inverse problem of structure prediction
- [Experimental Structure Probing Data](#) can be used to guide structure prediction, for instance using SHAPE reactivity data
- [Ligand Binding](#) adds more complexity to structure prediction by modelling the interaction between small chemical compounds or proteins and the RNA
- [\(Tertiary\) Structure Motifs](#) extend the abstraction of secondary structure beyond canonical base pair formation

## 3.1 RNA Structure

### 3.1.1 RNA Structures

### 3.1.2 Levels of Structure Abstraction

#### 3.1.2.1 Primary Structure

#### 3.1.2.2 Secondary Structure

#### 3.1.2.3 Tertiary Structure

#### 3.1.2.4 Quarternary Structure

#### 3.1.2.5 Pseudo-Knots

## 3.2 Distance Measures

A simple measure of dissimilarity between secondary structures of equal length is the base pair distance, given by the number of pairs present in only one of the two structures being compared. I.e. the number of base pairs that have to be opened or closed to transform one structure into the other. It is therefore particularly useful for comparing structures on the same sequence. It is implemented by

```
int bp_distance(const char *str1,
               const char *str2)
```

Compute the "base pair" distance between two secondary structures s1 and s2.

For other cases a distance measure that allows for gaps is preferable. We can define distances between structures as edit distances between trees or their string representations. In the case of string distances this is the same as "sequence alignment". Given a set of edit operations and edit costs, the edit distance is given by the minimum sum of the costs along an edit path converting one object into the other. Edit distances like these always define a metric. The edit operations used by us are insertion, deletion and replacement of nodes. String editing does not pay attention to the matching of brackets, while in tree editing matching brackets represent a single node of the tree. [Tree](#) editing is therefore usually preferable, although somewhat slower. String edit distances are always smaller or equal to tree edit distances.

The different level of detail in the structure representations defined above naturally leads to different measures of distance. For full structures we use a cost of 1 for deletion or insertion of an unpaired base and 2 for a base pair. Replacing an unpaired base for a pair incurs a cost of 1.

Two cost matrices are provided for coarse grained structures:

```

/* Null, H, B, I, M, S, E */
{ 0, 2, 2, 2, 2, 1, 1}, /* Null replaced */
{ 2, 0, 2, 2, 2, INF, INF}, /* H replaced */
{ 2, 2, 0, 1, 2, INF, INF}, /* B replaced */
{ 2, 2, 1, 0, 2, INF, INF}, /* I replaced */
{ 2, 2, 2, 2, 0, INF, INF}, /* M replaced */
{ 1, INF, INF, INF, INF, 0, INF}, /* S replaced */
{ 1, INF, INF, INF, INF, INF, 0}, /* E replaced */

/* Null, H, B, I, M, S, E */
{ 0, 100, 5, 5, 75, 5, 5}, /* Null replaced */
{ 100, 0, 8, 8, 8, INF, INF}, /* H replaced */
{ 5, 8, 0, 3, 8, INF, INF}, /* B replaced */
{ 5, 8, 3, 0, 8, INF, INF}, /* I replaced */
{ 75, 8, 8, 8, 0, INF, INF}, /* M replaced */
{ 5, INF, INF, INF, INF, 0, INF}, /* S replaced */
{ 5, INF, INF, INF, INF, INF, 0}, /* E replaced */

```

The lower matrix uses the costs given in [21]. All distance functions use the following global variables:

```
int cost_matrix;
```

Specify the cost matrix to be used for distance calculations.

```
int edit_backtrack;
```

Produce an alignment of the two structures being compared by tracing the editing path giving the minimum distance.

```
char *aligned_line[4];
```

Contains the two aligned structures after a call to one of the distance functions with [edit\\_backtrack](#) set to 1.

See also

[utils.h](#), [dist\\_vars.h](#) and [stringdist.h](#) for more details

### 3.2.1 Functions for Tree Edit Distances

```
Tree *make_tree (char *struc)
```

Constructs a [Tree](#) (essentially the postorder list) of the structure 'struc', for use in [tree\\_edit\\_distance\(\)](#).

```
float tree_edit_distance (Tree *T1,
                        Tree *T2)
```

Calculates the edit distance of the two trees.

```
void free_tree(Tree *t)
```

Free the memory allocated for [Tree](#) t.

See also

[dist\\_vars.h](#) and [treedist.h](#) for prototypes and more detailed descriptions

### 3.2.2 Functions for String Alignment

```
swString *Make_swString (char *string)
```

Convert a structure into a format suitable for [string\\_edit\\_distance\(\)](#).

```
float      string_edit_distance (swString *T1,  
                                swString *T2)
```

Calculate the string edit distance of T1 and T2.

See also

[dist\\_vars.h](#) and [stringdist.h](#) for prototypes and more detailed descriptions

### 3.2.3 Functions for Comparison of Base Pair Probabilities

For comparison of base pair probability matrices, the matrices are first condensed into probability profiles which are then compared by alignment.

```
float *Make_bp_profile_bppm ( double *bppm,  
                             int length)
```

condense pair probability matrix into a vector containing probabilities for unpaired, upstream paired and downstream paired.

```
float profile_edit_distance ( const float *T1,  
                             const float *T2)
```

Align the 2 probability profiles T1, T2

.

See also

[ProfileDist.h](#) for prototypes and more details of the above functions

## 3.3 Free Energy of Secondary Structures

A description on how secondary structures are decomposed into individual loops to eventually evaluate their stability in terms of free energy.

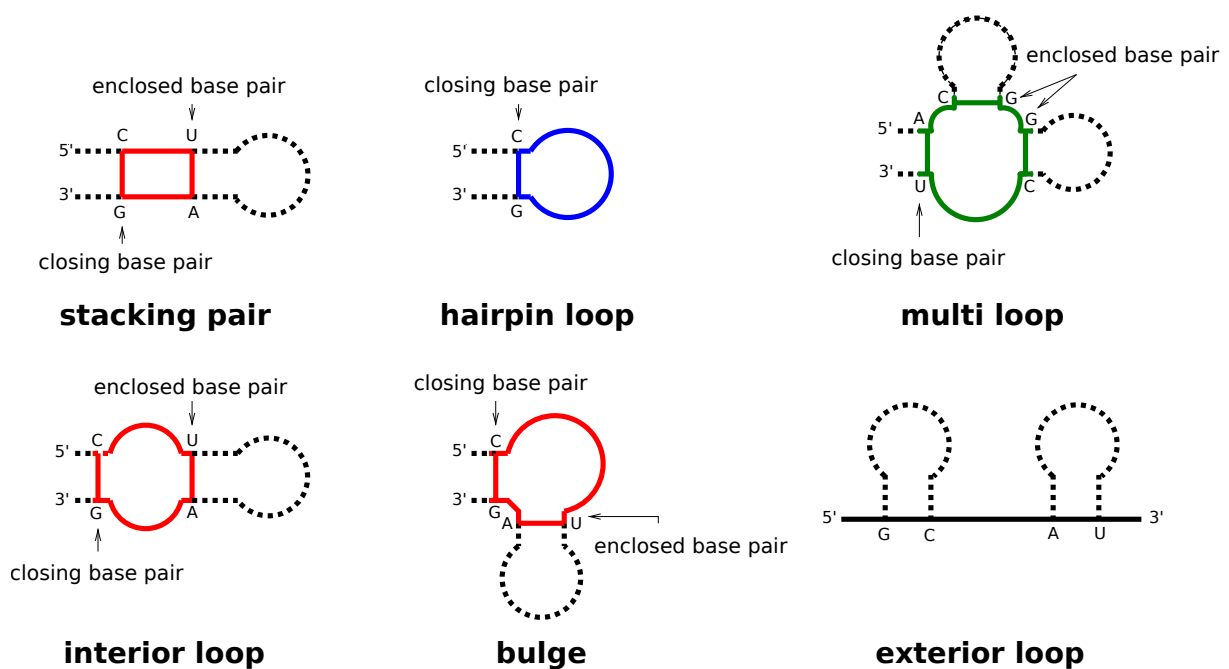


### 3.3.1 Secondary Structure Loop Decomposition

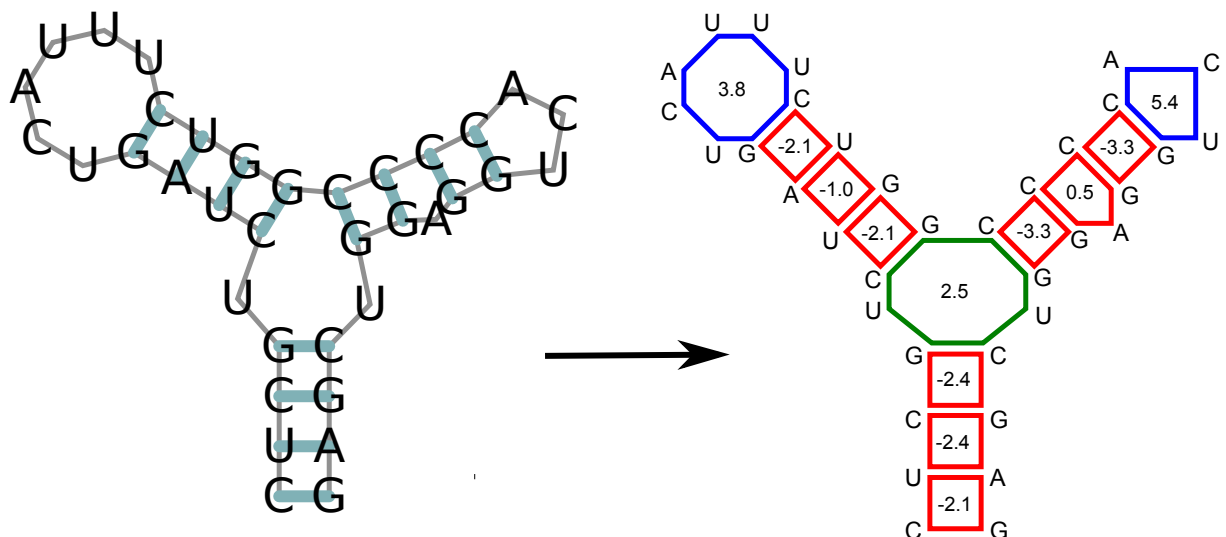
Each base pair in a secondary structure closes a loop, thereby directly enclosing unpaired nucleotides, and/or further base pairs. Our implementation distinguishes four basic types of loops:

- hairpin loops
- interior loops
- multibranch loops
- exterior loop

While the exterior loop is a special case without a closing pair, the other loops are determined by the number of base pairs involved in the loop formation, i.e. hairpin loops are 1-loops, since only a single base pair delimits the loop. interior loops are 2-loops due to their enclosing, and enclosed base pair. All loops where more than two base pairs are involved, are termed multibranch loops.



Any secondary structure can be decomposed into its loops. Each of the loops then can be scored in terms of free energy, and the free energy of an entire secondary structure is simply the sum of free energies of its loops.



### 3.3.1.1 Free Energy Evaluation API

While we implement some functions that decompose a secondary structure into its individual loops, the majority of methods provided in are dedicated to free energy evaluation. The corresponding modules are:

See also

[Free Energy Evaluation](#), [Energy Evaluation for Individual Loops](#)

### 3.3.2 Free Energy Parameters

For secondary structure free energy evaluation we usually utilize the set of Nearest Neighbor Parameters also used in other software, such as *UNAFold* and *RNAstructure*. While the *RNAlib* already contains a compiled-in set of the latest *Turner 2004 Free Energy Parameters*, we defined a file format that allows to change these parameters at runtime. The *ViennaRNA Package* already comes with a set of parameter files containing

- Turner 1999 RNA parameters
- Mathews 1999 DNA parameters
- Andronescu 2007 RNA parameters
- Mathews 2004 DNA parameters

### 3.3.2.1 Free Energy Parameters Modification API

See also

[Energy Parameters](#), [Reading/Writing Energy Parameter Sets from/to File](#)

### 3.3.3 Fine-tuning of the Energy Evaluation Model

See also

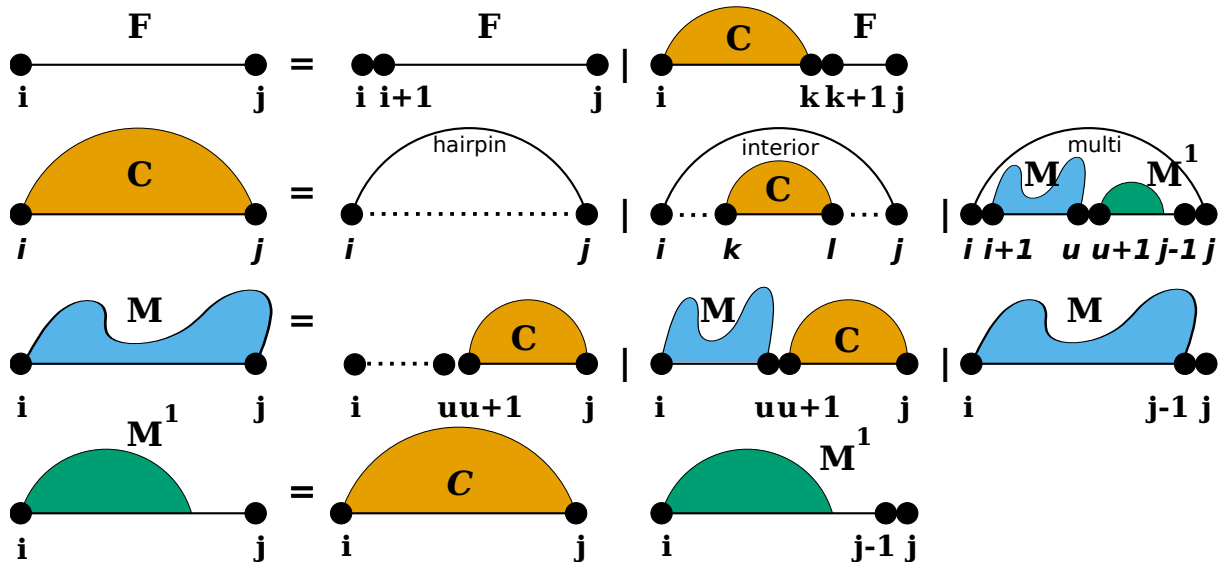
[Fine-tuning of the Implemented Models](#)

## 3.4 Secondary Structure Folding Grammar

A description of the basic grammar to generate secondary structures, used for almost all prediction algorithms in our library and how to modify it.

### 3.4.1 Secondary Structure Folding Recurrences

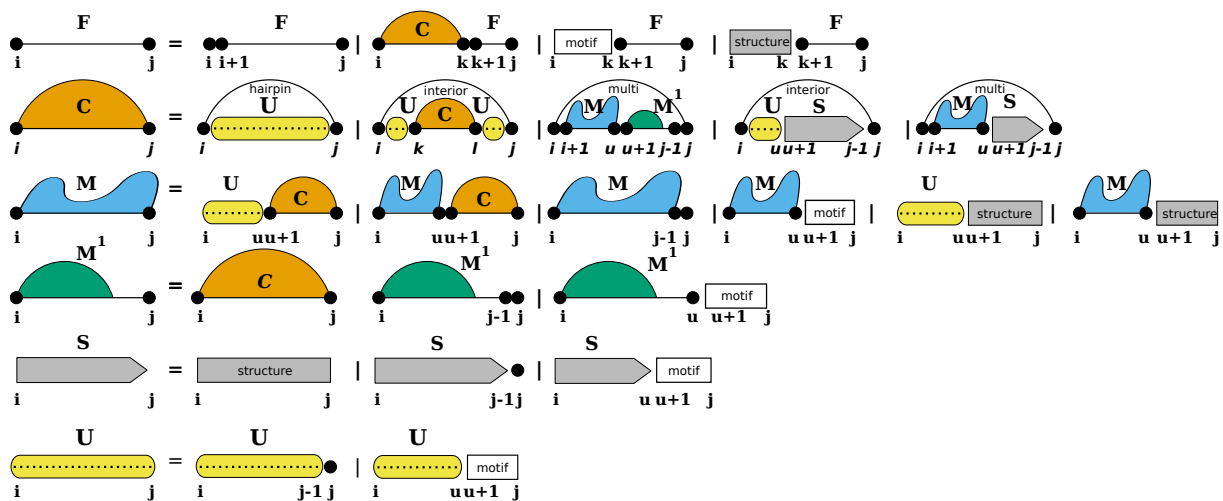
To predict secondary structures composed of the four distinguished loop types introduced before, all algorithms implemented in *RNAlib* follow a specific decomposition scheme, also known as the *RNA folding grammar*, or *Secondary Structure Folding Recurrences*.



However, compared to other RNA secondary structure prediction libraries, our implementation allows for a fine-grained control of the above recursions by constraining both, the individual derivations of the grammar as well as the evaluation of particular loop contributions. Furthermore, we provide a mechanism to extend the above grammar with additional derivation rules, so-called *Domains*.

### 3.4.2 Additional Structural Domains

Some applications of RNA secondary structure prediction require an extension of the *regular RNA folding grammar*. For instance one would like to include proteins and other ligands binding to unpaired loop regions while competing with conventional base pairing. Another application could be that one may want to include the formation of self-enclosed structural modules, such as *G-quadruplexes*. For such applications, we provide a pair of additional domains that extend the regular RNA folding grammar, [Structured Domains](#) and [Unstructured Domains](#).



While unstructured domains are usually determined by a more or less precise sequence motif, e.g. the binding site for a protein, structured domains are considered self-enclosed modules with a more or less complex pairing pattern. Our extension with these two domains introduces two production rules to fill additional dynamic processing matrices *S* and *U* where we store the pre-computed contributions of structured domains (*S*), and unstructured domains (*U*).

### 3.4.2.1 Structured Domains

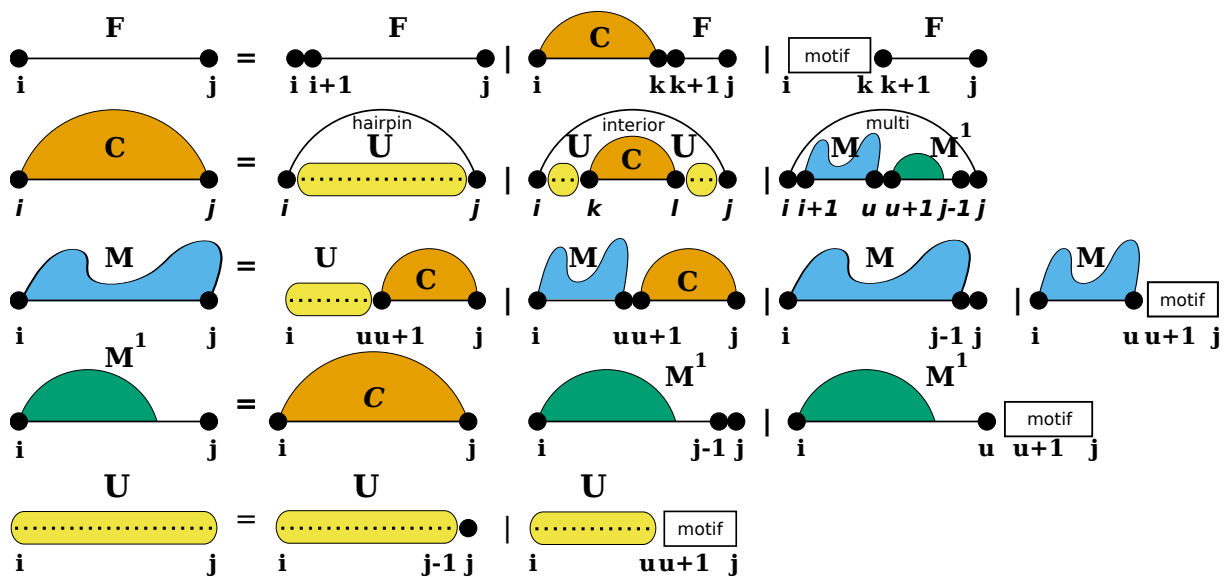
Usually, structured domains represent self-enclosed structural modules that exhibit a more or less complex base pairing pattern. This can be more or less well-defined 3D motifs, such as *G-Quadruplexes*, or loops with additional non-canonical base pair interactions, such as *kink-turns*.

#### Note

Currently, our implementation only provides the specialized case of *G-Quadruplexes*.

### 3.4.2.2 Unstructured Domains

Unstructured domains appear in the production rules of the RNA folding grammar wherever new unpaired nucleotides are attached to a growing substructure (see also [15]):



The white boxes represent the stretch of RNA bound to the ligand and represented by a more or less specific sequence motif. The motif itself is considered unable to form base pairs. The additional production rule  $U$  is used to precompute the contribution of unpaired stretches possibly bound by one or more ligands. The auxiliary DP matrix for this production rule is filled right before processing the other (regular) production rules of the RNA folding grammar.

### 3.4.2.3 Domain Extension API

For the sake of flexibility, each of the domains is associated with a specific data structure serving as an abstract interface to the extension. The interface uses callback functions to

- pre-compute arbitrary data, e.g. filling up additional dynamic programming matrices, and
- evaluate the contribution of a paired or unpaired structural feature of the RNA.

Implementations of these callbacks are separate for regular free energy evaluation, e.g. MFE prediction, and partition function applications. A data structure holding arbitrary data required for the callback functions can be associated to the domain as well. While *RNAlib* comes with a default implementation for structured and unstructured domains, the system is entirely user-customizable.

#### See also

[Unstructured Domains](#), [Structured Domains](#), [G-Quadruplexes](#), [Ligands Binding to Unstructured Domains](#)

### 3.4.3 Constraints on the Folding Grammar

Secondary Structure constraints can be subdivided into two groups:

- Hard Constraints
- Soft Constraints

While Hard-Constraints directly influence the production rules used in the folding recursions by allowing, disallowing, or enforcing certain decomposition steps, Soft-constraints on the other hand are used to change position specific contributions in the recursions by adding bonuses/penalties in form of pseudo free energies to certain loop configurations.

#### Note

Secondary structure constraints are always applied at decomposition level, i.e. in each step of the recursive structure decomposition, for instance during MFE prediction.

#### 3.4.3.1 Hard Constraints API

Hard constraints as implemented in our library can be specified for individual loop types, i.e. the atomic derivations of the RNA folding grammar rules. Hence, the pairing behavior of both, single nucleotides and pairs of bases, can be constrained in every loop context separately. Additionally, an abstract implementation using a callback mechanism allows for full control of more complex hard constraints.

#### See also

[Hard Constraints](#)

#### 3.4.3.2 Soft Constraints API

For the sake of memory efficiency, we do not implement a loop context aware version of soft constraints. The *static* soft constraints as implemented only distinguish unpaired from paired nucleotides. This is usually sufficient for most use-case scenarios. However, similar to hard constraints, an abstract soft constraints implementation using a callback mechanism exists, that allows for any soft constraint that is compatible with the RNA folding grammar. Thus, loop contexts and even individual derivation rules can be addressed separately for maximum flexibility in soft-constraints application.

#### See also

[Soft Constraints, Incorporating Ligands Binding to Specific Sequence/Structure Motifs using Soft Constraints, SHAPE Reactivity Data](#)

## 3.5 RNA Secondary Structure Landscapes

A description of the implicit landscape-like network of structures that appears upon modelling the transition of one structure into another

### 3.5.1 The Neighborhood of a Secondary Structure

### 3.5.2 The Secondary Structure Landscape API

## 3.6 Minimum Free Energy Algorithm(s)

Computing the Minimum Free Energy (MFE), i.e. the most stable conformation in thermodynamic equilibrium

### 3.6.1 Zuker's Algorithm

Our library provides fast dynamic programming Minimum Free Energy (MFE) folding algorithms derived from the decomposition scheme as described by "Zuker & Stiegler (1981)" [27].

### 3.6.2 MFE for circular RNAs

Folding of *circular* RNA sequences is handled as a post-processing step of the forward recursions. See [12] for further details.

### 3.6.3 MFE Algorithm API

We provide interfaces for the prediction of

- MFE and corresponding secondary structure for single sequences,
- consensus MFE structures of sequence alignments, and
- MFE structure for two hybridized RNA strands

See also

[Minimum Free Energy \(MFE\) Algorithms](#), `consensus_mfe_fold`, `mfe_cofold`, [Computing MFE representatives of a Distance Base](#)

## 3.7 Partition Function and Equilibrium Probability Algorithm(s)

### 3.7.1 Equilibrium Ensemble Statistics

In contrast to methods that compute the property of a single structure in the ensemble, e.g. [Minimum Free Energy Algorithm\(s\)](#), the partition function algorithms always consider the entire equilibrium ensemble. For that purpose, the McCaskill algorithm [18] and its variants can be used to efficiently compute

- the partition function, and from that
- various equilibrium probabilities, for instance base pair probabilities, probabilities of individual structure motifs, and many more.

The principal idea behind this approach is that in equilibrium, statistical mechanics and polymer theory tells us that the frequency or probability  $p(s)$  of a particular state  $s$  depends on its energy  $E(s)$  and follows a Boltzmann distribution, i.e.

$$p(s) \propto e^{-\beta E(s)} \text{ with } \beta = \frac{1}{kT}$$

where  $k \approx 1.987 \cdot 10^{-3} \frac{\text{kcal}}{\text{mol K}}$  is the Boltzmann constant, and  $T$  the thermodynamic temperature. From that relation, the actual probability of state  $s$  can then be obtained using a proper scaling factor, the *canonical partition function*

$$Z = \sum_{s \in \Omega} e^{-\beta E(s)}$$

where  $\Omega$  is the finite set of all states. Finally, the equilibrium probability of state  $s$  can be computed as

$$p(s) = \frac{e^{-\beta E(s)}}{Z}$$

Instead of enumerating all states exhaustively to compute  $Z$  one can apply the [Secondary Structure Folding Recurrences](#) again for an efficient computation in cubic time. An *outside* variant of the same recursions is then used to compute probabilities for base pairs, stretches of consecutive unpaired nucleotides, or structural motifs.

See also

Further details of the Partition function and Base Pair Probability algorithm can be obtained from McCaskill 1990 [18]

### 3.7.2 Partition Function and Equilibrium Probability API

We implement a wide variety of variants of the partition function algorithm according to McCaskill 1990 [18]. See the corresponding submodules for specific implementation details.

See also

[Partition Function and Equilibrium Properties](#), [consensus\\_pf\\_fold](#), [Partition Function for Two Hybridized Sequences](#), [Partition Function for two Hybridized Sequences as a Stepwise Process](#), [local\\_pf\\_fold](#), [Computing Partition Functions of a Distribution](#)

## 3.8 Suboptimals and (other) Representative Structures

### 3.8.1 Suboptimal Secondary Structures

### 3.8.2 Sampling Secondary Structures from the Ensemble

### 3.8.3 Structure Enumeration and Sampling API

See also

[Suboptimal Structures sensu Stiegler et al. 1984 / Zuker et al. 1989](#), [Suboptimal Structures within an Energy Band around the Minimum Free Energy Structure](#), [Random Structure Samples from the Ensemble](#), [Compute the Structure with Maximum Expected Accuracy \(MEA\)](#), [Compute the Centroid Structure](#)

## 3.9 RNA-RNA Interaction

### 3.9.1 rip\_intro

The function of an RNA molecule often depends on its interaction with other RNAs. The following routines therefore allows one to predict structures formed by two RNA molecules upon hybridization.

### 3.9.2 Concatenating RNA sequences

One approach to co-folding two RNAs consists of concatenating the two sequences and keeping track of the concatenation point in all energy evaluations. Correspondingly, many of the `cofold()` and `co_pf_fold()` routines take one sequence string as argument and use the the global variable `cut_point` to mark the concatenation point. Note that while the *RNAcofold* program uses the `'&'` character to mark the chain break in its input, you should not use an `'&'` when using the library routines (set `cut_point` instead).

### 3.9.3 RNA-RNA interaction as a Stepwise Process

In a second approach to co-folding two RNAs, cofolding is seen as a stepwise process. In the first step the probability of an unpaired region is calculated and in a second step this probability of an unpaired region is multiplied with the probability of an interaction between the two RNAs. This approach is implemented for the interaction between a long target sequence and a short ligand RNA. Function `pf_unstru()` calculates the partition function over all unpaired regions in the input sequence. Function `pf_interact()`, which calculates the partition function over all possible interactions between two sequences, needs both sequence as separate strings as input.

### 3.9.4 RNA-RNA Interaction API

## 3.10 Locally Stable Secondary Structures

### 3.10.1 local\_intro

### 3.10.2 local\_mfe

### 3.10.3 local\_pf

### 3.10.4 Locally Stable Secondary Structure API

## 3.11 Comparative Structure Prediction

### 3.11.1 Incorporate Evolutionary Information

Consensus structures can be predicted by a modified version of the `fold()` algorithm that takes a set of aligned sequences instead of a single sequence. The energy function consists of the mean energy averaged over the sequences, plus a covariance term that favors pairs with consistent and compensatory mutations and penalizes pairs that cannot be formed by all structures. For details see [10] and [1].



### 3.11.2 Comparative Structure Prediction API

## 3.12 Classified DP variations

### 3.12.1 The Idea of Classified Dynamic Programming

Usually, thermodynamic properties using the basic recursions for [Minimum Free Energy Algorithm\(s\)](#), [Partition Function and Equilibrium](#) and so forth, are computed over the entire structure space. However, sometimes it is desired to partition the structure space *a priori* and compute the above properties for each of the resulting partitions. This approach directly leads to *Classified Dynamic Programming*.

### 3.12.2 Distance Class Partitioning

The secondary structure space is divided into partitions according to the base pair distance to two given reference structures and all relevant properties are calculated for each of the resulting partitions.

See also

For further details, we refer to Lorenz et al. 2009 [[14](#)]

### 3.12.3 Density of States (DOS)

### 3.12.4 Classified DP API

## 3.13 RNA Sequence Design

### 3.13.1 Generate Sequences that fold into particular Secondary Structures

### 3.13.2 RNA Sequence Design API

See also

[Inverse Folding \(Design\)](#)

## 3.14 Experimental Structure Probing Data

### 3.14.1 Guide the Structure Prediction using Experimental Data

#### 3.14.1.1 SHAPE reactivities

### 3.14.2 Structure Probing Data API

See also

[Experimental Structure Probing Data](#), [SHAPE Reactivity Data](#), [perturbation](#)

## 3.15 Ligand Binding

### 3.15.1 Small Molecules and Proteins that bind to specific RNA Structures

#### 3.15.2 `ligand_binding_api`

In our library, we provide two different ways to incorporate ligand binding to RNA structures:

- [Ligands Binding to Unstructured Domains](#), and
- [Incorporating Ligands Binding to Specific Sequence/Structure Motifs using Soft Constraints](#)

The first approach is implemented as an actual extension of the folding grammar. It adds auxiliary derivation rules for each case when consecutive unpaired nucleotides are evaluated. Therefore, this model is applicable to ligand binding to any loop context.

The second approach, on the other hand, uses the soft-constraints feature to change the energy evaluation of hairpin- or interior-loops. Hence, it can only be applied when a ligand binds to a hairpin-like, or interior-loop like motif.

See also

[Ligands Binding to Unstructured Domains](#), [Incorporating Ligands Binding to Specific Sequence/Structure Motifs using Soft Constraints](#)

## 3.16 (Tertiary) Structure Motifs

### 3.16.1 Incorporating Higher-Order (Tertiary) Structure Motifs

#### 3.16.2 RNA G-Quadruplexes

#### 3.16.3 (Tertiary) Structure Motif API

## Chapter 4

# I/O Formats

Below, you'll find a listing of different sections that introduce the most common notations of sequence and structure data, specifications of bioinformatics sequence and structure file formats, and various output file formats produced by our library.

- [RNA Structure Notations](#) describes the different notations and representations of RNA secondary structures
- [File Formats](#) gives an overview of the file formats compatible with our library
- [Plotting](#) shows the different (PostScript) plotting functions for RNA secondary structures, feature probabilities, and multiple sequence alignments

### 4.1 RNA Structure Notations

#### 4.1.1 Representations of Secondary Structures

The standard representation of a secondary structure in our library is the [Dot-Bracket Notation \(a.k.a. Dot-Parenthesis Notation\)](#), where matching brackets symbolize base pairs and unpaired bases are shown as dots. Based on that notation, more elaborate representations have been developed to include additional information, such as the loop context a nucleotide belongs to and to annotated pseudo-knots.

See also

[Extended Dot-Bracket Notation, Washington University Secondary Structure \(WUSS\) notation](#)

#### 4.1.2 Dot-Bracket Notation (a.k.a. Dot-Parenthesis Notation)

The Dot-Bracket notation as introduced already in the early times of the ViennaRNA Package denotes base pairs by matching pairs of parenthesis ( ) and unpaired nucleotides by dots . .

Example: A simple helix of size 4 enclosing a hairpin of size 4 is annotated as

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

See also

[vrna\\_ptable\\_from\\_string\(\)](#), [vrna\\_db\\_flatten\(\)](#), [vrna\\_db\\_flatten\\_to\(\)](#)

### 4.1.3 Extended Dot-Bracket Notation

A more generalized version of the original Dot-Bracket notation may use additional pairs of brackets, such as <>, {}, and [], and matching pairs of uppercase/lowercase letters. This allows for annotating pseudo-knots, since different pairs of brackets are not required to be nested.

Example: The following annotations of a simple structure with two crossing helices of size 4 are equivalent:

```
<<<<[[[...>>>]]]
(((AAAA...)))aaaa
AAAA{ {{...aaaa}} }
```

See also

[vrna\\_ptable\\_from\\_string\(\)](#), [vrna\\_db\\_flatten\(\)](#), [vrna\\_db\\_flatten\\_to\(\)](#)

### 4.1.4 Washington University Secondary Structure (WUSS) notation

The WUSS notation, as frequently used for consensus secondary structures in [Stockholm 1.0 format](#) allows for a fine-grained annotation of base pairs and unpaired nucleotides, including pseudo-knots.

Below, you'll find a list of secondary structure elements and their corresponding WUSS annotation (See also the [infernal user guide at http://eddylab.org/infernal/Userguide.pdf](http://eddylab.org/infernal/Userguide.pdf))

- **Base pairs**

Nested base pairs are annotated by matching pairs of the symbols <>, (), {}, and []. Each of the matching pairs of parenthesis have their special meaning, however, when used as input in our programs, e.g. structure constraint, these details are usually ignored. Furthermore, base pairs that constitute as pseudo-knot are denoted by letters from the latin alphabet and are, if not denoted otherwise, ignored entirely in our programs.

- **Hairpin loops**

Unpaired nucleotides that constitute the hairpin loop are indicated by underscores, \_.

Example:

```
<<<<_____>>>>
```

- **Bulges and interior loops**

Residues that constitute a bulge or interior loop are denoted by dashes, -.

Example:

```
(((--<_____>-)) )
```

- **Multibranch loops**

Unpaired nucleotides in multibranch loops are indicated by commas, ,.

Example:

```
(( ( , <_____> , <_____> ) ) )
```

- **External residues**

Single stranded nucleotides in the exterior loop, i.e. not enclosed by any other pair are denoted by colons, :.

Example:

```
<<<_____>>>:::
```

- **Insertions**

In cases where an alignment represents the consensus with a known structure, insertions relative to the known structure are denoted by periods, .. Regions where local structural alignment was invoked, leaving regions of both target and query sequence unaligned, are indicated by tildes, ~.

**Note**

These symbols only appear in alignments of a known (query) structure annotation to a target sequence of unknown structure.

- **Pseudo-knots**

The WUSS notation allows for annotation of pseudo-knots using pairs of upper-case/lower-case letters.

**Note**

Our programs and library functions usually ignore pseudo-knots entirely treating them as unpaired nucleotides, if not stated otherwise.

Example:

```
<<<_AAA____>>>aaa
```

See also

[vrna\\_db\\_from\\_WUSS\(\)](#)

### 4.1.5 Tree Representations of Secondary Structures

Alternatively, one may find representations with two types of node labels, 'P' for paired and 'U' for unpaired; a dot is then replaced by '(U)', and each closed bracket is assigned an additional identifier 'P'. We call this the expanded notation. In [8] a condensed representation of the secondary structure is proposed, the so-called homeomorphically irreducible tree (HIT) representation. Here a stack is represented as a single pair of matching brackets labeled 'P' and weighted by the number of base pairs. Correspondingly, a contiguous strain of unpaired bases is shown as one pair of matching brackets labeled 'U' and weighted by its length. Generally any string consisting of matching brackets and identifiers is equivalent to a plane tree with as many different types of nodes as there are identifiers.

Bruce Shapiro proposed a coarse grained representation [20], which, does not retain the full information of the secondary structure. He represents the different structure elements by single matching brackets and labels them as

- H (hairpin loop),
- I (interior loop),
- B (bulge),
- M (multi-loop), and
- S (stack).

We extend his alphabet by an extra letter for external elements E. Again these identifiers may be followed by a weight corresponding to the number of unpaired bases or base pairs in the structure element. All tree representations (except for the dot-bracket form) can be encapsulated into a virtual root (labeled R).

The following example illustrates the different linear tree representations used by the package:

Consider the secondary structure represented by the dot-bracket string (full tree)

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

which is the most convenient condensed notation used by our programs and library functions.

Then, the following tree representations are equivalent:

- Expanded tree:

```
((U) ((U) (U) (((U) (U) (U) P) P) P) (U) (U) ((U) (U) P) P) P) (U) R)
```

- HIT representation (Fontana et al. 1993 [8]):

```
((U1) ((U2) ((U3) P3) (U2) ((U2) P2) P2) (U1) R)
```

- Coarse Grained **Tree** Representation (Shapiro 1988 [20]):

- Short (with root node R, without stem nodes S):

```
((H) ((H) M) R)
```

- Full (with root node R):

```
(((((H) S) ((H) S) M) S) R)
```

- Extended (with root node R, with external nodes E):

```
(((((H) S) ((H) S) M) S) E) R)
```

- Weighted (with root node R, with external nodes E):

```
(((((H3) S3) ((H2) S2) M4) S2) E2) R)
```

The Expanded tree is rather clumsy and mostly included for the sake of completeness. The different versions of Coarse Grained **Tree** Representations are variations of Shapiro's linear tree notation.

For the output of aligned structures from string editing, different representations are needed, where we put the label on both sides. The above examples for tree representations would then look like:

- (UU) (P (P (P (P (UU) (UU) (P (P (P (UU) (UU) (UU) P) P) P) (UU) (UU) (P (P (UU) (U . . .
- (UU) (P2 (P2 (U2U2) (P2 (U3U3) P3) (U2U2) (P2 (U2U2) P2) P2) (UU) P2) (UU)
- (B (M (HH) (HH) M) B)  
(S (B (S (M (S (HH) S) (S (HH) S) M) S) B) S)  
(E (S (B (S (M (S (HH) S) (S (HH) S) M) S) B) S) E)
- (R (E2 (S2 (B1 (S2 (M4 (S3 (H3) S3) ((H2) S2) M4) S2) B1) S2) E2) R)

Aligned structures additionally contain the gap character '\_'.

#### 4.1.6 Examples for Structure Parsing and Conversion

##### 4.1.7 Structure Parsing and Conversion API

Several functions are provided for parsing structures and converting to different representations.

```
char *expand_Full(const char *structure)
```

Convert the full structure from bracket notation to the expanded notation including root.

```
char *b2HIT (const char *structure)
```

Converts the full structure from bracket notation to the HIT notation including root.

```
char *b2C (const char *structure)
```

Converts the full structure from bracket notation to the a coarse grained notation using the 'H' 'B' 'I' 'M' and 'R' identifiers.

```
char *b2Shapiro (const char *structure)
```

Converts the full structure from bracket notation to the *weighted* coarse grained notation using the 'H' 'B' 'I' 'M' 'S' 'E' and 'R' identifiers.

```
char *expand_Shapiro (const char *coarse);
```

Inserts missing 'S' identifiers in unweighted coarse grained structures as obtained from [b2C\(\)](#).

```
char *add_root (const char *structure)
```

Adds a root to an un-rooted tree in any except bracket notation.

```
char *unexpand_Full (const char *ffull)
```

Restores the bracket notation from an expanded full or HIT tree, that is any tree using only identifiers 'U' 'P' and 'R'.

```
char *unweight (const char *wcoarse)
```

Strip weights from any weighted tree.

```
void unexpand_aligned_F (char *align[2])
```

Converts two aligned structures in expanded notation.

```
void parse_structure (const char *structure)
```

Collects a statistic of structure elements of the full structure in bracket notation.

See also

[RNAstruct.h](#) for prototypes and more detailed description

## 4.2 File Formats

### 4.2.1 File formats for Multiple Sequence Alignments (MSA)

#### 4.2.1.1 ClustalW format

The *ClustalW* format is a relatively simple text file containing a single multiple sequence alignment of DNA, RNA, or protein sequences. It was first used as an output format for the *clustalw* programs, but nowadays it may also be generated by various other sequence alignment tools. The specification is straight forward:

- The first line starts with the words

```
CLUSTAL W
```

or

```
CLUSTALW
```

- After the above header there is at least one empty line
- Finally, one or more blocks of sequence data are following, where each block is separated by at least one empty line

Each line in a blocks of sequence data consists of the sequence name followed by the sequence symbols, separated by at least one whitespace character. Usually, the length of a sequence in one block does not exceed 60 symbols. Optionally, an additional whitespace separated cumulative residue count may follow the sequence symbols. Optionally, a block may be followed by a line depicting the degree of conservation of the respective alignment columns.

#### Note

Sequence names and the sequences must not contain whitespace characters! Allowed gap symbols are the hyphen ("-"), and dot (".").

#### Warning

Please note that many programs that output this format tend to truncate the sequence names to a limited number of characters, for instance the first 15 characters. This can destroy the uniqueness of identifiers in your MSA.

Here is an example alignment in ClustalW format:

```
CLUSTAL W (1.83) multiple sequence alignment
```

```
AL031296.1/85969-86120      CUGCCUCACAAACGUUUGUGCCUCAGUUACCCGUAGAUGUAGUGAGGGUAACAAUACUUAC
AANU01225121.1/438-603     CUGCCUCACAAACAUUUGUGCCUCAGUUACUAGUAGUAGUGAGGGUGACAAUACUUAC
AAWR02037329.1/29294-29150 ---CUCGACACCACU---GCCUCGGUUAACCAUCGGUGCAGUGCGGGUAGUAGUACCAAU

AL031296.1/85969-86120      UCUCGUUGGUGAUAAAGGAACAGCU
AANU01225121.1/438-603     UCUCGUUGGUGAUAAAGGAACAGCU
AAWR02037329.1/29294-29150 GCUAAUUAGUUGUGAGGACCAACU
```





#### 4.2.1.4 MAF format

The multiple alignment format (MAF) is usually used to store multiple alignments on DNA level between entire genomes. It consists of independent blocks of aligned sequences which are annotated by their genomic location. Consequently, an MAF formatted MSA file may contain multiple records. MAF files start with a line

```
##maf
```

which is optionally extended by whitespace delimited key=value pairs. Lines starting with the character ("#") are considered comments and usually ignored.

A MAF block starts with character ("a") at the beginning of a line, optionally followed by whitespace delimited key=value pairs. The next lines start with character ("s") and contain sequence information of the form

```
s src start size strand srcSize sequence
```

where

- *src* is the name of the sequence source
- *start* is the start of the aligned region within the source (0-based)
- *size* is the length of the aligned region without gap characters
- *strand* is either ("+") or ("-"), depicting the location of the aligned region relative to the source
- *srcSize* is the size of the entire sequence source, e.g. the full chromosome
- *sequence* is the aligned sequence including gaps depicted by the hyphen ("-")

Here is an example alignment in MAF format (bluntly taken from the [UCSC Genome browser website](#)):

```
##maf version=1 scoring=tba.v8
# tba.v8 ((human chimp) baboon) (mouse rat))
# multiz.v7
# maf_project.v5 _tba_right.maf3 mouse _tba_C
# single_cov2.v4 single_cov2 /dev/stdin

a score=23262.0
s hg16.chr7 27578828 38 + 158545518 AAA-GGGAATGTTAACCAAATGA---ATTGTCTCTTACGGTG
s panTro1.chr6 28741140 38 + 161576975 AAA-GGGAATGTTAACCAAATGA---ATTGTCTCTTACGGTG
s baboon 116834 38 + 4622798 AAA-GGGAATGTTAACCAAATGA---GTTGTCTCTTATGGTG
s mm4.chr6 53215344 38 + 151104725 -AATGGGAATGTTAAGCAAACGA---ATTGTCTCTCAGTGTG
s rn3.chr4 81344243 40 + 187371129 -AA-GGGGATGCTAAGCCAATGAGTTGTTGTCTCTCAATGTG

a score=5062.0
s hg16.chr7 27699739 6 + 158545518 TAAAGA
s panTro1.chr6 28862317 6 + 161576975 TAAAGA
s baboon 241163 6 + 4622798 TAAAGA
s mm4.chr6 53303881 6 + 151104725 TAAAGA
s rn3.chr4 81444246 6 + 187371129 taagga

a score=6636.0
s hg16.chr7 27707221 13 + 158545518 gcagctgaaaaca
s panTro1.chr6 28869787 13 + 161576975 gcagctgaaaaca
s baboon 249182 13 + 4622798 gcagctgaaaaca
s mm4.chr6 53310102 13 + 151104725 ACAGCTGAAAATA
```

## 4.2.2 File formats to manipulate the RNA folding grammar

### 4.2.2.1 Command Files

The RNAlib and many programs of the ViennaRNA Package can parse and apply data from so-called command files. These commands may refer to structure constraints or even extensions of the RNA folding grammar (such as [Unstructured Domains](#)). Commands are given as a line of whitespace delimited data fields. The syntax we use extends the constraint definitions used in the `mfold` / `UNAFold` software, where each line begins with a command character followed by a set of positions.

However, we introduce several new commands, and allow for an optional loop type context specifier in form of a sequence of characters, and an orientation flag that enables one to force a nucleotide to pair upstream, or downstream.

#### 4.2.2.1.1 Constraint commands

The following set of commands is recognized:

- F ... Force
- P ... Prohibit
- C ... Conflicts/Context dependency
- A ... Allow (for non-canonical pairs)
- E ... Soft constraints for unpaired position(s), or base pair(s)

#### 4.2.2.1.2 RNA folding grammar extensions

- UD ... Add ligand binding using the [Unstructured Domains](#) feature

#### 4.2.2.1.3 Specification of the loop type context

The optional loop type context specifier [LOOP] may be a combination of the following:

- E ... Exterior loop
- H ... Hairpin loop
- I ... Interior loop
- M ... Multibranch loop
- A ... All loops

For structure constraints, we additionally allow one to address base pairs enclosed by a particular kind of loop, which results in the specifier [WHERE] which consists of [LOOP] plus the following character:

- i ... enclosed pair of an Interior loop
- m ... enclosed pair of a Multibranch loop

If no [LOOP] or [WHERE] flags are set, all contexts are considered (equivalent to A )

#### 4.2.2.1.4 Controlling the orientation of base pairing

For particular nucleotides that are forced to pair, the following [ORIENTATION] flags may be used:

- U ... Upstream
- D ... Downstream

If no [ORIENTATION] flag is set, both directions are considered.

#### 4.2.2.1.5 Sequence coordinates

Sequence positions of nucleotides/base pairs are 1-based and consist of three positions  $i$ ,  $j$ , and  $k$ . Alternatively, four positions may be provided as a pair of two position ranges  $[i : j]$ , and  $[k : l]$  using the '-' sign as delimiter within each range, i.e.  $i - j$ , and  $k - l$ .

#### 4.2.2.1.6 Valid constraint commands

Below are resulting general cases that are considered *valid* constraints:

##### 1. "Forcing a range of nucleotide positions to be paired":

Syntax:

```
F i 0 k [WHERE] [ORIENTATION]
```

Description:

Enforces the set of  $k$  consecutive nucleotides starting at position  $i$  to be paired. The optional loop type specifier [WHERE] allows to force them to appear as closing/enclosed pairs of certain types of loops.

##### 2. "Forcing a set of consecutive base pairs to form":

Syntax:

```
F i j k [WHERE]
```

Description:

Enforces the base pairs  $(i, j), \dots, (i + (k - 1), j - (k - 1))$  to form. The optional loop type specifier [WHERE] allows to specify in which loop context the base pair must appear.

##### 3. "Prohibiting a range of nucleotide positions to be paired":

Syntax:

```
P i 0 k [WHERE]
```

Description:

Prohibit a set of  $k$  consecutive nucleotides to participate in base pairing, i.e. make these positions unpaired. The optional loop type specifier [WHERE] allows to force the nucleotides to appear within the loop of specific types.

##### 4. "Prohibiting a set of consecutive base pairs to form":

Syntax:

```
P i j k [WHERE]
```

Description:

Prohibit the base pairs  $(i, j), \dots, (i + (k - 1), j - (k - 1))$  to form. The optional loop type specifier [WHERE] allows to specify the type of loop they are disallowed to be the closing or an enclosed pair of.

5. **"Prohibiting two ranges of nucleotides to pair with each other":**

Syntax:

```
P i-j k-l [WHERE]
```

Description:

Prohibit any nucleotide  $p \in [i : j]$  to pair with any other nucleotide  $q \in [k : l]$ . The optional loop type specifier [WHERE] allows to specify the type of loop they are disallowed to be the closing or an enclosed pair of.

6. **"Enforce a loop context for a range of nucleotide positions":**

Syntax:

```
C i 0 k [WHERE]
```

Description:

This command enforces nucleotides to be unpaired similar to *prohibiting* nucleotides to be paired, as described above. It too marks the corresponding nucleotides to be unpaired, however, the [WHERE] flag can be used to enforce specific loop types the nucleotides must appear in.

7. **"Remove pairs that conflict with a set of consecutive base pairs":**

Syntax:

```
C i j k
```

Description:

Remove all base pairs that conflict with a set of consecutive base pairs  $(i, j), \dots, (i + (k - 1), j - (k - 1))$ . Two base pairs  $(i, j)$  and  $(p, q)$  conflict with each other if  $i < p < j < q$ , or  $p < i < q < j$ .

8. **"Allow a set of consecutive (non-canonical) base pairs to form":**

Syntax:

```
A i j k [WHERE]
```

Description:

This command enables the formation of the consecutive base pairs  $(i, j), \dots, (i + (k - 1), j - (k - 1))$ , no matter if they are *canonical*, or *non-canonical*. In contrast to the above F and W commands, which remove conflicting base pairs, the A command does not. Therefore, it may be used to allow *non-canonical* base pair interactions. Since the RNAlib does not contain free energy contributions  $E_{ij}$  for non-canonical base pairs  $(i, j)$ , they are scored as the *maximum* of similar, known contributions. In terms of a *Nussinov* like scoring function the free energy of non-canonical base pairs is therefore estimated as

$$E_{ij} = \min \left[ \max_{(i,k) \in \{GC,CG,AU,UA,GU,UG\}} E_{ik}, \max_{(k,j) \in \{GC,CG,AU,UA,GU,UG\}} E_{kj} \right].$$

The optional loop type specifier [WHERE] allows to specify in which loop context the base pair may appear.

9. **"Apply pseudo free energy to a range of unpaired nucleotide positions":**

Syntax:

```
E i 0 k e
```

Description:

Use this command to apply a pseudo free energy of  $e$  to the set of  $k$  consecutive nucleotides, starting at position  $i$ . The pseudo free energy is applied only if these nucleotides are considered unpaired in the recursions, or evaluations, and is expected to be given in *kcal/mol*.

#### 10. "Apply pseudo free energy to a set of consecutive base pairs":

Syntax

```
E i j k e
```

Use this command to apply a pseudo free energy of  $e$  to the set of base pairs  $(i, j), \dots, (i + (k - 1), j - (k - 1))$ . Energies are expected to be given in *kcal/mol*.

#### 4.2.2.1.7 Valid domain extensions commands

##### 1. "Add ligand binding to unpaired motif (a.k.a. unstructured domains)":

Syntax:

```
UD m e [LOOP]
```

Description:

Add ligand binding to unpaired sequence motif  $m$  (given in IUPAC format, capital letters) with binding energy  $e$  in particular loop type(s).

Example:

```
UD AAA -5.0 A
```

The above example applies a binding free energy of  $-5\text{kcal/mol}$  for a motif AAA that may be present in all loop types.

## 4.3 Plotting

Create Plots of Secondary Structures, Feature Motifs, and Sequence Alignments

### 4.3.1 Producing secondary structure graphs

```
int PS_rna_plot ( char *string,
                  char *structure,
                  char *file)
```

Produce a secondary structure graph in PostScript and write it to 'filename'.

```
int PS_rna_plot_a (
    char *string,
    char *structure,
    char *file,
    char *pre,
    char *post)
```

Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename'.

```
int gmlRNA (char *string,
            char *structure,
            char *ssfile,
            char option)
```

Produce a secondary structure graph in Graph Meta Language (gml) and write it to a file.

```
int ssv_rna_plot (char *string,
                 char *structure,
                 char *ssfile)
```

Produce a secondary structure graph in SStructView format.

```
int svg_rna_plot (char *string,
                 char *structure,
                 char *ssfile)
```

Produce a secondary structure plot in SVG format and write it to a file.

```
int xrna_plot ( char *string,
                char *structure,
                char *ssfile)
```

Produce a secondary structure plot for further editing in XRNA.

```
int rna_plot_type
```

Switch for changing the secondary structure layout algorithm.

Two low-level functions provide direct access to the graph lauyouting algorithms:

```
int simple_xy_coordinates ( short *pair_table,
                           float *X,
                           float *Y)
```

Calculate nucleotide coordinates for secondary structure plot the *Simple way*

```
int naview_xy_coordinates ( short *pair_table,
                           float *X,
                           float *Y)
```

See also

[PS\\_dot.h](#) and [naview.h](#) for more detailed descriptions.

### 4.3.2 Producing (colored) dot plots for base pair probabilities

```
int PS_color_dot_plot ( char *string,
                       cpair *pi,
                       char *filename)
```

```
int PS_color_dot_plot_turn (char *seq,
                           cpair *pi,
                           char *filename,
                           int winSize)
```

```
int PS_dot_plot_list (char *seq,
                     char *filename,
                     plist *pl,
                     plist *mf,
                     char *comment)
```

Produce a postscript dot-plot from two pair lists.

```
int PS_dot_plot_turn (char *seq,
                     struct plist *pl,
                     char *filename,
                     int winSize)
```

See also

[PS\\_dot.h](#) for more detailed descriptions.

### 4.3.3 Producing (colored) alignments

```
int PS_color_aln (  
    const char *structure,  
    const char *filename,  
    const char *seqs[],  
    const char *names[])
```

Produce PostScript sequence alignment color-annotated by consensus structure.



## Chapter 5

# Basic Data Structures

- [Sequence and Structure Data](#) shows the most common types for sequence or structure data
- [The 'Fold Compound'](#) is the basic, central container for our implementations of prediction-, evaluation, and other algorithms
- [Model Details](#) provides the means to store the different model parameters

### 5.1 Sequence and Structure Data

See also

[Secondary Structure Utilities](#)

### 5.2 The 'Fold Compound'

See also

[The Fold Compound](#)

### 5.3 Model Details

See also

[Fine-tuning of the Implemented Models](#)



## Chapter 6

# API Features

- [RNAlib API v3.0](#)
- [Callback Functions](#)
- [Scripting Language interface\(s\)](#)

### 6.1 RNAlib API v3.0

#### 6.1.1 Introduction

With version 2.2 we introduce the new API that will take over the old one in the future version 3.0. By then, backwards compatibility will be broken, and third party applications using RNAlib need to be ported. This switch of API became necessary, since many new features found their way into the RNAlib where a balance between threadsafety and easy-to-use library functions is hard or even impossible to establish. Furthermore, many old functions of the library are present as slightly modified copies of themselves to provide a crude way to overload functions.

Therefore, we introduce the new v3.0 API very early in our development stage such that developers have enough time to migrate to the new functions and interfaces. We also started to provide encapsulation of the RNAlib functions, data structures, typedefs, and macros by prefixing them with *vrna\_* and *VRNA\_*, respectively. Header files should also be included using the *ViennaRNA/*namespace, e.g.

```
#include <ViennaRNA/fold.h>
```

instead of just using

```
#include <fold.h>
```

as required for RNAlib 1.x and 2.x.

This eases the work for programmers of third party applications that would otherwise need to put much effort into renaming functions and data types in their own implementations if their names appear in our library. Since we still provide backward compatibility up to the last version of RNAlib 2.x, this advantage may be fully exploited only starting from v3.0 which will be released in the future. However, our plan is to provide the possibility for an early switch-off mechanism of the backward compatibility in one of our next releases of ViennaRNA Package 2.x.

### 6.1.2 What are the major changes?

...

### 6.1.3 How to port your program to the new API

...

### 6.1.4 Some Examples using RNAlib API v3.0

Examples on how to use the new v3.0 API can be found in the `examples_c_new_API` section.

## 6.2 Callback Functions

With the new [RNAlib API v3.0](#) we introduce so-called callback mechanisms for several functions.

### 6.2.1 The purpose of Callback mechanisms

Using callback mechanisms, our library enables users not only to retrieve computed data without the need for parsing complicated data structures, but also allows one to tweak our implementation to do additional tasks without the requirement of a re-implementation of basic algorithms.

Our implementation of the callback mechanisms always follows the same scheme: The user:

- defines a function that complies with the interface we've defined, and
- passes a pointer to said function to our implementations

In addition to the specific arguments of our callback interfaces, virtually all callbacks receive an additional *pass-through-pointer* as their last argument. This enables one to:

- encapsulate data, and
- provide thread-safe operations,

since this pointer is simply passed through by our library functions. It may therefore hold the address of an arbitrary, user-defined data structure.

### 6.2.2 List of available Callbacks

Below, you find an enumeration of the individual callback functions that are available in *RNAlib*.

**Global `vrna_callback_free_auxdata` (void \*data)**

This callback is supposed to free memory occupied by an auxiliary data structure. It will be called when the `vrna_fold_compound_t` is erased from memory through a call to `vrna_fold_compound_free()` and will be passed the address of memory previously bound to the `vrna_fold_compound_t` via `vrna_fold_compound_add_auxdata()`.

**Global `vrna_callback_hc_evaluate` (int i, int j, int k, int l, unsigned char d, void \*data)**

This callback enables one to over-rule default hard constraints in secondary structure decompositions.

**Global `vrna_callback_recursion_status` (unsigned char status, void \*data)**

This function will be called to notify a third-party implementation about the status of a currently ongoing recursion. The purpose of this callback mechanism is to provide users with a simple way to ensure pre- and post conditions for auxiliary mechanisms attached to our implementations.

**Global `vrna_callback_sc_backtrack` (int i, int j, int k, int l, unsigned char d, void \*data)**

This callback enables one to add auxiliary base pairs in the backtracking steps of hairpin- and interior loops.

**Global `vrna_callback_sc_energy` (int i, int j, int k, int l, unsigned char d, void \*data)**

This callback enables one to add (pseudo-)energy contributions to individual decompositions of the secondary structure.

**Global `vrna_callback_sc_exp_energy` (int i, int j, int k, int l, unsigned char d, void \*data)**

This callback enables one to add (pseudo-)energy contributions to individual decompositions of the secondary structure (Partition function variant, i.e. contributions must be returned as Boltzmann factors).

**Global `vrna_callback_ud_energy` (vrna\_fold\_compound\_t \*vc, int i, int j, unsigned int loop\_type, void \*data)**

This function will be called to determine the additional energy contribution of a specific unstructured domain, e.g. the binding free energy of some ligand.

**Global `vrna_callback_ud_exp_energy` (vrna\_fold\_compound\_t \*vc, int i, int j, unsigned int loop\_type, void \*data)**

This function will be called to determine the additional energy contribution of a specific unstructured domain, e.g. the binding free energy of some ligand (Partition function variant, i.e. the Boltzmann factors instead of actual free energies).

**Global `vrna_callback_ud_exp_production` (vrna\_fold\_compound\_t \*vc, void \*data)**

The production rule for the unstructured domain grammar extension (Partition function variant)

**Global `vrna_callback_ud_probs_add` (vrna\_fold\_compound\_t \*vc, int i, int j, unsigned int loop\_type, FLT\_OR\_DBL exp\_energy, void \*data)**

A callback function to store equilibrium probabilities for the unstructured domain feature

**Global `vrna_callback_ud_probs_get` (vrna\_fold\_compound\_t \*vc, int i, int j, unsigned int loop\_type, int motif, void \*data)**

A callback function to retrieve equilibrium probabilities for the unstructured domain feature

**Global `vrna_callback_ud_production` (vrna\_fold\_compound\_t \*vc, void \*data)**

The production rule for the unstructured domain grammar extension

**Global `vrna_mfe_window_callback` (int start, int end, const char \*structure, float en, void \*data)**

This function will be called for each hit in a sliding window MFE prediction.

**Global `vrna_probs_window_callback` (FLT\_OR\_DBL \*pr, int pr\_size, int i, int max, unsigned int type, void \*data)**

This function will be called for each probability data set in the sliding window probability computation implementation of `vrna_probs_window()`. The argument *type* specifies the type of probability that is passed to this function.

**Global `vrna_subopt_callback` (const char \*structure, float energy, void \*data)**

This function will be called for each suboptimal secondary structure that is successfully backtraced.

## 6.3 Scripting Language interface(s)

### 6.3.1 Introduction

For an easy integration into scripting languages, we provide an automatically generated interface to the RNAlib C-library, generated with SWIG.

### 6.3.2 Function Renaming

To provide a namespace-like separation of function symbols from our C library and third-party code, we use the prefix `vrna_` or `VRNA_` whenever possible. This, however, is not necessary for the scripting language interface, as it uses the separate namespace or package `RNA` anyway. Consequently, symbols that appear to have the `vrna_` or `VRNA_` prefix in the C-library have the corresponding prefix stripped away.

For instance, the C code

```
mfe = vrna_fold(sequence, structure);
```

translates to

```
my ($structure, $mfe) = RNA::fold($sequence)
```

in the Perl 5 interface, and

```
structure,mfe = RNA.fold(sequence)
```

for Python 2/3. Note, that in this example we also make use of the possibility to return multiple data at once in the scripting language, while the C library function uses additional parameters to return multiple data.

Functions that are dedicated to work on specific data structures only, e.g. the `vrna_fold_compound_t`, are usually not exported at all. Instead, they are attached as object methods of a corresponding class (see [Object oriented Interface for Data Structures](#) for detailed information).

#### 6.3.2.1 Global Variables

For the Python interface(s) SWIG places global variables of the C-library into an additional namespace `cvar`. For instance, changing the global temperature variable thus becomes

```
RNA.cvar.temperature = 25
```

### 6.3.3 Object oriented Interface for Data Structures

For data structures, typedefs, and enumerations the `vrna_` prefixes are dropped as well, together with their suffixes `_s`, `_t`, and `_e`, respectively. Furthermore, data structures are usually transformed into classes and relevant functions of the C-library are attached as methods.

### 6.3.4 Examples

Examples on the basic usage of the scripting language interfaces can be found in the `scripting_perl_examples` and `scripting_python_examples` section.

### 6.3.5 SWIG generated Wrapper notes

Special notes on how functions, structures, enums, and macro definitions are actually wrapped, can be found below

#### Global `vrna_aln_conservation_col` (const char \*\*alignment, const vrna\_md\_t \*md\_p, unsigned int options)

This function is available in an overloaded form where the last two parameters may be omitted, indicating `md = NULL`, and `options = VRNA_MEASURE_SHANNON_ENTROPY`, respectively.

#### Global `vrna_aln_conservation_struct` (const char \*\*alignment, const char \*structure, const vrna\_md\_t \*md)

This function is available in an overloaded form where the last parameter may be omitted, indicating `md = NULL`

#### Global `vrna_db_flatten` (char \*structure, unsigned int options)

This function flattens an input structure string in-place! The second parameter is optional and defaults to `VRNA_BRACKETS_DEFAULT`.

An overloaded version of this function exists, where an additional second parameter can be passed to specify the target brackets, i.e. the type of matching pair characters all brackets will be flattened to. Therefore, in the scripting language interface this function is a replacement for `vrna_db_flatten_to()`.

#### Global `vrna_db_flatten_to` (char \*string, const char target[3], unsigned int options)

This function is available as an overloaded version of `vrna_db_flatten()`

#### Global `vrna_enumerate_necklaces` (const unsigned int \*type\_counts)

This function is available as global function `enumerate_necklaces()` which accepts lists input, and produces list of lists output.

#### Global `vrna_eval_circ_consensus_structure` (const char \*\*alignment, const char \*structure)

This function is available through an overloaded version of `vrna_eval_circ_structure()`. Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument

#### Global `vrna_eval_circ_consensus_structure_v` (const char \*\*alignment, const char \*structure, int verbosity\_level, FILE \*file)

This function is available through an overloaded version of `vrna_eval_circ_structure()`. Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument. The last two arguments are optional and default to `VRNA_VERBOSITY_QUIET` and `NULL`, respectively.

#### Global `vrna_eval_circ_gquad_consensus_structure` (const char \*\*alignment, const char \*structure)

This function is available through an overloaded version of `vrna_eval_circ_gquad_structure()`. Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument

#### Global `vrna_eval_circ_gquad_consensus_structure_v` (const char \*\*alignment, const char \*structure, int verbosity\_level, FILE \*file)

This function is available through an overloaded version of `vrna_eval_circ_gquad_structure()`. Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument. The last two arguments are optional and default to `VRNA_VERBOSITY_QUIET` and `NULL`, respectively.

#### Global `vrna_eval_circ_gquad_structure` (const char \*string, const char \*structure)

In the target scripting language, this function serves as a wrapper for `vrna_eval_circ_gquad_structure_v()` and, thus, allows for two additional, optional arguments, the verbosity level and a file handle which default to `VRNA_VERBOSITY_QUIET` and `NULL`, respectively.

#### Global `vrna_eval_circ_gquad_structure_v` (const char \*string, const char \*structure, int verbosity\_level, FILE \*file)

This function is available through an overloaded version of `vrna_eval_circ_gquad_structure()`. The last two arguments for this function are optional and default to `VRNA_VERBOSITY_QUIET` and `NULL`, respectively.

**Global `vrna_eval_circ_structure` (`const char *string`, `const char *structure`)**

In the target scripting language, this function serves as a wrapper for `vrna_eval_circ_structure_v()` and, thus, allows for two additional, optional arguments, the verbosity level and a file handle which default to `VRNA_VERBOSITY_QUIET` and `NULL`, respectively.

**Global `vrna_eval_circ_structure_v` (`const char *string`, `const char *structure`, `int verbosity_level`, `FILE *file`)**

This function is available through an overloaded version of `vrna_eval_circ_structure()`. The last two arguments for this function are optional and default to `VRNA_VERBOSITY_QUIET` and `NULL`, respectively.

**Global `vrna_eval_consensus_structure_pt_simple` (`const char **alignment`, `const short *pt`)**

This function is available through an overloaded version of `vrna_eval_structure_pt_simple()`. Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument

**Global `vrna_eval_consensus_structure_simple` (`const char **alignment`, `const char *structure`)**

This function is available through an overloaded version of `vrna_eval_structure_simple()`. Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument

**Global `vrna_eval_consensus_structure_simple_v` (`const char **alignment`, `const char *structure`, `int verbosity_level`, `FILE *file`)**

This function is available through an overloaded version of `vrna_eval_structure_simple()`. Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument. The last two arguments are optional and default to `VRNA_VERBOSITY_QUIET` and `NULL`, respectively.

**Global `vrna_eval_consensus_structure_simple_verbose` (`const char **alignment`, `const char *structure`, `FILE *file`)**

This function is not available. Use `vrna_eval_consensus_structure_simple_v()` instead!

**Global `vrna_eval_covar_structure` (`vrna_fold_compound_t *vc`, `const char *structure`)**

This function is attached as method `eval_covar_structure()` to objects of type `fold_compound`

**Global `vrna_eval_gquad_consensus_structure` (`const char **alignment`, `const char *structure`)**

This function is available through an overloaded version of `vrna_eval_gquad_structure()`. Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument

**Global `vrna_eval_gquad_consensus_structure_v` (`const char **alignment`, `const char *structure`, `int verbosity_level`, `FILE *file`)**

This function is available through an overloaded version of `vrna_eval_gquad_structure()`. Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument. The last two arguments are optional and default to `VRNA_VERBOSITY_QUIET` and `NULL`, respectively.

**Global `vrna_eval_gquad_structure` (`const char *string`, `const char *structure`)**

In the target scripting language, this function serves as a wrapper for `vrna_eval_gquad_structure_v()` and, thus, allows for two additional, optional arguments, the verbosity level and a file handle which default to `VRNA_VERBOSITY_QUIET` and `NULL`, respectively.

**Global `vrna_eval_gquad_structure_v` (`const char *string`, `const char *structure`, `int verbosity_level`, `FILE *file`)**

This function is available through an overloaded version of `vrna_eval_gquad_structure()`. The last two arguments for this function are optional and default to `VRNA_VERBOSITY_QUIET` and `NULL`, respectively.

**Global `vrna_eval_hp_loop` (`vrna_fold_compound_t *fc`, `int i`, `int j`)**

This function is attached as method `eval_hp_loop()` to objects of type `fold_compound`

**Global `vrna_eval_int_loop` (`vrna_fold_compound_t *fc`, `int i`, `int j`, `int k`, `int l`)**

This function is attached as method `eval_int_loop()` to objects of type `fold_compound`

**Global `vrna_eval_loop_pt` (`vrna_fold_compound_t *vc`, `int i`, `const short *pt`)**

This function is attached as method `eval_loop_pt()` to objects of type `fold_compound`

**Global `vrna_eval_move` (`vrna_fold_compound_t *vc`, `const char *structure`, `int m1`, `int m2`)**

This function is attached as method `eval_move()` to objects of type `fold_compound`

**Global `vrna_eval_move_pt` (`vrna_fold_compound_t *vc`, `short *pt`, `int m1`, `int m2`)**

This function is attached as method `eval_move_pt()` to objects of type `fold_compound`



Global **[vrna\\_eval\\_structure](#)** (**[vrna\\_fold\\_compound\\_t](#)** \*vc, const char \*structure)

This function is attached as method **[eval\\_structure\(\)](#)** to objects of type *fold\_compound*

Global **[vrna\\_eval\\_structure\\_pt](#)** (**[vrna\\_fold\\_compound\\_t](#)** \*vc, const short \*pt)

This function is attached as method **[eval\\_structure\\_pt\(\)](#)** to objects of type *fold\_compound*

Global **[vrna\\_eval\\_structure\\_pt\\_simple](#)** (const char \*string, const short \*pt)

In the target scripting language, this function serves as a wrapper for **[vrna\\_eval\\_structure\\_pt\\_v\(\)](#)** and, thus, allows for two additional, optional arguments, the verbosity level and a file handle which default to **[VRNA\\_VERBOSITY\\_QUIET](#)** and **[NULL](#)**, respectively.

Global **[vrna\\_eval\\_structure\\_pt\\_verbose](#)** (**[vrna\\_fold\\_compound\\_t](#)** \*vc, const short \*pt, FILE \*file)

This function is attached as method **[eval\\_structure\\_pt\\_verbose\(\)](#)** to objects of type *fold\_compound*

Global **[vrna\\_eval\\_structure\\_simple](#)** (const char \*string, const char \*structure)

In the target scripting language, this function serves as a wrapper for **[vrna\\_eval\\_structure\\_simple\\_v\(\)](#)** and, thus, allows for two additional, optional arguments, the verbosity level and a file handle which default to **[VRNA\\_VERBOSITY\\_QUIET](#)** and **[NULL](#)**, respectively.

Global **[vrna\\_eval\\_structure\\_simple\\_v](#)** (const char \*string, const char \*structure, int verbosity\_level, FILE \*file)

This function is available through an overloaded version of **[vrna\\_eval\\_structure\\_simple\(\)](#)**. The last two arguments for this function are optional and default to **[VRNA\\_VERBOSITY\\_QUIET](#)** and **[NULL](#)**, respectively.

Global **[vrna\\_eval\\_structure\\_simple\\_verbose](#)** (const char \*string, const char \*structure, FILE \*file)

This function is not available. Use **[vrna\\_eval\\_structure\\_simple\\_v\(\)](#)** instead!

Global **[vrna\\_eval\\_structure\\_verbose](#)** (**[vrna\\_fold\\_compound\\_t](#)** \*vc, const char \*structure, FILE \*file)

This function is attached as method **[eval\\_structure\\_verbose\(\)](#)** to objects of type *fold\_compound*

Global **[vrna\\_exp\\_params\\_rescale](#)** (**[vrna\\_fold\\_compound\\_t](#)** \*vc, double \*mfe)

This function is attached to **[vrna\\_fc\\_s](#)** objects as overloaded **[exp\\_params\\_rescale\(\)](#)** method.

When no parameter is passed to this method, the resulting action is the same as passing **[NULL](#)** as second parameter to **[vrna\\_exp\\_params\\_rescale\(\)](#)**, i.e. default scaling of the partition function. Passing an energy in kcal/mol, e.g. as retrieved by a previous call to the **[mfe\(\)](#)** method, instructs all subsequent calls to scale the partition function accordingly.

Global **[vrna\\_exp\\_params\\_reset](#)** (**[vrna\\_fold\\_compound\\_t](#)** \*vc, **[vrna\\_md\\_t](#)** \*md\_p)

This function is attached to **[vrna\\_fc\\_s](#)** objects as overloaded **[exp\\_params\\_reset\(\)](#)** method.

When no parameter is passed to this method, the resulting action is the same as passing **[NULL](#)** as second parameter to **[vrna\\_exp\\_params\\_reset\(\)](#)**, i.e. global default model settings are used. Passing an object of type **[vrna\\_md\\_s](#)** resets the fold compound according to the specifications stored within the **[vrna\\_md\\_s](#)** object.

Class **[vrna\\_fc\\_s](#)**

This data structure is wrapped as an object **[fold\\_compound](#)** with several related functions attached as methods.

A new **[fold\\_compound](#)** can be obtained by calling one of its constructors:

- **[fold\\_compound\(seq\)](#)** – Initialize with a single sequence, or two concatenated sequences separated by an ampersand character '&' (for cofolding)
- **[fold\\_compound\(aln\)](#)** – Initialize with a sequence alignment *aln* stored as a list of sequences (with gap characters)

The resulting object has a list of attached methods which in most cases directly correspond to functions that mainly operate on the corresponding C data structure:

- **[type\(\)](#)** – Get the type of the *fold\_compound* (See **[vrna\\_fc\\_type\\_e](#)**)
- **[length\(\)](#)** – Get the length of the sequence(s) or alignment stored within the *fold\_compound*

Global **`vrna_file_commands_apply`** (`vrna_fold_compound_t *vc`, `const char *filename`, `unsigned int options`)

This function is attached as method **`file_commands_apply()`** to objects of type *fold\_compound*

Global **`vrna_file_msa_detect_format`** (`const char *filename`, `unsigned int options`)

This function exists as an overloaded version where the `options` parameter may be omitted! In that case, the `options` parameter defaults to [VRNA\\_FILE\\_FORMAT\\_MSA\\_DEFAULT](#).

Global **`vrna_file_msa_read`** (`const char *filename`, `char ***names`, `char ***aln`, `char **id`, `char **structure`, `unsigned int options`)

In the target scripting language, only the first and last argument, `filename` and `options`, are passed to the corresponding function. The other arguments, which serve as output in the C-library, are available as additional return values. Hence, a function call in python may look like this:

```
num_seq, names, aln, id, structure = RNA.file_msa_read("msa.stk", RNA.FILE_FORMAT_MSA_STOCKHOLM)
```

After successfully reading the first record, the variable `num_seq` contains the number of sequences in the alignment (the actual return value of the C-function), while the variables `names`, `aln`, `id`, and `structure` are lists of the sequence names and aligned sequences, as well as strings holding the alignment ID and the structure as stated in the `SS_cons` line, respectively. Note, the last two return values may be empty strings in case the alignment does not provide the required data.

This function exists as an overloaded version where the `options` parameter may be omitted! In that case, the `options` parameter defaults to [VRNA\\_FILE\\_FORMAT\\_MSA\\_STOCKHOLM](#).

Global **`vrna_file_msa_read_record`** (`FILE *fp`, `char ***names`, `char ***aln`, `char **id`, `char **structure`, `unsigned int options`)

In the target scripting language, only the first and last argument, `fp` and `options`, are passed to the corresponding function. The other arguments, which serve as output in the C-library, are available as additional return values. Hence, a function call in python may look like this:

```
f = open('msa.stk', 'r')
num_seq, names, aln, id, structure = RNA.file_msa_read_record(f, RNA.FILE_FORMAT_MSA_STOCKHOLM)
f.close()
```

After successfully reading the first record, the variable `num_seq` contains the number of sequences in the alignment (the actual return value of the C-function), while the variables `names`, `aln`, `id`, and `structure` are lists of the sequence names and aligned sequences, as well as strings holding the alignment ID and the structure as stated in the `SS_cons` line, respectively. Note, the last two return values may be empty strings in case the alignment does not provide the required data.

This function exists as an overloaded version where the `options` parameter may be omitted! In that case, the `options` parameter defaults to [VRNA\\_FILE\\_FORMAT\\_MSA\\_STOCKHOLM](#).

Global **`vrna_file_msa_write`** (`const char *filename`, `const char **names`, `const char **aln`, `const char *id`, `const char *structure`, `const char *source`, `unsigned int options`)

In the target scripting language, this function exists as a set of overloaded versions, where the last four parameters may be omitted. If the `options` parameter is missing the options default to ([VRNA\\_FILE\\_FORMAT\\_MSA\\_STOCKHOLM](#) | [VRNA\\_FILE\\_FORMAT\\_MSA\\_APPEND](#)).

Global **`vrna_hc_add_from_db`** (`vrna_fold_compound_t *vc`, `const char *constraint`, `unsigned int options`)

This function is attached as method **`hc_add_from_db()`** to objects of type *fold\_compound*

Global **`vrna_hc_init`** (`vrna_fold_compound_t *vc`)

This function is attached as method **`hc_init()`** to objects of type *fold\_compound*

Global **`vrna_maximum_matching`** (`vrna_fold_compound_t *fc`)

This function is attached as method **`maximum_matching()`** to objects of type *fold\_compound* (i.e. *vrna\_fold\_compound\_t*).

Global **`vrna_maximum_matching_simple`** (`const char *sequence`)

This function is available as global function **`maximum_matching()`**.

**Class `vrna_md_s`**

This data structure is wrapped as an object **md** with multiple related functions attached as methods.

A new set of default parameters can be obtained by calling the constructor of **md**:

- `md()` – Initialize with default settings

The resulting object has a list of attached methods which directly correspond to functions that mainly operate on the corresponding C data structure:

- `reset()` – `vrna_md_set_default()`
- `set_from_globals()` – `set_model_details()`
- `option_string()` – `vrna_md_option_string()`

Note, that default parameters can be modified by directly setting any of the following global variables. Internally, getting/setting default parameters using their global variable representative translates into calls of the following functions, therefore these wrappers for these functions do not exist in the scripting language interface(s):

global variable	C getter	C setter
temperature	<a href="#">vrna_md_defaults_temperature_get()</a>	<a href="#">vrna_md_defaults_temperature()</a>
dangles	<a href="#">vrna_md_defaults_dangles_get()</a>	<a href="#">vrna_md_defaults_dangles()</a>
betaScale	<a href="#">vrna_md_defaults_betaScale_get()</a>	<a href="#">vrna_md_defaults_betaScale()</a>
tetra_loop	this is an alias of <i>special_hp</i>	
special_hp	<a href="#">vrna_md_defaults_special_hp_get()</a>	<a href="#">vrna_md_defaults_special_hp()</a>
noLonelyPairs	this is an alias of <i>noLP</i>	
noLP	<a href="#">vrna_md_defaults_noLP_get()</a>	<a href="#">vrna_md_defaults_noLP()</a>
noGU	<a href="#">vrna_md_defaults_noGU_get()</a>	<a href="#">vrna_md_defaults_noGU()</a>
no_closingGU	this is an alias of <i>noGUclosure</i>	
noGUclosure	<a href="#">vrna_md_defaults_noGUclosure_get()</a>	<a href="#">vrna_md_defaults_noGUclosure()</a>
logML	<a href="#">vrna_md_defaults_logML_get()</a>	<a href="#">vrna_md_defaults_logML()</a>
circ	<a href="#">vrna_md_defaults_circ_get()</a>	<a href="#">vrna_md_defaults_circ()</a>
gquad	<a href="#">vrna_md_defaults_gquad_get()</a>	<a href="#">vrna_md_defaults_gquad()</a>
uniq_ML	<a href="#">vrna_md_defaults_uniq_ML_get()</a>	<a href="#">vrna_md_defaults_uniq_ML()</a>
energy_set	<a href="#">vrna_md_defaults_energy_set_get()</a>	<a href="#">vrna_md_defaults_energy_set()</a>
backtrack	<a href="#">vrna_md_defaults_backtrack_get()</a>	<a href="#">vrna_md_defaults_backtrack()</a>
backtrack_type	<a href="#">vrna_md_defaults_backtrack_type_get()</a>	<a href="#">vrna_md_defaults_backtrack_type()</a>
do_backtrack	this is an alias of <i>compute_bpp</i>	
compute_bpp	<a href="#">vrna_md_defaults_compute_bpp_get()</a>	<a href="#">vrna_md_defaults_compute_bpp()</a>
max_bp_span	<a href="#">vrna_md_defaults_max_bp_span_get()</a>	<a href="#">vrna_md_defaults_max_bp_span()</a>
min_loop_size	<a href="#">vrna_md_defaults_min_loop_size_get()</a>	<a href="#">vrna_md_defaults_min_loop_size()</a>
window_size	<a href="#">vrna_md_defaults_window_size_get()</a>	<a href="#">vrna_md_defaults_window_size()</a>
oldAliEn	<a href="#">vrna_md_defaults_oldAliEn_get()</a>	<a href="#">vrna_md_defaults_oldAliEn()</a>
ribo	<a href="#">vrna_md_defaults_ribo_get()</a>	<a href="#">vrna_md_defaults_ribo()</a>
cv_fact	<a href="#">vrna_md_defaults_cv_fact_get()</a>	<a href="#">vrna_md_defaults_cv_fact()</a>
nc_fact	<a href="#">vrna_md_defaults_nc_fact_get()</a>	<a href="#">vrna_md_defaults_nc_fact()</a>
sfact	<a href="#">vrna_md_defaults_sfact_get()</a>	<a href="#">vrna_md_defaults_sfact()</a>

**Global `vrna_mean_bp_distance` (`vrna_fold_compound_t *vc`)**

This function is attached as method `mean_bp_distance()` to objects of type `fold_compound`

**Global `vrna_mfe` (`vrna_fold_compound_t *vc`, `char *structure`)**

This function is attached as method `mfe()` to objects of type `fold_compound`

**Global `vrna_mfe_dimer` (`vrna_fold_compound_t *vc`, `char *structure`)**

This function is attached as method `mfe_dimer()` to objects of type `fold_compound`

**Global `vrna_mfe_window` (`vrna_fold_compound_t *vc`, `FILE *file`)**

This function is attached as method `mfe_window()` to objects of type `fold_compound`

**Global `vrna_neighbors` (`vrna_fold_compound_t *vc`, `const short *pt`, `unsigned int options`)**

This function is attached as an overloaded method `neighbors()` to objects of type `fold_compound`. The optional parameter `options` defaults to `VRNA_MOVESET_DEFAULT` if it is omitted.

**Global `vrna_params_reset` (`vrna_fold_compound_t *vc`, `vrna_md_t *md_p`)**

This function is attached to `vrna_fc_s` objects as overloaded `params_reset()` method.

When no parameter is passed to this method, the resulting action is the same as passing `NULL` as second parameter to `vrna_params_reset()`, i.e. global default model settings are used. Passing an object of type `vrna_md_s` resets the fold compound according to the specifications stored within the `vrna_md_s` object.

**Global `vrna_params_subst` (`vrna_fold_compound_t *vc`, `vrna_param_t *par`)**

This function is attached to `vrna_fc_s` objects as `params_subst()` method.

**Global `vrna_path` (`vrna_fold_compound_t *vc`, `short *pt`, `unsigned int steps`, `unsigned int options`)**

This function is attached as an overloaded method `path()` to objects of type `fold_compound`. The optional parameter `options` defaults to `VRNA_PATH_DEFAULT` if it is omitted.

**Global `vrna_path_findpath` (`vrna_fold_compound_t *vc`, `const char *s1`, `const char *s2`, `int width`)**

This function is attached as an overloaded method `path_findpath()` to objects of type `fold_compound`. The optional parameter `width` defaults to 1 if it is omitted.

**Global `vrna_path_findpath_saddle` (`vrna_fold_compound_t *vc`, `const char *s1`, `const char *s2`, `int width`)**

This function is attached as an overloaded method `path_findpath_saddle()` to objects of type `fold_compound`. The optional parameter `width` defaults to 1 if it is omitted.

**Global `vrna_path_findpath_saddle_ub` (`vrna_fold_compound_t *vc`, `const char *s1`, `const char *s2`, `int width`, `int maxE`)**

This function is attached as an overloaded method `path_findpath_saddle()` to objects of type `fold_compound`. The optional parameter `width` defaults to 1 if it is omitted, while the optional parameter `maxE` defaults to `INF`. In case the function did not find a path with  $E_{saddle} < E_{max}$  the function returns a `NULL` object, i.e. `undef` for Perl and `None` for Python.

**Global `vrna_path_findpath_ub` (`vrna_fold_compound_t *vc`, `const char *s1`, `const char *s2`, `int width`, `int maxE`)**

This function is attached as an overloaded method `path_findpath()` to objects of type `fold_compound`. The optional parameter `width` defaults to 1 if it is omitted, while the optional parameter `maxE` defaults to `INF`. In case the function did not find a path with  $E_{saddle} < E_{max}$  the function returns an empty list.

**Global `vrna_path_gradient` (`vrna_fold_compound_t *vc`, `short *pt`, `unsigned int options`)**

This function is attached as an overloaded method `path_gradient()` to objects of type `fold_compound`. The optional parameter `options` defaults to `VRNA_PATH_DEFAULT` if it is omitted.

**Global `vrna_path_random` (`vrna_fold_compound_t *vc`, `short *pt`, `unsigned int steps`, `unsigned int options`)**

This function is attached as an overloaded method `path_gradient()` to objects of type `fold_compound`. The optional parameter `options` defaults to `VRNA_PATH_DEFAULT` if it is omitted.

**Global `vrna_pbacktrack` (`vrna_fold_compound_t *vc`)**

This function is attached as overloaded method `pbacktrack()` to objects of type `fold_compound` that accepts an optional `length` argument. Hence, it serves as a replacement for `vrna_pbacktrack()`.

**Global `vrna_pbacktrack5` (`vrna_fold_compound_t *vc`, `int length`)**

This function is attached as overloaded method `pbacktrack()` to objects of type `fold_compound`

Global **vrna\_pbacktrack\_nr** (vrna\_fold\_compound\_t \*vc, int num\_samples)

This function is attached as method **pbacktrack\_nr()** to objects of type *fold\_compound*.

Global **vrna\_pf** (vrna\_fold\_compound\_t \*vc, char \*structure)

This function is attached as method **pf()** to objects of type *fold\_compound*

Global **vrna\_pf\_dimer** (vrna\_fold\_compound\_t \*vc, char \*structure)

This function is attached as method **pf\_dimer()** to objects of type *fold\_compound*

Global **vrna\_rotational\_symmetry** (const char \*string)

This function is available as global function **rotational\_symmetry()**. See **vrna\_rotational\_symmetry\_pos()** for details.

Global **vrna\_rotational\_symmetry\_db** (vrna\_fold\_compound\_t \*fc, const char \*structure)

This function is attached as method **rotational\_symmetry\_db()** to objects of type *fold\_compound* (i.e. *vrna\_fold\_compound\_t*). See **vrna\_rotational\_symmetry\_db\_pos()** for details.

Global **vrna\_rotational\_symmetry\_db\_pos** (vrna\_fold\_compound\_t \*fc, const char \*structure, unsigned int \*\*positions)

This function is attached as method **rotational\_symmetry\_db()** to objects of type *fold\_compound* (i.e. *vrna\_fold\_compound\_t*). Thus, the first argument must be omitted. In contrast to our C-implementation, this function doesn't simply return the order of rotational symmetry of the secondary structure, but returns the list *position* of cyclic permutation shifts that result in a rotationally symmetric structure. The length of the list then determines the order of rotational symmetry.

Global **vrna\_rotational\_symmetry\_num** (const unsigned int \*string, size\_t string\_length)

This function is available as global function **rotational\_symmetry()**. See **vrna\_rotational\_symmetry\_pos()** for details. Note, that in the target language the length of the list *string* is always known a-priori, so the parameter *string\_length* must be omitted.

Global **vrna\_rotational\_symmetry\_pos\_num** (const unsigned int \*string, size\_t string\_length, unsigned int \*\*positions)

This function is available as global function **rotational\_symmetry()**. See **vrna\_rotational\_symmetry\_pos()** for details. Note, that in the target language the length of the list *string* is always known a-priori, so the parameter *string\_length* must be omitted.

Global **vrna\_sc\_add\_bp** (vrna\_fold\_compound\_t \*vc, int i, int j, FLT\_OR\_DBL energy, unsigned int options)

This function is attached as an overloaded method **sc\_add\_bp()** to objects of type *fold\_compound*. The method either takes arguments for a single base pair (i,j) with the corresponding energy value:

```
fold_compound.sc_add_bp(i, j, energy, options)
```

or an entire 2-dimensional matrix with dimensions  $n \times n$  that stores free energy contributions for any base pair (i,j) with  $1 \leq i < j \leq n$ :

```
fold_compound.sc_add_bp(matrix, options)
```

In both variants, the *options* argument is optional and may be omitted.

Global **vrna\_sc\_add\_bt** (vrna\_fold\_compound\_t \*vc, vrna\_callback\_sc\_backtrack \*f)

This function is attached as method **sc\_add\_bt()** to objects of type *fold\_compound*

Global **vrna\_sc\_add\_data** (vrna\_fold\_compound\_t \*vc, void \*data, vrna\_callback\_free\_auxdata \*free\_data)

This function is attached as method **sc\_add\_data()** to objects of type *fold\_compound*

Global **vrna\_sc\_add\_exp\_f** (vrna\_fold\_compound\_t \*vc, vrna\_callback\_sc\_exp\_energy \*exp\_f)

This function is attached as method **sc\_add\_exp\_f()** to objects of type *fold\_compound*

Global **vrna\_sc\_add\_f** (vrna\_fold\_compound\_t \*vc, vrna\_callback\_sc\_energy \*f)

This function is attached as method **sc\_add\_f()** to objects of type *fold\_compound*

Global **vrna\_sc\_add\_hi\_motif** (vrna\_fold\_compound\_t \*vc, const char \*seq, const char \*structure, FLT\_OR\_DBL energy, unsigned int options)

This function is attached as method **sc\_add\_hi\_motif()** to objects of type *fold\_compound*

Global **vrna\_sc\_add\_SHAPE\_deigan** (vrna\_fold\_compound\_t \*vc, const double \*reactivities, double m, double b, unsigned int options)

This function is attached as method **sc\_add\_SHAPE\_deigan()** to objects of type *fold\_compound*

Global **vrna\_sc\_add\_SHAPE\_deigan\_ali** (vrna\_fold\_compound\_t \*vc, const char \*\*shape\_files, const int \*shape\_file\_association, double m, double b, unsigned int options)

This function is attached as method **sc\_add\_SHAPE\_deigan\_ali()** to objects of type *fold\_compound*

Global **vrna\_sc\_add\_SHAPE\_zarringhalam** (vrna\_fold\_compound\_t \*vc, const double \*reactivities, double b, double default\_value, const char \*shape\_conversion, unsigned int options)

This function is attached as method **sc\_add\_SHAPE\_zarringhalam()** to objects of type *fold\_compound*

Global **vrna\_sc\_add\_up** (vrna\_fold\_compound\_t \*vc, int i, FLT\_OR\_DBL energy, unsigned int options)

This function is attached as an overloaded method **sc\_add\_up()** to objects of type *fold\_compound*. The method either takes arguments for a single nucleotide *i* with the corresponding energy value:

```
fold_compound.sc_add_up(i, energy, options)
```

or an entire vector that stores free energy contributions for each nucleotide *i* with  $1 \leq i \leq n$ :

```
fold_compound.sc_add_bp(vector, options)
```

In both variants, the *options* argument is optional and may be omitted.

Global **vrna\_sc\_init** (vrna\_fold\_compound\_t \*vc)

This function is attached as method **sc\_init()** to objects of type *fold\_compound*

Global **vrna\_sc\_remove** (vrna\_fold\_compound\_t \*vc)

This function is attached as method **sc\_remove()** to objects of type *fold\_compound*

Global **vrna\_sc\_set\_bp** (vrna\_fold\_compound\_t \*vc, const FLT\_OR\_DBL \*\*constraints, unsigned int options)

This function is attached as method **sc\_set\_bp()** to objects of type *fold\_compound*

Global **vrna\_sc\_set\_up** (vrna\_fold\_compound\_t \*vc, const FLT\_OR\_DBL \*constraints, unsigned int options)

This function is attached as method **sc\_set\_up()** to objects of type *fold\_compound*

Global **vrna\_subopt** (vrna\_fold\_compound\_t \*vc, int delta, int sorted, FILE \*fp)

This function is attached as method **subopt()** to objects of type *fold\_compound*

Global **vrna\_subopt\_cb** (vrna\_fold\_compound\_t \*vc, int delta, vrna\_subopt\_callback \*cb, void \*data)

This function is attached as method **subopt\_cb()** to objects of type *fold\_compound*

Global **vrna\_subopt\_zuker** (vrna\_fold\_compound\_t \*vc)

This function is attached as method **subopt\_zuker()** to objects of type *fold\_compound*

Global **vrna\_ud\_remove** (vrna\_fold\_compound\_t \*vc)

This function is attached as method **ud\_remove()** to objects of type *fold\_compound*

Global **vrna\_ud\_set\_data** (vrna\_fold\_compound\_t \*vc, void \*data, vrna\_callback\_free\_auxdata \*free\_cb)

This function is attached as method **ud\_set\_data()** to objects of type *fold\_compound*

Global **vrna\_ud\_set\_exp\_prod\_rule\_cb** (vrna\_fold\_compound\_t \*vc, vrna\_callback\_ud\_exp\_production \*pre\_cb, vrna\_callback\_ud\_exp\_energy \*exp\_e\_cb)

This function is attached as method **ud\_set\_exp\_prod\_rule\_cb()** to objects of type *fold\_compound*

Global **vrna\_ud\_set\_prob\_cb** (vrna\_fold\_compound\_t \*vc, vrna\_callback\_ud\_probs\_add \*setter, vrna\_callback\_ud\_probs\_get \*getter)

This function is attached as method **ud\_set\_prob\_cb()** to objects of type *fold\_compound*

Global **vrna\_ud\_set\_prod\_rule\_cb** (vrna\_fold\_compound\_t \*vc, vrna\_callback\_ud\_production \*pre\_cb, vrna\_callback\_ud\_energy \*e\_cb)

This function is attached as method **ud\_set\_prod\_rule\_cb()** to objects of type *fold\_compound*

## **Chapter 7**

### **Additional Utilities**





# Chapter 8

## Examples

- [C Examples](#)
- [Perl5 Examples](#)
- [Python Examples](#)

### 8.1 C Examples

#### 8.1.1 Hello World Examples

##### helloworld\_mfe.c

The following is an example showing the minimal requirements to compute the Minimum Free Energy (MFE) and corresponding secondary structure of an RNA sequence

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <ViennaRNA/fold.h>
#include <ViennaRNA/Utils/basic.h>

int
main()
{
    /* The RNA sequence */
    char *seq = "GAGUAGUGGAACCAGGCUAUGUUUGUGACUCGCAGACUAACA";

    /* allocate memory for MFE structure (length + 1) */
    char *structure = (char *)vrna_alloc(sizeof(char) * (strlen(seq) + 1));

    /* predict Minimum Free Energy and corresponding secondary structure */
    float mfe = vrna_fold(seq, structure);

    /* print sequence, structure and MFE */
    printf("%s\n%s [ %6.2f ]\n", seq, structure, mfe);

    /* cleanup memory */
    free(structure);

    return 0;
}
```

##### See also

examples/helloworld\_mfe.c in the source code tarball

## helloworld\_mfe\_comparative.c

Instead of using a single sequence as done above, this example predicts a consensus structure for a multiple sequence alignment

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <ViennaRNA/alifold.h>
#include <ViennaRNA/utils/basic.h>
#include <ViennaRNA/utils/alignments.h>

int
main()
{
    /* The RNA sequence alignment */
    const char *sequences[] = {
        "CUGCCUCACAACGUUUUGGCCUCAGUUACCCGUAGAUGUAGUGAGGGU",
        "CUGCCUCACAACAUUUUGGCCUCAGUUACUACAUAGAUGUAGUGAGGGU",
        "---CUCGACACACU---GCCUCGGUUACCAUCGGUGCAGUGCGGGU",
        NULL /* indicates end of alignment */
    };

    /* compute the consensus sequence */
    char *cons = consensus(sequences);

    /* allocate memory for MFE consensus structure (length + 1) */
    char *structure = (char *)vrna_alloc(sizeof(char) * (strlen(sequences[0]) + 1));

    /* predict Minnum Free Energy and corresponding secondary structure */
    float mfe = vrna_alifold(sequences, structure);

    /* print consensus sequence, structure and MFE */
    printf("%s\n%s [ %.2f ]\n", cons, structure, mfe);

    /* cleanup memory */
    free(cons);
    free(structure);

    return 0;
}
```

### See also

examples/helloworld\_mfe\_comparative.c in the source code tarball

## helloworld\_probabilities.c

This example shows how to compute the partition function and base pair probabilities with minimal implementation effort.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <ViennaRNA/fold.h>
#include <ViennaRNA/part_func.h>
#include <ViennaRNA/utils/basic.h>

int
main()
{
    /* The RNA sequence */
    char *seq = "GAGUAGUGGAACAGGCCUAUGUUUGUGACUCGCAGACUAACA";

    /* allocate memory for pairing propensity string (length + 1) */
    char *propensity = (char *)vrna_alloc(sizeof(char) * (strlen(seq) + 1));

    /* pointers for storing and navigating through base pair probabilities */
    vrna_ep_t *ptr, *pair_probabilities = NULL;

    float en = vrna_pf_fold(seq, propensity, &pair_probabilities);
}
```

```

/* print sequence, pairing propensity string and ensemble free energy */
printf("%s\n%s [ %6.2f ]\n", seq, propensity, en);

/* print all base pairs with probability above 50% */
for (ptr = pair_probabilities; ptr->i != 0; ptr++)
    if (ptr->p > 0.5)
        printf("p(%d, %d) = %g\n", ptr->i, ptr->j, ptr->p);

/* cleanup memory */
free(pair_probabilities);
free(propensity);

return 0;
}

```

#### See also

examples/helloworld\_probabilities.c in the source code tarball

### 8.1.2 First Steps with the Fold Compound

#### fold\_compound\_mfe.c

Instead of calling the simple MFE folding interface [vrna\\_fold\(\)](#), this example shows how to first create a [vrna\\_fold\\_compound\\_t](#) container with the RNA sequence to finally compute the MFE using this container. This is especially useful if non-default model settings are applied or the dynamic programming (DP) matrices of the MFE prediction are required for post-processing operations, or other tasks on the same sequence will be performed.

```

#include <stdlib.h>
#include <stdio.h>

#include <ViennaRNA/fold_compound.h>
#include <ViennaRNA/utils/basic.h>
#include <ViennaRNA/utils/strings.h>
#include <ViennaRNA/mfe.h>

int
main()
{
    /* initialize random number generator */
    vrna_init_rand();

    /* Generate a random sequence of 50 nucleotides */
    char *seq = vrna_random_string(50, "ACGU");

    /* Create a fold compound for the sequence */
    vrna_fold_compound_t *fc = vrna_fold_compound(seq, NULL,
        VRNA_OPTION_DEFAULT);

    /* allocate memory for MFE structure (length + 1) */
    char *structure = (char *)vrna_alloc(sizeof(char) * (strlen(seq) + 1));

    /* predict Minimum Free Energy and corresponding secondary structure */
    float mfe = vrna_mfe(fc, structure);

    /* print sequence, structure and MFE */
    printf("%s\n%s [ %6.2f ]\n", seq, structure, mfe);

    /* cleanup memory */
    free(seq);
    free(structure);
    vrna_fold_compound_free(fc);

    return 0;
}

```

#### See also

examples/fold\_compound\_mfe.c in the source code tarball

## fold\_compound\_md.c

In the following, we change the model settings (model details) to a temperature of 25 Degree Celcius, and activate G-Quadruplex prediction.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <ViennaRNA/model.h>
#include <ViennaRNA/fold_compound.h>
#include <ViennaRNA/utils/basic.h>
#include <ViennaRNA/utils/strings.h>
#include <ViennaRNA/mfe.h>

int
main()
{
    /* initialize random number generator */
    vrna_init_rand();

    /* Generate a random sequence of 50 nucleotides */
    char *seq = vrna_random_string(50, "ACGU");

    /* allocate memory for MFE structure (length + 1) */
    char *structure = (char *)vrna_alloc(sizeof(char) * (strlen(seq) + 1));

    /* create a new model details structure to store the Model Settings */
    vrna_md_t md;

    /* ALWAYS set default model settings first! */
    vrna_md_set_default(&md);

    /* change temperature and activate G-Quadruplex prediction */
    md.temperature = 25.0; /* 25 Deg Celcius */
    md.gquad = 1; /* Turn-on G-Quadruples support */

    /* create a fold compound */
    vrna_fold_compound_t *fc = vrna_fold_compound(seq, &md,
        VRNA_OPTION_DEFAULT);

    /* predict Minum Free Energy and corresponding secondary structure */
    float mfe = vrna_mfe(fc, structure);

    /* print sequence, structure and MFE */
    printf("%s\n%s [ %6.2f ]\n", seq, structure, mfe);

    /* cleanup memory */
    free(structure);
    vrna_fold_compound_free(fc);

    return 0;
}
```

### See also

examples/fold\_compound\_md.c in the source code tarball

## 8.1.3 Writing Callback Functions

### callback\_subopt.c

Here is a basic example how to use the callback mechanism in `vrna_subopt_cb()`. It simply defines a callback function (see interface definition for `vrna_subopt_callback`) that prints the result and increases a counter variable.

```
#include <stdlib.h>
#include <stdio.h>

#include <ViennaRNA/fold_compound.h>
#include <ViennaRNA/utils/basic.h>
#include <ViennaRNA/utils/strings.h>
```

```

#include <ViennaRNA/subopt.h>

void
subopt_callback(const char *structure,
               float energy,
               void *data)
{
    /* simply print the result and increase the counter variable by 1 */
    if (structure)
        printf("%d.\t%s\t%6.2f\n", ((int *)data)++, structure, energy);
}

int
main()
{
    /* initialize random number generator */
    vrna_init_rand();

    /* Generate a random sequence of 50 nucleotides */
    char *seq = vrna_random_string(50, "ACGU");

    /* Create a fold compound for the sequence */
    vrna_fold_compound_t *fc = vrna_fold_compound(seq, NULL,
        VRNA_OPTION_DEFAULT);

    int counter = 0;

    /*
     * call subopt to enumerate all secondary structures in an energy band of
     * 5 kcal/mol of the MFE and pass it the address of the callback and counter
     * variable
     */
    vrna_subopt_cb(fc, 500, &subopt_callback, (void *)&counter);

    /* cleanup memory */
    free(seq);
    vrna_fold_compound_free(fc);

    return 0;
}

```

**See also**

examples/callback\_subopt.c in the source code tarball

**8.1.4 Application of Soft Constraints****soft\_constraints\_up.c**

In this example, a random RNA sequence is generated to predict its MFE under the constraint that a particular nucleotide receives an additional bonus energy if it remains unpaired.

```

#include <stdlib.h>
#include <stdio.h>

#include <ViennaRNA/fold_compound.h>
#include <ViennaRNA/utils/basic.h>
#include <ViennaRNA/utils/strings.h>
#include <ViennaRNA/constraints/soft.h>
#include <ViennaRNA/mfe.h>

int
main()
{
    /* initialize random number generator */
    vrna_init_rand();

    /* Generate a random sequence of 50 nucleotides */
    char *seq = vrna_random_string(50, "ACGU");

    /* Create a fold compound for the sequence */
    vrna_fold_compound_t *fc = vrna_fold_compound(seq, NULL,
        VRNA_OPTION_DEFAULT);

```

```

/* Add soft constraint of -1.7 kcal/mol to nucleotide 5 whenever it appears in an unpaired context */
vrna_sc_add_up(fc, 5, -1.7, VRNA_OPTION_DEFAULT);

/* allocate memory for MFE structure (length + 1) */
char *structure = (char *)vrna_alloc(sizeof(char) * 51);

/* predict Minimum Free Energy and corresponding secondary structure */
float mfe = vrna_mfe(fc, structure);

/* print sequence, structure and MFE */
printf("%s\n%s [ %.2f ]\n", seq, structure, mfe);

/* cleanup memory */
free(seq);
free(structure);
vrna_fold_compound_free(fc);

return 0;
}

```

### See also

examples/soft\_constraints\_up.c in the source code tarball

## 8.1.5 Other Examples

### example1.c

A more extensive example including MFE, Partition Function, and Centroid structure prediction.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <ViennaRNA/data_structures.h>
#include <ViennaRNA/params/basic.h>
#include <ViennaRNA/utils/basic.h>
#include <ViennaRNA/eval.h>
#include <ViennaRNA/fold.h>
#include <ViennaRNA/part_func.h>

int
main(int argc,
     char *argv[])
{
    char *seq =
        "AGACGACAAGGUUGAAUCGCACCCACAGUCUAUGAGUCGGUGACAACAUAUACGAAAGGCUGUAAAAUCAAUUAUUCACCACAGGGGGCCCCGUGUCUAG";
    char *mfe_structure = vrna_alloc(sizeof(char) * (strlen(seq) + 1));
    char *prob_string = vrna_alloc(sizeof(char) * (strlen(seq) + 1));

    /* get a vrna_fold_compound with default settings */
    vrna_fold_compound_t *vc = vrna_fold_compound(seq, NULL,
        VRNA_OPTION_DEFAULT);

    /* call MFE function */
    double mfe = (double)vrna_mfe(vc, mfe_structure);

    printf("%s\n%s (%.2f)\n", seq, mfe_structure, mfe);

    /* rescale parameters for Boltzmann factors */
    vrna_exp_params_rescale(vc, &mfe);

    /* call PF function */
    FLT_OR_DBL en = vrna_pf(vc, prob_string);

    /* print probability string and free energy of ensemble */
    printf("%s (%.2f)\n", prob_string, en);

    /* compute centroid structure */
    double dist;
    char *cent = vrna_centroid(vc, &dist);

    /* print centroid structure, its free energy and mean distance to the ensemble */
    printf("%s (%.2f d=%.2f)\n", cent, vrna_eval_structure(vc, cent), dist);
}

```

```

/* free centroid structure */
free(cent);

/* free pseudo dot-bracket probability string */
free(prob_string);

/* free mfe structure */
free(mfe_structure);

/* free memory occupied by vrna_fold_compound */
vrna_fold_compound_free(vc);

return EXIT_SUCCESS;
}

```

### See also

examples/example1.c in the source code tarball

## 8.1.6 Deprecated Examples

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "utils.h"
#include "fold_vars.h"
#include "fold.h"
#include "part_func.h"
#include "inverse.h"
#include "RNAstruct.h"
#include "treedist.h"
#include "stringdist.h"
#include "profiledist.h"

void
main()
{
    char          *seq1 = "CGCAGGGAUACCCGCG", *seq2 = "GCGCCCAUAGGGACGC",
                 *struct1, *struct2, *xstruc;
    float         e1, e2, tree_dist, string_dist, profile_dist, kT;
    Tree          *T1, *T2;
    swString      *S1, *S2;
    float         *pfl, *pf2;
    FLT_OR_DBL    *bppm;

    /* fold at 30C instead of the default 37C */
    temperature = 30.;          /* must be set *before* initializing */

    /* allocate memory for structure and fold */
    struct1 = (char *)space(sizeof(char) * (strlen(seq1) + 1));
    e1      = fold(seq1, struct1);

    struct2 = (char *)space(sizeof(char) * (strlen(seq2) + 1));
    e2      = fold(seq2, struct2);

    free_arrays();              /* free arrays used in fold() */

    /* produce tree and string representations for comparison */
    xstruc = expand_Full(struct1);
    T1     = make_tree(xstruc);
    S1     = Make_swString(xstruc);
    free(xstruc);

    xstruc = expand_Full(struct2);
    T2     = make_tree(xstruc);
    S2     = Make_swString(xstruc);
    free(xstruc);

    /* calculate tree edit distance and aligned structures with gaps */
    edit_backtrack = 1;
    tree_dist      = tree_edit_distance(T1, T2);
    free_tree(T1);
    free_tree(T2);
    unexpand_aligned_F(aligned_line);
    printf("%s\n%s  %3.2f\n", aligned_line[0], aligned_line[1], tree_dist);

    /* same thing using string edit (alignment) distance */
    string_dist = string_edit_distance(S1, S2);
}

```

```

free(S1);
free(S2);
printf("%s mfe=%5.2f\n%s mfe=%5.2f dist=%3.2f\n",
       aligned_line[0], e1, aligned_line[1], e2, string_dist);

/* for longer sequences one should also set a scaling factor for
 * partition function folding, e.g: */
kT      = (temperature + 273.15) * 1.98717 / 1000.; /* kT in kcal/mol */
pf_scale = exp(-e1 / kT / strlen(seq1));

/* calculate partition function and base pair probabilities */
e1 = pf_fold(seq1, struct1);
/* get the base pair probability matrix for the previous run of pf_fold() */
bppm = export_bppm();
pf1   = Make_bp_profile_bppm(bppm, strlen(seq1));

e2 = pf_fold(seq2, struct2);
/* get the base pair probability matrix for the previous run of pf_fold() */
bppm = export_bppm();
pf2   = Make_bp_profile_bppm(bppm, strlen(seq2));

free_pf_arrays(); /* free space allocated for pf_fold() */

profile_dist = profile_edit_distance(pf1, pf2);
printf("%s free energy=%5.2f\n%s free energy=%5.2f dist=%3.2f\n",
       aligned_line[0], e1, aligned_line[1], e2, profile_dist);

free_profile(pf1);
free_profile(pf2);
}

```

#### See also

examples/example\_old.c in the source code tarball

## 8.2 Perl5 Examples

### Hello World Examples

#### Using the flat interface

- MFE prediction

```

use RNA;

# The RNA sequence
my $seq = "GAGUAGUGGAACCAGGCUAUGUUUGACUCGACAGACUAACA";

# compute minimum free energy (MFE) and corresponding structure
my ($ss, $mfe) = RNA::fold($seq);

# print output
printf "%s\n%s [ %6.2f ]\n", $seq, $ss, $mfe;

```

#### Using the object oriented interface

- MFE prediction

```

#!/usr/bin/perl

use warnings;
use strict;

use RNA;

my $seq1 = "CGCAGGGAUACCCGCG";

# create new fold_compound object
my $fc = new RNA::fold_compound($seq1);

# compute minimum free energy (mfe) and corresponding structure
my ($ss, $mfe) = $fc->mfe();

# print output
printf "%s [ %6.2f ]\n", $ss, $mfe;

```



## Changing the Model Settings

### Using the flat interface

- MFE prediction at different temperature and dangle model

```
use RNA;

# The RNA sequence
my $seq = "GAGUAGUGGAACCGAGCUAUGUUUGUGACUCGACAGACUAACA";

# create a new model details structure
my $md = new RNA::md();

# change temperature and dangle model
$md->{temperature} = 20.0; # 20 Deg Celcius
$md->{dangles}      = 1;    # Dangle Model 1

# create a fold compound
my $fc = new RNA::fold_compound($seq, $md);

# predict Minmum Free Energy and corresponding secondary structure
my ($ss, $mfe) = $fc->mfe();

# print sequence, structure and MFE
printf "%s\n%s [ %6.2f ]\n", $seq, $ss, $mfe;
```

### Using the object oriented interface

- MFE prediction at different temperature and dangle model

## 8.3 Python Examples

### MFE Prediction (flat interface)

```
import RNA

# The RNA sequence
seq = "GAGUAGUGGAACCGAGCUAUGUUUGUGACUCGACAGACUAACA"

# compute minimum free energy (MFE) and corresponding structure
(ss, mfe) = RNA.fold(seq)

# print output
print "%s\n%s [ %6.2f ]" % (seq, ss, mfe)
```

### MFE Prediction (object oriented interface)

```
import RNA;

sequence = "CGCAGGGAUACCCGCG"

# create new fold_compound object
fc = RNA.fold_compound(sequence)

# compute minimum free energy (mfe) and corresponding structure
(ss, mfe) = fc.mfe()

# print output
print "%s [ %6.2f ]" % (ss, mfe)
```

```

import RNA

sequence = "GGGGAAACCCC"

# Set global switch for unique ML decomposition
RNA.cvar.uniq_ML = 1

subopt_data = { 'counter' : 1, 'sequence' : sequence }

# Print a subopt result as FASTA record
def print_subopt_result(structure, energy, data):
    if not structure == None:
        print ">subopt %d" % data['counter']
        print "%s" % data['sequence']
        print "%s [%6.2f]" % (structure, energy)
        # increase structure counter
        data['counter'] = data['counter'] + 1

# Create a 'fold_compound' for our sequence
a = RNA.fold_compound(sequence)

# Enumerate all structures 500 dcal/mol = 5 kcal/mol around
# the MFE and print each structure using the function above
a.subopt_cb(500, print_subopt_result, subopt_data);

import RNA

seq1 = "CUCGUCGCCUUAUCCAGUGCGGCGCUAGACAUCUAGUUAUCGCCGCAA"

# Turn-off dangles globally
RNA.cvar.dangles = 0

# Data structure that will be passed to our MaximumMatching() callback with two components:
# 1. a 'dummy' fold_compound to evaluate loop energies w/o constraints, 2. a fresh set of energy parameters
mm_data = { 'dummy': RNA.fold_compound(seq1), 'params': RNA.param() }

# Nearest Neighbor Parameter reversal functions
revert_NN = {
    RNA.DECOMP_PAIR_HP:      lambda i, j, k, l, f, p: - f.eval_hp_loop(i, j) - 100,
    RNA.DECOMP_PAIR_IL:      lambda i, j, k, l, f, p: - f.eval_int_loop(i, j, k, l) - 100,
    RNA.DECOMP_PAIR_ML:      lambda i, j, k, l, f, p: - p.MLclosing - p.MLintern[0] - (j - i - k + l - 2)
                          * p.MLbase - 100,
    RNA.DECOMP_ML_ML_STEM:   lambda i, j, k, l, f, p: - p.MLintern[0] - (l - k - 1) * p.MLbase,
    RNA.DECOMP_ML_STEM:      lambda i, j, k, l, f, p: - p.MLintern[0] - (j - i - k + l) * p.MLbase,
    RNA.DECOMP_ML_ML:        lambda i, j, k, l, f, p: - (j - i - k + l) * p.MLbase,
    RNA.DECOMP_ML_UP:         lambda i, j, k, l, f, p: - (j - i + 1) * p.MLbase,
    RNA.DECOMP_EXT_STEM:      lambda i, j, k, l, f, p: - f.E_ext_loop(k, l),
    RNA.DECOMP_EXT_STEM_EXT:  lambda i, j, k, l, f, p: - f.E_ext_loop(i, k),
    RNA.DECOMP_EXT_EXT_STEM:  lambda i, j, k, l, f, p: - f.E_ext_loop(l, j),
    RNA.DECOMP_EXT_EXT_STEM1: lambda i, j, k, l, f, p: - f.E_ext_loop(l, j-1),
}

# Maximum Matching callback function (will be called by RNAlib in each decomposition step)
def MaximumMatching(i, j, k, l, d, data):
    return revert_NN[d](i, j, k, l, data['dummy'], data['params'])

# Create a 'fold_compound' for our sequence
fc = RNA.fold_compound(seq1)

# Add maximum matching soft-constraints
fc.sc_add_f(MaximumMatching)
fc.sc_add_data(mm_data, None)

# Call MFE algorithm
(s, mm) = fc.mfe()

# print result
print "%s\n%s (MM: %d)\n" % (seq1, s, -mm)

```

## **Chapter 9**

## **Changelog**



# Chapter 10

## Deprecated List

Global **alifold** (const char \*\*strings, char \*structure)

Usage of this function is discouraged! Use [vrna\\_alifold\(\)](#), or [vrna\\_mfe\(\)](#) instead!

Global **alimake\_pair\_table** (const char \*structure)

Use [vrna\\_pt\\_ali\\_get\(\)](#) instead!

Global **alipbacktrack** (double \*prob)

Use [vrna\\_pbacktrack\(\)](#) instead!

Global **alipf\_circ\_fold** (const char \*\*sequences, char \*structure, vrna\_ep\_t \*\*pl)

Use [vrna\\_pf\(\)](#) instead

Global **alipf\_fold** (const char \*\*sequences, char \*structure, vrna\_ep\_t \*\*pl)

Use [vrna\\_pf\(\)](#) instead

Global **alipf\_fold\_par** (const char \*\*sequences, char \*structure, vrna\_ep\_t \*\*pl, vrna\_exp\_param\_t \*parameters, int calculate\_bppm, int is\_constrained, int is\_circular)

Use [vrna\\_pf\(\)](#) instead

File **aln\_util.h**

Use [ViennaRNA/utis/alignments.h](#) instead

Global **assign\_plist\_from\_db** (vrna\_ep\_t \*\*pl, const char \*struc, float pr)

Use [vrna\\_plist\(\)](#) instead

Global **assign\_plist\_from\_pr** (vrna\_ep\_t \*\*pl, FLT\_OR\_DBL \*probs, int length, double cutoff)

Use [vrna\\_plist\\_from\\_probs\(\)](#) instead!

Global **b2C** (const char \*structure)

See [vrna\\_db\\_to\\_tree\\_string\(\)](#) and [VRNA\\_STRUCTURE\\_TREE\\_SHAPIRO\\_SHORT](#) for a replacement

Global **b2HIT** (const char \*structure)

See [vrna\\_db\\_to\\_tree\\_string\(\)](#) and [VRNA\\_STRUCTURE\\_TREE\\_HIT](#) for a replacement

Global **b2Shapiro** (const char \*structure)

See [vrna\\_db\\_to\\_tree\\_string\(\)](#) and [VRNA\\_STRUCTURE\\_TREE\\_SHAPIRO\\_WEIGHT](#) for a replacement

Global **base\_pair**

Do not use this variable anymore!

Global **bondT**

Use [vrna\\_bp\\_stack\\_t](#) instead!

Global **bp\_distance** (const char \*str1, const char \*str2)

Use [vrna\\_bp\\_distance](#) instead

Global **bppm\_symbol** (const float \*x)

Use [vrna\\_bpp\\_symbol\(\)](#) instead!

Global **bppm\_to\_structure** (char \*structure, FLT\_OR\_DBL \*pr, unsigned int length)

Use [vrna\\_db\\_from\\_probs\(\)](#) instead!

Global **centroid** (int length, double \*dist)

This function is deprecated and should not be used anymore as it is not threadsafe!

File **char\_stream.h**

Use [ViennaRNA/datastructures/char\\_stream.h](#) instead

Global **circularfold** (const char \*\*strings, char \*structure)

Usage of this function is discouraged! Use [vrna\\_alicircfold\(\)](#), and [vrna\\_mfe\(\)](#) instead!

Global **circfold** (const char \*sequence, char \*structure)

Use [vrna\\_circfold\(\)](#), or [vrna\\_mfe\(\)](#) instead!

Global **co\_pf\_fold** (char \*sequence, char \*structure)

{Use [vrna\\_pf\\_dimer\(\)](#) instead!}

Global **co\_pf\_fold\_par** (char \*sequence, char \*structure, vrna\_exp\_param\_t \*parameters, int calculate\_↵  
bppm, int is\_constrained)

Use [vrna\\_pf\\_dimer\(\)](#) instead!

Global **cofold** (const char \*sequence, char \*structure)

use [vrna\\_mfe\\_dimer\(\)](#) instead

Global **cofold\_par** (const char \*string, char \*structure, vrna\_param\_t \*parameters, int is\_constrained)

use [vrna\\_mfe\\_dimer\(\)](#) instead

Global **compute\_BPdifferences** (short \*pt1, short \*pt2, unsigned int turn)

Use [vrna\\_refBPdist\\_matrix\(\)](#) instead

Global **compute\_probabilities** (double FAB, double FEA, double FEB, vrna\_ep\_t \*prAB, vrna\_ep\_t \*prA,  
vrna\_ep\_t \*prB, int Alength)

{ Use [vrna\\_pf\\_dimer\\_probs\(\)](#) instead!}

Global **constrain\_ptypes** (const char \*constraint, unsigned int length, char \*ptype, int \*BP, int min\_loop↵  
\_size, unsigned int idx\_type)

Do not use this function anymore! Structure constraints are now handled through [vrna\\_hc\\_t](#) and related functions.

File **constraints.h**

Use [ViennaRNA/constraints/basic.h](#) instead

File **constraints\_hard.h**

Use [ViennaRNA/constraints/hard.h](#) instead

File **constraints\_ligand.h**

Use [ViennaRNA/constraints/ligand.h](#) instead

File **constraints\_SHAPE.h**

Use [ViennaRNA/constraints/SHAPE.h](#) instead

File **constraints\_soft.h**

Use [ViennaRNA/constraints/soft.h](#) instead

File **convert\_epars.h**

Use [ViennaRNA/params/convert.h](#) instead

Global **copy\_pair\_table** (const short \*pt)

Use [vrna\\_ptable\\_copy\(\)](#) instead

Global **cpair**

Use [vrna\\_cpair\\_t](#) instead!

**Global `cv_fact`**

See `vrna_md_t.cv_fact`, and `vrna_mfe()` to avoid using global variables

**File `data_structures.h`**

Use `ViennaRNA/datastructures/basic.h` instead

**Global `destroy_TwoDfold_variables` (`TwoDfold_vars` \*our\_variables)**

Use the new API that relies on `vrna_fold_compound_t` and the corresponding functions `vrna_fold_compound↵_TwoD()`, `vrna_mfe_TwoD()`, and `vrna_fold_compound_free()` instead!

**Global `destroy_TwoDpfold_variables` (`TwoDpfold_vars` \*vars)**

Use the new API that relies on `vrna_fold_compound_t` and the corresponding functions `vrna_fold_compound↵_TwoD()`, `vrna_pf_TwoD()`, and `vrna_fold_compound_free()` instead!

**Global `E_Stem` (int type, int si1, int sj1, int extLoop, vrna\_param\_t \*P)**

Please use one of the functions `vrna_E_ext_stem()` and `E_MLstem()` instead! Use the former for cases where `extLoop != 0` and the latter otherwise.

**File `energy_const.h`**

Use `ViennaRNA/params/constants.h` instead

**Global `energy_of_alistruct` (const char \*\*sequences, const char \*structure, int n\_seq, float \*energy)**

Usage of this function is discouraged! Use `vrna_eval_structure()`, and `vrna_eval_covar_structure()` instead!

**Global `energy_of_circ_struct` (const char \*string, const char \*structure)**

This function is deprecated and should not be used in future programs Use `energy_of_circ_structure()` instead!

**Global `energy_of_circ_struct_par` (const char \*string, const char \*structure, vrna\_param\_t \*parameters, int verbosity\_level)**

Use `vrna_eval_structure()` or `vrna_eval_structure_verbose()` instead!

**Global `energy_of_circ_structure` (const char \*string, const char \*structure, int verbosity\_level)**

Use `vrna_eval_structure()` or `vrna_eval_structure_verbose()` instead!

**Global `energy_of_move` (const char \*string, const char \*structure, int m1, int m2)**

Use `vrna_eval_move()` instead!

**Global `energy_of_move_pt` (short \*pt, short \*s, short \*s1, int m1, int m2)**

Use `vrna_eval_move_pt()` instead!

**Global `energy_of_struct` (const char \*string, const char \*structure)**

This function is deprecated and should not be used in future programs! Use `energy_of_structure()` instead!

**Global `energy_of_struct_par` (const char \*string, const char \*structure, vrna\_param\_t \*parameters, int verbosity\_level)**

Use `vrna_eval_structure()` or `vrna_eval_structure_verbose()` instead!

**Global `energy_of_struct_pt` (const char \*string, short \*ptable, short \*s, short \*s1)**

This function is deprecated and should not be used in future programs! Use `energy_of_structure_pt()` instead!

**Global `energy_of_struct_pt_par` (const char \*string, short \*ptable, short \*s, short \*s1, vrna\_param\_t↵ \*parameters, int verbosity\_level)**

Use `vrna_eval_structure_pt()` or `vrna_eval_structure_pt_verbose()` instead!

**Global `energy_of_structure` (const char \*string, const char \*structure, int verbosity\_level)**

Use `vrna_eval_structure()` or `vrna_eval_structure_verbose()` instead!

**Global `energy_of_structure_pt` (const char \*string, short \*ptable, short \*s, short \*s1, int verbosity\_level)**

Use `vrna_eval_structure_pt()` or `vrna_eval_structure_pt_verbose()` instead!

**File `energy_par.h`**

Use `ViennaRNA/params/default.h` instead

**Global `exp_E_ExtLoop` (int type, int si1, int sj1, vrna\_exp\_param\_t \*P)**

Use `vrna_exp_E_ext_stem()` instead!

Global **expHairpinEnergy** (int u, int type, short si1, short sj1, const char \*string)

Use [exp\\_E\\_Hairpin\(\)](#) from [loop\\_energies.h](#) instead

Global **expLoopEnergy** (int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short sq1)

Use [exp\\_E\\_IntLoop\(\)](#) from [loop\\_energies.h](#) instead

Global **export\_ali\_bppm** (void)

Usage of this function is discouraged! The new [vrna\\_fold\\_compound\\_t](#) allows direct access to the folding matrices, including the pair probabilities! The pair probability array returned here reflects the one of the latest call to [vrna\\_pf\(\)](#), or any of the old API calls for consensus structure partition function folding.

Global **export\_circfold\_arrays** (int \*Fc\_p, int \*FcH\_p, int \*Fcl\_p, int \*FcM\_p, int \*\*fM2\_p, int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*indx\_p, char \*\*ptype\_p)

See [vrna\\_mfe\(\)](#) and [vrna\\_fold\\_compound\\_t](#) for the usage of the new API!

Global **export\_circfold\_arrays\_par** (int \*Fc\_p, int \*FcH\_p, int \*Fcl\_p, int \*FcM\_p, int \*\*fM2\_p, int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*indx\_p, char \*\*ptype\_p, vrna\_param\_t \*\*P\_p)

See [vrna\\_mfe\(\)](#) and [vrna\\_fold\\_compound\\_t](#) for the usage of the new API!

Global **export\_co\_bppm** (void)

This function is deprecated and will be removed soon! The base pair probability array is available through the [vrna\\_fold\\_compound\\_t](#) data structure, and its associated [vrna\\_mx\\_pf\\_t](#) member.

Global **export\_cofold\_arrays** (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*fc\_p, int \*\*indx\_p, char \*\*ptype\_p)

folding matrices now reside within the [vrna\\_fold\\_compound\\_t](#). Thus, this function will only work in conjunction with a prior call to the deprecated functions [cofold\(\)](#) or [cofold\\_par\(\)](#)

Global **export\_cofold\_arrays\_gg** (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*fc\_p, int \*\*ggg\_p, int \*\*indx\_p, char \*\*ptype\_p)

folding matrices now reside within the fold compound. Thus, this function will only work in conjunction with a prior call to [cofold\(\)](#) or [cofold\\_par\(\)](#)

Global **export\_fold\_arrays** (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*indx\_p, char \*\*ptype\_p)

See [vrna\\_mfe\(\)](#) and [vrna\\_fold\\_compound\\_t](#) for the usage of the new API!

Global **export\_fold\_arrays\_par** (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*indx\_p, char \*\*ptype\_p, vrna\_param\_t \*\*P\_p)

See [vrna\\_mfe\(\)](#) and [vrna\\_fold\\_compound\\_t](#) for the usage of the new API!

File **exterior\_loops.h**

Use [ViennaRNA/loops/external.h](#) instead

File **file\_formats.h**

Use [ViennaRNA/io/file\\_formats.h](#) instead

File **file\_formats\_msa.h**

Use [ViennaRNA/io/file\\_formats\\_msa.h](#) instead

File **file\_utils.h**

Use [ViennaRNA/io/utils.h](#) instead

Global **filecopy** (FILE \*from, FILE \*to)

Use [vrna\\_file\\_copy\(\)](#) instead!

Global **fold** (const char \*sequence, char \*structure)

use [vrna\\_fold\(\)](#), or [vrna\\_mfe\(\)](#) instead!

Global **fold\_par** (const char \*sequence, char \*structure, vrna\_param\_t \*parameters, int is\_constrained, int is\_circular)

use [vrna\\_mfe\(\)](#) instead!

Global **free\_alifold\_arrays** (void)

Usage of this function is discouraged! It only affects memory being free'd that was allocated by an old API function before. Release of memory occupied by the newly introduced [vrna\\_fold\\_compound\\_t](#) is handled by [vrna\\_fold\\_compound\\_free\(\)](#)



**Global [free\\_alipf\\_arrays](#) (void)**

Usage of this function is discouraged! This function only free's memory allocated by old API function calls. Memory allocated by any of the new API calls (starting with `vrna_`) will be not affected!

**Global [free\\_arrays](#) (void)**

See [vrna\\_fold\(\)](#), [vrna\\_circfold\(\)](#), or [vrna\\_mfe\(\)](#) and [vrna\\_fold\\_compound\\_t](#) for the usage of the new API!

**Global [free\\_co\\_arrays](#) (void)**

This function will only free memory allocated by a prior call of [cofold\(\)](#) or [cofold\\_par\(\)](#). See [vrna\\_mfe\\_dimer\(\)](#) for how to use the new API

**Global [free\\_co\\_pf\\_arrays](#) (void)**

This function will be removed for the new API soon! See [vrna\\_pf\\_dimer\(\)](#), [vrna\\_fold\\_compound\(\)](#), and [vrna\\_fold\\_compound\\_free\(\)](#) for an alternative

**Global [free\\_pf\\_arrays](#) (void)**

See [vrna\\_fold\\_compound\\_t](#) and its related functions for how to free memory occupied by the dynamic programming matrices

**Global [get\\_alipf\\_arrays](#) (short \*\*\*S\_p, short \*\*\*S5\_p, short \*\*\*S3\_p, unsigned short \*\*\*a2s\_p, char \*\*\*Ss\_p, FLT\_OR\_DBL \*\*qb\_p, FLT\_OR\_DBL \*\*qm\_p, FLT\_OR\_DBL \*\*q1k\_p, FLT\_OR\_DBL \*\*qln\_p, short \*\*pscore)**

It is discouraged to use this function! The new [vrna\\_fold\\_compound\\_t](#) allows direct access to all necessary consensus structure prediction related variables!

**Global [get\\_boltzmann\\_factor\\_copy](#) (vrna\_exp\_param\_t \*parameters)**

Use [vrna\\_exp\\_params\\_copy\(\)](#) instead!

**Global [get\\_boltzmann\\_factors](#) (double temperature, double betaScale, vrna\_md\_t md, double pf\_scale)**

Use [vrna\\_exp\\_params\(\)](#) instead!

**Global [get\\_boltzmann\\_factors\\_al](#) (unsigned int n\_seq, double temperature, double betaScale, vrna\_md\_t md, double pf\_scale)**

Use [vrna\\_exp\\_params\\_comparative\(\)](#) instead!

**Global [get\\_centroid\\_struct\\_gquad\\_pr](#) (int length, double \*dist)**

This function is deprecated and should not be used anymore as it is not threadsafe!

**Global [get\\_centroid\\_struct\\_pl](#) (int length, double \*dist, vrna\_ep\_t \*pl)**

This function was renamed to [vrna\\_centroid\\_from\\_plist\(\)](#)

**Global [get\\_centroid\\_struct\\_pr](#) (int length, double \*dist, FLT\_OR\_DBL \*pr)**

This function was renamed to [vrna\\_centroid\\_from\\_probs\(\)](#)

**Global [get\\_concentrations](#) (double FEAB, double FEAA, double FEBB, double FEA, double FEB, double \*startconc)**

{ Use [vrna\\_pf\\_dimer\\_concentrations\(\)](#) instead! }

**Global [get\\_line](#) (FILE \*fp)**

Use [vrna\\_read\\_line\(\)](#) as a substitute!

**Global [get\\_monomere\\_mfes](#) (float \*e1, float \*e2)**

{ This function is obsolete and will be removed soon! }

**Global [get\\_mpi](#) (char \*Aseq[], int n\_seq, int length, int \*mini)**

Use [vrna\\_aln\\_mpi\(\)](#) as a replacement

**Global [get\\_plist](#) (vrna\_ep\_t \*pl, int length, double cut\_off)**

{ This function is deprecated and will be removed soon! } use [assign\\_plist\\_from\\_pr\(\)](#) instead!

**Global [get\\_scaled\\_alipf\\_parameters](#) (unsigned int n\_seq)**

Use [vrna\\_exp\\_params\\_comparative\(\)](#) instead!

**Global [get\\_scaled\\_parameters](#) (double temperature, vrna\_md\_t md)**

Use [vrna\\_params\(\)](#) instead!

**Global [get\\_scaled\\_pf\\_parameters](#) (void)**

Use [vrna\\_exp\\_params\(\)](#) instead!

**Global [get\\_TwoDfold\\_variables](#) (const char \*seq, const char \*structure1, const char \*structure2, int circ)**

Use the new API that relies on [vrna\\_fold\\_compound\\_t](#) and the corresponding functions [vrna\\_fold\\_compound↔\\_TwoD\(\)](#), [vrna\\_mfe\\_TwoD\(\)](#), and [vrna\\_fold\\_compound\\_free\(\)](#) instead!

**Global [get\\_TwoDpfold\\_variables](#) (const char \*seq, const char \*structure1, char \*structure2, int circ)**

Use the new API that relies on [vrna\\_fold\\_compound\\_t](#) and the corresponding functions [vrna\\_fold\\_compound↔\\_TwoD\(\)](#), [vrna\\_pf\\_TwoD\(\)](#), and [vrna\\_fold\\_compound\\_free\(\)](#) instead!

**File [hairpin\\_loops.h](#)**

Use [ViennaRNA/loops/hairpin.h](#) instead

**Global [HairpinE](#) (int size, int type, int si1, int sj1, const char \*string)**

{This function is deprecated and will be removed soon. Use [E\\_Hairpin\(\)](#) instead!}

**Global [hamming](#) (const char \*s1, const char \*s2)**

Use [vrna\\_hamming\\_distance\(\)](#) instead!

**Global [hamming\\_bound](#) (const char \*s1, const char \*s2, int n)**

Use [vrna\\_hamming\\_distance\\_bound\(\)](#) instead!

**Global [iindx](#)**

Do not use this variable anymore!

**Global [init\\_co\\_pf\\_fold](#) (int length)**

{ This function is deprecated and will be removed soon!}

**Global [init\\_pf\\_fold](#) (int length)**

This function is obsolete and will be removed soon!

**Global [init\\_rand](#) (void)**

Use [vrna\\_init\\_rand\(\)](#) instead!

**Global [initialize\\_cofold](#) (int length)**

{This function is obsolete and will be removed soon!}

**Global [initialize\\_fold](#) (int length)**

See [vrna\\_mfe\(\)](#) and [vrna\\_fold\\_compound\\_t](#) for the usage of the new API!

**Global [int\\_urn](#) (int from, int to)**

Use [vrna\\_int\\_urn\(\)](#) instead!

**File [interior\\_loops.h](#)**

Use [ViennaRNA/loops/internal.h](#) instead

**Global [Lfold](#) (const char \*string, const char \*structure, int maxdist)**

Use [vrna\\_mfe\\_window\(\)](#) instead!

**Global [Lfoldz](#) (const char \*string, const char \*structure, int maxdist, int zsc, double min\_z)**

Use [vrna\\_mfe\\_window\\_zscore\(\)](#) instead!

**File [loop\\_energies.h](#)**

Use [ViennaRNA/loops/all.h](#) instead

**Global [loop\\_energy](#) (short \*ptable, short \*s, short \*s1, int i)**

Use [vrna\\_eval\\_loop\\_pt\(\)](#) instead!

**Global [LoopEnergy](#) (int n1, int n2, int type, int type\_2, int si1, int sj1, int sp1, int sq1)**

{This function is deprecated and will be removed soon. Use [E\\_IntLoop\(\)](#) instead!}

**Global [Make\\_bp\\_profile](#) (int length)**

This function is deprecated and will be removed soon! See [Make\\_bp\\_profile\\_bppm\(\)](#) for a replacement

Global **make\_pair\_table** (const char \*structure)

Use [vrna\\_ptable\(\)](#) instead

Global **make\_pair\_table\_snoop** (const char \*structure)

Use [vrna\\_pt\\_snoop\\_get\(\)](#) instead!

Global **make\_referenceBP\_array** (short \*reference\_pt, unsigned int turn)

Use [vrna\\_refBPcnt\\_matrix\(\)](#) instead

Global **mean\_bp\_dist** (int length)

This function is not threadsafe and should not be used anymore. Use [mean\\_bp\\_distance\(\)](#) instead!

Global **mean\_bp\_distance** (int length)

Use [vrna\\_mean\\_bp\\_distance\(\)](#) or [vrna\\_mean\\_bp\\_distance\\_pr\(\)](#) instead!

Global **mean\_bp\_distance\_pr** (int length, FLT\_OR\_DBL \*pr)

Use [vrna\\_mean\\_bp\\_distance\(\)](#) or [vrna\\_mean\\_bp\\_distance\\_pr\(\)](#) instead!

File **multibranch\_loops.h**

Use [ViennaRNA/loops/multibranch.h](#) instead

File **naview.h**

Use [ViennaRNA/plotting/naview.h](#) instead

Global **nc\_fact**

See [vrna\\_md\\_t.nc\\_fact](#), and [vrna\\_mfe\(\)](#) to avoid using global variables

Global **nrerror** (const char message[])

Use [vrna\\_message\\_error\(\)](#) instead!

Global **pack\_structure** (const char \*struc)

Use [vrna\\_db\\_pack\(\)](#) as a replacement

Global **PAIR**

Use [vrna\\_basepair\\_t](#) instead!

Global **pair\_info**

Use [vrna\\_pinfo\\_t](#) instead!

File **params.h**

Use [ViennaRNA/params/basic.h](#) instead

Global **paramT**

Use [vrna\\_param\\_t](#) instead!

Global **parenthesis\_structure** (char \*structure, vrna\_bp\_stack\_t \*bp, int length)

use [vrna\\_parenthesis\\_structure\(\)](#) instead

Global **parenthesis\_zuker** (char \*structure, vrna\_bp\_stack\_t \*bp, int length)

use [vrna\\_parenthesis\\_zuker](#) instead

Global **path\_t**

Use [vrna\\_path\\_t](#) instead!

Global **pbacktrack\_circ** (char \*sequence)

Use [vrna\\_pbacktrack\(\)](#) instead.

Global **pf\_circ\_fold** (const char \*sequence, char \*structure)

Use [vrna\\_pf\(\)](#) instead!

Global **pf\_fold\_par** (const char \*sequence, char \*structure, vrna\_exp\_param\_t \*parameters, int calculate↵  
\_bppm, int is\_constrained, int is\_circular)

Use [vrna\\_pf\(\)](#) instead

Global **pf\_paramT**

Use [vrna\\_exp\\_param\\_t](#) instead!

**Global `plist`**

Use [vrna\\_ep\\_t](#) or [vrna\\_elem\\_prob\\_s](#) instead!

**File `plot_aln.h`**

Use [ViennaRNA/plotting/alignments.h](#) instead

**File `plot_layouts.h`**

Use [ViennaRNA/plotting/layouts.h](#) instead

**File `plot_structure.h`**

Use [ViennaRNA/plotting/structures.h](#) instead

**File `plot_utils.h`**

Use [ViennaRNA/plotting/utils.h](#) instead

**Global `pr`**

Do not use this variable anymore!

**Global `print_tty_constraint` (unsigned int option)**

Use [vrna\\_message\\_constraints\(\)](#) instead!

**Global `print_tty_constraint_full` (void)**

Use [vrna\\_message\\_constraint\\_options\\_all\(\)](#) instead!

**Global `print_tty_input_seq` (void)**

Use [vrna\\_message\\_input\\_seq\\_simple\(\)](#) instead!

**Global `print_tty_input_seq_str` (const char \*s)**

Use [vrna\\_message\\_input\\_seq\(\)](#) instead!

**File `PS_dot.h`**

Use [ViennaRNA/plotting/probabilities.h](#) instead

**Global `PS_dot_plot` (char \*string, char \*file)**

This function is deprecated and will be removed soon! Use [PS\\_dot\\_plot\\_list\(\)](#) instead!

**Global `PS_rna_plot` (char \*string, char \*structure, char \*file)**

Use [vrna\\_file\\_PS\\_rnaplot\(\)](#) instead!

**Global `PS_rna_plot_a` (char \*string, char \*structure, char \*file, char \*pre, char \*post)**

Use [vrna\\_file\\_PS\\_rnaplot\\_a\(\)](#) instead!

**Global `PS_rna_plot_a_gquad` (char \*string, char \*structure, char \*ssfile, char \*pre, char \*post)**

Use [vrna\\_file\\_PS\\_rnaplot\\_a\(\)](#) instead!

**Global `random_string` (int l, const char symbols[])**

Use [vrna\\_random\\_string\(\)](#) instead!

**File `read_epars.h`**

Use [ViennaRNA/params/io.h](#) instead

**Global `read_record` (char \*\*header, char \*\*sequence, char \*\*\*rest, unsigned int options)**

This function is deprecated! Use [vrna\\_file\\_fasta\\_read\\_record\(\)](#) as a replacment.

**Global `scale_parameters` (void)**

Use [vrna\\_params\(\)](#) instead!

**Global `sect`**

Use [vrna\\_sect\\_t](#) instead!

**Global `set_model_details` (vrna\_md\_t \*md)**

This function will vanish as soon as backward compatibility of RNAlib is dropped (expected in version 3). Use [vrna\\_md\\_set\\_default\(\)](#) instead!

**Global `SOLUTION`**

Use [vrna\\_subopt\\_solution\\_t](#) instead!

**Global `space` (unsigned size)**

Use `vrna_alloc()` instead!

**Global `st_back`**

set the `uniq_ML` flag in `vrna_md_t` before passing it to `vrna_fold_compound()`.

**Global `stackProb` (double cutoff)**

Use `vrna_stack_prob()` instead!

**Global `str_DNA2RNA` (char \*sequence)**

Use `vrna_seq_toRNA()` instead!

**Global `str_uppercase` (char \*sequence)**

Use `vrna_seq_toupper()` instead!

**File `stream_output.h`**

Use `ViennaRNA/datastructures/stream_output.h` instead

**File `string_utils.h`**

Use `ViennaRNA/utills/strings.h` instead

**File `structure_utils.h`**

Use `ViennaRNA/utills/structures.h` instead

**File `svm_utils.h`**

Use `ViennaRNA/utills/svm.h` instead

**Global `temperature`**

Use `vrna_md_defaults_temperature()`, and `vrna_md_defaults_temperature_get()` to change, and read the global default temperature settings

**Global `time_stamp` (void)**

Use `vrna_time_stamp()` instead!

**Global `TwoDfold_backtrack_f5` (unsigned int j, int k, int l, `TwoDfold_vars` \*vars)**

Use the new API that relies on `vrna_fold_compound_t` and the corresponding functions `vrna_fold_compound↔_TwoD()`, `vrna_mfe_TwoD()`, `vrna_backtrack5_TwoD()`, and `vrna_fold_compound_free()` instead!

**Global `TwoDfold_vars`**

This data structure will be removed from the library soon! Use `vrna_fold_compound_t` and the corresponding functions `vrna_fold_compound_TwoD()`, `vrna_mfe_TwoD()`, and `vrna_fold_compound_free()` instead!

**Global `TwoDfoldList` (`TwoDfold_vars` \*vars, int distance1, int distance2)**

Use the new API that relies on `vrna_fold_compound_t` and the corresponding functions `vrna_fold_compound↔_TwoD()`, `vrna_mfe_TwoD()`, and `vrna_fold_compound_free()` instead!

**Global `TwoDpfold_pbacktrack` (`TwoDpfold_vars` \*vars, int d1, int d2)**

Use the new API that relies on `vrna_fold_compound_t` and the corresponding functions `vrna_fold_compound↔_TwoD()`, `vrna_pf_TwoD()`, `vrna_pbacktrack_TwoD()`, and `vrna_fold_compound_free()` instead!

**Global `TwoDpfold_pbacktrack5` (`TwoDpfold_vars` \*vars, int d1, int d2, unsigned int length)**

Use the new API that relies on `vrna_fold_compound_t` and the corresponding functions `vrna_fold_compound↔_TwoD()`, `vrna_pf_TwoD()`, `vrna_pbacktrack5_TwoD()`, and `vrna_fold_compound_free()` instead!

**Class `TwoDpfold_vars`**

This data structure will be removed from the library soon! Use `vrna_fold_compound_t` and the corresponding functions `vrna_fold_compound_TwoD()`, `vrna_pf_TwoD()`, and `vrna_fold_compound_free()` instead!

**Global `TwoDpfoldList` (`TwoDpfold_vars` \*vars, int maxDistance1, int maxDistance2)**

Use the new API that relies on `vrna_fold_compound_t` and the corresponding functions `vrna_fold_compound↔_TwoD()`, `vrna_pf_TwoD()`, and `vrna_fold_compound_free()` instead!

**Global `unpack_structure` (const char \*packed)**

Use `vrna_db_unpack()` as a replacement

**Global `update_alifold_params` (void)**

Usage of this function is discouraged! The new API uses `vrna_fold_compound_t` to lump all folding related necessities together, including the energy parameters. Use `vrna_update_fold_params()` to update the energy parameters within a `vrna_fold_compound_t`.

**Global `update_co_pf_params` (int length)**

Use `vrna_exp_params_subst()` instead!

**Global `update_co_pf_params_par` (int length, `vrna_exp_param_t` \*parameters)**

Use `vrna_exp_params_subst()` instead!

**Global `update_cofold_params` (void)**

See `vrna_params_subst()` for an alternative using the new API

**Global `update_cofold_params_par` (`vrna_param_t` \*parameters)**

See `vrna_params_subst()` for an alternative using the new API

**Global `update_fold_params` (void)**

For non-default model settings use the new API with `vrna_params_subst()` and `vrna_mfe()` instead!

**Global `update_fold_params_par` (`vrna_param_t` \*parameters)**

For non-default model settings use the new API with `vrna_params_subst()` and `vrna_mfe()` instead!

**Global `update_pf_params` (int length)**

Use `vrna_exp_params_subst()` instead

**Global `update_pf_params_par` (int length, `vrna_exp_param_t` \*parameters)**

Use `vrna_exp_params_subst()` instead

**Global `urn` (void)**

Use `vrna_urn()` instead!

**File `utils.h`**

Use `ViennaRNA/utils/basic.h` instead

**Global `VRNA_CONSTRAINT_FILE`**

Use 0 instead!

**Global `VRNA_CONSTRAINT_MULTILINE`**

see `vrna_extract_record_rest_structure()`

**Global `VRNA_CONSTRAINT_NO_HEADER`**

This mode is not supported anymore!

**Global `VRNA_CONSTRAINT_SOFT_MFE`**

This flag has no meaning anymore, since constraints are now always stored!

**Global `VRNA_CONSTRAINT_SOFT_PF`**

Use `VRNA_OPTION_PF` instead!

**Global `vrna_exp_param_s::id`**

This attribute will be removed in version 3

**Global `vrna_extract_record_rest_constraint` (char \*\*cstruc, const char \*\*lines, unsigned int option)**

Use `vrna_extract_record_rest_structure()` instead!

**Global `vrna_fc_s::pscore_pf_compat`**

This attribute will vanish in the future!

**Global `vrna_fc_s::ptype_pf_compat`**

This attribute will vanish in the future! It's meant for backward compatibility only!

**Global `warn_user` (const char message[])**

Use `vrna_message_warning()` instead!

Global **xrealloc** (void \*p, unsigned size)

Use **vrna\_realloc**() instead!

Global **zukersubopt** (const char \*string)

use **vrna\_zukersubopt**() instead

Global **zukersubopt\_par** (const char \*string, vrna\_param\_t \*parameters)

use **vrna\_zukersubopt**() instead





# Chapter 11

## Bug List

### Module [domains\\_up](#)

Although the additional production rule(s) for unstructured domains as described in [Unstructured Domains](#) are always treated as 'segments possibly bound to one or more ligands', the current implementation requires that at least one ligand is bound. The default implementation already takes care of the required changes, however, upon using callback functions other than the default ones, one has to take care of this fact. Please also note, that this behavior might change in one of the next releases, such that the decomposition schemes as shown above comply with the actual implementation.

### Global [VRNA\\_PROBS\\_WINDOW\\_STACKP](#)

Currently, this flag is a placeholder doing nothing as the corresponding implementation for stack probability computation is missing.

### Global [vrna\\_subopt\\_zuker](#) ([vrna\\_fold\\_compound\\_t](#) \*vc)

Due to resizing, any pre-existing constraints will be lost!



## Chapter 12

# Module Index

### 12.1 The RNAlib API

Our library is grouped into several modules, each addressing different aspects of RNA secondary structure related problems. You can find an overview of the different groups below.

Free Energy Evaluation . . . . .	95
Energy Evaluation for Individual Loops . . . . .	121
Exterior Loops . . . . .	364
Hairpin Loops . . . . .	368
Internal Loops . . . . .	373
Multibranch Loops . . . . .	374
Energy Evaluation for Atomic Moves . . . . .	124
Deprecated Interface for Free Energy Evaluation . . . . .	126
The RNA Folding Grammar . . . . .	141
Fine-tuning of the Implemented Models . . . . .	142
Energy Parameters . . . . .	178
Reading/Writing Energy Parameter Sets from/to File . . . . .	417
Converting Energy Parameter Files . . . . .	419
Extending the Folding Grammar with Additional Domains . . . . .	192
Unstructured Domains . . . . .	193
Structured Domains . . . . .	205
Constraining the RNA Folding Grammar . . . . .	206
Hard Constraints . . . . .	222
Soft Constraints . . . . .	234
The RNA Secondary Structure Landscape . . . . .	247
Neighborhood Relation and Move Sets for Secondary Structures . . . . .	325
Refolding Paths of Secondary Structures . . . . .	333
Direct Refolding Paths between two Secondary Structures . . . . .	425
Minimum Free Energy (MFE) Algorithms . . . . .	248
Global MFE Prediction . . . . .	251
Computing MFE representatives of a Distance Based Partitioning . . . . .	307
Deprecated Interface for Global MFE Prediction . . . . .	376
Local (sliding window) MFE Prediction . . . . .	258
Deprecated Interface for Local (Sliding Window) MFE Prediction . . . . .	389
Backtracking MFE structures . . . . .	263
Partition Function and Equilibrium Properties . . . . .	249
Global Partition Function and Equilibrium Probabilities . . . . .	265

Computing Partition Functions of a Distance Based Partitioning . . . . .	315
Deprecated Interface for Global Partition Function Computation . . . . .	390
Local (sliding window) Partition Function and Equilibrium Probabilities . . . . .	277
Deprecated Interface for Local (Sliding Window) Partition Function Computation . . . . .	408
Suboptimals and Representative Structures . . . . .	287
Suboptimal Structures sensu Stiegler et al. 1984 / Zuker et al. 1989 . . . . .	288
Suboptimal Structures within an Energy Band around the MFE . . . . .	290
Random Structure Samples from the Ensemble . . . . .	295
Stochastic Backtracking of Structures from Distance Based Partitioning . . . . .	318
Compute the Structure with Maximum Expected Accuracy (MEA) . . . . .	300
Compute the Centroid Structure . . . . .	301
RNA-RNA Interaction . . . . .	304
Partition Function for Two Hybridized Sequences . . . . .	411
Partition Function for two Hybridized Sequences as a Stepwise Process . . . . .	414
Classified Dynamic Programming Variants . . . . .	305
Distance Based Partitioning of the Secondary Structure Space . . . . .	306
Computing MFE representatives of a Distance Based Partitioning . . . . .	307
Computing Partition Functions of a Distance Based Partitioning . . . . .	315
Stochastic Backtracking of Structures from Distance Based Partitioning . . . . .	318
Compute the Density of States . . . . .	321
Inverse Folding (Design) . . . . .	322
Experimental Structure Probing Data . . . . .	338
SHAPE Reactivity Data . . . . .	339
Generate Soft Constraints from Data . . . . .	343
Ligands Binding to RNA Structures . . . . .	348
Ligands Binding to Unstructured Domains . . . . .	349
Incorporating Ligands Binding to Specific Sequence/Structure Motifs using Soft Constraints . . . . .	350
Complex Structured Modules . . . . .	352
G-Quadruplexes . . . . .	353
Utilities . . . . .	356
Utilities to deal with Nucleotide Alphabets . . . . .	433
(Nucleic Acid Sequence) String Utilitites . . . . .	437
Secondary Structure Utilities . . . . .	446
Dot-Bracket Notation of Secondary Structures . . . . .	450
Pair Table Representation of Secondary Structures . . . . .	457
Pair List Representation of Secondary Structures . . . . .	460
Helix List Representation of Secondary Structures . . . . .	462
Tree Representation of Secondary Structures . . . . .	464
Deprecated Interface for Secondary Structure Utilities . . . . .	470
Multiple Sequence Alignment Utilities . . . . .	481
Deprecated Interface for Multiple Sequence Alignment Utilities . . . . .	491
Files and I/O . . . . .	495
Nucleic Acid Sequences and Structures . . . . .	498
Multiple Sequence Alignments . . . . .	506
Command Files . . . . .	516
Plotting . . . . .	521
Annotation . . . . .	533
Search Algorithms . . . . .	534
Combinatorics Algorithms . . . . .	538
(Abstract) Data Structures . . . . .	544
The Fold Compound . . . . .	561
The Dynamic Programming Matrices . . . . .	578
Hash Tables . . . . .	583
Buffers . . . . .	594
Messages . . . . .	551
Unit Conversion . . . . .	557

## Chapter 13

# Data Structure Index

### 13.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">_struct_en</a>	
Data structure for <a href="#">energy_of_move()</a>	601
<a href="#">LIST</a>	601
<a href="#">LST_BUCKET</a>	601
<a href="#">Postorder_list</a>	
Postorder data structure	602
<a href="#">swString</a>	
Some other data structure	602
<a href="#">Tree</a>	
<a href="#">Tree</a> data structure	602
<a href="#">TwoDpfold_vars</a>	
Variables compound for 2Dfold partition function folding	603
<a href="#">vrna_dimer_conc_s</a>	
Data structure for concentration dependency computations	604
<a href="#">vrna_hc_bp_storage_t</a>	
A base pair hard constraint	604
<a href="#">vrna_sc_bp_storage_t</a>	
A base pair constraint	604
<a href="#">vrna_sc_motif_s</a>	605
<a href="#">vrna_structured_domains_s</a>	605
<a href="#">vrna_subopt_sol_s</a>	
Solution element from subopt.c	605
<a href="#">vrna_unstructured_domain_motif_s</a>	605



## Chapter 14

# File Index

### 14.1 File List

Here is a list of all documented files with brief descriptions:

ViennaRNA/2Dfold.h	
MFE structures for base pair distance classes . . . . .	607
ViennaRNA/2Dpfold.h	
Partition function implementations for base pair distance classes . . . . .	608
ViennaRNA/ali_plex.h . . . . .	??
ViennaRNA/alifold.h	
Functions for comparative structure prediction using RNA sequence alignments . . . . .	612
ViennaRNA/aln_util.h	
Use ViennaRNA/utis/alignments.h instead . . . . .	615
ViennaRNA/alphabet.h	
Functions to process, convert, and generally handle different nucleotide and/or base pair alphabets . . . . .	615
ViennaRNA/boltzmann_sampling.h	
Boltzmann Sampling of secondary structures from the ensemble . . . . .	616
ViennaRNA/centroid.h	
Centroid structure computation . . . . .	616
ViennaRNA/char_stream.h	
Use ViennaRNA/datastructures/char_stream.h instead . . . . .	617
ViennaRNA/cofold.h	
MFE implementations for RNA-RNA interaction . . . . .	618
ViennaRNA/combinatorics.h	
Various implementations that deal with combinatorial aspects of objects . . . . .	619
ViennaRNA/commands.h	
Parse and apply different commands that alter the behavior of secondary structure prediction and evaluation . . . . .	620
ViennaRNA/concentrations.h	
Concentration computations for RNA-RNA interactions . . . . .	621
ViennaRNA/constraints.h	
Use ViennaRNA/constraints/basic.h instead . . . . .	622
ViennaRNA/constraints_hard.h	
Use ViennaRNA/constraints/hard.h instead . . . . .	632
ViennaRNA/constraints_ligand.h	
Use ViennaRNA/constraints/ligand.h instead . . . . .	632
ViennaRNA/constraints_SHAPE.h	
Use ViennaRNA/constraints/SHAPE.h instead . . . . .	632

ViennaRNA/ <a href="#">constraints_soft.h</a>	
Use <a href="#">ViennaRNA/constraints/soft.h</a> instead	633
ViennaRNA/ <a href="#">convert_epars.h</a>	
Use <a href="#">ViennaRNA/params/convert.h</a> instead	633
ViennaRNA/ <a href="#">data_structures.h</a>	
Use <a href="#">ViennaRNA/datastructures/basic.h</a> instead	633
ViennaRNA/ <a href="#">dist_vars.h</a>	
Global variables for Distance-Package	635
ViennaRNA/ <a href="#">dp_matrices.h</a>	
Functions to deal with standard dynamic programming (DP) matrices	636
ViennaRNA/ <a href="#">duplex.h</a>	
Functions for simple RNA-RNA duplex interactions	637
ViennaRNA/ <a href="#">edit_cost.h</a>	
Global variables for Edit Costs included by treedist.c and stringdist.c	637
ViennaRNA/ <a href="#">energy_const.h</a>	
Use <a href="#">ViennaRNA/params/constants.h</a> instead	637
ViennaRNA/ <a href="#">energy_par.h</a>	
Use <a href="#">ViennaRNA/params/default.h</a> instead	638
ViennaRNA/ <a href="#">equilibrium_probs.h</a>	
Equilibrium Probability implementations	638
ViennaRNA/ <a href="#">eval.h</a>	
Functions and variables related to energy evaluation of sequence/structure pairs	639
ViennaRNA/ <a href="#">exterior_loops.h</a>	
Use <a href="#">ViennaRNA/loops/external.h</a> instead	642
ViennaRNA/ <a href="#">file_formats.h</a>	
Use <a href="#">ViennaRNA/io/file_formats.h</a> instead	642
ViennaRNA/ <a href="#">file_formats_msa.h</a>	
Use <a href="#">ViennaRNA/io/file_formats_msa.h</a> instead	644
ViennaRNA/ <a href="#">file_utils.h</a>	
Use <a href="#">ViennaRNA/io/utils.h</a> instead	645
ViennaRNA/ <a href="#">findpath.h</a>	
A breadth-first search heuristic for optimal direct folding paths	645
ViennaRNA/ <a href="#">fold.h</a>	
MFE calculations for single RNA sequences	646
ViennaRNA/ <a href="#">fold_compound.h</a>	
The Basic Fold Compound API	647
ViennaRNA/ <a href="#">fold_vars.h</a>	
Here all all declarations of the global variables used throughout RNALib	649
ViennaRNA/ <a href="#">gquad.h</a>	
G-quadruplexes	651
ViennaRNA/ <a href="#">grammar.h</a>	
Implementations for the RNA folding grammar	651
ViennaRNA/ <a href="#">hairpin_loops.h</a>	
Use <a href="#">ViennaRNA/loops/hairpin.h</a> instead	652
ViennaRNA/ <a href="#">interior_loops.h</a>	
Use <a href="#">ViennaRNA/loops/internal.h</a> instead	652
ViennaRNA/ <a href="#">inverse.h</a>	
Inverse folding routines	652
ViennaRNA/ <a href="#">Lfold.h</a>	
Functions for locally optimal MFE structure prediction	653
ViennaRNA/ <a href="#">loop_energies.h</a>	
Use <a href="#">ViennaRNA/loops/all.h</a> instead	653
ViennaRNA/ <a href="#">LPfold.h</a>	
Partition function and equilibrium probability implementation for the sliding window algorithm	658
ViennaRNA/ <a href="#">MEA.h</a>	
Computes a MEA (maximum expected accuracy) structure	658



ViennaRNA/ <a href="#">mfe.h</a>	
Compute Minimum Free energy (MFE) and backtrace corresponding secondary structures from RNA sequence data	659
ViennaRNA/ <a href="#">mfe_window.h</a>	
Compute local Minimum Free Energy (MFE) using a sliding window approach and backtrace corresponding secondary structures	660
ViennaRNA/ <a href="#">mm.h</a>	
Several Maximum Matching implementations	661
ViennaRNA/ <a href="#">model.h</a>	
The model details data structure and its corresponding modifiers	662
ViennaRNA/ <a href="#">move_set.h</a>	??
ViennaRNA/ <a href="#">multibranch_loops.h</a>	
Use <a href="#">ViennaRNA/loops/multibranch.h</a> instead	666
ViennaRNA/ <a href="#">naview.h</a>	
Use <a href="#">ViennaRNA/plotting/naview.h</a> instead	666
ViennaRNA/ <a href="#">neighbor.h</a>	
Methods to compute the neighbors of an RNA secondary structure	667
ViennaRNA/ <a href="#">pair_mat.h</a>	??
ViennaRNA/ <a href="#">params.h</a>	
Use <a href="#">ViennaRNA/params/basic.h</a> instead	668
ViennaRNA/ <a href="#">part_func.h</a>	
Partition function implementations	682
ViennaRNA/ <a href="#">part_func_co.h</a>	
Partition function for two RNA sequences	686
ViennaRNA/ <a href="#">part_func_up.h</a>	
Implementations for accessibility and RNA-RNA interaction as a stepwise process	687
ViennaRNA/ <a href="#">part_func_window.h</a>	
Partition function and equilibrium probability implementation for the sliding window algorithm	687
ViennaRNA/ <a href="#">perturbation_fold.h</a>	
Find a vector of perturbation energies that minimizes the discrepancies between predicted and observed pairing probabilities and the amount of necessary adjustments	689
ViennaRNA/ <a href="#">PKplex.h</a>	??
ViennaRNA/ <a href="#">plex.h</a>	??
ViennaRNA/ <a href="#">plot_aln.h</a>	
Use <a href="#">ViennaRNA/plotting/alignments.h</a> instead	690
ViennaRNA/ <a href="#">plot_layouts.h</a>	
Use <a href="#">ViennaRNA/plotting/layouts.h</a> instead	690
ViennaRNA/ <a href="#">plot_structure.h</a>	
Use <a href="#">ViennaRNA/plotting/structures.h</a> instead	690
ViennaRNA/ <a href="#">plot_utils.h</a>	
Use <a href="#">ViennaRNA/plotting/utils.h</a> instead	691
ViennaRNA/ <a href="#">ProfileAln.h</a>	??
ViennaRNA/ <a href="#">profiledist.h</a>	699
ViennaRNA/ <a href="#">PS_dot.h</a>	
Use <a href="#">ViennaRNA/plotting/probabilities.h</a> instead	701
ViennaRNA/ <a href="#">read_epars.h</a>	
Use <a href="#">ViennaRNA/params/io.h</a> instead	701
ViennaRNA/ <a href="#">ribo.h</a>	
Parse RiboSum Scoring Matrices for Covariance Scoring of Alignments	701
ViennaRNA/ <a href="#">RNAstruct.h</a>	
Parsing and Coarse Graining of Structures	702
ViennaRNA/ <a href="#">sequence.h</a>	
Functions and data structures related to sequence representations ,	703
ViennaRNA/ <a href="#">snofold.h</a>	??
ViennaRNA/ <a href="#">snoop.h</a>	??
ViennaRNA/ <a href="#">special_const.h</a>	??
ViennaRNA/ <a href="#">stream_output.h</a>	
Use <a href="#">ViennaRNA/datastructures/stream_output.h</a> instead	704

ViennaRNA/string_utils.h	
Use <a href="#">ViennaRNA/utls/strings.h</a> instead	705
ViennaRNA/stringdist.h	
Functions for String Alignment	705
ViennaRNA/structure_utils.h	
Use <a href="#">ViennaRNA/utls/structures.h</a> instead	707
ViennaRNA/structured_domains.h	
This module provides interfaces that deal with additional structured domains in the folding grammar	707
ViennaRNA/subopt.h	
RNAsubopt and density of states declarations	707
ViennaRNA/svm_utils.h	
Use <a href="#">ViennaRNA/utls/svm.h</a> instead	709
ViennaRNA/treedist.h	
Functions for Tree Edit Distances	709
ViennaRNA/ugly_bt.h	??
ViennaRNA/units.h	
Physical Units and Functions to convert them into each other	711
ViennaRNA/unstructured_domains.h	
Functions to modify unstructured domains, e.g. to incorporate ligands binding to unpaired stretches	712
ViennaRNA/utls.h	
Use <a href="#">ViennaRNA/utls/basic.h</a> instead	714
ViennaRNA/vrna_config.h	??
ViennaRNA/walk.h	
Methods to generate particular paths such as gradient or random walks through the energy landscape of an RNA sequence	718
ViennaRNA/constraints/basic.h	
Functions and data structures for constraining secondary structure predictions and evaluation	671
ViennaRNA/constraints/hard.h	
Functions and data structures for handling of secondary structure hard constraints	623
ViennaRNA/constraints/ligand.h	
Functions for incorporation of ligands binding to hairpin and interior loop motifs using the soft constraints framework	628
ViennaRNA/constraints/SHAPE.h	
This module provides function to incorporate SHAPE reactivity data into the folding recursions by means of soft constraints	628
ViennaRNA/constraints/soft.h	
Functions and data structures for secondary structure soft constraints	630
ViennaRNA/datastructures/basic.h	
Various data structures and pre-processor macros	678
ViennaRNA/datastructures/char_stream.h	
Implementation of a dynamic, buffered character stream	618
ViennaRNA/datastructures/hash_tables.h	
Implementations of hash table functions	634
ViennaRNA/datastructures/lists.h	??
ViennaRNA/datastructures/stream_output.h	
An implementation of a buffered, ordered stream output data structure	704
ViennaRNA/io/file_formats.h	
Read and write different file formats for RNA sequences, structures	643
ViennaRNA/io/file_formats_msa.h	
Functions dealing with file formats for Multiple Sequence Alignments (MSA)	644
ViennaRNA/io/utls.h	
Several utilities for file handling	714
ViennaRNA/loops/all.h	
Energy evaluation for MFE and partition function calculations	654
ViennaRNA/loops/external.h	
Energy evaluation of exterior loops for MFE and partition function calculations	654

ViennaRNA/loops/ <a href="#">hairpin.h</a>	
Energy evaluation of hairpin loops for MFE and partition function calculations . . . . .	655
ViennaRNA/loops/ <a href="#">internal.h</a>	
Energy evaluation of interior loops for MFE and partition function calculations . . . . .	656
ViennaRNA/loops/ <a href="#">multibranch.h</a>	
Energy evaluation of multibranch loops for MFE and partition function calculations . . . . .	657
ViennaRNA/params/ <a href="#">1.8.4_epars.h</a>	
Free energy parameters for parameter file conversion . . . . .	668
ViennaRNA/params/ <a href="#">1.8.4_intloops.h</a>	
Free energy parameters for interior loop contributions needed by the parameter file conversion functions . . . . .	669
ViennaRNA/params/ <a href="#">basic.h</a>	
Functions to deal with sets of energy parameters . . . . .	669
ViennaRNA/params/ <a href="#">constants.h</a>	
Energy parameter constants . . . . .	679
ViennaRNA/params/ <a href="#">convert.h</a>	
Functions and definitions for energy parameter file format conversion . . . . .	681
ViennaRNA/params/ <a href="#">default.h</a>	??
ViennaRNA/params/ <a href="#">intl11.h</a>	??
ViennaRNA/params/ <a href="#">intl11dH.h</a>	??
ViennaRNA/params/ <a href="#">intl21.h</a>	??
ViennaRNA/params/ <a href="#">intl21dH.h</a>	??
ViennaRNA/params/ <a href="#">intl22.h</a>	??
ViennaRNA/params/ <a href="#">intl22dH.h</a>	??
ViennaRNA/params/ <a href="#">io.h</a>	
Read and write energy parameter files . . . . .	682
ViennaRNA/plotting/ <a href="#">alignments.h</a>	
Various functions for plotting Sequence / Structure Alignments . . . . .	691
ViennaRNA/plotting/ <a href="#">layouts.h</a>	
Secondary structure plot layout algorithms . . . . .	693
ViennaRNA/plotting/ <a href="#">naview.h</a>	667
ViennaRNA/plotting/ <a href="#">probabilities.h</a>	
Various functions for plotting RNA secondary structures, dot-plots and other visualizations . . .	694
ViennaRNA/plotting/ <a href="#">structures.h</a>	
Various functions for plotting RNA secondary structures . . . . .	695
ViennaRNA/plotting/ <a href="#">utils.h</a>	
Various utilities to assist in plotting secondary structures and consensus structures . . . . .	715
ViennaRNA/search/ <a href="#">BoyerMoore.h</a>	
Variants of the Boyer-Moore string search algorithm . . . . .	703
ViennaRNA/utis/ <a href="#">alignments.h</a>	
Various utility- and helper-functions for sequence alignments and comparative structure prediction . . . . .	692
ViennaRNA/utis/ <a href="#">basic.h</a>	
General utility- and helper-functions used throughout the <i>ViennaRNA Package</i> . . . . .	672
ViennaRNA/utis/ <a href="#">cpu.h</a>	??
ViennaRNA/utis/ <a href="#">higher_order_functions.h</a>	??
ViennaRNA/utis/ <a href="#">strings.h</a>	
General utility- and helper-functions for RNA sequence and structure strings used throughout the ViennaRNA Package . . . . .	715
ViennaRNA/utis/ <a href="#">structures.h</a>	
Various utility- and helper-functions for secondary structure parsing, converting, etc . . . . .	696
ViennaRNA/utis/ <a href="#">svm.h</a>	??



## Chapter 15

# Module Documentation

### 15.1 Free Energy Evaluation

Functions and variables related to free energy evaluation of sequence/structure pairs.

#### 15.1.1 Detailed Description

Functions and variables related to free energy evaluation of sequence/structure pairs.

Several different functions to evaluate the free energy of a particular secondary structure under a particular set of parameters and the Nearest Neighbor Energy model are available. For most of them, two different forms of representations for the secondary structure may be used:

- The Dot-Bracket string
- A pair table representation

Furthermore, the evaluation functions are divided into `basic` and `simplified` variants, where `basic` functions require the use of a `vrna_fold_compound_t` data structure holding the sequence string, and model configuration (settings and parameters). The `simplified` functions, on the other hand, provide often used default model settings that may be called directly with only sequence and structure data.

Finally, `verbose` options exist for some functions that allow one to print the (individual) free energy contributions to some `FILE` stream. Collaboration diagram for Free Energy Evaluation:

#### Modules

- [Energy Evaluation for Individual Loops](#)  
*Functions to evaluate the free energy of particular types of loops.*
- [Energy Evaluation for Atomic Moves](#)  
*Functions to evaluate the free energy change of a structure after application of (a set of) atomic moves.*
- [Deprecated Interface for Free Energy Evaluation](#)  
*Deprecated Energy Evaluation functions.*

## Files

- file [eval.h](#)  
*Functions and variables related to energy evaluation of sequence/structure pairs.*
- file [all.h](#)  
*Energy evaluation for MFE and partition function calculations.*
- file [external.h](#)  
*Energy evaluation of exterior loops for MFE and partition function calculations.*
- file [hairpin.h](#)  
*Energy evaluation of hairpin loops for MFE and partition function calculations.*
- file [internal.h](#)  
*Energy evaluation of interior loops for MFE and partition function calculations.*
- file [multibranch.h](#)  
*Energy evaluation of multibranch loops for MFE and partition function calculations.*

## Macros

- `#define VRNA_VERBOSE_QUIET -1`  
*Quiet level verbosity setting.*
- `#define VRNA_VERBOSE_DEFAULT 1`  
*Default level verbosity setting.*

## Basic Energy Evaluation Interface with Dot-Bracket Structure String

- float [vrna\\_eval\\_structure](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*structure)  
*Calculate the free energy of an already folded RNA.*
- float [vrna\\_eval\\_covar\\_structure](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*structure)  
*Calculate the pseudo energy derived by the covariance scores of a set of aligned sequences.*
- float [vrna\\_eval\\_structure\\_verbose](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*structure, FILE \*file)  
*Calculate the free energy of an already folded RNA and print contributions on a per-loop base.*
- float [vrna\\_eval\\_structure\\_v](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*structure, int verbosity\_level, FILE \*file)  
*Calculate the free energy of an already folded RNA and print contributions on a per-loop base.*
- float [vrna\\_eval\\_structure\\_cstr](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*structure, int verbosity\_level, [vrna\\_cstr\\_t](#) output\_stream)

## Basic Energy Evaluation Interface with Structure Pair Table

- int [vrna\\_eval\\_structure\\_pt](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const short \*pt)  
*Calculate the free energy of an already folded RNA.*
- int [vrna\\_eval\\_structure\\_pt\\_verbose](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const short \*pt, FILE \*file)  
*Calculate the free energy of an already folded RNA.*
- int [vrna\\_eval\\_structure\\_pt\\_v](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const short \*pt, int verbosity\_level, FILE \*file)  
*Calculate the free energy of an already folded RNA.*

## Simplified Energy Evaluation with Sequence and Dot-Bracket Strings

- float `vrna_eval_structure_simple` (const char \*string, const char \*structure)  
*Calculate the free energy of an already folded RNA.*
- float `vrna_eval_circ_structure` (const char \*string, const char \*structure)  
*Evaluate the free energy of a sequence/structure pair where the sequence is circular.*
- float `vrna_eval_gquad_structure` (const char \*string, const char \*structure)  
*Evaluate the free energy of a sequence/structure pair where the structure may contain G-Quadruplexes.*
- float `vrna_eval_circ_gquad_structure` (const char \*string, const char \*structure)  
*Evaluate the free energy of a sequence/structure pair where the sequence is circular and the structure may contain G-Quadruplexes.*
- float `vrna_eval_structure_simple_verbose` (const char \*string, const char \*structure, FILE \*file)  
*Calculate the free energy of an already folded RNA and print contributions per loop.*
- float `vrna_eval_structure_simple_v` (const char \*string, const char \*structure, int verbosity\_level, FILE \*file)  
*Calculate the free energy of an already folded RNA and print contributions per loop.*
- float `vrna_eval_circ_structure_v` (const char \*string, const char \*structure, int verbosity\_level, FILE \*file)  
*Evaluate free energy of a sequence/structure pair, assume sequence to be circular and print contributions per loop.*
- float `vrna_eval_gquad_structure_v` (const char \*string, const char \*structure, int verbosity\_level, FILE \*file)  
*Evaluate free energy of a sequence/structure pair, allow for G-Quadruplexes in the structure and print contributions per loop.*
- float `vrna_eval_circ_gquad_structure_v` (const char \*string, const char \*structure, int verbosity\_level, FILE \*file)  
*Evaluate free energy of a sequence/structure pair, assume sequence to be circular, allow for G-Quadruplexes in the structure, and print contributions per loop.*

## Simplified Energy Evaluation with Sequence Alignments and Consensus Structure Dot-Bracket String

- float `vrna_eval_consensus_structure_simple` (const char \*\*alignment, const char \*structure)  
*Calculate the free energy of an already folded RNA sequence alignment.*
- float `vrna_eval_circ_consensus_structure` (const char \*\*alignment, const char \*structure)  
*Evaluate the free energy of a multiple sequence alignment/consensus structure pair where the sequences are circular.*
- float `vrna_eval_gquad_consensus_structure` (const char \*\*alignment, const char \*structure)  
*Evaluate the free energy of a multiple sequence alignment/consensus structure pair where the structure may contain G-Quadruplexes.*
- float `vrna_eval_circ_gquad_consensus_structure` (const char \*\*alignment, const char \*structure)  
*Evaluate the free energy of a multiple sequence alignment/consensus structure pair where the sequence is circular and the structure may contain G-Quadruplexes.*
- float `vrna_eval_consensus_structure_simple_verbose` (const char \*\*alignment, const char \*structure, FILE \*file)  
*Evaluate the free energy of a consensus structure for an RNA sequence alignment and print contributions per loop.*
- float `vrna_eval_consensus_structure_simple_v` (const char \*\*alignment, const char \*structure, int verbosity\_level, FILE \*file)  
*Evaluate the free energy of a consensus structure for an RNA sequence alignment and print contributions per loop.*
- float `vrna_eval_circ_consensus_structure_v` (const char \*\*alignment, const char \*structure, int verbosity\_level, FILE \*file)  
*Evaluate the free energy of a consensus structure for an alignment of circular RNA sequences and print contributions per loop.*
- float `vrna_eval_gquad_consensus_structure_v` (const char \*\*alignment, const char \*structure, int verbosity\_level, FILE \*file)  
*Evaluate the free energy of a consensus structure for an RNA sequence alignment, allow for annotated G-Quadruplexes in the structure and print contributions per loop.*
- float `vrna_eval_circ_gquad_consensus_structure_v` (const char \*\*alignment, const char \*structure, int verbosity\_level, FILE \*file)  
*Evaluate the free energy of a consensus structure for an alignment of circular RNA sequences, allow for annotated G-Quadruplexes in the structure and print contributions per loop.*

## Simplified Energy Evaluation with Sequence String and Structure Pair Table

- `int vrna_eval_structure_pt_simple` (const char \*string, const short \*pt)  
*Calculate the free energy of an already folded RNA.*
- `int vrna_eval_structure_pt_simple_verbose` (const char \*string, const short \*pt, FILE \*file)  
*Calculate the free energy of an already folded RNA.*
- `int vrna_eval_structure_pt_simple_v` (const char \*string, const short \*pt, int verbosity\_level, FILE \*file)  
*Calculate the free energy of an already folded RNA.*

## Simplified Energy Evaluation with Sequence Alignment and Consensus Structure Pair Table

- `int vrna_eval_consensus_structure_pt_simple` (const char \*\*alignment, const short \*pt)  
*Evaluate the Free Energy of a Consensus Secondary Structure given a Sequence Alignment.*
- `int vrna_eval_consensus_structure_pt_simple_verbose` (const char \*\*alignment, const short \*pt, FILE \*file)
- `int vrna_eval_consensus_structure_pt_simple_v` (const char \*\*alignment, const short \*pt, int verbosity\_level, FILE \*file)

### 15.1.2 Function Documentation

#### 15.1.2.1 vrna\_eval\_structure()

```
float vrna_eval_structure (
    vrna_fold_compound_t * vc,
    const char * structure )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA.

This function allows for energy evaluation of a given pair of structure and sequence (alignment). Model details, energy parameters, and possibly soft constraints are used as provided via the parameter 'vc'. The `vrna_fold_compound_t` does not need to contain any DP matrices, but requires all most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_EVAL_ONLY);
```

#### Note

Accepts `vrna_fold_compound_t` of type `VRNA_FC_TYPE_SINGLE` and `VRNA_FC_TYPE_COMPARATIVE`

#### See also

`vrna_eval_structure_pt()`, `vrna_eval_structure_verbose()`, `vrna_eval_structure_pt_verbose()`, `vrna_fold_compound()`, `vrna_fold_compound_comparative()`, `vrna_eval_covar_structure()`



## Parameters

<i>vc</i>	A <code>vrna_fold_compound_t</code> containing the energy parameters and model details
<i>structure</i>	Secondary structure in dot-bracket notation

## Returns

The free energy of the input structure given the input sequence in kcal/mol

**SWIG Wrapper Notes** This function is attached as method **eval\_structure()** to objects of type *fold\_compound*

## 15.1.2.2 vrna\_eval\_covar\_structure()

```
float vrna_eval_covar_structure (
    vrna_fold_compound_t * vc,
    const char * structure )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the pseudo energy derived by the covariance scores of a set of aligned sequences.

Consensus structure prediction is driven by covariance scores of base pairs in rows of the provided alignment. This function allows one to retrieve the total amount of this covariance pseudo energy scores. The `vrna_fold_compound_t` does not need to contain any DP matrices, but requires all most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound_comparative(alignment, NULL,
    VRNA_OPTION_EVAL_ONLY);
```

## Note

Accepts `vrna_fold_compound_t` of type `VRNA_FC_TYPE_COMPARATIVE` only!

## See also

[vrna\\_fold\\_compound\\_comparative\(\)](#), [vrna\\_eval\\_structure\(\)](#)

## Parameters

<i>vc</i>	A <code>vrna_fold_compound_t</code> containing the energy parameters and model details
<i>structure</i>	Secondary (consensus) structure in dot-bracket notation

## Returns

The covariance pseudo energy score of the input structure given the input sequence alignment in kcal/mol

**SWIG Wrapper Notes** This function is attached as method **eval\_covar\_structure()** to objects of type *fold\_compound*

### 15.1.2.3 `vrna_eval_structure_verbose()`

```
float vrna_eval_structure_verbose (
    vrna_fold_compound_t * vc,
    const char * structure,
    FILE * file )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA and print contributions on a per-loop base.

This function is a simplified version of `vrna_eval_structure_v()` that uses the *default* verbosity level. (

See also

`vrna_eval_structure_pt()`, `vrna_eval_structure_verbose()`, `vrna_eval_structure_pt_verbose()`,

#### Parameters

<i>vc</i>	A <code>vrna_fold_compound_t</code> containing the energy parameters and model details
<i>structure</i>	Secondary structure in dot-bracket notation
<i>file</i>	A file handle where this function should print to (may be NULL).

#### Returns

The free energy of the input structure given the input sequence in kcal/mol

**SWIG Wrapper Notes** This function is attached as method `eval_structure_verbose()` to objects of type `fold_compound`

### 15.1.2.4 `vrna_eval_structure_v()`

```
float vrna_eval_structure_v (
    vrna_fold_compound_t * vc,
    const char * structure,
    int verbosity_level,
    FILE * file )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA and print contributions on a per-loop base.

This function allows for detailed energy evaluation of a given sequence/structure pair. In contrast to `vrna_eval_structure()` this function prints detailed energy contributions based on individual loops to a file handle. If NULL is passed as file handle, this function defaults to print to stdout. Any positive `verbosity_level` activates potential warning message of the energy evaluating functions, while values  $\geq 1$  allow for detailed control of what data is printed. A negative parameter `verbosity_level` turns off printing all together.

Model details, energy parameters, and possibly soft constraints are used as provided via the parameter 'vc'. The `fold_compound` does not need to contain any DP matrices, but all the most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_EVAL_ONLY);
```

#### See also

[vrna\\_eval\\_structure\\_pt\(\)](#), [vrna\\_eval\\_structure\\_verbose\(\)](#), [vrna\\_eval\\_structure\\_pt\\_verbose\(\)](#),

#### Parameters

<i>vc</i>	A <code>vrna_fold_compound_t</code> containing the energy parameters and model details
<i>structure</i>	Secondary structure in dot-bracket notation
<i>verbosity_level</i>	The level of verbosity of this function
<i>file</i>	A file handle where this function should print to (may be NULL).

#### Returns

The free energy of the input structure given the input sequence in kcal/mol

#### 15.1.2.5 vrna\_eval\_structure\_pt()

```
int vrna_eval_structure_pt (
    vrna_fold_compound_t * vc,
    const short * pt )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA.

This function allows for energy evaluation of a given sequence/structure pair where the structure is provided in pair\_table format as obtained from [vrna\\_ptable\(\)](#). Model details, energy parameters, and possibly soft constraints are used as provided via the parameter 'vc'. The fold\_compound does not need to contain any DP matrices, but all the most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_EVAL_ONLY);
```

#### See also

[vrna\\_ptable\(\)](#), [vrna\\_eval\\_structure\(\)](#), [vrna\\_eval\\_structure\\_pt\\_verbose\(\)](#)

#### Parameters

<i>vc</i>	A <code>vrna_fold_compound_t</code> containing the energy parameters and model details
<i>pt</i>	Secondary structure as pair_table

#### Returns

The free energy of the input structure given the input sequence in 10cal/mol

**SWIG Wrapper Notes** This function is attached as method `eval_structure_pt()` to objects of type `fold_compound`

#### 15.1.2.6 `vrna_eval_structure_pt_verbose()`

```
int vrna_eval_structure_pt_verbose (
    vrna_fold_compound_t * vc,
    const short * pt,
    FILE * file )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA.

This function is a simplified version of `vrna_eval_structure_simple_v()` that uses the *default* verbosity level.

See also

`vrna_eval_structure_pt_v()`, `vrna_ptable()`, `vrna_eval_structure_pt()`, `vrna_eval_structure_verbose()`

#### Parameters

<i>vc</i>	A <code>vrna_fold_compound_t</code> containing the energy parameters and model details
<i>pt</i>	Secondary structure as <code>pair_table</code>
<i>file</i>	A file handle where this function should print to (may be NULL).

#### Returns

The free energy of the input structure given the input sequence in 10cal/mol

**SWIG Wrapper Notes** This function is attached as method `eval_structure_pt_verbose()` to objects of type `fold_compound`

#### 15.1.2.7 `vrna_eval_structure_pt_v()`

```
int vrna_eval_structure_pt_v (
    vrna_fold_compound_t * vc,
    const short * pt,
    int verbosity_level,
    FILE * file )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA.

This function allows for energy evaluation of a given sequence/structure pair where the structure is provided in `pair_table` format as obtained from `vrna_ptable()`. Model details, energy parameters, and possibly soft constraints are used as provided via the parameter 'vc'. The `fold_compound` does not need to contain any DP matrices, but all the most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_EVAL_ONLY);
```

In contrast to `vrna_eval_structure_pt()` this function prints detailed energy contributions based on individual loops to a file handle. If NULL is passed as file handle, this function defaults to print to stdout. Any positive `verbosity_level` activates potential warning message of the energy evaluating functions, while values  $\geq 1$  allow for detailed control of what data is printed. A negative parameter `verbosity_level` turns off printing all together.

See also

`vrna_ptable()`, `vrna_eval_structure_pt()`, `vrna_eval_structure_verbose()`

#### Parameters

<i>vc</i>	A <code>vrna_fold_compound_t</code> containing the energy parameters and model details
<i>pt</i>	Secondary structure as <code>pair_table</code>
<i>verbosity_level</i>	The level of verbosity of this function
<i>file</i>	A file handle where this function should print to (may be NULL).

#### Returns

The free energy of the input structure given the input sequence in 10cal/mol

#### 15.1.2.8 vrna\_eval\_structure\_simple()

```
int vrna_eval_structure_simple (
    const char * string,
    const char * structure )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA.

This function allows for energy evaluation of a given sequence/structure pair. In contrast to `vrna_eval_structure()` this function assumes default model details and default energy parameters in order to evaluate the free energy of the secondary structure. Therefore, it serves as a simple interface function for energy evaluation for situations where no changes on the energy model are required.

See also

`vrna_eval_structure()`, `vrna_eval_structure_pt()`, `vrna_eval_structure_verbose()`, `vrna_eval_structure_pt_verbose()`,

#### Parameters

<i>string</i>	RNA sequence in uppercase letters
<i>structure</i>	Secondary structure in dot-bracket notation

**Returns**

The free energy of the input structure given the input sequence in kcal/mol

**SWIG Wrapper Notes** In the target scripting language, this function serves as a wrapper for `vrna_eval_structure_simple_v()` and, thus, allows for two additional, optional arguments, the verbosity level and a file handle which default to `VRNA_VERBOSITY_QUIET` and `NULL`, respectively.

**15.1.2.9 vrna\_eval\_circ\_structure()**

```
int vrna_eval_circ_structure (
    const char * string,
    const char * structure )
```

```
#include <ViennaRNA/eval.h>
```

Evaluate the free energy of a sequence/structure pair where the sequence is circular.

**See also**

[vrna\\_eval\\_structure\\_simple\(\)](#), [vrna\\_eval\\_gquad\\_structure\(\)](#), [vrna\\_eval\\_circ\\_consensus\\_structure\(\)](#), [vrna\\_eval\\_circ\\_structure\\_v\(\)](#), [vrna\\_eval\\_structure\(\)](#)

**Parameters**

<i>string</i>	RNA sequence in uppercase letters
<i>structure</i>	Secondary structure in dot-bracket notation

**Returns**

The free energy of the structure given the circular input sequence in kcal/mol

**SWIG Wrapper Notes** In the target scripting language, this function serves as a wrapper for `vrna_eval_circ_structure_v()` and, thus, allows for two additional, optional arguments, the verbosity level and a file handle which default to `VRNA_VERBOSITY_QUIET` and `NULL`, respectively.

**15.1.2.10 vrna\_eval\_gquad\_structure()**

```
int vrna_eval_gquad_structure (
    const char * string,
    const char * structure )
```

```
#include <ViennaRNA/eval.h>
```

Evaluate the free energy of a sequence/structure pair where the structure may contain G-Quadruplexes.

G-Quadruplexes are annotated as plus signs ('+') for each G involved in the motif. Linker sequences must be denoted by dots('.') as they are considered unpaired. Below is an example of a 2-layer G-quadruplex:

```
GGAAGGAAAGGAGG
++..++...++..++
```

## See also

[vrna\\_eval\\_structure\\_simple\(\)](#), [vrna\\_eval\\_circ\\_structure\(\)](#), [vrna\\_eval\\_gquad\\_consensus\\_structure\(\)](#), [vrna\\_eval\\_gquad\\_structure\\_v\(\)](#), [vrna\\_eval\\_structure\(\)](#)

## Parameters

<i>string</i>	RNA sequence in uppercase letters
<i>structure</i>	Secondary structure in dot-bracket notation

## Returns

The free energy of the structure including contributions of G-quadruplexes in kcal/mol

**SWIG Wrapper Notes** In the target scripting language, this function serves as a wrapper for [vrna\\_eval\\_gquad\\_structure\\_v\(\)](#) and, thus, allows for two additional, optional arguments, the verbosity level and a file handle which default to [VRNA\\_VERBOSITY\\_QUIET](#) and NULL, respectively.

15.1.2.11 `vrna_eval_circ_gquad_structure()`

```
int vrna_eval_circ_gquad_structure (
    const char * string,
    const char * structure )
```

```
#include <ViennaRNA/eval.h>
```

Evaluate the free energy of a sequence/structure pair where the sequence is circular and the structure may contain G-Quadruplexes.

G-Quadruplexes are annotated as plus signs ('+') for each G involved in the motif. Linker sequences must be denoted by dots('.') as they are considered unpaired. Below is an example of a 2-layer G-quadruplex:

```
GGAAGGAAAGGAGG
++..++..++..++
```

## See also

[vrna\\_eval\\_structure\\_simple\(\)](#), [vrna\\_eval\\_circ\\_gquad\\_consensus\\_structure\(\)](#), [vrna\\_eval\\_circ\\_gquad\\_structure\\_v\(\)](#), [vrna\\_eval\\_structure\(\)](#)

## Parameters

<i>string</i>	RNA sequence in uppercase letters
<i>structure</i>	Secondary structure in dot-bracket notation

**Returns**

The free energy of the structure including contributions of G-quadruplexes in kcal/mol

**SWIG Wrapper Notes** In the target scripting language, this function serves as a wrapper for `vrna_eval_circ_gquad_structure_v()` and, thus, allows for two additional, optional arguments, the verbosity level and a file handle which default to `VRNA_VERBOSITY_QUIET` and `NULL`, respectively.

**15.1.2.12 vrna\_eval\_structure\_simple\_verbose()**

```
int vrna_eval_structure_simple_verbose (
    const char * string,
    const char * structure,
    FILE * file )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA and print contributions per loop.

This function is a simplyfied version of `vrna_eval_structure_simple_v()` that uses the *default* verbosity level.

**See also**

`vrna_eval_structure_simple_v()`, `vrna_eval_structure_verbose()`, `vrna_eval_structure_pt()`, `vrna_eval_structure_verbose()`, `vrna_eval_structure_pt_verbose()`

**Parameters**

<i>string</i>	RNA sequence in uppercase letters
<i>structure</i>	Secondary structure in dot-bracket notation
<i>file</i>	A file handle where this function should print to (may be <code>NULL</code> ).

**Returns**

The free energy of the input structure given the input sequence in kcal/mol

**SWIG Wrapper Notes** This function is not available. Use `vrna_eval_structure_simple_v()` instead!

**15.1.2.13 vrna\_eval\_structure\_simple\_v()**

```
int vrna_eval_structure_simple_v (
    const char * string,
    const char * structure,
    int verbosity_level,
    FILE * file )
```



```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA and print contributions per loop.

This function allows for detailed energy evaluation of a given sequence/structure pair. In contrast to [vrna\\_eval\\_structure\(\)](#) this function prints detailed energy contributions based on individual loops to a file handle. If NULL is passed as file handle, this function defaults to print to stdout. Any positive `verbosity_level` activates potential warning message of the energy evaluating functions, while values  $\geq 1$  allow for detailed control of what data is printed. A negative parameter `verbosity_level` turns off printing all together.

In contrast to [vrna\\_eval\\_structure\\_verbose\(\)](#) this function assumes default model details and default energy parameters in order to evaluate the free energy of the secondary structure. Therefore, it serves as a simple interface function for energy evaluation for situations where no changes on the energy model are required.

See also

[vrna\\_eval\\_structure\\_verbose\(\)](#), [vrna\\_eval\\_structure\\_pt\(\)](#), [vrna\\_eval\\_structure\\_pt\\_verbose\(\)](#),

#### Parameters

<i>string</i>	RNA sequence in uppercase letters
<i>structure</i>	Secondary structure in dot-bracket notation
<i>verbosity_level</i>	The level of verbosity of this function
<i>file</i>	A file handle where this function should print to (may be NULL).

#### Returns

The free energy of the input structure given the input sequence in kcal/mol

**SWIG Wrapper Notes** This function is available through an overloaded version of [vrna\\_eval\\_structure\\_simple\(\)](#). The last two arguments for this function are optional and default to `VRNA_VERBOSITY_QUIET` and NULL, respectively.

#### 15.1.2.14 vrna\_eval\_circ\_structure\_v()

```
int vrna_eval_circ_structure_v (
    const char * string,
    const char * structure,
    int verbosity_level,
    FILE * file )
```

```
#include <ViennaRNA/eval.h>
```

Evaluate free energy of a sequence/structure pair, assume sequence to be circular and print contributions per loop.

This function is the same as [vrna\\_eval\\_structure\\_simple\\_v\(\)](#) but assumes the input sequence to be circularized.

See also

[vrna\\_eval\\_structure\\_simple\\_v\(\)](#), [vrna\\_eval\\_circ\\_structure\(\)](#), [vrna\\_eval\\_structure\\_verbose\(\)](#)

**Parameters**

<i>string</i>	RNA sequence in uppercase letters
<i>structure</i>	Secondary structure in dot-bracket notation
<i>verbosity_level</i>	The level of verbosity of this function
<i>file</i>	A file handle where this function should print to (may be NULL).

**Returns**

The free energy of the input structure given the input sequence in kcal/mol

**SWIG Wrapper Notes** This function is available through an overloaded version of [vrna\\_eval\\_circ\\_structure\(\)](#). The last two arguments for this function are optional and default to [VRNA\\_VERBOSITY\\_QUIET](#) and NULL, respectively.

**15.1.2.15 vrna\_eval\_gquad\_structure\_v()**

```
int vrna_eval_gquad_structure_v (
    const char * string,
    const char * structure,
    int verbosity_level,
    FILE * file )
```

```
#include <ViennaRNA/eval.h>
```

Evaluate free energy of a sequence/structure pair, allow for G-Quadruplexes in the structure and print contributions per loop.

This function is the same as [vrna\\_eval\\_structure\\_simple\\_v\(\)](#) but allows for annotated G-Quadruplexes in the dot-bracket structure input.

G-Quadruplexes are annotated as plus signs ('+') for each G involved in the motif. Linker sequences must be denoted by dots('.') as they are considered unpaired. Below is an example of a 2-layer G-quadruplex:

```
GGAAGGAAAGGAGG
++..++..++..++
```

**See also**

[vrna\\_eval\\_structure\\_simple\\_v\(\)](#), [vrna\\_eval\\_gquad\\_structure\(\)](#), [vrna\\_eval\\_structure\\_verbose\(\)](#)

**Parameters**

<i>string</i>	RNA sequence in uppercase letters
<i>structure</i>	Secondary structure in dot-bracket notation
<i>verbosity_level</i>	The level of verbosity of this function
<i>file</i>	A file handle where this function should print to (may be NULL).

**Returns**

The free energy of the input structure given the input sequence in kcal/mol

**SWIG Wrapper Notes** This function is available through an overloaded version of [vrna\\_eval\\_gquad\\_structure\(\)](#). The last two arguments for this function are optional and default to [VRNA\\_VERBOSITY\\_QUIET](#) and `NULL`, respectively.

**15.1.2.16 vrna\_eval\_circ\_gquad\_structure\_v()**

```
int vrna_eval_circ_gquad_structure_v (
    const char * string,
    const char * structure,
    int verbosity_level,
    FILE * file )
```

```
#include <ViennaRNA/eval.h>
```

Evaluate free energy of a sequence/structure pair, assume sequence to be circular, allow for G-Quadruplexes in the structure, and print contributions per loop.

This function is the same as [vrna\\_eval\\_structure\\_simple\\_v\(\)](#) but assumes the input sequence to be circular and allows for annotated G-Quadruplexes in the dot-bracket structure input.

G-Quadruplexes are annotated as plus signs ('+') for each G involved in the motif. Linker sequences must be denoted by dots('.') as they are considered unpaired. Below is an example of a 2-layer G-quadruplex:

```
GGAAGGAAAGGAGG
++..++..++..++
```

**Parameters**

<i>string</i>	RNA sequence in uppercase letters
<i>structure</i>	Secondary structure in dot-bracket notation
<i>verbosity_level</i>	The level of verbosity of this function
<i>file</i>	A file handle where this function should print to (may be <code>NULL</code> ).

**Returns**

The free energy of the input structure given the input sequence in kcal/mol

**SWIG Wrapper Notes** This function is available through an overloaded version of [vrna\\_eval\\_circ\\_gquad\\_structure\(\)](#). The last two arguments for this function are optional and default to [VRNA\\_VERBOSITY\\_QUIET](#) and `NULL`, respectively.

15.1.2.17 `vrna_eval_consensus_structure_simple()`

```
int vrna_eval_consensus_structure_simple (
    const char ** alignment,
    const char * structure )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA sequence alignment.

This function allows for energy evaluation for a given multiple sequence alignment and consensus structure pair. In contrast to `vrna_eval_structure()` this function assumes default model details and default energy parameters in order to evaluate the free energy of the secondary structure. Therefore, it serves as a simple interface function for energy evaluation for situations where no changes on the energy model are required.

**Note**

The free energy returned from this function already includes the covariation pseudo energies that is used for comparative structure prediction within this library.

**See also**

[vrna\\_eval\\_covar\\_structure\(\)](#), [vrna\\_eval\\_structure\(\)](#), [vrna\\_eval\\_structure\\_pt\(\)](#), [vrna\\_eval\\_structure\\_verbose\(\)](#), [vrna\\_eval\\_structure\\_pt\\_verbose\(\)](#)

**Parameters**

<i>alignment</i>	RNA sequence alignment in uppercase letters and hyphen ('-') to denote gaps
<i>structure</i>	Consensus Secondary structure in dot-bracket notation

**Returns**

The free energy of the consensus structure given the input alignment in kcal/mol

**SWIG Wrapper Notes** This function is available through an overloaded version of `vrna_eval_structure_simple()`. Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument

15.1.2.18 `vrna_eval_circ_consensus_structure()`

```
int vrna_eval_circ_consensus_structure (
    const char ** alignment,
    const char * structure )
```

```
#include <ViennaRNA/eval.h>
```

Evaluate the free energy of a multiple sequence alignment/consensus structure pair where the sequences are circular.

**Note**

The free energy returned from this function already includes the covariation pseudo energies that is used for comparative structure prediction within this library.

**See also**

[vrna\\_eval\\_covar\\_structure\(\)](#), [vrna\\_eval\\_consensus\\_structure\\_simple\(\)](#), [vrna\\_eval\\_gquad\\_consensus\\_structure\(\)](#), [vrna\\_eval\\_circ\\_structure\(\)](#), [vrna\\_eval\\_circ\\_consensus\\_structure\\_v\(\)](#), [vrna\\_eval\\_structure\(\)](#)

**Parameters**

<i>alignment</i>	RNA sequence alignment in uppercase letters
<i>structure</i>	Consensus secondary structure in dot-bracket notation

**Returns**

The free energy of the consensus structure given the circular input sequence in kcal/mol

**SWIG Wrapper Notes** This function is available through an overloaded version of [vrna\\_eval\\_circ\\_structure\(\)](#). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument

**15.1.2.19 vrna\_eval\_gquad\_consensus\_structure()**

```
int vrna_eval_gquad_consensus_structure (
    const char ** alignment,
    const char * structure )
```

```
#include <ViennaRNA/eval.h>
```

Evaluate the free energy of a multiple sequence alignment/consensus structure pair where the structure may contain G-Quadruplexes.

G-Quadruplexes are annotated as plus signs ('+') for each G involved in the motif. Linker sequences must be denoted by dots('.') as they are considered unpaired. Below is an example of a 2-layer G-quadruplex:

```
GGAAGGAAAGGAGG
++..++..++..++
```

**Note**

The free energy returned from this function already includes the covariation pseudo energies that is used for comparative structure prediction within this library.

**See also**

[vrna\\_eval\\_covar\\_structure\(\)](#), [vrna\\_eval\\_consensus\\_structure\\_simple\(\)](#), [vrna\\_eval\\_circ\\_consensus\\_structure\(\)](#), [vrna\\_eval\\_gquad\\_structure\(\)](#), [vrna\\_eval\\_gquad\\_consensus\\_structure\\_v\(\)](#), [vrna\\_eval\\_structure\(\)](#)

**Parameters**

<i>alignment</i>	RNA sequence alignment in uppercase letters
<i>structure</i>	Consensus secondary structure in dot-bracket notation

**Returns**

The free energy of the consensus structure including contributions of G-quadruplexes in kcal/mol

**SWIG Wrapper Notes** This function is available through an overloaded version of [vrna\\_eval\\_gquad\\_structure\(\)](#). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument

**15.1.2.20 vrna\_eval\_circ\_gquad\_consensus\_structure()**

```
int vrna_eval_circ_gquad_consensus_structure (
    const char ** alignment,
    const char * structure )
```

```
#include <ViennaRNA/eval.h>
```

Evaluate the free energy of a multiple sequence alignment/consensus structure pair where the sequence is circular and the structure may contain G-Quadruplexes.

G-Quadruplexes are annotated as plus signs ('+') for each G involved in the motif. Linker sequences must be denoted by dots('.') as they are considered unpaired. Below is an example of a 2-layer G-quadruplex:

```
GGAAGGAAAGGAGG
++..++..++..++
```

**Note**

The free energy returned from this function already includes the covariation pseudo energies that is used for comparative structure prediction within this library.

**See also**

[vrna\\_eval\\_covar\\_structure\(\)](#), [vrna\\_eval\\_consensus\\_structure\\_simple\(\)](#), [vrna\\_eval\\_circ\\_consensus\\_structure\(\)](#), [vrna\\_eval\\_gquad\\_structure\(\)](#), [vrna\\_eval\\_circ\\_gquad\\_consensus\\_structure\\_v\(\)](#), [vrna\\_eval\\_structure\(\)](#)

**Parameters**

<i>alignment</i>	RNA sequence alignment in uppercase letters
<i>structure</i>	Consensus secondary structure in dot-bracket notation

**Returns**

The free energy of the consensus structure including contributions of G-quadruplexes in kcal/mol

**SWIG Wrapper Notes** This function is available through an overloaded version of [vrna\\_eval\\_circ\\_gquad\\_structure\(\)](#). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument

**15.1.2.21 vrna\_eval\_consensus\_structure\_simple\_verbose()**

```
int vrna_eval_consensus_structure_simple_verbose (
    const char ** alignment,
    const char * structure,
    FILE * file )
```

```
#include <ViennaRNA/eval.h>
```

Evaluate the free energy of a consensus structure for an RNA sequence alignment and print contributions per loop.

This function is a simplified version of [vrna\\_eval\\_consensus\\_structure\\_simple\\_v\(\)](#) that uses the *default* verbosity level.

**Note**

The free energy returned from this function already includes the covariation pseudo energies that is used for comparative structure prediction within this library.

**See also**

[vrna\\_eval\\_consensus\\_structure\\_simple\\_v\(\)](#), [vrna\\_eval\\_structure\\_verbose\(\)](#), [vrna\\_eval\\_structure\\_pt\(\)](#), [vrna\\_eval\\_structure\\_pt\\_verbose\(\)](#)

**Parameters**

<i>alignment</i>	RNA sequence alignment in uppercase letters. Gaps are denoted by hyphens ('-')
<i>structure</i>	Consensus secondary structure in dot-bracket notation
<i>file</i>	A file handle where this function should print to (may be NULL).

**Returns**

The free energy of the consensus structure given the aligned input sequences in kcal/mol

**SWIG Wrapper Notes** This function is not available. Use [vrna\\_eval\\_consensus\\_structure\\_simple\\_v\(\)](#) instead!

15.1.2.22 `vrna_eval_consensus_structure_simple_v()`

```
int vrna_eval_consensus_structure_simple_v (
    const char ** alignment,
    const char * structure,
    int verbosity_level,
    FILE * file )
```

```
#include <ViennaRNA/eval.h>
```

Evaluate the free energy of a consensus structure for an RNA sequence alignment and print contributions per loop.

This function allows for detailed energy evaluation of a given sequence alignment/consensus structure pair. In contrast to `vrna_eval_consensus_structure_simple()` this function prints detailed energy contributions based on individual loops to a file handle. If NULL is passed as file handle, this function defaults to print to stdout. Any positive `verbosity_level` activates potential warning message of the energy evaluating functions, while values  $\geq 1$  allow for detailed control of what data is printed. A negative parameter `verbosity_level` turns off printing all together.

**Note**

The free energy returned from this function already includes the covariation pseudo energies that is used for comparative structure prediction within this library.

**See also**

`vrna_eval_consensus_structure()`, `vrna_eval_structure()`

**Parameters**

<i>alignment</i>	RNA sequence alignment in uppercase letters. Gaps are denoted by hyphens ('-')
<i>structure</i>	Consensus secondary structure in dot-bracket notation
<i>verbosity_level</i>	The level of verbosity of this function
<i>file</i>	A file handle where this function should print to (may be NULL).

**Returns**

The free energy of the consensus structure given the sequence alignment in kcal/mol

**SWIG Wrapper Notes** This function is available through an overloaded version of `vrna_eval_structure_simple()`. Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument. The last two arguments are optional and default to `VRNA_VERBOSITY_QUIET` and NULL, respectively.

15.1.2.23 `vrna_eval_circ_consensus_structure_v()`

```
int vrna_eval_circ_consensus_structure_v (
    const char ** alignment,
```



```
const char * structure,
int verbosity_level,
FILE * file )
```

```
#include <ViennaRNA/eval.h>
```

Evaluate the free energy of a consensus structure for an alignment of circular RNA sequences and print contributions per loop.

This function is identical with [vrna\\_eval\\_consensus\\_structure\\_simple\\_v\(\)](#) but assumed the aligned sequences to be circular.

#### Note

The free energy returned from this function already includes the covariation pseudo energies that is used for comparative structure prediction within this library.

#### See also

[vrna\\_eval\\_consensus\\_structure\\_simple\\_v\(\)](#), [vrna\\_eval\\_circ\\_consensus\\_structure\(\)](#), [vrna\\_eval\\_structure\(\)](#)

#### Parameters

<i>alignment</i>	RNA sequence alignment in uppercase letters. Gaps are denoted by hyphens ('-')
<i>structure</i>	Consensus secondary structure in dot-bracket notation
<i>verbosity_level</i>	The level of verbosity of this function
<i>file</i>	A file handle where this function should print to (may be NULL).

#### Returns

The free energy of the consensus structure given the sequence alignment in kcal/mol

**SWIG Wrapper Notes** This function is available through an overloaded version of [vrna\\_eval\\_circ\\_structure\(\)](#). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument. The last two arguments are optional and default to [VRNA\\_VERBOSITY\\_QUIET](#) and NULL, respectively.

#### 15.1.2.24 vrna\_eval\_gquad\_consensus\_structure\_v()

```
int vrna_eval_gquad_consensus_structure_v (
    const char ** alignment,
    const char * structure,
    int verbosity_level,
    FILE * file )
```

```
#include <ViennaRNA/eval.h>
```

Evaluate the free energy of a consensus structure for an RNA sequence alignment, allow for annotated G↔ Quadruplexes in the structure and print contributions per loop.

This function is identical with [vrna\\_eval\\_consensus\\_structure\\_simple\\_v\(\)](#) but allows for annotated G-Quadruplexes in the consensus structure.

G-Quadruplexes are annotated as plus signs ('+') for each G involved in the motif. Linker sequences must be denoted by dots('.') as they are considered unpaired. Below is an example of a 2-layer G-quadruplex:

```
GGAAGGAAAGGAGG
++..++...++..++
```

#### Note

The free energy returned from this function already includes the covariation pseudo energies that is used for comparative structure prediction within this library.

#### See also

[vrna\\_eval\\_consensus\\_structure\\_simple\\_v\(\)](#), [vrna\\_eval\\_gquad\\_consensus\\_structure\(\)](#), [vrna\\_eval\\_structure\(\)](#)

#### Parameters

<i>alignment</i>	RNA sequence alignment in uppercase letters. Gaps are denoted by hyphens ('-')
<i>structure</i>	Consensus secondary structure in dot-bracket notation
<i>verbosity_level</i>	The level of verbosity of this function
<i>file</i>	A file handle where this function should print to (may be NULL).

#### Returns

The free energy of the consensus structure given the sequence alignment in kcal/mol

**SWIG Wrapper Notes** This function is available through an overloaded version of [vrna\\_eval\\_gquad\\_structure\(\)](#). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument. The last two arguments are optional and default to [VRNA\\_VERBOSITY\\_QUIET](#) and NULL, respectively.

#### 15.1.2.25 vrna\_eval\_circ\_gquad\_consensus\_structure\_v()

```
int vrna_eval_circ_gquad_consensus_structure_v (
    const char ** alignment,
    const char * structure,
    int verbosity_level,
    FILE * file )
```

```
#include <ViennaRNA/eval.h>
```

Evaluate the free energy of a consensus structure for an alignment of circular RNA sequences, allow for annotated G-Quadruplexes in the structure and print contributions per loop.

This function is identical with [vrna\\_eval\\_consensus\\_structure\\_simple\\_v\(\)](#) but assumes the sequences in the alignment to be circular and allows for annotated G-Quadruplexes in the consensus structure.

G-Quadruplexes are annotated as plus signs ('+') for each G involved in the motif. Linker sequences must be denoted by dots('.') as they are considered unpaired. Below is an example of a 2-layer G-quadruplex:

```
GGAAGGAAAGGAGG
++..++..++..++
```

**Note**

The free energy returned from this function already includes the covariation pseudo energies that is used for comparative structure prediction within this library.

**See also**

[vrna\\_eval\\_consensus\\_structure\\_simple\\_v\(\)](#), [vrna\\_eval\\_circ\\_gquad\\_consensus\\_structure\(\)](#), [vrna\\_eval\\_structure\(\)](#)

**Parameters**

<i>alignment</i>	RNA sequence alignment in uppercase letters. Gaps are denoted by hyphens ('-')
<i>structure</i>	Consensus secondary structure in dot-bracket notation
<i>verbosity_level</i>	The level of verbosity of this function
<i>file</i>	A file handle where this function should print to (may be NULL).

**Returns**

The free energy of the consensus structure given the sequence alignment in kcal/mol

**SWIG Wrapper Notes** This function is available through an overloaded version of [vrna\\_eval\\_circ\\_gquad\\_structure\(\)](#). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument. The last two arguments are optional and default to [VRNA\\_VERBOSITY\\_QUIET](#) and NULL, respectively.

**15.1.2.26 vrna\_eval\_structure\_pt\_simple()**

```
int vrna_eval_structure_pt_simple (
    const char * string,
    const short * pt )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA.

In contrast to [vrna\\_eval\\_structure\\_pt\(\)](#) this function assumes default model details and default energy parameters in order to evaluate the free energy of the secondary structure. Therefore, it serves as a simple interface function for energy evaluation for situations where no changes on the energy model are required.

**See also**

[vrna\\_ptable\(\)](#), [vrna\\_eval\\_structure\\_simple\(\)](#), [vrna\\_eval\\_structure\\_pt\(\)](#)

## Parameters

<i>string</i>	RNA sequence in uppercase letters
<i>pt</i>	Secondary structure as pair_table

## Returns

The free energy of the input structure given the input sequence in 10cal/mol

**SWIG Wrapper Notes** In the target scripting language, this function serves as a wrapper for [vrna\\_eval\\_structure\\_pt\\_v\(\)](#) and, thus, allows for two additional, optional arguments, the verbosity level and a file handle which default to [VRNA\\_VERBOSITY\\_QUIET](#) and NULL, respectively.

## 15.1.2.27 vrna\_eval\_structure\_pt\_simple\_verbose()

```
int vrna_eval_structure_pt_simple_verbose (
    const char * string,
    const short * pt,
    FILE * file )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA.

This function is a simplified version of [vrna\\_eval\\_structure\\_pt\\_simple\\_v\(\)](#) that uses the *default* verbosity level.

## See also

[vrna\\_eval\\_structure\\_pt\\_simple\\_v\(\)](#), [vrna\\_ptable\(\)](#), [vrna\\_eval\\_structure\\_pt\\_verbose\(\)](#), [vrna\\_eval\\_structure\\_simple\(\)](#)

## Parameters

<i>string</i>	RNA sequence in uppercase letters
<i>pt</i>	Secondary structure as pair_table
<i>file</i>	A file handle where this function should print to (may be NULL).

## Returns

The free energy of the input structure given the input sequence in 10cal/mol

## 15.1.2.28 vrna\_eval\_structure\_pt\_simple\_v()

```
int vrna_eval_structure_pt_simple_v (
    const char * string,
```

```
const short * pt,
int verbosity_level,
FILE * file )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA.

This function allows for energy evaluation of a given sequence/structure pair where the structure is provided in pair\_table format as obtained from [vrna\\_ptable\(\)](#). Model details, energy parameters, and possibly soft constraints are used as provided via the parameter 'vc'. The fold\_compound does not need to contain any DP matrices, but all the most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_EVAL_ONLY);
```

In contrast to [vrna\\_eval\\_structure\\_pt\\_verbose\(\)](#) this function assumes default model details and default energy parameters in order to evaluate the free energy of the secondary structure. Therefore, it serves as a simple interface function for energy evaluation for situations where no changes on the energy model are required.

See also

[vrna\\_ptable\(\)](#), [vrna\\_eval\\_structure\\_pt\\_v\(\)](#), [vrna\\_eval\\_structure\\_simple\(\)](#)

#### Parameters

<i>string</i>	RNA sequence in uppercase letters
<i>pt</i>	Secondary structure as pair_table
<i>verbosity_level</i>	The level of verbosity of this function
<i>file</i>	A file handle where this function should print to (may be NULL).

#### Returns

The free energy of the input structure given the input sequence in 10cal/mol

#### 15.1.2.29 vrna\_eval\_consensus\_structure\_pt\_simple()

```
int vrna_eval_consensus_structure_pt_simple (
    const char ** alignment,
    const short * pt )
```

```
#include <ViennaRNA/eval.h>
```

Evaluate the Free Energy of a Consensus Secondary Structure given a Sequence Alignment.

#### Note

The free energy returned from this function already includes the covariation pseudo energies that is used for comparative structure prediction within this library.

See also

[vrna\\_eval\\_consensus\\_structure\\_simple\(\)](#), [vrna\\_eval\\_structure\\_pt\(\)](#), [vrna\\_eval\\_structure\(\)](#), [vrna\\_eval\\_covar\\_structure\(\)](#)

**Parameters**

<i>alignment</i>	RNA sequence alignment in uppercase letters. Gaps are denoted by hyphens ('-')
<i>pt</i>	Secondary structure in pair table format

**Returns**

Free energy of the consensus structure in 10cal/mol

**SWIG Wrapper Notes** This function is available through an overloaded version of [vrna\\_eval\\_structure\\_pt\\_simple\(\)](#). Simply pass a sequence alignment as list of strings (including gaps) as first, and the consensus structure as second argument

## 15.2 Energy Evaluation for Individual Loops

Functions to evaluate the free energy of particular types of loops.

### 15.2.1 Detailed Description

Functions to evaluate the free energy of particular types of loops.

To assess the free energy contribution of a particular loop within a secondary structure, two variants are provided:

- The `bare` free energy  $E$  (usually in deka-calories, i.e. multiples of  $10\text{cal/mol}$ ), and
- The Boltzmann weight  $q = \exp(-\beta E)$  of the free energy  $E$  (with  $\beta = \frac{1}{RT}$ , gas constant  $R$  and temperature  $T$ )

The latter is usually required for partition function computations. Collaboration diagram for Energy Evaluation for Individual Loops:

### Modules

- [Exterior Loops](#)  
*Functions to evaluate the free energy contributions for exterior loops.*
- [Hairpin Loops](#)  
*Functions to evaluate the free energy contributions for hairpin loops.*
- [Internal Loops](#)  
*Functions to evaluate the free energy contributions for internal loops.*
- [Multibranch Loops](#)  
*Functions to evaluate the free energy contributions for multibranch loops.*

### Files

- file [all.h](#)  
*Energy evaluation for MFE and partition function calculations.*
- file [external.h](#)  
*Energy evaluation of exterior loops for MFE and partition function calculations.*
- file [hairpin.h](#)  
*Energy evaluation of hairpin loops for MFE and partition function calculations.*
- file [internal.h](#)  
*Energy evaluation of interior loops for MFE and partition function calculations.*
- file [multibranch.h](#)  
*Energy evaluation of multibranch loops for MFE and partition function calculations.*

### Functions

- int [vrna\\_eval\\_loop\\_pt](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int i, const short \*pt)  
*Calculate energy of a loop.*
- int [vrna\\_eval\\_loop\\_pt\\_v](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int i, const short \*pt, int verbosity\_level)  
*Calculate energy of a loop.*

## 15.2.2 Function Documentation

### 15.2.2.1 vrna\_eval\_loop\_pt()

```
int vrna_eval_loop_pt (
    vrna_fold_compound_t * vc,
    int i,
    const short * pt )
```

```
#include <ViennaRNA/eval.h>
```

Calculate energy of a loop.

#### Parameters

<i>vc</i>	A <code>vrna_fold_compound_t</code> containing the energy parameters and model details
<i>i</i>	position of covering base pair
<i>pt</i>	the pair table of the secondary structure

#### Returns

free energy of the loop in 10cal/mol

**SWIG Wrapper Notes** This function is attached as method **eval\_loop\_pt()** to objects of type *fold\_compound*

### 15.2.2.2 vrna\_eval\_loop\_pt\_v()

```
int vrna_eval_loop_pt_v (
    vrna_fold_compound_t * vc,
    int i,
    const short * pt,
    int verbosity_level )
```

```
#include <ViennaRNA/eval.h>
```

Calculate energy of a loop.

#### Parameters

<i>vc</i>	A <code>vrna_fold_compound_t</code> containing the energy parameters and model details
<i>i</i>	position of covering base pair
<i>pt</i>	the pair table of the secondary structure
<i>verbosity_level</i>	The level of verbosity of this function



Returns

free energy of the loop in 10cal/mol

## 15.3 Energy Evaluation for Atomic Moves

Functions to evaluate the free energy change of a structure after application of (a set of) atomic moves.

### 15.3.1 Detailed Description

Functions to evaluate the free energy change of a structure after application of (a set of) atomic moves.

Here, atomic moves are not to be confused with moves of actual physical atoms. Instead, an atomic move is considered the smallest conformational change a secondary structure can undergo to form another, distinguishable structure. We currently support the following moves

Atomic Moves:

- Opening (dissociation) of a single base pair
- Closing (formation) of a single base pair
- Shifting one pairing partner of an existing pair to a different location

Collaboration diagram for Energy Evaluation for Atomic Moves:

### Functions

- float [vrna\\_eval\\_move](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*structure, int m1, int m2)  
*Calculate energy of a move (closing or opening of a base pair)*
- int [vrna\\_eval\\_move\\_pt](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, short \*pt, int m1, int m2)  
*Calculate energy of a move (closing or opening of a base pair)*

### 15.3.2 Function Documentation

#### 15.3.2.1 vrna\_eval\_move()

```
float vrna_eval_move (
    vrna_fold_compound_t * vc,
    const char * structure,
    int m1,
    int m2 )
```

```
#include <ViennaRNA/eval.h>
```

Calculate energy of a move (closing or opening of a base pair)

If the parameters m1 and m2 are negative, it is deletion (opening) of a base pair, otherwise it is insertion (opening).

See also

[vrna\\_eval\\_move\\_pt\(\)](#)

## Parameters

<i>vc</i>	A <code>vrna_fold_compound_t</code> containing the energy parameters and model details
<i>structure</i>	secondary structure in dot-bracket notation
<i>m1</i>	first coordinate of base pair
<i>m2</i>	second coordinate of base pair

## Returns

energy change of the move in kcal/mol ([INF](#) / 100. upon any error)

**SWIG Wrapper Notes** This function is attached as method `eval_move()` to objects of type `fold_compound`

15.3.2.2 `vrna_eval_move_pt()`

```
int vrna_eval_move_pt (
    vrna_fold_compound_t * vc,
    short * pt,
    int m1,
    int m2 )
```

```
#include <ViennaRNA/eval.h>
```

Calculate energy of a move (closing or opening of a base pair)

If the parameters `m1` and `m2` are negative, it is deletion (opening) of a base pair, otherwise it is insertion (opening).

## See also

[vrna\\_eval\\_move\(\)](#)

## Parameters

<i>vc</i>	A <code>vrna_fold_compound_t</code> containing the energy parameters and model details
<i>pt</i>	the pair table of the secondary structure
<i>m1</i>	first coordinate of base pair
<i>m2</i>	second coordinate of base pair

## Returns

energy change of the move in 10cal/mol

**SWIG Wrapper Notes** This function is attached as method `eval_move_pt()` to objects of type `fold_compound`

## 15.4 Deprecated Interface for Free Energy Evaluation

Deprecated Energy Evaluation functions.

### 15.4.1 Detailed Description

Deprecated Energy Evaluation functions.

Using the functions below is discouraged as they have been marked deprecated and will be removed from the library in the (near) future! Collaboration diagram for Deprecated Interface for Free Energy Evaluation:

### Functions

- float [energy\\_of\\_structure](#) (const char \*string, const char \*structure, int verbosity\_level)  
*Calculate the free energy of an already folded RNA using global model detail settings.*
- float [energy\\_of\\_struct\\_par](#) (const char \*string, const char \*structure, [vrna\\_param\\_t](#) \*parameters, int verbosity\_level)  
*Calculate the free energy of an already folded RNA.*
- float [energy\\_of\\_circ\\_structure](#) (const char \*string, const char \*structure, int verbosity\_level)  
*Calculate the free energy of an already folded circular RNA.*
- float [energy\\_of\\_circ\\_struct\\_par](#) (const char \*string, const char \*structure, [vrna\\_param\\_t](#) \*parameters, int verbosity\_level)  
*Calculate the free energy of an already folded circular RNA.*
- int [energy\\_of\\_structure\\_pt](#) (const char \*string, short \*ptable, short \*s, short \*s1, int verbosity\_level)  
*Calculate the free energy of an already folded RNA.*
- int [energy\\_of\\_struct\\_pt\\_par](#) (const char \*string, short \*ptable, short \*s, short \*s1, [vrna\\_param\\_t](#) \*parameters, int verbosity\_level)  
*Calculate the free energy of an already folded RNA.*
- float [energy\\_of\\_move](#) (const char \*string, const char \*structure, int m1, int m2)  
*Calculate energy of a move (closing or opening of a base pair)*
- int [energy\\_of\\_move\\_pt](#) (short \*pt, short \*s, short \*s1, int m1, int m2)  
*Calculate energy of a move (closing or opening of a base pair)*
- int [loop\\_energy](#) (short \*ptable, short \*s, short \*s1, int i)  
*Calculate energy of a loop.*
- float [energy\\_of\\_struct](#) (const char \*string, const char \*structure)
- int [energy\\_of\\_struct\\_pt](#) (const char \*string, short \*ptable, short \*s, short \*s1)
- float [energy\\_of\\_circ\\_struct](#) (const char \*string, const char \*structure)
- int [E\\_Stem](#) (int type, int si1, int sj1, int extLoop, [vrna\\_param\\_t](#) \*P)  
*Compute the energy contribution of a stem branching off a loop-region.*
- [FLT\\_OR\\_DBL exp\\_E\\_ExtLoop](#) (int type, int si1, int sj1, [vrna\\_exp\\_param\\_t](#) \*P)
- [FLT\\_OR\\_DBL exp\\_E\\_Stem](#) (int type, int si1, int sj1, int extLoop, [vrna\\_exp\\_param\\_t](#) \*P)
- PRIVATE int [E\\_IntLoop](#) (int n1, int n2, int type, int type\_2, int si1, int sj1, int sp1, int sq1, [vrna\\_param\\_t](#) \*P)
- PRIVATE [FLT\\_OR\\_DBL exp\\_E\\_IntLoop](#) (int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short sq1, [vrna\\_exp\\_param\\_t](#) \*P)

## Variables

- int [cut\\_point](#)  
*first pos of second seq for cofolding*
- int [eos\\_debug](#)  
*verbose info from energy\_of\_struct*

## 15.4.2 Function Documentation

### 15.4.2.1 energy\_of\_structure()

```
float energy_of_structure (
    const char * string,
    const char * structure,
    int verbosity_level )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA using global model detail settings.

If verbosity level is set to a value >0, energies of structure elements are printed to stdout

#### Note

OpenMP: This function relies on several global model settings variables and thus is not to be considered threadsafe. See [energy\\_of\\_struct\\_par\(\)](#) for a completely threadsafe implementation.

**Deprecated** Use [vrna\\_eval\\_structure\(\)](#) or [vrna\\_eval\\_structure\\_verbose\(\)](#) instead!

#### See also

[vrna\\_eval\\_structure\(\)](#)

#### Parameters

<i>string</i>	RNA sequence
<i>structure</i>	secondary structure in dot-bracket notation
<i>verbosity_level</i>	a flag to turn verbose output on/off

#### Returns

the free energy of the input structure given the input sequence in kcal/mol

### 15.4.2.2 energy\_of\_struct\_par()

```
float energy_of_struct_par (
    const char * string,
    const char * structure,
    vrna_param_t * parameters,
    int verbosity_level )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA.

If verbosity level is set to a value >0, energies of structure elements are printed to stdout

**Deprecated** Use [vrna\\_eval\\_structure\(\)](#) or [vrna\\_eval\\_structure\\_verbose\(\)](#) instead!

See also

[vrna\\_eval\\_structure\(\)](#)

#### Parameters

<i>string</i>	RNA sequence in uppercase letters
<i>structure</i>	Secondary structure in dot-bracket notation
<i>parameters</i>	A data structure containing the prescaled energy contributions and the model details.
<i>verbosity_level</i>	A flag to turn verbose output on/off

#### Returns

The free energy of the input structure given the input sequence in kcal/mol

### 15.4.2.3 energy\_of\_circ\_structure()

```
float energy_of_circ_structure (
    const char * string,
    const char * structure,
    int verbosity_level )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded circular RNA.

#### Note

OpenMP: This function relies on several global model settings variables and thus is not to be considered threadsafe. See [energy\\_of\\_circ\\_struct\\_par\(\)](#) for a completely threadsafe implementation.

If verbosity level is set to a value >0, energies of structure elements are printed to stdout

**Deprecated** Use [vrna\\_eval\\_structure\(\)](#) or [vrna\\_eval\\_structure\\_verbose\(\)](#) instead!

See also

[vrna\\_eval\\_structure\(\)](#)

## Parameters

<i>string</i>	RNA sequence
<i>structure</i>	Secondary structure in dot-bracket notation
<i>verbosity_level</i>	A flag to turn verbose output on/off

## Returns

The free energy of the input structure given the input sequence in kcal/mol

15.4.2.4 `energy_of_circ_struct_par()`

```
float energy_of_circ_struct_par (
    const char * string,
    const char * structure,
    vrna_param_t * parameters,
    int verbosity_level )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded circular RNA.

If verbosity level is set to a value >0, energies of structure elements are printed to stdout

**Deprecated** Use `vrna_eval_structure()` or `vrna_eval_structure_verbose()` instead!

## See also

[vrna\\_eval\\_structure\(\)](#)

## Parameters

<i>string</i>	RNA sequence
<i>structure</i>	Secondary structure in dot-bracket notation
<i>parameters</i>	A data structure containing the prescaled energy contributions and the model details.
<i>verbosity_level</i>	A flag to turn verbose output on/off

## Returns

The free energy of the input structure given the input sequence in kcal/mol

15.4.2.5 `energy_of_structure_pt()`

```
int energy_of_structure_pt (
    const char * string,
```

```

    short * ptable,
    short * s,
    short * s1,
    int verbosity_level )

```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA.

If verbosity level is set to a value >0, energies of structure elements are printed to stdout

#### Note

OpenMP: This function relies on several global model settings variables and thus is not to be considered threadsafe. See [energy\\_of\\_struct\\_pt\\_par\(\)](#) for a completely threadsafe implementation.

**Deprecated** Use [vrna\\_eval\\_structure\\_pt\(\)](#) or [vrna\\_eval\\_structure\\_pt\\_verbose\(\)](#) instead!

#### See also

[vrna\\_eval\\_structure\\_pt\(\)](#)

#### Parameters

<i>string</i>	RNA sequence
<i>ptable</i>	the pair table of the secondary structure
<i>s</i>	encoded RNA sequence
<i>s1</i>	encoded RNA sequence
<i>verbosity_level</i>	a flag to turn verbose output on/off

#### Returns

the free energy of the input structure given the input sequence in 10kcal/mol

#### 15.4.2.6 energy\_of\_struct\_pt\_par()

```

int energy_of_struct_pt_par (
    const char * string,
    short * ptable,
    short * s,
    short * s1,
    vrna_param_t * parameters,
    int verbosity_level )

```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA.

If verbosity level is set to a value >0, energies of structure elements are printed to stdout

**Deprecated** Use [vrna\\_eval\\_structure\\_pt\(\)](#) or [vrna\\_eval\\_structure\\_pt\\_verbose\(\)](#) instead!



See also

[vrna\\_eval\\_structure\\_pt\(\)](#)

#### Parameters

<i>string</i>	RNA sequence in uppercase letters
<i>ptable</i>	The pair table of the secondary structure
<i>s</i>	Encoded RNA sequence
<i>s1</i>	Encoded RNA sequence
<i>parameters</i>	A data structure containing the prescaled energy contributions and the model details.
<i>verbosity_level</i>	A flag to turn verbose output on/off

#### Returns

The free energy of the input structure given the input sequence in 10kcal/mol

#### 15.4.2.7 energy\_of\_move()

```
float energy_of_move (
    const char * string,
    const char * structure,
    int m1,
    int m2 )
```

```
#include <ViennaRNA/eval.h>
```

Calculate energy of a move (closing or opening of a base pair)

If the parameters m1 and m2 are negative, it is deletion (opening) of a base pair, otherwise it is insertion (opening).

**Deprecated** Use [vrna\\_eval\\_move\(\)](#) instead!

See also

[vrna\\_eval\\_move\(\)](#)

#### Parameters

<i>string</i>	RNA sequence
<i>structure</i>	secondary structure in dot-bracket notation
<i>m1</i>	first coordinate of base pair
<i>m2</i>	second coordinate of base pair

**Returns**

energy change of the move in kcal/mol

**15.4.2.8 energy\_of\_move\_pt()**

```
int energy_of_move_pt (
    short * pt,
    short * s,
    short * s1,
    int m1,
    int m2 )
```

```
#include <ViennaRNA/eval.h>
```

Calculate energy of a move (closing or opening of a base pair)

If the parameters m1 and m2 are negative, it is deletion (opening) of a base pair, otherwise it is insertion (opening).

**Deprecated** Use [vrna\\_eval\\_move\\_pt\(\)](#) instead!

**See also**

[vrna\\_eval\\_move\\_pt\(\)](#)

**Parameters**

<i>pt</i>	the pair table of the secondary structure
<i>s</i>	encoded RNA sequence
<i>s1</i>	encoded RNA sequence
<i>m1</i>	first coordinate of base pair
<i>m2</i>	second coordinate of base pair

**Returns**

energy change of the move in 10cal/mol

**15.4.2.9 loop\_energy()**

```
int loop_energy (
    short * ptable,
    short * s,
    short * s1,
    int i )
```

```
#include <ViennaRNA/eval.h>
```

Calculate energy of a loop.

**Deprecated** Use [vrna\\_eval\\_loop\\_pt\(\)](#) instead!

See also

[vrna\\_eval\\_loop\\_pt\(\)](#)

Parameters

<i>ptable</i>	the pair table of the secondary structure
<i>s</i>	encoded RNA sequence
<i>s1</i>	encoded RNA sequence
<i>i</i>	position of covering base pair

Returns

free energy of the loop in 10cal/mol

#### 15.4.2.10 `energy_of_struct()`

```
float energy_of_struct (
    const char * string,
    const char * structure )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA

Note

This function is not entirely threadsafe! Depending on the state of the global variable [eos\\_debug](#) it prints energy information to stdout or not...

**Deprecated** This function is deprecated and should not be used in future programs! Use [energy\\_of\\_structure\(\)](#) instead!

See also

[energy\\_of\\_structure](#), [energy\\_of\\_circ\\_struct\(\)](#), [energy\\_of\\_struct\\_pt\(\)](#)

Parameters

<i>string</i>	RNA sequence
<i>structure</i>	secondary structure in dot-bracket notation

**Returns**

the free energy of the input structure given the input sequence in kcal/mol

**15.4.2.11 energy\_of\_struct\_pt()**

```
int energy_of_struct_pt (
    const char * string,
    short * ptable,
    short * s,
    short * s1 )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded RNA

**Note**

This function is not entirely threadsafe! Depending on the state of the global variable [eos\\_debug](#) it prints energy information to stdout or not...

**Deprecated** This function is deprecated and should not be used in future programs! Use [energy\\_of\\_structure\\_pt\(\)](#) instead!

**See also**

[make\\_pair\\_table\(\)](#), [energy\\_of\\_structure\(\)](#)

**Parameters**

<i>string</i>	RNA sequence
<i>ptable</i>	the pair table of the secondary structure
<i>s</i>	encoded RNA sequence
<i>s1</i>	encoded RNA sequence

**Returns**

the free energy of the input structure given the input sequence in 10kcal/mol

**15.4.2.12 energy\_of\_circ\_struct()**

```
float energy_of_circ_struct (
    const char * string,
    const char * structure )
```

```
#include <ViennaRNA/eval.h>
```

Calculate the free energy of an already folded circular RNA

**Note**

This function is not entirely threadsafe! Depending on the state of the global variable `eos_debug` it prints energy information to stdout or not...

**Deprecated** This function is deprecated and should not be used in future programs Use `energy_of_circ_structure()` instead!

**See also**

`energy_of_circ_structure()`, `energy_of_struct()`, `energy_of_struct_pt()`

**Parameters**

<i>string</i>	RNA sequence
<i>structure</i>	secondary structure in dot-bracket notation

**Returns**

the free energy of the input structure given the input sequence in kcal/mol

**15.4.2.13 E\_Stem()**

```
int E_Stem (
    int type,
    int si1,
    int sj1,
    int extLoop,
    vrna_param_t * P )

#include <ViennaRNA/loops/external.h>
```

Compute the energy contribution of a stem branching off a loop-region.

This function computes the energy contribution of a stem that branches off a loop region. This can be the case in multiloops, when a stem branching off increases the degree of the loop but also *immediately interior base pairs* of an exterior loop contribute free energy. To switch the behavior of the function according to the evaluation of a multiloop- or exterior-loop-stem, you pass the flag 'extLoop'. The returned energy contribution consists of a TerminalAU penalty if the pair type is greater than 2, dangling end contributions of mismatching nucleotides adjacent to the stem if only one of the si1, sj1 parameters is greater than 0 and mismatch energies if both mismatching nucleotides are positive values. Thus, to avoid incorporating dangling end or mismatch energies just pass a negative number, e.g. -1 to the mismatch argument.

This is an illustration of how the energy contribution is assembled:

```

      3'   5'
      |   |
      X - Y
5'-si1      sj1-3'
```

Here, (X,Y) is the base pair that closes the stem that branches off a loop region. The nucleotides si1 and sj1 are the 5'- and 3'- mismatches, respectively. If the base pair type of (X,Y) is greater than 2 (i.e. an A-U or G-U pair, the TerminalAU penalty will be included in the energy contribution returned. If si1 and sj1 are both nonnegative numbers, mismatch energies will also be included. If one of si1 or sj1 is a negative value, only 5' or 3' dangling end contributions are taken into account. To prohibit any of these mismatch contributions to be incorporated, just pass a negative number to both, si1 and sj1. In case the argument extLoop is 0, the returned energy contribution also includes the *internal-loop-penalty* of a multiloop stem with closing pair type.

See also

E\_MLstem()  
E\_ExtLoop()

Note

This function is threadsafe

**Deprecated** Please use one of the functions [vrna\\_E\\_ext\\_stem\(\)](#) and E\_MLstem() instead! Use the former for cases where extLoop != 0 and the latter otherwise.

Parameters

<i>type</i>	The pair type of the first base pair un the stem
<i>si1</i>	The 5'-mismatching nucleotide
<i>sj1</i>	The 3'-mismatching nucleotide
<i>extLoop</i>	A flag that indicates whether the contribution reflects the one of an exterior loop or not
<i>P</i>	The data structure containing scaled energy parameters

Returns

The Free energy of the branch off the loop in dcal/mol

#### 15.4.2.14 exp\_E\_ExtLoop()

```
FLT_OR_DBL exp_E_ExtLoop (
    int type,
    int si1,
    int sj1,
    vrna_exp_param_t * P )
#include <ViennaRNA/loops/external.h>
```

This is the partition function variant of E\_ExtLoop()

**Deprecated** Use [vrna\\_exp\\_E\\_ext\\_stem\(\)](#) instead!

See also

E\_ExtLoop()

Returns

The Boltzmann weighted energy contribution of the introduced exterior-loop stem

#### 15.4.2.15 exp\_E\_Stem()

```
FLT_OR_DBL exp_E_Stem (
    int type,
    int sil,
    int sj1,
    int extLoop,
    vrna_exp_param_t * P )
```

```
#include <ViennaRNA/loops/external.h>
```

Compute the Boltzmann weighted energy contribution of a stem branching off a loop-region

This is the partition function variant of [E\\_Stem\(\)](#)

See also

[E\\_Stem\(\)](#)

Note

This function is threadsafe

Returns

The Boltzmann weighted energy contribution of the branch off the loop

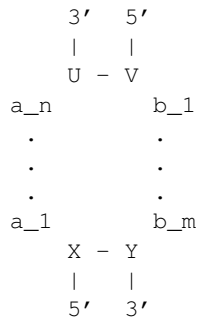
#### 15.4.2.16 E\_IntLoop()

```
PRIVATE int E_IntLoop (
    int n1,
    int n2,
    int type,
    int type_2,
    int sil,
    int sj1,
    int sp1,
    int sq1,
    vrna_param_t * P )
```

```
#include <ViennaRNA/loops/internal.h>
```

### Compute the Energy of an interior-loop

This function computes the free energy  $\Delta G$  of an interior-loop with the following structure:



This general structure depicts an interior-loop that is closed by the base pair (X,Y). The enclosed base pair is (V,U) which leaves the unpaired bases  $a_1$ - $a_n$  and  $b_1$ - $b_n$  that constitute the loop. In this example, the length of the interior-loop is  $(n + m)$  where  $n$  or  $m$  may be 0 resulting in a bulge-loop or base pair stack. The mismatching nucleotides for the closing pair (X,Y) are:

5'-mismatch:  $a_1$

3'-mismatch:  $b_m$

and for the enclosed base pair (V,U):

5'-mismatch:  $b_1$

3'-mismatch:  $a_n$

#### Note

Base pairs are always denoted in 5'->3' direction. Thus the enclosed base pair must be 'turned around' when evaluating the free energy of the interior-loop

#### See also

[scale\\_parameters\(\)](#)  
[vrna\\_param\\_t](#)

#### Note

This function is threadsafe

#### Parameters

$n1$	The size of the 'left'-loop (number of unpaired nucleotides)
$n2$	The size of the 'right'-loop (number of unpaired nucleotides)
$type$	The pair type of the base pair closing the interior loop
$type_{\leftarrow 2}$	The pair type of the enclosed base pair
$si1$	The 5'-mismatching nucleotide of the closing pair
$sj1$	The 3'-mismatching nucleotide of the closing pair
$sp1$	The 3'-mismatching nucleotide of the enclosed pair
$sq1$	The 5'-mismatching nucleotide of the enclosed pair
$P$	The datastructure containing scaled energy parameters



**Returns**

The Free energy of the Interior-loop in dcal/mol

**15.4.2.17 exp\_E\_IntLoop()**

```
PRIVATE FLT_OR_DBL exp_E_IntLoop (
    int u1,
    int u2,
    int type,
    int type2,
    short si1,
    short sj1,
    short sp1,
    short sq1,
    vrna_exp_param_t * P )

#include <ViennaRNA/loops/internal.h>
```

Compute Boltzmann weight  $e^{-\Delta G/kT}$  of interior loop

multiply by scale[u1+u2+2] for scaling

**See also**

[get\\_scaled\\_pf\\_parameters\(\)](#)  
[vrna\\_exp\\_param\\_t](#)  
[E\\_IntLoop\(\)](#)

**Note**

This function is threadsafe

**Parameters**

<i>u1</i>	The size of the 'left'-loop (number of unpaired nucleotides)
<i>u2</i>	The size of the 'right'-loop (number of unpaired nucleotides)
<i>type</i>	The pair type of the base pair closing the interior loop
<i>type2</i>	The pair type of the enclosed base pair
<i>si1</i>	The 5'-mismatching nucleotide of the closing pair
<i>sj1</i>	The 3'-mismatching nucleotide of the closing pair
<i>sp1</i>	The 3'-mismatching nucleotide of the enclosed pair
<i>sq1</i>	The 5'-mismatching nucleotide of the enclosed pair
<i>P</i>	The datastructure containing scaled Boltzmann weights of the energy parameters

**Returns**

The Boltzmann weight of the Interior-loop

## 15.5 The RNA Folding Grammar

The RNA folding grammar as implemented in RNAlib.

### 15.5.1 Detailed Description

The RNA folding grammar as implemented in RNAlib.

Collaboration diagram for The RNA Folding Grammar:

#### Modules

- [Fine-tuning of the Implemented Models](#)  
*Functions and data structures to fine-tune the implemented secondary structure evaluation model.*
- [Energy Parameters](#)  
*All relevant functions to retrieve and copy pre-calculated energy parameter sets as well as reading/writing the energy parameter set from/to file(s).*
- [Extending the Folding Grammar with Additional Domains](#)  
*This module covers simple and straight-forward extensions to the RNA folding grammar.*
- [Constraining the RNA Folding Grammar](#)  
*This module provides general functions that allow for an easy control of constrained secondary structure prediction and evaluation.*

#### Files

- file [grammar.h](#)  
*Implementations for the RNA folding grammar.*

#### Data Structures

- struct [vrna\\_gr\\_aux\\_s](#)

### 15.5.2 Data Structure Documentation

#### 15.5.2.1 struct vrna\_gr\_aux\_s

##### Data Fields

- `vrna_callback_gr_cond * cb\_proc`  
*A callback for pre- and post-processing of auxiliary grammar rules.*

## 15.6 Fine-tuning of the Implemented Models

Functions and data structures to fine-tune the implemented secondary structure evaluation model.

### 15.6.1 Detailed Description

Functions and data structures to fine-tune the implemented secondary structure evaluation model.

Collaboration diagram for Fine-tuning of the Implemented Models:

#### Files

- file [model.h](#)  
*The model details data structure and its corresponding modifiers.*

#### Data Structures

- struct [vrna\\_md\\_s](#)  
*The data structure that contains the complete model details used throughout the calculations. [More...](#)*

#### Macros

- `#define VRNA_MODEL_DEFAULT_TEMPERATURE 37.0`  
*Default temperature for structure prediction and free energy evaluation in °C*
- `#define VRNA_MODEL_DEFAULT_PF_SCALE -1`  
*Default scaling factor for partition function computations.*
- `#define VRNA_MODEL_DEFAULT_BETA_SCALE 1.`  
*Default scaling factor for absolute thermodynamic temperature in Boltzmann factors.*
- `#define VRNA_MODEL_DEFAULT_DANGLES 2`  
*Default dangling end model.*
- `#define VRNA_MODEL_DEFAULT_SPECIAL_HP 1`  
*Default model behavior for lookup of special tri-, tetra-, and hexa-loops.*
- `#define VRNA_MODEL_DEFAULT_NO_LP 0`  
*Default model behavior for so-called 'lonely pairs'.*
- `#define VRNA_MODEL_DEFAULT_NO_GU 0`  
*Default model behavior for G-U base pairs.*
- `#define VRNA_MODEL_DEFAULT_NO_GU_CLOSURE 0`  
*Default model behavior for G-U base pairs closing a loop.*
- `#define VRNA_MODEL_DEFAULT_CIRC 0`  
*Default model behavior to treat a molecule as a circular RNA (DNA)*
- `#define VRNA_MODEL_DEFAULT_GQUAD 0`  
*Default model behavior regarding the treatment of G-Quadruplexes.*
- `#define VRNA_MODEL_DEFAULT_UNIQ_ML 0`  
*Default behavior of the model regarding unique multi-branch loop decomposition.*
- `#define VRNA_MODEL_DEFAULT_ENERGY_SET 0`

- *Default model behavior on which energy set to use.*  
• #define `VRNA_MODEL_DEFAULT_BACKTRACK` 1
- *Default model behavior with regards to backtracking of structures.*  
• #define `VRNA_MODEL_DEFAULT_BACKTRACK_TYPE` 'F'
- *Default model behavior on what type of backtracking to perform.*  
• #define `VRNA_MODEL_DEFAULT_COMPUTE_BPP` 1
- *Default model behavior with regards to computing base pair probabilities.*  
• #define `VRNA_MODEL_DEFAULT_MAX_BP_SPAN` -1
- *Default model behavior for the allowed maximum base pair span.*  
• #define `VRNA_MODEL_DEFAULT_WINDOW_SIZE` -1
- *Default model behavior for the sliding window approach.*  
• #define `VRNA_MODEL_DEFAULT_LOG_ML` 0
- *Default model behavior on how to evaluate the energy contribution of multi-branch loops.*  
• #define `VRNA_MODEL_DEFAULT_ALI_OLD_EN` 0
- *Default model behavior for consensus structure energy evaluation.*  
• #define `VRNA_MODEL_DEFAULT_ALI_RIBO` 0
- *Default model behavior for consensus structure co-variance contribution assessment.*  
• #define `VRNA_MODEL_DEFAULT_ALI_CV_FACT` 1.
- *Default model behavior for weighting the co-variance score in consensus structure prediction.*  
• #define `VRNA_MODEL_DEFAULT_ALI_NC_FACT` 1.
- *Default model behavior for weighting the nucleotide conservation? in consensus structure prediction.*  
• #define `MAXALPHA` 20
- *Maximal length of alphabet.*

## Typedefs

- typedef struct `vrna_md_s` `vrna_md_t`  
*Typename for the model details data structure `vrna_md_s`.*

## Functions

- void `vrna_md_set_default` (`vrna_md_t` \*md)  
*Apply default model details to a provided `vrna_md_t` data structure.*
- void `vrna_md_update` (`vrna_md_t` \*md)  
*Update the model details data structure.*
- `vrna_md_t` \* `vrna_md_copy` (`vrna_md_t` \*md\_to, const `vrna_md_t` \*md\_from)  
*Copy/Clone a `vrna_md_t` model.*
- char \* `vrna_md_option_string` (`vrna_md_t` \*md)  
*Get a corresponding commandline parameter string of the options in a `vrna_md_t`.*
- void `vrna_md_defaults_reset` (`vrna_md_t` \*md\_p)  
*Reset the global default model details to a specific set of parameters, or their initial values.*
- void `vrna_md_defaults_temperature` (double T)  
*Set default temperature for energy evaluation of loops.*
- double `vrna_md_defaults_temperature_get` (void)  
*Get default temperature for energy evaluation of loops.*
- void `vrna_md_defaults_betaScale` (double b)  
*Set default scaling factor of thermodynamic temperature in Boltzmann factors.*
- double `vrna_md_defaults_betaScale_get` (void)  
*Get default scaling factor of thermodynamic temperature in Boltzmann factors.*

- void [vrna\\_md\\_defaults\\_dangles](#) (int d)  
*Set default dangle model for structure prediction.*
- int [vrna\\_md\\_defaults\\_dangles\\_get](#) (void)  
*Get default dangle model for structure prediction.*
- void [vrna\\_md\\_defaults\\_special\\_hp](#) (int flag)  
*Set default behavior for lookup of tabulated free energies for special hairpin loops, such as Tri-, Tetra-, or Hexa-loops.*
- int [vrna\\_md\\_defaults\\_special\\_hp\\_get](#) (void)  
*Get default behavior for lookup of tabulated free energies for special hairpin loops, such as Tri-, Tetra-, or Hexa-loops.*
- void [vrna\\_md\\_defaults\\_noLP](#) (int flag)  
*Set default behavior for prediction of canonical secondary structures.*
- int [vrna\\_md\\_defaults\\_noLP\\_get](#) (void)  
*Get default behavior for prediction of canonical secondary structures.*
- void [vrna\\_md\\_defaults\\_noGU](#) (int flag)  
*Set default behavior for treatment of G-U wobble pairs.*
- int [vrna\\_md\\_defaults\\_noGU\\_get](#) (void)  
*Get default behavior for treatment of G-U wobble pairs.*
- void [vrna\\_md\\_defaults\\_noGUclosure](#) (int flag)  
*Set default behavior for G-U pairs as closing pair for loops.*
- int [vrna\\_md\\_defaults\\_noGUclosure\\_get](#) (void)  
*Get default behavior for G-U pairs as closing pair for loops.*
- void [vrna\\_md\\_defaults\\_logML](#) (int flag)  
*Set default behavior recomputing free energies of multi-branch loops using a logarithmic model.*
- int [vrna\\_md\\_defaults\\_logML\\_get](#) (void)  
*Get default behavior recomputing free energies of multi-branch loops using a logarithmic model.*
- void [vrna\\_md\\_defaults\\_circ](#) (int flag)  
*Set default behavior whether input sequences are circularized.*
- int [vrna\\_md\\_defaults\\_circ\\_get](#) (void)  
*Get default behavior whether input sequences are circularized.*
- void [vrna\\_md\\_defaults\\_gquad](#) (int flag)  
*Set default behavior for treatment of G-Quadruplexes.*
- int [vrna\\_md\\_defaults\\_gquad\\_get](#) (void)  
*Get default behavior for treatment of G-Quadruplexes.*
- void [vrna\\_md\\_defaults\\_uniq\\_ML](#) (int flag)  
*Set default behavior for creating additional matrix for unique multi-branch loop prediction.*
- int [vrna\\_md\\_defaults\\_uniq\\_ML\\_get](#) (void)  
*Get default behavior for creating additional matrix for unique multi-branch loop prediction.*
- void [vrna\\_md\\_defaults\\_energy\\_set](#) (int e)  
*Set default energy set.*
- int [vrna\\_md\\_defaults\\_energy\\_set\\_get](#) (void)  
*Get default energy set.*
- void [vrna\\_md\\_defaults\\_backtrack](#) (int flag)  
*Set default behavior for whether to backtrack secondary structures.*
- int [vrna\\_md\\_defaults\\_backtrack\\_get](#) (void)  
*Get default behavior for whether to backtrack secondary structures.*
- void [vrna\\_md\\_defaults\\_backtrack\\_type](#) (char t)  
*Set default backtrack type, i.e. which DP matrix is used.*
- char [vrna\\_md\\_defaults\\_backtrack\\_type\\_get](#) (void)  
*Get default backtrack type, i.e. which DP matrix is used.*
- void [vrna\\_md\\_defaults\\_compute\\_bpp](#) (int flag)  
*Set the default behavior for whether to compute base pair probabilities after partition function computation.*
- int [vrna\\_md\\_defaults\\_compute\\_bpp\\_get](#) (void)

- Get the default behavior for whether to compute base pair probabilities after partition function computation.*

  - void `vrna_md_defaults_max_bp_span` (int span)

*Set default maximal base pair span.*
- int `vrna_md_defaults_max_bp_span_get` (void)

*Get default maximal base pair span.*
- void `vrna_md_defaults_min_loop_size` (int size)

*Set default minimal loop size.*
- int `vrna_md_defaults_min_loop_size_get` (void)

*Get default minimal loop size.*
- void `vrna_md_defaults_window_size` (int size)

*Set default window size for sliding window structure prediction approaches.*
- int `vrna_md_defaults_window_size_get` (void)

*Get default window size for sliding window structure prediction approaches.*
- void `vrna_md_defaults_oldAliEn` (int flag)

*Set default behavior for whether to use old energy model for comparative structure prediction.*
- int `vrna_md_defaults_oldAliEn_get` (void)

*Get default behavior for whether to use old energy model for comparative structure prediction.*
- void `vrna_md_defaults_ribo` (int flag)

*Set default behavior for whether to use Ribosum Scoring in comparative structure prediction.*
- int `vrna_md_defaults_ribo_get` (void)

*Get default behavior for whether to use Ribosum Scoring in comparative structure prediction.*
- void `vrna_md_defaults_cv_fact` (double factor)

*Set the default co-variance scaling factor used in comparative structure prediction.*
- double `vrna_md_defaults_cv_fact_get` (void)

*Get the default co-variance scaling factor used in comparative structure prediction.*
- void `vrna_md_defaults_nc_fact` (double factor)
- double `vrna_md_defaults_nc_fact_get` (void)
- void `vrna_md_defaults_sfact` (double factor)

*Set the default scaling factor used to avoid under-/overflows in partition function computation.*
- double `vrna_md_defaults_sfact_get` (void)

*Get the default scaling factor used to avoid under-/overflows in partition function computation.*
- void `set_model_details` (`vrna_md_t` \*md)

*Set default model details.*

## Variables

- double `temperature`

*Rescale energy parameters to a temperature in degC.*
- double `pf_scale`

*A scaling factor used by `pf_fold()` to avoid overflows.*
- int `dangles`

*Switch the energy model for dangling end contributions (0, 1, 2, 3)*
- int `tetra_loop`

*Include special stabilizing energies for some tri-, tetra- and hexa-loops;.*
- int `noLonelyPairs`

*Global switch to avoid/allow helices of length 1.*
- int `noGU`

*Global switch to forbid/allow GU base pairs at all.*
- int `no_closingGU`

*GU allowed only inside stacks if set to 1.*

- int `circ`  
*backward compatibility variable.. this does not effect anything*
- int `gquad`  
*Allow G-quadruplex formation.*
- int `uniq_ML`  
*do ML decomposition uniquely (for subopt)*
- int `energy_set`  
*0 = BP; 1=any with GC; 2=any with AU-parameter*
- int `do_backtrack`  
*do backtracking, i.e. compute secondary structures or base pair probabilities*
- char `backtrack_type`  
*A backtrack array marker for `inverse_fold()`*
- char \* `nonstandards`  
*contains allowed non standard base pairs*
- int `max_bp_span`  
*Maximum allowed base pair span.*
- int `oldAlfEn`  
*use old alifold energies (with gaps)*
- int `ribo`  
*use ribosum matrices*
- int `logML`  
*if nonzero use logarithmic ML energy in `energy_of_struct`*

## 15.6.2 Data Structure Documentation

### 15.6.2.1 struct `vrna_md_s`

The data structure that contains the complete model details used throughout the calculations.

For convenience reasons, we provide the type name `vrna_md_t` to address this data structure without the use of the struct keyword

See also

`vrna_md_set_default()`, `set_model_details()`, `vrna_md_update()`, `vrna_md_t`

**SWIG Wrapper Notes** This data structure is wrapped as an object **md** with multiple related functions attached as methods.

A new set of default parameters can be obtained by calling the constructor of **md**:

- `md()` – Initialize with default settings

The resulting object has a list of attached methods which directly correspond to functions that mainly operate on the corresponding C data structure:

- `reset()` – `vrna_md_set_default()`
- `set_from_globals()` – `set_model_details()`
- `option_string()` – `vrna_md_option_string()`

Note, that default parameters can be modified by directly setting any of the following global variables. Internally, getting/setting default parameters using their global variable representative translates into calls of the following functions, therefore these wrappers for these functions do not exist in the scripting language interface(s):



global variable	C getter	C setter
temperature	<a href="#">vrna_md_defaults_temperature_get()</a>	<a href="#">vrna_md_defaults_temperature()</a>
dangles	<a href="#">vrna_md_defaults_dangles_get()</a>	<a href="#">vrna_md_defaults_dangles()</a>
betaScale	<a href="#">vrna_md_defaults_betaScale_get()</a>	<a href="#">vrna_md_defaults_betaScale()</a>
tetra_loop	this is an alias of <i>special_hp</i>	
special_hp	<a href="#">vrna_md_defaults_special_hp_get()</a>	<a href="#">vrna_md_defaults_special_hp()</a>
noLonelyPairs	this is an alias of <i>noLP</i>	
noLP	<a href="#">vrna_md_defaults_noLP_get()</a>	<a href="#">vrna_md_defaults_noLP()</a>
noGU	<a href="#">vrna_md_defaults_noGU_get()</a>	<a href="#">vrna_md_defaults_noGU()</a>
no_closingGU	this is an alias of <i>noGUclosure</i>	
noGUclosure	<a href="#">vrna_md_defaults_noGUclosure_get()</a>	<a href="#">vrna_md_defaults_noGUclosure()</a>
logML	<a href="#">vrna_md_defaults_logML_get()</a>	<a href="#">vrna_md_defaults_logML()</a>
circ	<a href="#">vrna_md_defaults_circ_get()</a>	<a href="#">vrna_md_defaults_circ()</a>
gquad	<a href="#">vrna_md_defaults_gquad_get()</a>	<a href="#">vrna_md_defaults_gquad()</a>
uniq_ML	<a href="#">vrna_md_defaults_uniq_ML_get()</a>	<a href="#">vrna_md_defaults_uniq_ML()</a>
energy_set	<a href="#">vrna_md_defaults_energy_set_get()</a>	<a href="#">vrna_md_defaults_energy_set()</a>
backtrack	<a href="#">vrna_md_defaults_backtrack_get()</a>	<a href="#">vrna_md_defaults_backtrack()</a>
backtrack_type	<a href="#">vrna_md_defaults_backtrack_type_get()</a>	<a href="#">vrna_md_defaults_backtrack_type()</a>
do_backtrack	this is an alias of <i>compute_bpp</i>	
compute_bpp	<a href="#">vrna_md_defaults_compute_bpp_get()</a>	<a href="#">vrna_md_defaults_compute_bpp()</a>
max_bp_span	<a href="#">vrna_md_defaults_max_bp_span_get()</a>	<a href="#">vrna_md_defaults_max_bp_span()</a>
min_loop_size	<a href="#">vrna_md_defaults_min_loop_size_get()</a>	<a href="#">vrna_md_defaults_min_loop_size()</a>
window_size	<a href="#">vrna_md_defaults_window_size_get()</a>	<a href="#">vrna_md_defaults_window_size()</a>
oldAliEn	<a href="#">vrna_md_defaults_oldAliEn_get()</a>	<a href="#">vrna_md_defaults_oldAliEn()</a>
ribo	<a href="#">vrna_md_defaults_ribo_get()</a>	<a href="#">vrna_md_defaults_ribo()</a>
cv_fact	<a href="#">vrna_md_defaults_cv_fact_get()</a>	<a href="#">vrna_md_defaults_cv_fact()</a>
nc_fact	<a href="#">vrna_md_defaults_nc_fact_get()</a>	<a href="#">vrna_md_defaults_nc_fact()</a>
sfact	<a href="#">vrna_md_defaults_sfact_get()</a>	<a href="#">vrna_md_defaults_sfact()</a>

#### Data Fields

- double [temperature](#)  
The temperature used to scale the thermodynamic parameters.
- double [betaScale](#)  
A scaling factor for the thermodynamic temperature of the Boltzmann factors.
- int [dangles](#)  
Specifies the dangle model used in any energy evaluation (0, 1, 2 or 3)
- int [special\\_hp](#)  
Include special hairpin contributions for tri, tetra and hexaloops.
- int [noLP](#)  
Only consider canonical structures, i.e. no 'lonely' base pairs.
- int [noGU](#)  
Do not allow GU pairs.

- int [noGUclosure](#)  
*Do not allow loops to be closed by GU pair.*
- int [logML](#)  
*Use logarithmic scaling for multiloops.*
- int [circ](#)  
*Assume RNA to be circular instead of linear.*
- int [gquad](#)  
*Include G-quadruplexes in structure prediction.*
- int [uniq\\_ML](#)  
*Flag to ensure unique multi-branch loop decomposition during folding.*
- int [energy\\_set](#)  
*Specifies the energy set that defines set of compatible base pairs.*
- int [backtrack](#)  
*Specifies whether or not secondary structures should be backtraced.*
- char [backtrack\\_type](#)  
*Specifies in which matrix to backtrack.*
- int [compute\\_bpp](#)  
*Specifies whether or not backward recursions for base pair probability (bpp) computation will be performed.*
- char [nonstandards](#) [64]  
*contains allowed non standard bases*
- int [max\\_bp\\_span](#)  
*maximum allowed base pair span*
- int [min\\_loop\\_size](#)  
*Minimum size of hairpin loops.*
- int [window\\_size](#)  
*Size of the sliding window for locally optimal structure prediction.*
- int [oldAliEn](#)  
*Use old alifold energy model.*
- int [ribo](#)  
*Use ribosum scoring table in alifold energy model.*
- double [cv\\_fact](#)  
*Co-variance scaling factor for consensus structure prediction.*
- double [nc\\_fact](#)  
*Scaling factor to weight co-variance contributions of non-canonical pairs.*
- double [sfact](#)  
*Scaling factor for partition function scaling.*
- int [rtype](#) [8]  
*Reverse base pair type array.*
- short [alias](#) [MAXALPHA+1]  
*alias of an integer nucleotide representation*
- int [pair](#) [MAXALPHA+1][MAXALPHA+1]  
*Integer representation of a base pair.*

#### 15.6.2.1.1 Field Documentation

#### 15.6.2.1.1.1 dangles

```
int vrna_md_s::dangles
```

Specifies the dangle model used in any energy evaluation (0,1,2 or 3)

If set to 0 no stabilizing energies are assigned to bases adjacent to helices in free ends and multiloops (so called dangling ends). Normally (dangles = 1) dangling end energies are assigned only to unpaired bases and a base cannot participate simultaneously in two dangling ends. In the partition function algorithm [vrna\\_pf\(\)](#) these checks are neglected. To provide comparability between free energy minimization and partition function algorithms, the default setting is 2. This treatment of dangling ends gives more favorable energies to helices directly adjacent to one another, which can be beneficial since such helices often do engage in stabilizing interactions through co-axial stacking.

If set to 3 co-axial stacking is explicitly included for adjacent helices in multiloops. The option affects only mfe folding and energy evaluation ([vrna\\_mfe\(\)](#) and [vrna\\_eval\\_structure\(\)](#)), as well as suboptimal folding ([vrna\\_subopt\(\)](#)) via re-evaluation of energies. Co-axial stacking with one intervening mismatch is not considered so far.

#### Note

Some function do not implement all dangle model but only a subset of (0,1,2,3). In particular, partition function algorithms can only handle 0 and 2. Read the documentation of the particular recurrences or energy evaluation function for information about the provided dangle model.

#### 15.6.2.1.1.2 min\_loop\_size

```
int vrna_md_s::min_loop_size
```

Minimum size of hairpin loops.

#### Note

The default value for this field is [TURN](#), however, it may be 0 in cofolding context.

### 15.6.3 Macro Definition Documentation

#### 15.6.3.1 VRNA\_MODEL\_DEFAULT\_TEMPERATURE

```
#define VRNA_MODEL_DEFAULT_TEMPERATURE 37.0
```

```
#include <ViennaRNA/model.h>
```

Default temperature for structure prediction and free energy evaluation in °C

#### See also

[vrna\\_md\\_t.temperature](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

### 15.6.3.2 VRNA\_MODEL\_DEFAULT\_PF\_SCALE

```
#define VRNA_MODEL_DEFAULT_PF_SCALE -1  
  
#include <ViennaRNA/model.h>
```

Default scaling factor for partition function computations.

See also

[vrna\\_exp\\_param\\_t.pf\\_scale](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

### 15.6.3.3 VRNA\_MODEL\_DEFAULT\_BETA\_SCALE

```
#define VRNA_MODEL_DEFAULT_BETA_SCALE 1.  
  
#include <ViennaRNA/model.h>
```

Default scaling factor for absolute thermodynamic temperature in Boltzmann factors.

See also

[vrna\\_exp\\_param\\_t.alpha](#), [vrna\\_md\\_t.betaScale](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

### 15.6.3.4 VRNA\_MODEL\_DEFAULT\_DANGLES

```
#define VRNA_MODEL_DEFAULT_DANGLES 2  
  
#include <ViennaRNA/model.h>
```

Default dangling end model.

See also

[vrna\\_md\\_t.dangles](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

### 15.6.3.5 VRNA\_MODEL\_DEFAULT\_SPECIAL\_HP

```
#define VRNA_MODEL_DEFAULT_SPECIAL_HP 1  
  
#include <ViennaRNA/model.h>
```

Default model behavior for lookup of special tri-, tetra-, and hexa-loops.

See also

[vrna\\_md\\_t.special\\_hp](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

#### 15.6.3.6 VRNA\_MODEL\_DEFAULT\_NO\_LP

```
#define VRNA_MODEL_DEFAULT_NO_LP 0  
  
#include <ViennaRNA/model.h>
```

Default model behavior for so-called 'lonely pairs'.

See also

[vrna\\_md\\_t.noLP](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

#### 15.6.3.7 VRNA\_MODEL\_DEFAULT\_NO\_GU

```
#define VRNA_MODEL_DEFAULT_NO_GU 0  
  
#include <ViennaRNA/model.h>
```

Default model behavior for G-U base pairs.

See also

[vrna\\_md\\_t.noGU](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

#### 15.6.3.8 VRNA\_MODEL\_DEFAULT\_NO\_GU\_CLOSURE

```
#define VRNA_MODEL_DEFAULT_NO_GU_CLOSURE 0  
  
#include <ViennaRNA/model.h>
```

Default model behavior for G-U base pairs closing a loop.

See also

[vrna\\_md\\_t.noGUclosure](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

#### 15.6.3.9 VRNA\_MODEL\_DEFAULT\_CIRC

```
#define VRNA_MODEL_DEFAULT_CIRC 0  
  
#include <ViennaRNA/model.h>
```

Default model behavior to treat a molecule as a circular RNA (DNA)

See also

[vrna\\_md\\_t.circ](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

#### 15.6.3.10 VRNA\_MODEL\_DEFAULT\_GQUAD

```
#define VRNA_MODEL_DEFAULT_GQUAD 0  
  
#include <ViennaRNA/model.h>
```

Default model behavior regarding the treatment of G-Quadruplexes.

See also

[vrna\\_md\\_t.gquad](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

#### 15.6.3.11 VRNA\_MODEL\_DEFAULT\_UNIQ\_ML

```
#define VRNA_MODEL_DEFAULT_UNIQ_ML 0  
  
#include <ViennaRNA/model.h>
```

Default behavior of the model regarding unique multi-branch loop decomposition.

See also

[vrna\\_md\\_t.uniq\\_ML](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

#### 15.6.3.12 VRNA\_MODEL\_DEFAULT\_ENERGY\_SET

```
#define VRNA_MODEL_DEFAULT_ENERGY_SET 0  
  
#include <ViennaRNA/model.h>
```

Default model behavior on which energy set to use.

See also

[vrna\\_md\\_t.energy\\_set](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

#### 15.6.3.13 VRNA\_MODEL\_DEFAULT\_BACKTRACK

```
#define VRNA_MODEL_DEFAULT_BACKTRACK 1  
  
#include <ViennaRNA/model.h>
```

Default model behavior with regards to backtracking of structures.

See also

[vrna\\_md\\_t.backtrack](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

#### 15.6.3.14 VRNA\_MODEL\_DEFAULT\_BACKTRACK\_TYPE

```
#define VRNA_MODEL_DEFAULT_BACKTRACK_TYPE 'F'  
  
#include <ViennaRNA/model.h>
```

Default model behavior on what type of backtracking to perform.

See also

[vrna\\_md\\_t.backtrack\\_type](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

#### 15.6.3.15 VRNA\_MODEL\_DEFAULT\_COMPUTE\_BPP

```
#define VRNA_MODEL_DEFAULT_COMPUTE_BPP 1  
  
#include <ViennaRNA/model.h>
```

Default model behavior with regards to computing base pair probabilities.

See also

[vrna\\_md\\_t.compute\\_bpp](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

#### 15.6.3.16 VRNA\_MODEL\_DEFAULT\_MAX\_BP\_SPAN

```
#define VRNA_MODEL_DEFAULT_MAX_BP_SPAN -1  
  
#include <ViennaRNA/model.h>
```

Default model behavior for the allowed maximum base pair span.

See also

[vrna\\_md\\_t.max\\_bp\\_span](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

#### 15.6.3.17 VRNA\_MODEL\_DEFAULT\_WINDOW\_SIZE

```
#define VRNA_MODEL_DEFAULT_WINDOW_SIZE -1  
  
#include <ViennaRNA/model.h>
```

Default model behavior for the sliding window approach.

See also

[vrna\\_md\\_t.window\\_size](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

#### 15.6.3.18 VRNA\_MODEL\_DEFAULT\_LOG\_ML

```
#define VRNA_MODEL_DEFAULT_LOG_ML 0  
  
#include <ViennaRNA/model.h>
```

Default model behavior on how to evaluate the energy contribution of multi-branch loops.

See also

[vrna\\_md\\_t.logML](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

#### 15.6.3.19 VRNA\_MODEL\_DEFAULT\_ALI\_OLD\_EN

```
#define VRNA_MODEL_DEFAULT_ALI_OLD_EN 0  
  
#include <ViennaRNA/model.h>
```

Default model behavior for consensus structure energy evaluation.

See also

[vrna\\_md\\_t.oldAliEn](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

#### 15.6.3.20 VRNA\_MODEL\_DEFAULT\_ALI\_RIBO

```
#define VRNA_MODEL_DEFAULT_ALI_RIBO 0  
  
#include <ViennaRNA/model.h>
```

Default model behavior for consensus structure co-variance contribution assessment.

See also

[vrna\\_md\\_t.ribo](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

#### 15.6.3.21 VRNA\_MODEL\_DEFAULT\_ALI\_CV\_FACT

```
#define VRNA_MODEL_DEFAULT_ALI_CV_FACT 1.  
  
#include <ViennaRNA/model.h>
```

Default model behavior for weighting the co-variance score in consensus structure prediction.

See also

[vrna\\_md\\_t.cv\\_fact](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)



## 15.6.3.22 VRNA\_MODEL\_DEFAULT\_ALI\_NC\_FACT

```
#define VRNA_MODEL_DEFAULT_ALI_NC_FACT 1.
```

```
#include <ViennaRNA/model.h>
```

Default model behavior for weighting the nucleotide conservation? in consensus structure prediction.

See also

[vrna\\_md\\_t.nc\\_fact](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

## 15.6.4 Function Documentation

## 15.6.4.1 vrna\_md\_set\_default()

```
void vrna_md_set_default (
    vrna_md_t * md )
```

```
#include <ViennaRNA/model.h>
```

Apply default model details to a provided [vrna\\_md\\_t](#) data structure.

Use this function to initialize a [vrna\\_md\\_t](#) data structure with its default values

Parameters

<i>md</i>	A pointer to the data structure that is about to be initialized
-----------	---

## 15.6.4.2 vrna\_md\_update()

```
void vrna_md_update (
    vrna_md_t * md )
```

```
#include <ViennaRNA/model.h>
```

Update the model details data structure.

This function should be called after changing the [vrna\\_md\\_t.energy\\_set](#) attribute since it re-initializes base pairing related arrays within the [vrna\\_md\\_t](#) data structure. In particular, [vrna\\_md\\_t.pair](#), [vrna\\_md\\_t.alias](#), and [vrna\\_md\\_t.rtype](#) are set to the values that correspond to the specified [vrna\\_md\\_t.energy\\_set](#) option

See also

[vrna\\_md\\_t](#), [vrna\\_md\\_t.energy\\_set](#), [vrna\\_md\\_t.pair](#), [vrna\\_md\\_t.rtype](#), [vrna\\_md\\_t.alias](#), [vrna\\_md\\_set\\_default\(\)](#)

#### 15.6.4.3 vrna\_md\_copy()

```
vrna_md_t* vrna_md_copy (
    vrna_md_t * md_to,
    const vrna_md_t * md_from )
```

```
#include <ViennaRNA/model.h>
```

Copy/Clone a `vrna_md_t` model.

Use this function to clone a given model either inplace (target container `md_to` given) or create a copy by cloning the source model and returning it (`md_to == NULL`).

##### Parameters

<code>md_to</code>	The model to be overwritten (if non-NULL and <code>md_to != md_from</code> )
<code>md_from</code>	The model to copy (if non-NULL)

##### Returns

A pointer to the copy model (or NULL if `md_from == NULL`)

#### 15.6.4.4 vrna\_md\_option\_string()

```
char* vrna_md_option_string (
    vrna_md_t * md )
```

```
#include <ViennaRNA/model.h>
```

Get a corresponding commandline parameter string of the options in a `vrna_md_t`.

##### Note

This function is not threadsafe!

#### 15.6.4.5 vrna\_md\_defaults\_reset()

```
void vrna_md_defaults_reset (
    vrna_md_t * md_p )
```

```
#include <ViennaRNA/model.h>
```

Reset the global default model details to a specific set of parameters, or their initial values.

This function resets the global default model details to their initial values, i.e. as specified by the ViennaRNA Package release, upon passing NULL as argument. Alternatively it resets them according to a set of provided parameters.

**Note**

The global default parameters affect all function calls of RNAlib where model details are not explicitly provided. Hence, any change of them is not considered threadsafe

**Warning**

This function first resets the global default settings to factory defaults, and only then applies user provided settings (if any). User settings that do not meet specifications are skipped.

**See also**

[vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#)

**Parameters**

<i>md</i> <sub>↔</sub> <i>_p</i>	A set of model details to use as global default (if NULL is passed, factory defaults are restored)
-------------------------------------	--

**15.6.4.6 vrna\_md\_defaults\_temperature()**

```
void vrna_md_defaults_temperature (
    double T )
```

```
#include <ViennaRNA/model.h>
```

Set default temperature for energy evaluation of loops.

**See also**

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_TEMPERATURE](#)

**Parameters**

<i>T</i>	Temperature in centigrade
----------	---------------------------

**15.6.4.7 vrna\_md\_defaults\_temperature\_get()**

```
double vrna_md_defaults_temperature_get (
    void )
```

```
#include <ViennaRNA/model.h>
```

Get default temperature for energy evaluation of loops.

**See also**

[vrna\\_md\\_defaults\\_temperature\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_T](#)

**Returns**

The global default settings for temperature in centigrade

**15.6.4.8 vrna\_md\_defaults\_betaScale()**

```
void vrna_md_defaults_betaScale (
    double b )
```

```
#include <ViennaRNA/model.h>
```

Set default scaling factor of thermodynamic temperature in Boltzmann factors.

Boltzmann factors are then computed as  $\exp(-E/(b \cdot kT))$ .

**See also**

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_BETA\\_SCALE](#)

**Parameters**

<i>b</i>	The scaling factor, default is 1.0
----------	------------------------------------

**15.6.4.9 vrna\_md\_defaults\_betaScale\_get()**

```
double vrna_md_defaults_betaScale_get (
    void )
```

```
#include <ViennaRNA/model.h>
```

Get default scaling factor of thermodynamic temperature in Boltzmann factors.

**See also**

[vrna\\_md\\_defaults\\_betaScale\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_BET](#)

**Returns**

The global default thermodynamic temperature scaling factor

#### 15.6.4.10 `vrna_md_defaults_dangles()`

```
void vrna_md_defaults_dangles (
    int d )
```

```
#include <ViennaRNA/model.h>
```

Set default dangle model for structure prediction.

See also

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_DANGLES](#)

Parameters

<i>d</i>	The dangle model
----------	------------------

#### 15.6.4.11 `vrna_md_defaults_dangles_get()`

```
int vrna_md_defaults_dangles_get (
    void )
```

```
#include <ViennaRNA/model.h>
```

Get default dangle model for structure prediction.

See also

[vrna\\_md\\_defaults\\_dangles\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_DANGLES](#)

Returns

The global default settings for the dangle model

#### 15.6.4.12 `vrna_md_defaults_special_hp()`

```
void vrna_md_defaults_special_hp (
    int flag )
```

```
#include <ViennaRNA/model.h>
```

Set default behavior for lookup of tabulated free energies for special hairpin loops, such as Tri-, Tetra-, or Hexa-loops.

See also

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_SPECIAL\\_HP](#)

## Parameters

<i>flag</i>	On/Off switch (0 = OFF, else = ON)
-------------	------------------------------------

15.6.4.13 `vrna_md_defaults_special_hp_get()`

```
int vrna_md_defaults_special_hp_get (  
    void )
```

```
#include <ViennaRNA/model.h>
```

Get default behavior for lookup of tabulated free energies for special hairpin loops, such as Tri-, Tetra-, or Hexa-loops.

## See also

[vrna\\_md\\_defaults\\_special\\_hp\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_SP](#)

## Returns

The global default settings for the treatment of special hairpin loops

15.6.4.14 `vrna_md_defaults_noLP()`

```
void vrna_md_defaults_noLP (  
    int flag )
```

```
#include <ViennaRNA/model.h>
```

Set default behavior for prediction of canonical secondary structures.

## See also

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_NO\\_LP](#)

## Parameters

<i>flag</i>	On/Off switch (0 = OFF, else = ON)
-------------	------------------------------------

15.6.4.15 `vrna_md_defaults_noLP_get()`

```
int vrna_md_defaults_noLP_get (  
    void )
```

```
void )

#include <ViennaRNA/model.h>
```

Get default behavior for prediction of canonical secondary structures.

See also

[vrna\\_md\\_defaults\\_noLP\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_NO\\_LP](#)

Returns

The global default settings for predicting canonical secondary structures

#### 15.6.4.16 vrna\_md\_defaults\_noGU()

```
void vrna_md_defaults_noGU (
    int flag )

#include <ViennaRNA/model.h>
```

Set default behavior for treatment of G-U wobble pairs.

See also

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_NO\\_GU](#)

Parameters

<i>flag</i>	On/Off switch (0 = OFF, else = ON)
-------------	------------------------------------

#### 15.6.4.17 vrna\_md\_defaults\_noGU\_get()

```
int vrna_md_defaults_noGU_get (
    void )

#include <ViennaRNA/model.h>
```

Get default behavior for treatment of G-U wobble pairs.

See also

[vrna\\_md\\_defaults\\_noGU\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_NO\\_GU](#)

Returns

The global default settings for treatment of G-U wobble pairs

**15.6.4.18 vrna\_md\_defaults\_noGUclosure()**

```
void vrna_md_defaults_noGUclosure (
    int flag )
```

```
#include <ViennaRNA/model.h>
```

Set default behavior for G-U pairs as closing pair for loops.

See also

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_NO\\_GU\\_CLOSURE](#)

Parameters

<i>flag</i>	On/Off switch (0 = OFF, else = ON)
-------------	------------------------------------

**15.6.4.19 vrna\_md\_defaults\_noGUclosure\_get()**

```
int vrna_md_defaults_noGUclosure_get (
    void )
```

```
#include <ViennaRNA/model.h>
```

Get default behavior for G-U pairs as closing pair for loops.

See also

[vrna\\_md\\_defaults\\_noGUclosure\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_NO\\_GU\\_CLOSURE](#)

Returns

The global default settings for treatment of G-U pairs closing a loop

**15.6.4.20 vrna\_md\_defaults\_logML()**

```
void vrna_md_defaults_logML (
    int flag )
```

```
#include <ViennaRNA/model.h>
```

Set default behavior recomputing free energies of multi-branch loops using a logarithmic model.

See also

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_LOG\\_ML](#)



## Parameters

<i>flag</i>	On/Off switch (0 = OFF, else = ON)
-------------	------------------------------------

15.6.4.21 `vrna_md_defaults_logML_get()`

```
int vrna_md_defaults_logML_get (  
    void )
```

```
#include <ViennaRNA/model.h>
```

Get default behavior recomputing free energies of multi-branch loops using a logarithmic model.

## See also

[vrna\\_md\\_defaults\\_logML\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_LOG\\_M](#)

## Returns

The global default settings for logarithmic model in multi-branch loop free energy evaluation

15.6.4.22 `vrna_md_defaults_circ()`

```
void vrna_md_defaults_circ (  
    int flag )
```

```
#include <ViennaRNA/model.h>
```

Set default behavior whether input sequences are circularized.

## See also

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_CIRC](#)

## Parameters

<i>flag</i>	On/Off switch (0 = OFF, else = ON)
-------------	------------------------------------

15.6.4.23 `vrna_md_defaults_circ_get()`

```
int vrna_md_defaults_circ_get (  
    void )
```

```
#include <ViennaRNA/model.h>
```

Get default behavior whether input sequences are circularized.

See also

[vrna\\_md\\_defaults\\_circ\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_CIRC](#)

Returns

The global default settings for treating input sequences as circular

#### 15.6.4.24 vrna\_md\_defaults\_gquad()

```
void vrna_md_defaults_gquad (
    int flag )
```

```
#include <ViennaRNA/model.h>
```

Set default behavior for treatment of G-Quadruplexes.

See also

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_GQUAD](#)

Parameters

<i>flag</i>	On/Off switch (0 = OFF, else = ON)
-------------	------------------------------------

#### 15.6.4.25 vrna\_md\_defaults\_gquad\_get()

```
int vrna_md_defaults_gquad_get (
    void )
```

```
#include <ViennaRNA/model.h>
```

Get default behavior for treatment of G-Quadruplexes.

See also

[vrna\\_md\\_defaults\\_gquad\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_GQUAD](#)

Returns

The global default settings for treatment of G-Quadruplexes

15.6.4.26 `vrna_md_defaults_uniq_ML()`

```
void vrna_md_defaults_uniq_ML (
    int flag )
```

```
#include <ViennaRNA/model.h>
```

Set default behavior for creating additional matrix for unique multi-branch loop prediction.

**Note**

Activating this option usually results in higher memory consumption!

**See also**

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_UNIQ\\_ML](#)

**Parameters**

<i>flag</i>	On/Off switch (0 = OFF, else = ON)
-------------	------------------------------------

15.6.4.27 `vrna_md_defaults_uniq_ML_get()`

```
int vrna_md_defaults_uniq_ML_get (
    void )
```

```
#include <ViennaRNA/model.h>
```

Get default behavior for creating additional matrix for unique multi-branch loop prediction.

**See also**

[vrna\\_md\\_defaults\\_uniq\\_ML\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_UNIQ\\_ML](#)

**Returns**

The global default settings for creating additional matrices for unique multi-branch loop prediction

15.6.4.28 `vrna_md_defaults_energy_set()`

```
void vrna_md_defaults_energy_set (
    int e )
```

```
#include <ViennaRNA/model.h>
```

Set default energy set.

**See also**

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_ENERGY\\_SET](#)

## Parameters

<i>e</i>	Energy set (0, 1, 2, 3)
----------	-------------------------

15.6.4.29 `vrna_md_defaults_energy_set_get()`

```
int vrna_md_defaults_energy_set_get (  
    void )
```

```
#include <ViennaRNA/model.h>
```

Get default energy set.

## See also

[vrna\\_md\\_defaults\\_energy\\_set\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_EN](#)

## Returns

The global default settings for the energy set

15.6.4.30 `vrna_md_defaults_backtrack()`

```
void vrna_md_defaults_backtrack (  
    int flag )
```

```
#include <ViennaRNA/model.h>
```

Set default behavior for whether to backtrack secondary structures.

## See also

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_BACKTRACK](#)

## Parameters

<i>flag</i>	On/Off switch (0 = OFF, else = ON)
-------------	------------------------------------

15.6.4.31 `vrna_md_defaults_backtrack_get()`

```
int vrna_md_defaults_backtrack_get (  
    void )
```

```
#include <ViennaRNA/model.h>
```

Get default behavior for whether to backtrack secondary structures.

See also

[vrna\\_md\\_defaults\\_backtrack\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_BACKTRACK\\_TYPE](#)

Returns

The global default settings for backtracking structures

#### 15.6.4.32 vrna\_md\_defaults\_backtrack\_type()

```
void vrna_md_defaults_backtrack_type (
    char t )
```

```
#include <ViennaRNA/model.h>
```

Set default backtrack type, i.e. which DP matrix is used.

See also

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_BACKTRACK\\_TYPE](#)

Parameters

<i>t</i>	The type ('F', 'C', or 'M')
----------	-----------------------------

#### 15.6.4.33 vrna\_md\_defaults\_backtrack\_type\_get()

```
char vrna_md_defaults_backtrack_type_get (
    void )
```

```
#include <ViennaRNA/model.h>
```

Get default backtrack type, i.e. which DP matrix is used.

See also

[vrna\\_md\\_defaults\\_backtrack\\_type\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_BACKTRACK\\_TYPE](#)

Returns

The global default settings that specify which DP matrix is used for backtracking

**15.6.4.34 vrna\_md\_defaults\_compute\_bpp()**

```
void vrna_md_defaults_compute_bpp (
    int flag )
```

```
#include <ViennaRNA/model.h>
```

Set the default behavior for whether to compute base pair probabilities after partition function computation.

See also

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_COMPUTE\\_BPP](#)

Parameters

<i>flag</i>	On/Off switch (0 = OFF, else = ON)
-------------	------------------------------------

**15.6.4.35 vrna\_md\_defaults\_compute\_bpp\_get()**

```
int vrna_md_defaults_compute_bpp_get (
    void )
```

```
#include <ViennaRNA/model.h>
```

Get the default behavior for whether to compute base pair probabilities after partition function computation.

See also

[vrna\\_md\\_defaults\\_compute\\_bpp\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_COMPUTE\\_BPP](#)

Returns

The global default settings that specify whether base pair probabilities are computed together with partition function

**15.6.4.36 vrna\_md\_defaults\_max\_bp\_span()**

```
void vrna_md_defaults_max_bp_span (
    int span )
```

```
#include <ViennaRNA/model.h>
```

Set default maximal base pair span.

See also

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_MAX\\_BP\\_SPAN](#)

## Parameters

<i>span</i>	Maximal base pair span
-------------	------------------------

15.6.4.37 `vrna_md_defaults_max_bp_span_get()`

```
int vrna_md_defaults_max_bp_span_get (
    void )
```

```
#include <ViennaRNA/model.h>
```

Get default maximal base pair span.

## See also

[vrna\\_md\\_defaults\\_max\\_bp\\_span\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_](#)

## Returns

The global default settings for maximum base pair span

15.6.4.38 `vrna_md_defaults_min_loop_size()`

```
void vrna_md_defaults_min_loop_size (
    int size )
```

```
#include <ViennaRNA/model.h>
```

Set default minimal loop size.

## See also

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [TURN](#)

## Parameters

<i>size</i>	Minimal size, i.e. number of unpaired nucleotides for a hairpin loop
-------------	--

15.6.4.39 `vrna_md_defaults_min_loop_size_get()`

```
int vrna_md_defaults_min_loop_size_get (
    void )
```

```
#include <ViennaRNA/model.h>
```

Get default minimal loop size.

See also

[vrna\\_md\\_defaults\\_min\\_loop\\_size\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [TURN](#)

Returns

The global default settings for minimal size of hairpin loops

#### 15.6.4.40 vrna\_md\_defaults\_window\_size()

```
void vrna_md_defaults_window_size (
    int size )
```

```
#include <ViennaRNA/model.h>
```

Set default window size for sliding window structure prediction approaches.

See also

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_WINDOW\\_SIZE](#)

Parameters

<i>size</i>	The size of the sliding window
-------------	--------------------------------

#### 15.6.4.41 vrna\_md\_defaults\_window\_size\_get()

```
int vrna_md_defaults_window_size_get (
    void )
```

```
#include <ViennaRNA/model.h>
```

Get default window size for sliding window structure prediction approaches.

See also

[vrna\\_md\\_defaults\\_window\\_size\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_V](#)

Returns

The global default settings for the size of the sliding window



#### 15.6.4.42 vrna\_md\_defaults\_oldAliEn()

```
void vrna_md_defaults_oldAliEn (
    int flag )
```

```
#include <ViennaRNA/model.h>
```

Set default behavior for whether to use old energy model for comparative structure prediction.

##### Note

This option is outdated. Activating the old energy model usually results in worse consensus structure predictions.

##### See also

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_ALI\\_OLD\\_EN](#)

##### Parameters

<i>flag</i>	On/Off switch (0 = OFF, else = ON)
-------------	------------------------------------

#### 15.6.4.43 vrna\_md\_defaults\_oldAliEn\_get()

```
int vrna_md_defaults_oldAliEn_get (
    void )
```

```
#include <ViennaRNA/model.h>
```

Get default behavior for whether to use old energy model for comparative structure prediction.

##### See also

[vrna\\_md\\_defaults\\_oldAliEn\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_ALI\\_OLD\\_EN](#)

##### Returns

The global default settings for using old energy model for comparative structure prediction

#### 15.6.4.44 vrna\_md\_defaults\_ribo()

```
void vrna_md_defaults_ribo (
    int flag )
```

```
#include <ViennaRNA/model.h>
```

Set default behavior for whether to use Ribosum Scoring in comparative structure prediction.

##### See also

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_ALI\\_RIBO](#)

## Parameters

<i>flag</i>	On/Off switch (0 = OFF, else = ON)
-------------	------------------------------------

15.6.4.45 `vrna_md_defaults_ribo_get()`

```
int vrna_md_defaults_ribo_get (  
    void )
```

```
#include <ViennaRNA/model.h>
```

Get default behavior for whether to use Ribosum Scoring in comparative structure prediction.

## See also

[vrna\\_md\\_defaults\\_ribo\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_ALI\\_RIBO](#)

## Returns

The global default settings for using Ribosum scoring in comparative structure prediction

15.6.4.46 `vrna_md_defaults_cv_fact()`

```
void vrna_md_defaults_cv_fact (  
    double factor )
```

```
#include <ViennaRNA/model.h>
```

Set the default co-variance scaling factor used in comparative structure prediction.

## See also

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_ALI\\_CV\\_FACT](#)

## Parameters

<i>factor</i>	The co-variance factor
---------------	------------------------

15.6.4.47 `vrna_md_defaults_cv_fact_get()`

```
double vrna_md_defaults_cv_fact_get (  
    void )
```

```
#include <ViennaRNA/model.h>
```

Get the default co-variance scaling factor used in comparative structure prediction.

See also

[vrna\\_md\\_defaults\\_cv\\_fact\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_ALI\\_CV\\_FACT](#)

Returns

The global default settings for the co-variance factor

#### 15.6.4.48 vrna\_md\_defaults\_nc\_fact()

```
void vrna_md_defaults_nc_fact (
    double factor )
```

```
#include <ViennaRNA/model.h>
```

See also

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_ALI\\_NC\\_FACT](#)

Parameters

<i>factor</i>	
---------------	--

#### 15.6.4.49 vrna\_md\_defaults\_nc\_fact\_get()

```
double vrna_md_defaults_nc_fact_get (
    void )
```

```
#include <ViennaRNA/model.h>
```

See also

[vrna\\_md\\_defaults\\_nc\\_fact\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_ALI\\_NC\\_FACT](#)

Returns

15.6.4.50 `vrna_md_defaults_sfact()`

```
void vrna_md_defaults_sfact (
    double factor )
```

```
#include <ViennaRNA/model.h>
```

Set the default scaling factor used to avoid under-/overflows in partition function computation.

See also

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#)

## Parameters

<i>factor</i>	The scaling factor (default: 1.07)
---------------	------------------------------------

15.6.4.51 `vrna_md_defaults_sfact_get()`

```
double vrna_md_defaults_sfact_get (
    void )
```

```
#include <ViennaRNA/model.h>
```

Get the default scaling factor used to avoid under-/overflows in partition function computation.

See also

[vrna\\_md\\_defaults\\_sfact\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#)

## Returns

The global default settings of the scaling factor

15.6.4.52 `set_model_details()`

```
void set_model_details (
    vrna\_md\_t * md )
```

```
#include <ViennaRNA/model.h>
```

Set default model details.

Use this function if you wish to initialize a [vrna\\_md\\_t](#) data structure with its default values, i.e. the global model settings as provided by the deprecated global variables.

**Deprecated** This function will vanish as soon as backward compatibility of RNAlib is dropped (expected in version 3). Use [vrna\\_md\\_set\\_default\(\)](#) instead!

## Parameters

<i>md</i>	A pointer to the data structure that is about to be initialized
-----------	---

### 15.6.5 Variable Documentation

#### 15.6.5.1 temperature

```
double temperature
```

```
#include <ViennaRNA/model.h>
```

Rescale energy parameters to a temperature in degC.

Default is 37C. You have to call the `update___params()` functions after changing this parameter.

**Deprecated** Use `vrna_md_defaults_temperature()`, and `vrna_md_defaults_temperature_get()` to change, and read the global default temperature settings

See also

[vrna\\_md\\_defaults\\_temperature\(\)](#), [vrna\\_md\\_defaults\\_temperature\\_get\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#)

#### 15.6.5.2 pf\_scale

```
double pf_scale
```

```
#include <ViennaRNA/model.h>
```

A scaling factor used by [pf\\_fold\(\)](#) to avoid overflows.

Should be set to approximately  $\exp((-F/kT)/length)$ , where  $F$  is an estimate for the ensemble free energy, for example the minimum free energy. You must call [update\\_pf\\_params\(\)](#) after changing this parameter.

If `pf_scale` is -1 (the default) , an estimate will be provided automatically when computing partition functions, e.g. [pf\\_fold\(\)](#) The automatic estimate is usually insufficient for sequences more than a few hundred bases long.

### 15.6.5.3 dangles

```
int dangles
```

```
#include <ViennaRNA/model.h>
```

Switch the energy model for dangling end contributions (0, 1, 2, 3)

If set to 0 no stabilizing energies are assigned to bases adjacent to helices in free ends and multiloops (so called dangling ends). Normally (dangles = 1) dangling end energies are assigned only to unpaired bases and a base cannot participate simultaneously in two dangling ends. In the partition function algorithm `pf_fold()` these checks are neglected. If `dangles` is set to 2, all folding routines will follow this convention. This treatment of dangling ends gives more favorable energies to helices directly adjacent to one another, which can be beneficial since such helices often do engage in stabilizing interactions through co-axial stacking.

If dangles = 3 co-axial stacking is explicitly included for adjacent helices in multiloops. The option affects only mfe folding and energy evaluation (`fold()` and `energy_of_structure()`), as well as suboptimal folding (`subopt()`) via re-evaluation of energies. Co-axial stacking with one intervening mismatch is not considered so far.

Default is 2 in most algorithms, partition function algorithms can only handle 0 and 2

### 15.6.5.4 tetra\_loop

```
int tetra_loop
```

```
#include <ViennaRNA/model.h>
```

Include special stabilizing energies for some tri-, tetra- and hexa-loops;.

default is 1.

### 15.6.5.5 noLonelyPairs

```
int noLonelyPairs
```

```
#include <ViennaRNA/model.h>
```

Global switch to avoid/allow helices of length 1.

Disallow all pairs which can only occur as lonely pairs (i.e. as helix of length 1). This avoids lonely base pairs in the predicted structures in most cases.

### 15.6.5.6 energy\_set

```
int energy_set
```

```
#include <ViennaRNA/model.h>
```

0 = BP; 1=any with GC; 2=any with AU-parameter

If set to 1 or 2: fold sequences from an artificial alphabet ABCD..., where A pairs B, C pairs D, etc. using either GC (1) or AU parameters (2); default is 0, you probably don't want to change it.

#### 15.6.5.7 do\_backtrack

```
int do_backtrack
```

```
#include <ViennaRNA/model.h>
```

do backtracking, i.e. compute secondary structures or base pair probabilities

If 0, do not calculate pair probabilities in [pf\\_fold\(\)](#); this is about twice as fast. Default is 1.

#### 15.6.5.8 backtrack\_type

```
char backtrack_type
```

```
#include <ViennaRNA/model.h>
```

A backtrack array marker for [inverse\\_fold\(\)](#)

If set to 'C': force (1,N) to be paired, 'M' fold as if the sequence were inside a multiloop. Otherwise ('F') the usual mfe structure is computed.

#### 15.6.5.9 nonstandards

```
char* nonstandards
```

```
#include <ViennaRNA/model.h>
```

contains allowed non standard base pairs

Lists additional base pairs that will be allowed to form in addition to GC, CG, AU, UA, GU and UG. Nonstandard base pairs are given a stacking energy of 0.

#### 15.6.5.10 max\_bp\_span

```
int max_bp_span
```

```
#include <ViennaRNA/model.h>
```

Maximum allowed base pair span.

A value of -1 indicates no restriction for distant base pairs.

## 15.7 Energy Parameters

All relevant functions to retrieve and copy pre-calculated energy parameter sets as well as reading/writing the energy parameter set from/to file(s).

### 15.7.1 Detailed Description

All relevant functions to retrieve and copy pre-calculated energy parameter sets as well as reading/writing the energy parameter set from/to file(s).

This module covers all relevant functions for pre-calculation of the energy parameters necessary for the folding routines provided by RNAlib. Furthermore, the energy parameter set in the RNAlib can be easily exchanged by a user-defined one. It is also possible to write the current energy parameter set into a text file. Collaboration diagram for Energy Parameters:

### Modules

- [Reading/Writing Energy Parameter Sets from/to File](#)  
*Read and Write energy parameter sets from and to text files.*

### Files

- file [basic.h](#)  
*Functions to deal with sets of energy parameters.*
- file [constants.h](#)  
*Energy parameter constants.*
- file [convert.h](#)  
*Functions and definitions for energy parameter file format conversion.*
- file [io.h](#)  
*Read and write energy parameter files.*

### Data Structures

- struct [vrna\\_param\\_s](#)  
*The datastructure that contains temperature scaled energy parameters. [More...](#)*
- struct [vrna\\_exp\\_param\\_s](#)  
*The data structure that contains temperature scaled Boltzmann weights of the energy parameters. [More...](#)*

### Typedefs

- typedef struct [vrna\\_param\\_s](#) [vrna\\_param\\_t](#)  
*Typename for the free energy parameter data structure [vrna\\_params](#).*
- typedef struct [vrna\\_exp\\_param\\_s](#) [vrna\\_exp\\_param\\_t](#)  
*Typename for the Boltzmann factor data structure [vrna\\_exp\\_params](#).*
- typedef struct [vrna\\_param\\_s](#) paramT  
*Old typename of [vrna\\_param\\_s](#).*
- typedef struct [vrna\\_exp\\_param\\_s](#) pf\_paramT  
*Old typename of [vrna\\_exp\\_param\\_s](#).*



## Functions

- [vrna\\_param\\_t \\* vrna\\_params](#) ([vrna\\_md\\_t](#) \*md)  
*Get a data structure containing prescaled free energy parameters.*
- [vrna\\_param\\_t \\* vrna\\_params\\_copy](#) ([vrna\\_param\\_t](#) \*par)  
*Get a copy of the provided free energy parameters.*
- [vrna\\_exp\\_param\\_t \\* vrna\\_exp\\_params](#) ([vrna\\_md\\_t](#) \*md)  
*Get a data structure containing prescaled free energy parameters already transformed to Boltzmann factors.*
- [vrna\\_exp\\_param\\_t \\* vrna\\_exp\\_params\\_comparative](#) (unsigned int n\_seq, [vrna\\_md\\_t](#) \*md)  
*Get a data structure containing prescaled free energy parameters already transformed to Boltzmann factors (alifold version)*
- [vrna\\_exp\\_param\\_t \\* vrna\\_exp\\_params\\_copy](#) ([vrna\\_exp\\_param\\_t](#) \*par)  
*Get a copy of the provided free energy parameters (provided as Boltzmann factors)*
- void [vrna\\_params\\_subst](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_param\\_t](#) \*par)  
*Update/Reset energy parameters data structure within a [vrna\\_fold\\_compound\\_t](#).*
- void [vrna\\_exp\\_params\\_subst](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_exp\\_param\\_t](#) \*params)  
*Update the energy parameters for subsequent partition function computations.*
- void [vrna\\_exp\\_params\\_rescale](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, double \*mfe)  
*Rescale Boltzmann factors for partition function computations.*
- void [vrna\\_params\\_reset](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_md\\_t](#) \*md\_p)  
*Reset free energy parameters within a [vrna\\_fold\\_compound\\_t](#) according to provided, or default model details.*
- void [vrna\\_exp\\_params\\_reset](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_md\\_t](#) \*md\_p)  
*Reset Boltzmann factors for partition function computations within a [vrna\\_fold\\_compound\\_t](#) according to provided, or default model details.*
- [vrna\\_exp\\_param\\_t \\* get\\_scaled\\_pf\\_parameters](#) (void)
- [vrna\\_exp\\_param\\_t \\* get\\_boltzmann\\_factors](#) (double temperature, double betaScale, [vrna\\_md\\_t](#) md, double pf\_scale)  
*Get precomputed Boltzmann factors of the loop type dependent energy contributions with independent thermodynamic temperature.*
- [vrna\\_exp\\_param\\_t \\* get\\_boltzmann\\_factor\\_copy](#) ([vrna\\_exp\\_param\\_t](#) \*parameters)  
*Get a copy of already precomputed Boltzmann factors.*
- [vrna\\_exp\\_param\\_t \\* get\\_scaled\\_alipf\\_parameters](#) (unsigned int n\_seq)  
*Get precomputed Boltzmann factors of the loop type dependent energy contributions (alifold variant)*
- [vrna\\_exp\\_param\\_t \\* get\\_boltzmann\\_factors\\_ali](#) (unsigned int n\_seq, double temperature, double betaScale, [vrna\\_md\\_t](#) md, double pf\_scale)  
*Get precomputed Boltzmann factors of the loop type dependent energy contributions (alifold variant) with independent thermodynamic temperature.*
- [vrna\\_param\\_t \\* scale\\_parameters](#) (void)  
*Get precomputed energy contributions for all the known loop types.*
- [vrna\\_param\\_t \\* get\\_scaled\\_parameters](#) (double temperature, [vrna\\_md\\_t](#) md)  
*Get precomputed energy contributions for all the known loop types.*

## 15.7.2 Data Structure Documentation

### 15.7.2.1 struct vrna\_param\_s

The datastructure that contains temperature scaled energy parameters.

Collaboration diagram for [vrna\\_param\\_s](#):

### Data Fields

- double [temperature](#)  
*Temperature used for loop contribution scaling.*
- [vrna\\_md\\_t model\\_details](#)  
*Model details to be used in the recursions.*
- char [param\\_file](#) [256]  
*The filename the parameters were derived from, or empty string if they represent the default.*

#### 15.7.2.2 struct vrna\_exp\_param\_s

The data structure that contains temperature scaled Boltzmann weights of the energy parameters.

Collaboration diagram for vrna\_exp\_param\_s:

### Data Fields

- int [id](#)  
*An identifier for the data structure.*
- double [pf\\_scale](#)  
*Scaling factor to avoid over-/underflows.*
- double [temperature](#)  
*Temperature used for loop contribution scaling.*
- double [alpha](#)  
*Scaling factor for the thermodynamic temperature.*
- [vrna\\_md\\_t model\\_details](#)  
*Model details to be used in the recursions.*
- char [param\\_file](#) [256]  
*The filename the parameters were derived from, or empty string if they represent the default.*

#### 15.7.2.2.1 Field Documentation

##### 15.7.2.2.1.1 id

```
int vrna_exp_param_s::id
```

An identifier for the data structure.

**Deprecated** This attribute will be removed in version 3

#### 15.7.2.2.1.2 alpha

```
double vrna_exp_param_s::alpha
```

Scaling factor for the thermodynamic temperature.

This allows for temperature scaling in Boltzmann factors independently from the energy contributions. The resulting Boltzmann factors are then computed by  $e^{-E/(\alpha \cdot K \cdot T)}$

### 15.7.3 Typedef Documentation

#### 15.7.3.1 paramT

```
typedef struct vrna_param_s paramT  
  
#include <ViennaRNA/params/basic.h>  
  
Old typename of vrna_param_s.
```

**Deprecated** Use [vrna\\_param\\_t](#) instead!

#### 15.7.3.2 pf\_paramT

```
typedef struct vrna_exp_param_s pf_paramT  
  
#include <ViennaRNA/params/basic.h>  
  
Old typename of vrna_exp_param_s.
```

**Deprecated** Use [vrna\\_exp\\_param\\_t](#) instead!

### 15.7.4 Function Documentation

#### 15.7.4.1 vrna\_params()

```
vrna_param_t* vrna_params (  
    vrna_md_t * md )  
  
#include <ViennaRNA/params/basic.h>
```

Get a data structure containing prescaled free energy parameters.

If a NULL pointer is passed for the model details parameter, the default model parameters are stored within the requested [vrna\\_param\\_t](#) structure.

See also

[vrna\\_md\\_t](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_exp\\_params\(\)](#)

## Parameters

<i>md</i>	A pointer to the model details to store inside the structure (Maybe NULL)
-----------	---

## Returns

A pointer to the memory location where the requested parameters are stored

15.7.4.2 `vrna_params_copy()`

```
vrna_param_t* vrna_params_copy (
    vrna_param_t * par )
```

```
#include <ViennaRNA/params/basic.h>
```

Get a copy of the provided free energy parameters.

If NULL is passed as parameter, a default set of energy parameters is created and returned.

## See also

[vrna\\_params\(\)](#), [vrna\\_param\\_t](#)

## Parameters

<i>par</i>	The free energy parameters that are to be copied (Maybe NULL)
------------	---

## Returns

A copy or a default set of the (provided) parameters

15.7.4.3 `vrna_exp_params()`

```
vrna_exp_param_t* vrna_exp_params (
    vrna_md_t * md )
```

```
#include <ViennaRNA/params/basic.h>
```

Get a data structure containing prescaled free energy parameters already transformed to Boltzmann factors.

This function returns a data structure that contains all necessary precomputed energy contributions for each type of loop.

In contrast to [vrna\\_params\(\)](#), the free energies within this data structure are stored as their Boltzmann factors, i.e.

$\exp(-E/kT)$

where  $E$  is the free energy.

If a NULL pointer is passed for the model details parameter, the default model parameters are stored within the requested [vrna\\_exp\\_param\\_t](#) structure.

See also

[vrna\\_md\\_t](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_params\(\)](#), [vrna\\_rescale\\_pf\\_params\(\)](#)

Parameters

<i>md</i>	A pointer to the model details to store inside the structure (Maybe NULL)
-----------	---

Returns

A pointer to the memory location where the requested parameters are stored

#### 15.7.4.4 `vrna_exp_params_comparative()`

```
vrna_exp_param_t* vrna_exp_params_comparative (
    unsigned int n_seq,
    vrna_md_t * md )
```

```
#include <ViennaRNA/params/basic.h>
```

Get a data structure containing prescaled free energy parameters already transformed to Boltzmann factors (alifold version)

If a NULL pointer is passed for the model details parameter, the default model parameters are stored within the requested [vrna\\_exp\\_param\\_t](#) structure.

See also

[vrna\\_md\\_t](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_exp\\_params\(\)](#), [vrna\\_params\(\)](#)

Parameters

<i>n_seq</i>	The number of sequences in the alignment
<i>md</i>	A pointer to the model details to store inside the structure (Maybe NULL)

Returns

A pointer to the memory location where the requested parameters are stored

#### 15.7.4.5 `vrna_exp_params_copy()`

```
vrna_exp_param_t* vrna_exp_params_copy (
    vrna_exp_param_t * par )
```

```
#include <ViennaRNA/params/basic.h>
```

Get a copy of the provided free energy parameters (provided as Boltzmann factors)

If NULL is passed as parameter, a default set of energy parameters is created and returned.

See also

[vrna\\_exp\\_params\(\)](#), [vrna\\_exp\\_param\\_t](#)

Parameters

<i>par</i>	The free energy parameters that are to be copied (Maybe NULL)
------------	---

Returns

A copy or a default set of the (provided) parameters

#### 15.7.4.6 vrna\_params\_subst()

```
void vrna_params_subst (
    vrna_fold_compound_t * vc,
    vrna_param_t * par )
```

```
#include <ViennaRNA/params/basic.h>
```

Update/Reset energy parameters data structure within a [vrna\\_fold\\_compound\\_t](#).

Passing NULL as second argument leads to a reset of the energy parameters within vc to their default values. Otherwise, the energy parameters provided will be copied over into vc.

See also

[vrna\\_params\\_reset\(\)](#), [vrna\\_param\\_t](#), [vrna\\_md\\_t](#), [vrna\\_params\(\)](#)

Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> that is about to receive updated energy parameters
<i>par</i>	The energy parameters used to substitute those within vc (Maybe NULL)

**SWIG Wrapper Notes** This function is attached to [vrna\\_fc\\_s](#) objects as **params\_subst()** method.

#### 15.7.4.7 vrna\_exp\_params\_subst()

```
void vrna_exp_params_subst (
    vrna_fold_compound_t * vc,
    vrna_exp_param_t * params )
```

```
#include <ViennaRNA/params/basic.h>
```

Update the energy parameters for subsequent partition function computations.

This function can be used to properly assign new energy parameters for partition function computations to a [vrna\\_fold\\_compound\\_t](#). For this purpose, the data of the provided pointer `params` will be copied into `vc` and a recomputation of the partition function scaling factor is issued, if the `pf_scale` attribute of `params` is less than 1.0.

Passing NULL as second argument leads to a reset of the energy parameters within `vc` to their default values

See also

[vrna\\_exp\\_params\\_reset\(\)](#), [vrna\\_exp\\_params\\_rescale\(\)](#), [vrna\\_exp\\_param\\_t](#), [vrna\\_md\\_t](#), [vrna\\_exp\\_params\(\)](#)

#### Parameters

<code>vc</code>	The fold compound data structure
<code>params</code>	A pointer to the new energy parameters

#### 15.7.4.8 vrna\_exp\_params\_rescale()

```
void vrna_exp_params_rescale (
    vrna_fold_compound_t * vc,
    double * mfe )
#include <ViennaRNA/params/basic.h>
```

Rescale Boltzmann factors for partition function computations.

This function may be used to (automatically) rescale the Boltzmann factors used in partition function computations. Since partition functions over subsequences can easily become extremely large, the RNAlib internally rescales them to avoid numerical over- and/or underflow. Therefore, a proper scaling factor  $s$  needs to be chosen that in turn is then used to normalize the corresponding partition functions  $\hat{q}[i, j] = q[i, j]/s^{(j-i+1)}$ .

This function provides two ways to automatically adjust the scaling factor.

1. Automatic guess
2. Automatic adjustment according to MFE

Passing NULL as second parameter activates the *automatic guess mode*. Here, the scaling factor is recomputed according to a mean free energy of  $184.3 \cdot \text{length}$  cal for random sequences.

#### Note

This recomputation only takes place if the `pf_scale` attribute of the `exp_params` data structure contained in `vc` has a value below 1.0.

On the other hand, if the MFE for a sequence is known, it can be used to recompute a more robust scaling factor, since it represents the lowest free energy of the entire ensemble of structures, i.e. the highest Boltzmann factor. To activate this second mode of *automatic adjustment according to MFE*, a pointer to the MFE value needs to be passed as second argument. This value is then taken to compute the scaling factor as  $s = \exp((s_{fact} * MFE)/kT/length)$ , where `sfact` is an additional scaling weight located in the `vrna_md_t` data structure of `exp_params` in `vc`.

The computed scaling factor  $s$  will be stored as `pf_scale` attribute of the `exp_params` data structure in `vc`.

See also

[vrna\\_exp\\_params\\_subst\(\)](#), [vrna\\_md\\_t](#), [vrna\\_exp\\_param\\_t](#), [vrna\\_fold\\_compound\\_t](#)

## Parameters

<i>vc</i>	The fold compound data structure
<i>mfe</i>	A pointer to the MFE (in kcal/mol) or NULL

**SWIG Wrapper Notes** This function is attached to [vrna\\_fc\\_s](#) objects as overloaded **exp\_params\_rescale()** method.

When no parameter is passed to this method, the resulting action is the same as passing *NULL* as second parameter to [vrna\\_exp\\_params\\_rescale\(\)](#), i.e. default scaling of the partition function. Passing an energy in kcal/mol, e.g. as retrieved by a previous call to the *mfe()* method, instructs all subsequent calls to scale the partition function accordingly.

## 15.7.4.9 vrna\_params\_reset()

```
void vrna_params_reset (
    vrna_fold_compound_t * vc,
    vrna_md_t * md_p )

#include <ViennaRNA/params/basic.h>
```

Reset free energy parameters within a [vrna\\_fold\\_compound\\_t](#) according to provided, or default model details.

This function allows one to rescale free energy parameters for subsequent structure prediction or evaluation according to a set of model details, e.g. temperature values. To do so, the caller provides either a pointer to a set of model details to be used for rescaling, or NULL if global default setting should be used.

## See also

[vrna\\_exp\\_params\\_reset\(\)](#), [vrna\\_params\\_subs\(\)](#)

## Parameters

<i>vc</i>	The fold compound data structure
<i>md_p</i>	A pointer to the new model details (or NULL for reset to defaults)

**SWIG Wrapper Notes** This function is attached to [vrna\\_fc\\_s](#) objects as overloaded **params\_reset()** method.

When no parameter is passed to this method, the resulting action is the same as passing *NULL* as second parameter to [vrna\\_params\\_reset\(\)](#), i.e. global default model settings are used. Passing an object of type [vrna\\_md\\_s](#) resets the fold compound according to the specifications stored within the [vrna\\_md\\_s](#) object.



15.7.4.10 `vrna_exp_params_reset()`

```
vrna_exp_params_reset (
    vrna_fold_compound_t * vc,
    vrna_md_t * md_p )

#include <ViennaRNA/params/basic.h>
```

Reset Boltzmann factors for partition function computations within a `vrna_fold_compound_t` according to provided, or default model details.

This function allows one to rescale Boltzmann factors for subsequent partition function computations according to a set of model details, e.g. temperature values. To do so, the caller provides either a pointer to a set of model details to be used for rescaling, or NULL if global default setting should be used.

See also

[vrna\\_params\\_reset\(\)](#), [vrna\\_exp\\_params\\_subst\(\)](#), [vrna\\_exp\\_params\\_rescale\(\)](#)

## Parameters

<code>vc</code>	The fold compound data structure
<code>md_p</code>	A pointer to the new model details (or NULL for reset to defaults)

**SWIG Wrapper Notes** This function is attached to `vrna_fc_s` objects as overloaded `exp_params_reset()` method.

When no parameter is passed to this method, the resulting action is the same as passing `NULL` as second parameter to `vrna_exp_params_reset()`, i.e. global default model settings are used. Passing an object of type `vrna_md_s` resets the fold compound according to the specifications stored within the `vrna_md_s` object.

15.7.4.11 `get_scaled_pf_parameters()`

```
vrna_exp_param_t* get_scaled_pf_parameters (
    void )

#include <ViennaRNA/params/basic.h>
```

get a data structure of type `vrna_exp_param_t` which contains the Boltzmann weights of several energy parameters scaled according to the current temperature

**Deprecated** Use `vrna_exp_params()` instead!

## Returns

The data structure containing Boltzmann weights for use in partition function calculations

15.7.4.12 `get_boltzmann_factors()`

```
vrna_exp_param_t* get_boltzmann_factors (
    double temperature,
    double betaScale,
    vrna_md_t md,
    double pf_scale )
```

```
#include <ViennaRNA/params/basic.h>
```

Get precomputed Boltzmann factors of the loop type dependent energy contributions with independent thermodynamic temperature.

This function returns a data structure that contains all necessary precalculated Boltzmann factors for each loop type contribution.

In contrast to `get_scaled_pf_parameters()`, this function enables setting of independent temperatures for both, the individual energy contributions as well as the thermodynamic temperature used in  $\exp(-\Delta G/kT)$

**Deprecated** Use `vrna_exp_params()` instead!

See also

[get\\_scaled\\_pf\\_parameters\(\)](#), [get\\_boltzmann\\_factor\\_copy\(\)](#)

## Parameters

<i>temperature</i>	The temperature in degrees Celcius used for (re-)scaling the energy contributions
<i>betaScale</i>	A scaling value that is used as a multiplication factor for the absolute temperature of the system
<i>md</i>	The model details to be used
<i>pf_scale</i>	The scaling factor for the Boltzmann factors

## Returns

A set of precomputed Boltzmann factors

15.7.4.13 `get_boltzmann_factor_copy()`

```
vrna_exp_param_t* get_boltzmann_factor_copy (
    vrna_exp_param_t * parameters )
```

```
#include <ViennaRNA/params/basic.h>
```

Get a copy of already precomputed Boltzmann factors.

**Deprecated** Use `vrna_exp_params_copy()` instead!

See also

[get\\_boltzmann\\_factors\(\)](#), [get\\_scaled\\_pf\\_parameters\(\)](#)

## Parameters

<i>parameters</i>	The input data structure that shall be copied
-------------------	---

## Returns

A copy of the provided Boltzmann factor data set

15.7.4.14 `get_scaled_alipf_parameters()`

```
vrna_exp_param_t* get_scaled_alipf_parameters (
    unsigned int n_seq )
```

```
#include <ViennaRNA/params/basic.h>
```

Get precomputed Boltzmann factors of the loop type dependent energy contributions (alifold variant)

**Deprecated** Use `vrna_exp_params_comparative()` instead!

15.7.4.15 `get_boltzmann_factors_al()`

```
vrna_exp_param_t* get_boltzmann_factors_al (
    unsigned int n_seq,
    double temperature,
    double betaScale,
    vrna_md_t md,
    double pf_scale )
```

```
#include <ViennaRNA/params/basic.h>
```

Get precomputed Boltzmann factors of the loop type dependent energy contributions (alifold variant) with independent thermodynamic temperature.

**Deprecated** Use `vrna_exp_params_comparative()` instead!

#### 15.7.4.16 `scale_parameters()`

```
vrna_param_t* scale_parameters (
    void )
```

```
#include <ViennaRNA/params/basic.h>
```

Get precomputed energy contributions for all the known loop types.

##### Note

OpenMP: This function relies on several global model settings variables and thus is not to be considered threadsafe. See [get\\_scaled\\_parameters\(\)](#) for a completely threadsafe implementation.

**Deprecated** Use [vrna\\_params\(\)](#) instead!

##### Returns

A set of precomputed energy contributions

#### 15.7.4.17 `get_scaled_parameters()`

```
vrna_param_t* get_scaled_parameters (
    double temperature,
    vrna_md_t md )
```

```
#include <ViennaRNA/params/basic.h>
```

Get precomputed energy contributions for all the known loop types.

Call this function to retrieve precomputed energy contributions, i.e. scaled according to the temperature passed. Furthermore, this function assumes a data structure that contains the model details as well, such that subsequent folding recursions are able to retrieve the correct model settings

**Deprecated** Use [vrna\\_params\(\)](#) instead!

##### See also

[vrna\\_md\\_t](#), [set\\_model\\_details\(\)](#)

##### Parameters

<i>temperature</i>	The temperature in degrees Celcius
<i>md</i>	The model details

**Returns**

precomputed energy contributions and model settings

## 15.8 Extending the Folding Grammar with Additional Domains

This module covers simple and straight-forward extensions to the RNA folding grammar.

### 15.8.1 Detailed Description

This module covers simple and straight-forward extensions to the RNA folding grammar.

Collaboration diagram for Extending the Folding Grammar with Additional Domains:

#### Modules

- [Unstructured Domains](#)  
*Add and modify unstructured domains to the RNA folding grammar.*
- [Structured Domains](#)  
*Add and modify structured domains to the RNA folding grammar.*

## 15.9 Unstructured Domains

Add and modify unstructured domains to the RNA folding grammar.

### 15.9.1 Detailed Description

Add and modify unstructured domains to the RNA folding grammar.

This module provides the tools to add and modify unstructured domains to the production rules of the RNA folding grammar. Usually this functionality is utilized for incorporating ligand binding to unpaired stretches of an RNA.

**Bug** Although the additional production rule(s) for unstructured domains as described in [Unstructured Domains](#) are always treated as 'segments possibly bound to one or more ligands', the current implementation requires that at least one ligand is bound. The default implementation already takes care of the required changes, however, upon using callback functions other than the default ones, one has to take care of this fact. Please also note, that this behavior might change in one of the next releases, such that the decomposition schemes as shown above comply with the actual implementation.

A default implementation allows one to readily use this feature by simply adding sequence motifs and corresponding binding free energies with the function `vrna_ud_add_motif()` (see also [Ligands Binding to Unstructured Domains](#)).

The grammar extension is realized using a callback function that

- evaluates the binding free energy of a ligand to its target sequence segment (white boxes in the figures above), or
- returns the free energy of an unpaired stretch possibly bound by a ligand, stored in the additional *UDP* matrix.

The callback is passed the segment positions, the loop context, and which of the two above mentioned evaluations are required. A second callback implements the pre-processing step that prepares the *UDP* matrix by evaluating all possible cases of the additional production rule. Both callbacks have a default implementation in *RNAlib*, but may be over-written by a user-implementation, making it fully user-customizable.

For equilibrium probability computations, two additional callbacks exist. One to store/add and one to retrieve the probability of unstructured domains at particular positions. Our implementation already takes care of computing the probabilities, but users of the unstructured domain feature are required to provide a mechanism to efficiently store/add the corresponding values into some external data structure. Collaboration diagram for Unstructured Domains:

### Files

- file [unstructured\\_domains.h](#)

*Functions to modify unstructured domains, e.g. to incorporate ligands binding to unpaired stretches.*

### Data Structures

- struct [vrna\\_unstructured\\_domain\\_s](#)

*Data structure to store all functionality for ligand binding. [More...](#)*

## Macros

- `#define VRNA_UNSTRUCTURED_DOMAIN_EXT_LOOP 1U`  
*Flag to indicate ligand bound to unpaired stretch in the exterior loop.*
- `#define VRNA_UNSTRUCTURED_DOMAIN_HP_LOOP 2U`  
*Flag to indicate ligand bound to unpaired stretch in a hairpin loop.*
- `#define VRNA_UNSTRUCTURED_DOMAIN_INT_LOOP 4U`  
*Flag to indicate ligand bound to unpaired stretch in an interior loop.*
- `#define VRNA_UNSTRUCTURED_DOMAIN_MB_LOOP 8U`  
*Flag to indicate ligand bound to unpaired stretch in a multibranch loop.*
- `#define VRNA_UNSTRUCTURED_DOMAIN_MOTIF 16U`  
*Flag to indicate ligand binding without additional unbound nucleotides (motif-only)*
- `#define VRNA_UNSTRUCTURED_DOMAIN_ALL_LOOPS`  
*Flag to indicate ligand bound to unpaired stretch in any loop (convenience macro)*

## Typedefs

- `typedef struct vrna_unstructured_domain_s vrna_ud_t`  
*Typename for the ligand binding extension data structure `vrna_unstructured_domain_s`.*
- `typedef int() vrna_callback_ud_energy(vrna_fold_compound_t *vc, int i, int j, unsigned int loop_type, void *data)`  
*Callback to retrieve binding free energy of a ligand bound to an unpaired sequence segment.*
- `typedef FLT_OR_DBL() vrna_callback_ud_exp_energy(vrna_fold_compound_t *vc, int i, int j, unsigned int loop_type, void *data)`  
*Callback to retrieve Boltzmann factor of the binding free energy of a ligand bound to an unpaired sequence segment.*
- `typedef void() vrna_callback_ud_production(vrna_fold_compound_t *vc, void *data)`  
*Callback for pre-processing the production rule of the ligand binding to unpaired stretches feature.*
- `typedef void() vrna_callback_ud_exp_production(vrna_fold_compound_t *vc, void *data)`  
*Callback for pre-processing the production rule of the ligand binding to unpaired stretches feature (partition function variant)*
- `typedef void() vrna_callback_ud_probs_add(vrna_fold_compound_t *vc, int i, int j, unsigned int loop_type, FLT_OR_DBL exp_energy, void *data)`  
*Callback to store/add equilibrium probability for a ligand bound to an unpaired sequence segment.*
- `typedef FLT_OR_DBL() vrna_callback_ud_probs_get(vrna_fold_compound_t *vc, int i, int j, unsigned int loop_type, int motif, void *data)`  
*Callback to retrieve equilibrium probability for a ligand bound to an unpaired sequence segment.*

## Functions

- `vrna_ud_motif_t * vrna_ud_motifs_centroid(vrna_fold_compound_t *fc, const char *structure)`  
*Detect unstructured domains in centroid structure.*
- `vrna_ud_motif_t * vrna_ud_motifs_MEA(vrna_fold_compound_t *fc, const char *structure, vrna_ep_t *probability_list)`  
*Detect unstructured domains in MEA structure.*
- `vrna_ud_motif_t * vrna_ud_motifs_MFE(vrna_fold_compound_t *fc, const char *structure)`  
*Detect unstructured domains in MFE structure.*
- `void vrna_ud_add_motif(vrna_fold_compound_t *vc, const char *motif, double motif_en, const char *motif←_name, unsigned int loop_type)`  
*Add an unstructured domain motif, e.g. for ligand binding.*
- `void vrna_ud_remove(vrna_fold_compound_t *vc)`



*Remove ligand binding to unpaired stretches.*

- void [vrna\\_ud\\_set\\_data](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, void \*data, [vrna\\_callback\\_free\\_auxdata](#) \*free\_cb)

*Attach an auxiliary data structure.*

- void [vrna\\_ud\\_set\\_prod\\_rule\\_cb](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_callback\\_ud\\_production](#) \*pre\_cb, [vrna\\_callback\\_ud\\_energy](#) \*e\_cb)

*Attach production rule callbacks for free energies computations.*

- void [vrna\\_ud\\_set\\_exp\\_prod\\_rule\\_cb](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_callback\\_ud\\_exp\\_production](#) \*pre\_cb, [vrna\\_callback\\_ud\\_exp\\_energy](#) \*exp\_e\_cb)

*Attach production rule for partition function.*

## 15.9.2 Data Structure Documentation

### 15.9.2.1 struct [vrna\\_unstructured\\_domain\\_s](#)

Data structure to store all functionality for ligand binding.

#### Data Fields

- int [uniq\\_motif\\_count](#)  
*The unique number of motifs of different lengths.*
- unsigned int \* [uniq\\_motif\\_size](#)  
*An array storing a unique list of motif lengths.*
- int [motif\\_count](#)  
*Total number of distinguished motifs.*
- char \*\* [motif](#)  
*Motif sequences.*
- char \*\* [motif\\_name](#)  
*Motif identifier/name.*
- unsigned int \* [motif\\_size](#)  
*Motif lengths.*
- double \* [motif\\_en](#)  
*Ligand binding free energy contribution.*
- unsigned int \* [motif\\_type](#)  
*Type of motif, i.e. loop type the ligand binds to.*
- [vrna\\_callback\\_ud\\_production](#) \* [prod\\_cb](#)  
*Callback to ligand binding production rule, i.e. create/fill DP free energy matrices.*
- [vrna\\_callback\\_ud\\_exp\\_production](#) \* [exp\\_prod\\_cb](#)  
*Callback to ligand binding production rule, i.e. create/fill DP partition function matrices.*
- [vrna\\_callback\\_ud\\_energy](#) \* [energy\\_cb](#)  
*Callback to evaluate free energy of ligand binding to a particular unpaired stretch.*
- [vrna\\_callback\\_ud\\_exp\\_energy](#) \* [exp\\_energy\\_cb](#)  
*Callback to evaluate Boltzmann factor of ligand binding to a particular unpaired stretch.*
- void \* [data](#)  
*Auxiliary data structure passed to energy evaluation callbacks.*
- [vrna\\_callback\\_free\\_auxdata](#) \* [free\\_data](#)  
*Callback to free auxiliary data structure.*
- [vrna\\_callback\\_ud\\_probs\\_add](#) \* [probs\\_add](#)  
*Callback to store/add outside partition function.*
- [vrna\\_callback\\_ud\\_probs\\_get](#) \* [probs\\_get](#)  
*Callback to retrieve outside partition function.*

### 15.9.2.1.1 Field Documentation

#### 15.9.2.1.1.1 prod\_cb

```
vrna_callback_ud_production* vrna_unstructured_domain_s::prod_cb
```

Callback to ligand binding production rule, i.e. create/fill DP free energy matrices.

This callback will be executed right before the actual secondary structure decompositions, and, therefore, any implementation must not interleave with the regular DP matrices.

## 15.9.3 Typedef Documentation

### 15.9.3.1 vrna\_callback\_ud\_energy

```
typedef int() vrna_callback_ud_energy(vrna_fold_compound_t *vc, int i, int j, unsigned int loop_type, void *data)
```

```
#include <ViennaRNA/unstructured_domains.h>
```

Callback to retrieve binding free energy of a ligand bound to an unpaired sequence segment.

**Notes on Callback Functions** This function will be called to determine the additional energy contribution of a specific unstructured domain, e.g. the binding free energy of some ligand.

#### Parameters

<i>vc</i>	The current <a href="#">vrna_fold_compound_t</a>
<i>i</i>	The start of the unstructured domain (5' end)
<i>j</i>	The end of the unstructured domain (3' end)
<i>loop_type</i>	The loop context of the unstructured domain
<i>data</i>	Auxiliary data

#### Returns

The auxiliary energy contribution in deka-cal/mol

### 15.9.3.2 vrna\_callback\_ud\_exp\_energy

```
typedef FLT\_OR\_DBL() vrna_callback_ud_exp_energy(vrna_fold_compound_t *vc, int i, int j, unsigned int loop_type, void *data)
```

```
#include <ViennaRNA/unstructured_domains.h>
```

Callback to retrieve Boltzmann factor of the binding free energy of a ligand bound to an unpaired sequence segment.

**Notes on Callback Functions** This function will be called to determine the additional energy contribution of a specific unstructured domain, e.g. the binding free energy of some ligand (Partition function variant, i.e. the Boltzmann factors instead of actual free energies).

#### Parameters

<i>vc</i>	The current <code>vrna_fold_compound_t</code>
<i>i</i>	The start of the unstructured domain (5' end)
<i>j</i>	The end of the unstructured domain (3' end)
<i>loop_type</i>	The loop context of the unstructured domain
<i>data</i>	Auxiliary data

#### Returns

The auxiliary energy contribution as Boltzmann factor

#### 15.9.3.3 `vrna_callback_ud_production`

```
typedef void() vrna_callback_ud_production(vrna_fold_compound_t *vc, void *data)
```

```
#include <ViennaRNA/unstructured_domains.h>
```

Callback for pre-processing the production rule of the ligand binding to unpaired stretches feature.

**Notes on Callback Functions** The production rule for the unstructured domain grammar extension

#### 15.9.3.4 `vrna_callback_ud_exp_production`

```
typedef void() vrna_callback_ud_exp_production(vrna_fold_compound_t *vc, void *data)
```

```
#include <ViennaRNA/unstructured_domains.h>
```

Callback for pre-processing the production rule of the ligand binding to unpaired stretches feature (partition function variant)

**Notes on Callback Functions** The production rule for the unstructured domain grammar extension (Partition function variant)

### 15.9.3.5 vrna\_callback\_ud\_probs\_add

```
typedef void() vrna_callback_ud_probs_add(vrna_fold_compound_t *vc, int i, int j, unsigned int loop_type, FLT_OR_DBL exp_energy, void *data)
```

```
#include <ViennaRNA/unstructured_domains.h>
```

Callback to store/add equilibrium probability for a ligand bound to an unpaired sequence segment.

**Notes on Callback Functions** A callback function to store equilibrium probabilities for the unstructured domain feature

### 15.9.3.6 vrna\_callback\_ud\_probs\_get

```
typedef FLT_OR_DBL() vrna_callback_ud_probs_get(vrna_fold_compound_t *vc, int i, int j, unsigned int loop_type, int motif, void *data)
```

```
#include <ViennaRNA/unstructured_domains.h>
```

Callback to retrieve equilibrium probability for a ligand bound to an unpaired sequence segment.

**Notes on Callback Functions** A callback function to retrieve equilibrium probabilities for the unstructured domain feature

## 15.9.4 Function Documentation

### 15.9.4.1 vrna\_ud\_motifs\_centroid()

```
vrna_ud_motif_t* vrna_ud_motifs_centroid (
    vrna_fold_compound_t * fc,
    const char * structure )
```

```
#include <ViennaRNA/unstructured_domains.h>
```

Detect unstructured domains in centroid structure.

Given a centroid structure and a set of unstructured domains compute the list of unstructured domain motifs present in the centroid. Since we do not explicitly annotate unstructured domain motifs in dot-bracket strings, this function can be used to check for the presence and location of unstructured domain motifs under the assumption that the dot-bracket string is the centroid structure of the equilibrium ensemble.

See also

[vrna\\_centroid\(\)](#)

**Parameters**

<i>fc</i>	The fold_compound data structure with pre-computed equilibrium probabilities and model settings
<i>structure</i>	The centroid structure in dot-bracket notation

**Returns**

A list of unstructured domain motifs (possibly NULL). The last element terminates the list with `start=0`, `number=-1`

**15.9.4.2 vrna\_ud\_motifs\_MEA()**

```
vrna_ud_motif_t* vrna_ud_motifs_MEA (
    vrna_fold_compound_t * fc,
    const char * structure,
    vrna_ep_t * probability_list )
```

```
#include <ViennaRNA/unstructured_domains.h>
```

Detect unstructured domains in MEA structure.

Given an MEA structure and a set of unstructured domains compute the list of unstructured domain motifs present in the MEA structure. Since we do not explicitly annotate unstructured domain motifs in dot-bracket strings, this function can be used to check for the presence and location of unstructured domain motifs under the assumption that the dot-bracket string is the MEA structure of the equilibrium ensemble.

**See also**

[MEA\(\)](#)

**Parameters**

<i>fc</i>	The fold_compound data structure with pre-computed equilibrium probabilities and model settings
<i>structure</i>	The MEA structure in dot-bracket notation
<i>probability_list</i>	The list of probabilities to extract the MEA structure from

**Returns**

A list of unstructured domain motifs (possibly NULL). The last element terminates the list with `start=0`, `number=-1`

**15.9.4.3 vrna\_ud\_motifs\_MFE()**

```
vrna_ud_motif_t* vrna_ud_motifs_MFE (
    vrna_fold_compound_t * fc,
    const char * structure )

#include <ViennaRNA/unstructured_domains.h>
```

Detect unstructured domains in MFE structure.

Given an MFE structure and a set of unstructured domains compute the list of unstructured domain motifs present in the MFE structure. Since we do not explicitly annotate unstructured domain motifs in dot-bracket strings, this function can be used to check for the presence and location of unstructured domain motifs under the assumption that the dot-bracket string is the MFE structure of the equilibrium ensemble.

See also

[vrna\\_mfe\(\)](#)

#### Parameters

<i>fc</i>	The fold_compound data structure with model settings
<i>structure</i>	The MFE structure in dot-bracket notation

#### Returns

A list of unstructured domain motifs (possibly NULL). The last element terminates the list with `start=0`, `number=-1`

#### 15.9.4.4 vrna\_ud\_add\_motif()

```
void vrna_ud_add_motif (
    vrna_fold_compound_t * vc,
    const char * motif,
    double motif_en,
    const char * motif_name,
    unsigned int loop_type )

#include <ViennaRNA/unstructured_domains.h>
```

Add an unstructured domain motif, e.g. for ligand binding.

This function adds a ligand binding motif and the associated binding free energy to the [vrna\\_ud\\_t](#) attribute of a [vrna\\_fold\\_compound\\_t](#). The motif data will then be used in subsequent secondary structure predictions. Multiple calls to this function with different motifs append all additional data to a list of ligands, which all will be evaluated. Ligand motif data can be removed from the [vrna\\_fold\\_compound\\_t](#) again using the [vrna\\_ud\\_remove\(\)](#) function. The loop type parameter allows one to limit the ligand binding to particular loop type, such as the exterior loop, hairpin loops, interior loops, or multibranch loops.

See also

[VRNA\\_UNSTRUCTURED\\_DOMAIN\\_EXT\\_LOOP](#), [VRNA\\_UNSTRUCTURED\\_DOMAIN\\_HP\\_LOOP](#), [VRNA\\_UNSTRUCTURED\\_DOMAIN\\_MB\\_LOOP](#), [VRNA\\_UNSTRUCTURED\\_DOMAIN\\_ALL\\_LOOPS](#), [vrna\\_ud\\_remove\(\)](#)

## Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> data structure the ligand motif should be bound to
<i>motif</i>	The sequence motif the ligand binds to
<i>motif_en</i>	The binding free energy of the ligand in kcal/mol
<i>motif_name</i>	The name/id of the motif (may be <code>NULL</code> )
<i>loop_type</i>	The loop type the ligand binds to

15.9.4.5 `vrna_ud_remove()`

```
void vrna_ud_remove (
    vrna_fold_compound_t * vc )

#include <ViennaRNA/unstructured_domains.h>
```

Remove ligand binding to unpaired stretches.

This function removes all ligand motifs that were bound to a [vrna\\_fold\\_compound\\_t](#) using the [vrna\\_ud\\_add\\_motif\(\)](#) function.

## Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> data structure the ligand motif data should be removed from
-----------	--

**SWIG Wrapper Notes** This function is attached as method `ud_remove()` to objects of type *fold\_compound*

15.9.4.6 `vrna_ud_set_data()`

```
void vrna_ud_set_data (
    vrna_fold_compound_t * vc,
    void * data,
    vrna_callback_free_auxdata * free_cb )

#include <ViennaRNA/unstructured_domains.h>
```

Attach an auxiliary data structure.

This function binds an arbitrary, auxiliary data structure for user-implemented ligand binding. The optional callback `free_cb` will be passed the bound data structure whenever the [vrna\\_fold\\_compound\\_t](#) is removed from memory to avoid memory leaks.

## See also

[vrna\\_ud\\_set\\_prod\\_rule\\_cb\(\)](#), [vrna\\_ud\\_set\\_exp\\_prod\\_rule\\_cb\(\)](#), [vrna\\_ud\\_remove\(\)](#)





## Parameters

<code>vc</code>	The <a href="#">vrna_fold_compound_t</a> data structure the callback will be bound to
<code>pre_cb</code>	A pointer to a callback function for the $B$ production rule
<code>e_cb</code>	A pointer to a callback function for free energy evaluation

**SWIG Wrapper Notes** This function is attached as method `ud_set_prod_rule_cb()` to objects of type `fold_compound`

15.9.4.8 `vrna_ud_set_exp_prod_rule_cb()`

```
void vrna_ud_set_exp_prod_rule_cb (
    vrna_fold_compound_t * vc,
    vrna_callback_ud_exp_production * pre_cb,
    vrna_callback_ud_exp_energy * exp_e_cb )
```

```
#include <ViennaRNA/unstructured_domains.h>
```

Attach production rule for partition function.

This function is the partition function companion of [vrna\\_ud\\_set\\_prod\\_rule\\_cb\(\)](#).

Use it to bind callbacks to (i) fill the  $U$  production rule dynamic programming matrices and/or prepare the [vrna\\_unstructured\\_domain\\_s.data](#), and (ii) provide a callback to retrieve partition functions for subsegments  $[i, j]$ .

$$\begin{array}{c} \text{---} B \text{---} \\ i \qquad \qquad j \end{array} = \begin{array}{c} \text{---} B \text{---} \\ i \qquad \qquad j-1 \end{array} \bullet_j \mid \begin{array}{c} \text{---} B \text{---} \\ i \qquad \qquad u \end{array} \begin{array}{c} \boxed{\phantom{000}} \\ u+1 \qquad j \end{array}$$

$$f(i, j) = \begin{array}{c} \boxed{\phantom{000}} \\ i \qquad \qquad j \end{array} \mid \begin{array}{c} \text{---} B \text{---} \\ i \qquad \qquad j \end{array}$$

See also

[vrna\\_ud\\_set\\_prod\\_rule\\_cb\(\)](#)

## Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> data structure the callback will be bound to
<i>pre_cb</i>	A pointer to a callback function for the $\mathbb{B}$ production rule
<i>exp_e_cb</i>	A pointer to a callback function that retrieves the partition function for a segment $[i, j]$ that may be bound by one or more ligands.

**SWIG Wrapper Notes** This function is attached as method `ud_set_exp_prod_rule_cb()` to objects of type *fold↵\_compound*

## 15.10 Structured Domains

Add and modify structured domains to the RNA folding grammar.

### 15.10.1 Detailed Description

Add and modify structured domains to the RNA folding grammar.

This module provides the tools to add and modify structured domains to the production rules of the RNA folding grammar. Usually this functionality is utilized for incorporating self-enclosed structural modules that exhibit a more or less complex base pairing pattern. Collaboration diagram for Structured Domains:

#### Files

- file [structured\\_domains.h](#)

*This module provides interfaces that deal with additional structured domains in the folding grammar.*

## 15.11 Constraining the RNA Folding Grammar

This module provides general functions that allow for an easy control of constrained secondary structure prediction and evaluation.

### 15.11.1 Detailed Description

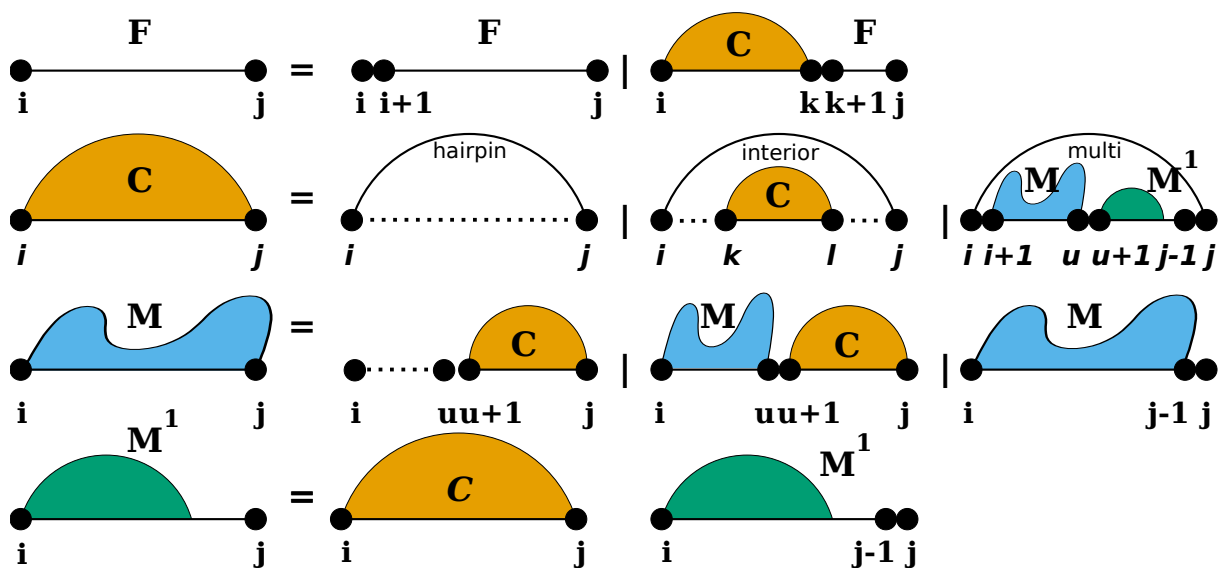
This module provides general functions that allow for an easy control of constrained secondary structure prediction and evaluation.

Secondary Structure constraints can be subdivided into two groups:

- [Hard Constraints](#), and
- [Soft Constraints](#).

While Hard-Constraints directly influence the production rules used in the folding recursions by allowing, disallowing, or enforcing certain decomposition steps, Soft-constraints on the other hand are used to change position specific contributions in the recursions by adding bonuses/penalties in form of pseudo free energies to certain loop configurations.

Secondary structure constraints are always applied at decomposition level, i.e. in each step of the recursive structure decomposition, for instance during MFE prediction. Below is a visualization of the decomposition scheme



For [Hard Constraints](#) the following option flags may be used to constrain the pairing behavior of single, or pairs of nucleotides:

- [VRNA\\_CONSTRAINT\\_CONTEXT\\_EXT\\_LOOP](#) - Hard constraints flag, base pair in the exterior loop.
- [VRNA\\_CONSTRAINT\\_CONTEXT\\_HP\\_LOOP](#) - Hard constraints flag, base pair encloses hairpin loop.
- [VRNA\\_CONSTRAINT\\_CONTEXT\\_INT\\_LOOP](#) - Hard constraints flag, base pair encloses an interior loop.
- [VRNA\\_CONSTRAINT\\_CONTEXT\\_INT\\_LOOP\\_ENC](#) - Hard constraints flag, base pair encloses a multi branch loop.

- [VRNA\\_CONSTRAINT\\_CONTEXT\\_MB\\_LOOP](#) - Hard constraints flag, base pair is enclosed in an interior loop.
- [VRNA\\_CONSTRAINT\\_CONTEXT\\_MB\\_LOOP\\_ENC](#) - Hard constraints flag, base pair is enclosed in a multi branch loop.
- [VRNA\\_CONSTRAINT\\_CONTEXT\\_ENFORCE](#) - Hard constraint flag to indicate enforcement of constraints.
- [VRNA\\_CONSTRAINT\\_CONTEXT\\_NO\\_REMOVE](#) - Hard constraint flag to indicate not to remove base pairs that conflict with a given constraint.
- [VRNA\\_CONSTRAINT\\_CONTEXT\\_ALL\\_LOOPS](#) - Constraint context flag indicating any loop context.

However, for [Soft Constraints](#) we do not allow for simple loop type dependent constraining. But soft constraints are equipped with generic constraint support. This enables the user to pass arbitrary callback functions that return auxiliary energy contributions for evaluation the evaluation of any decomposition.

The callback will then always be notified about the type of decomposition that is happening, and the corresponding delimiting sequence positions. The following decomposition steps are distinguished, and should be captured by the user's implementation of the callback:

- [VRNA\\_DECOMP\\_PAIR\\_HP](#) - Flag passed to generic softt constraints callback to indicate hairpin loop decomposition step.
- [VRNA\\_DECOMP\\_PAIR\\_IL](#) - Indicator for interior loop decomposition step.
- [VRNA\\_DECOMP\\_PAIR\\_ML](#) - Indicator for multibranch loop decomposition step.
- [VRNA\\_DECOMP\\_ML\\_ML\\_ML](#) - Indicator for decomposition of multibranch loop part.
- [VRNA\\_DECOMP\\_ML\\_STEM](#) - Indicator for decomposition of multibranch loop part.
- [VRNA\\_DECOMP\\_ML\\_ML](#) - Indicator for decomposition of multibranch loop part.
- [VRNA\\_DECOMP\\_ML\\_UP](#) - Indicator for decomposition of multibranch loop part.
- [VRNA\\_DECOMP\\_ML\\_ML\\_STEM](#) - Indicator for decomposition of multibranch loop part.
- [VRNA\\_DECOMP\\_ML\\_COAXIAL](#) - Indicator for decomposition of multibranch loop part.
- [VRNA\\_DECOMP\\_EXT\\_EXT](#) - Indicator for decomposition of exterior loop part.
- [VRNA\\_DECOMP\\_EXT\\_UP](#) - Indicator for decomposition of exterior loop part.
- [VRNA\\_DECOMP\\_EXT\\_STEM](#) - Indicator for decomposition of exterior loop part.
- [VRNA\\_DECOMP\\_EXT\\_EXT\\_EXT](#) - Indicator for decomposition of exterior loop part.
- [VRNA\\_DECOMP\\_EXT\\_STEM\\_EXT](#) - Indicator for decomposition of exterior loop part.
- [VRNA\\_DECOMP\\_EXT\\_STEM\\_OUTSIDE](#) - Indicator for decomposition of exterior loop part.
- [VRNA\\_DECOMP\\_EXT\\_EXT\\_STEM](#) - Indicator for decomposition of exterior loop part.
- [VRNA\\_DECOMP\\_EXT\\_EXT\\_STEM1](#) - Indicator for decomposition of exterior loop part.

Simplified interfaces to the soft constraints framework can be obtained by the implementations in the submodules

- [SHAPE Reactivity Data](#) and
- [ligands](#).

An implementation that generates soft constraints for unpaired nucleotides by minimizing the discrepancy between their predicted and expected pairing probability is available in submodule [Generate Soft Constraints from Data](#). Collaboration diagram for Constraining the RNA Folding Grammar:

## Modules

- [Hard Constraints](#)

*This module covers all functionality for hard constraints in secondary structure prediction.*

- [Soft Constraints](#)

*Functions and data structures for secondary structure soft constraints.*

## Files

- file [basic.h](#)

*Functions and data structures for constraining secondary structure predictions and evaluation.*

## Macros

- `#define VRNA_CONSTRAINT_FILE 0`

*Flag for [vrna\\_constraints\\_add\(\)](#) to indicate that constraints are present in a text file.*

- `#define VRNA_CONSTRAINT_SOFT_MFE 0`

*Indicate generation of constraints for MFE folding.*

- `#define VRNA_CONSTRAINT_SOFT_PF VRNA_OPTION_PF`

*Indicate generation of constraints for partition function computation.*

- `#define VRNA_DECOMP_PAIR_HP (unsigned char)1`

*Flag passed to generic softt constraints callback to indicate hairpin loop decomposition step.*

- `#define VRNA_DECOMP_PAIR_IL (unsigned char)2`

*Indicator for interior loop decomposition step.*

- `#define VRNA_DECOMP_PAIR_ML (unsigned char)3`

*Indicator for multibranch loop decomposition step.*

- `#define VRNA_DECOMP_ML_ML_ML (unsigned char)5`

*Indicator for decomposition of multibranch loop part.*

- `#define VRNA_DECOMP_ML_STEM (unsigned char)6`

*Indicator for decomposition of multibranch loop part.*

- `#define VRNA_DECOMP_ML_ML (unsigned char)7`

*Indicator for decomposition of multibranch loop part.*

- `#define VRNA_DECOMP_ML_UP (unsigned char)8`

*Indicator for decomposition of multibranch loop part.*

- `#define VRNA_DECOMP_ML_ML_STEM (unsigned char)9`

*Indicator for decomposition of multibranch loop part.*

- `#define VRNA_DECOMP_ML_COAXIAL (unsigned char)10`

*Indicator for decomposition of multibranch loop part.*

- `#define VRNA_DECOMP_ML_COAXIAL_ENC (unsigned char)11`

*Indicator for decomposition of multibranch loop part.*

- `#define VRNA_DECOMP_EXT_EXT (unsigned char)12`

*Indicator for decomposition of exterior loop part.*

- `#define VRNA_DECOMP_EXT_UP (unsigned char)13`

*Indicator for decomposition of exterior loop part.*

- `#define VRNA_DECOMP_EXT_STEM (unsigned char)14`

*Indicator for decomposition of exterior loop part.*

- `#define VRNA_DECOMP_EXT_EXT_EXT (unsigned char)15`

*Indicator for decomposition of exterior loop part.*

- `#define VRNA_DECOMP_EXT_STEM_EXT (unsigned char)16`

- Indicator for decomposition of exterior loop part.
- `#define VRNA_DECOMP_EXT_STEM_OUTSIDE` (unsigned char)17
- Indicator for decomposition of exterior loop part.
- `#define VRNA_DECOMP_EXT_EXT_STEM` (unsigned char)18
- Indicator for decomposition of exterior loop part.
- `#define VRNA_DECOMP_EXT_EXT_STEM1` (unsigned char)19
- Indicator for decomposition of exterior loop part.

## Functions

- void `vrna_constraints_add` (`vrna_fold_compound_t` \*vc, const char \*constraint, unsigned int options)  
Add constraints to a `vrna_fold_compound_t` data structure.
- void `vrna_message_constraint_options` (unsigned int option)  
Print a help message for pseudo dot-bracket structure constraint characters to stdout. (constraint support is specified by option parameter)
- void `vrna_message_constraint_options_all` (void)  
Print structure constraint characters to stdout (full constraint support)

### 15.11.2 Macro Definition Documentation

#### 15.11.2.1 VRNA\_CONSTRAINT\_FILE

```
#define VRNA_CONSTRAINT_FILE 0

#include <ViennaRNA/constraints/basic.h>
```

Flag for `vrna_constraints_add()` to indicate that constraints are present in a text file.

See also

`vrna_constraints_add()`

**Deprecated** Use 0 instead!

#### 15.11.2.2 VRNA\_CONSTRAINT\_SOFT\_MFE

```
#define VRNA_CONSTRAINT_SOFT_MFE 0

#include <ViennaRNA/constraints/basic.h>
```

Indicate generation of constraints for MFE folding.

**Deprecated** This flag has no meaning anymore, since constraints are now always stored!

### 15.11.2.3 VRNA\_CONSTRAINT\_SOFT\_PF

```
#define VRNA_CONSTRAINT_SOFT_PF VRNA_OPTION_PF  
  
#include <ViennaRNA/constraints/basic.h>
```

Indicate generation of constraints for partition function computation.

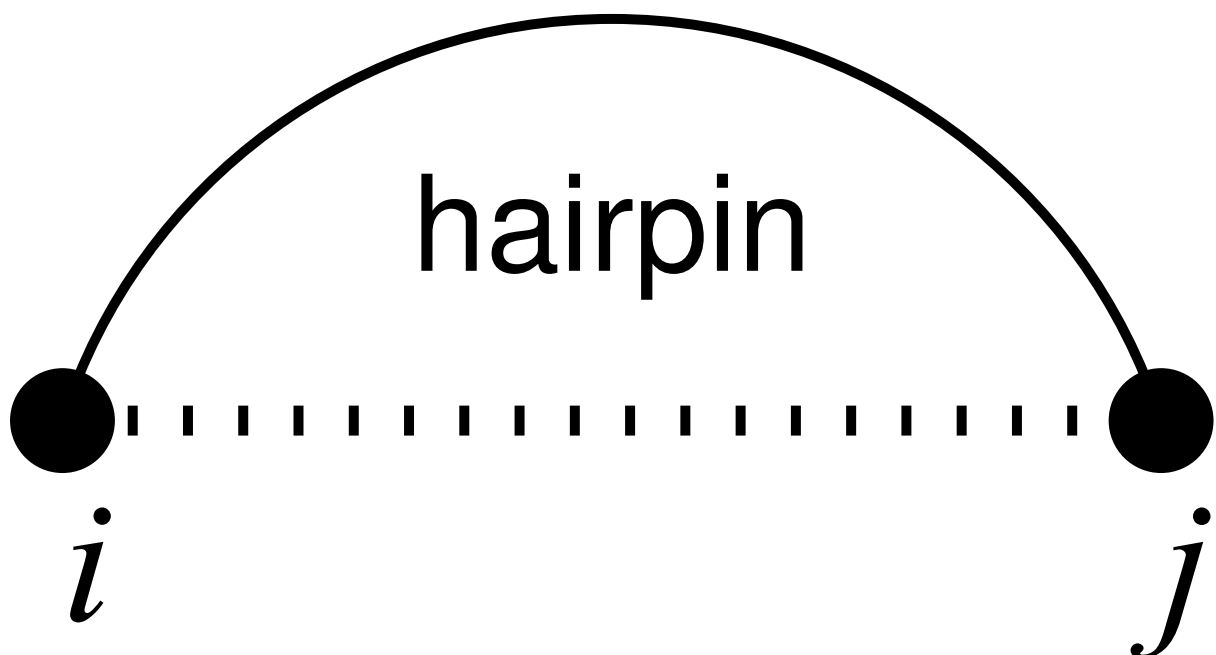
**Deprecated** Use `VRNA_OPTION_PF` instead!

### 15.11.2.4 VRNA\_DECOMP\_PAIR\_HP

```
#define VRNA_DECOMP_PAIR_HP (unsigned char)1  
  
#include <ViennaRNA/constraints/basic.h>
```

Flag passed to generic softt constraints callback to indicate hairpin loop decomposition step.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a hairpin loop enclosed by the base pair  $(i, j)$ .





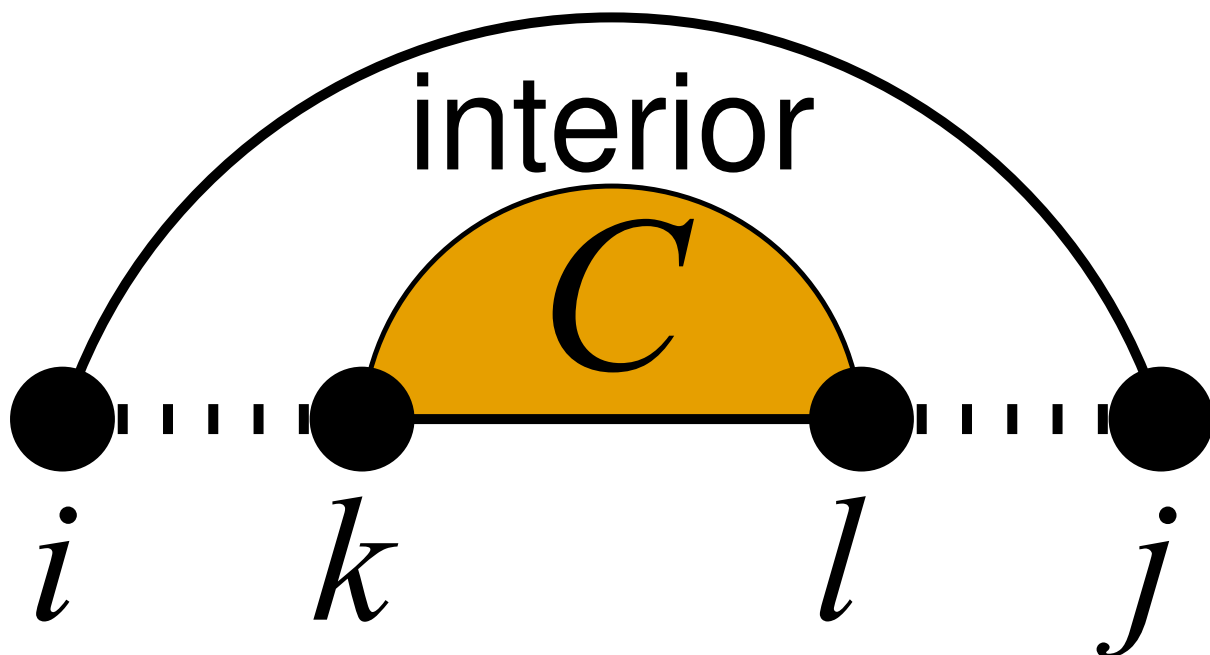
## 15.11.2.5 VRNA\_DECOMP\_PAIR\_IL

```
#define VRNA_DECOMP_PAIR_IL (unsigned char)2
```

```
#include <ViennaRNA/constraints/basic.h>
```

Indicator for interior loop decomposition step.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an interior loop enclosed by the base pair  $(i, j)$ , and enclosing the base pair  $(k, l)$ .



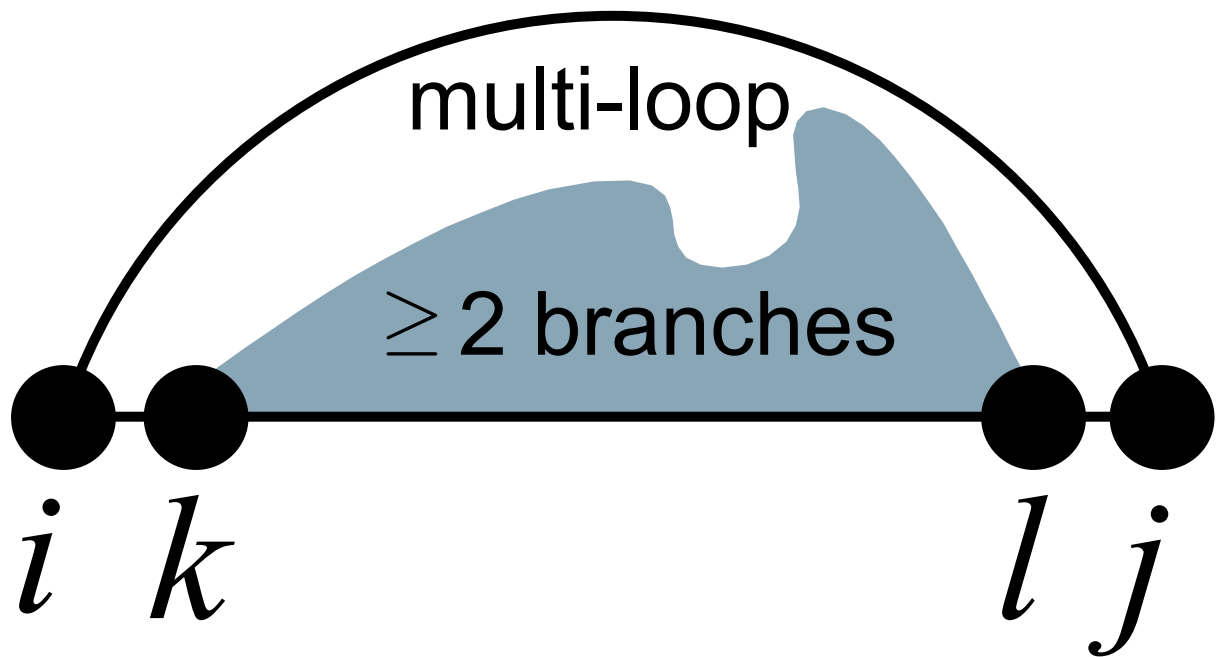
## 15.11.2.6 VRNA\_DECOMP\_PAIR\_ML

```
#define VRNA_DECOMP_PAIR_ML (unsigned char)3
```

```
#include <ViennaRNA/constraints/basic.h>
```

Indicator for multibranch loop decomposition step.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multibranch loop enclosed by the base pair  $(i, j)$ , and consisting of some enclosed multi loop content from k to l.



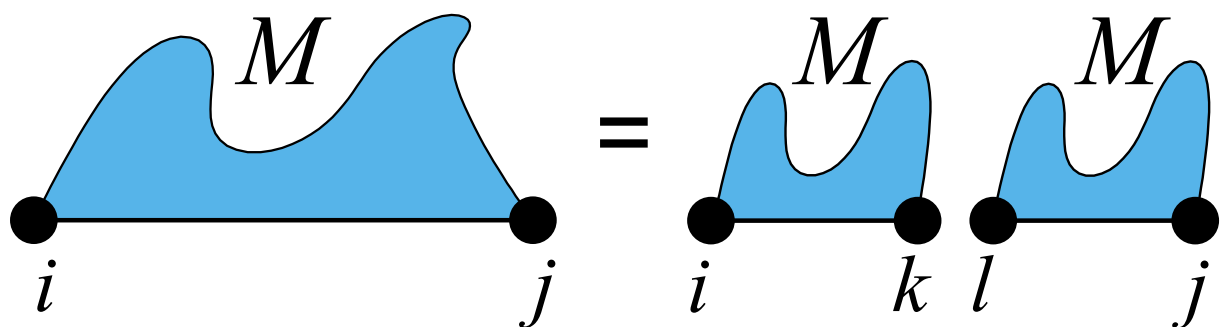
#### 15.11.2.7 VRNA\_DECOMP\_ML\_ML\_ML

```
#define VRNA_DECOMP_ML_ML_ML (unsigned char)5

#include <ViennaRNA/constraints/basic.h>
```

Indicator for decomposition of multibranch loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multi-branch loop part in the interval  $[i : j]$ , which will be decomposed into two multibranch loop parts  $[i : k]$ , and  $[l : j]$ .



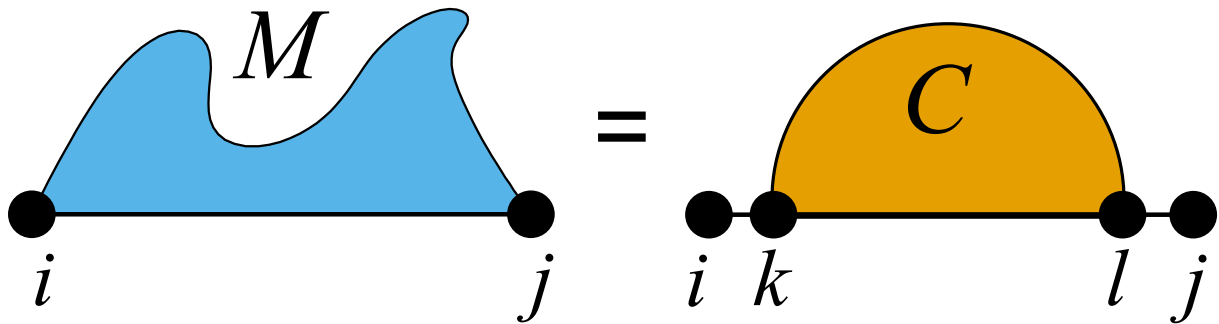
## 15.11.2.8 VRNA\_DECOMP\_ML\_STEM

```
#define VRNA_DECOMP_ML_STEM (unsigned char)6
```

```
#include <ViennaRNA/constraints/basic.h>
```

Indicator for decomposition of multibranch loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multibranch loop part in the interval  $[i : j]$ , which will be considered a single stem branching off with base pair  $(k, l)$ .



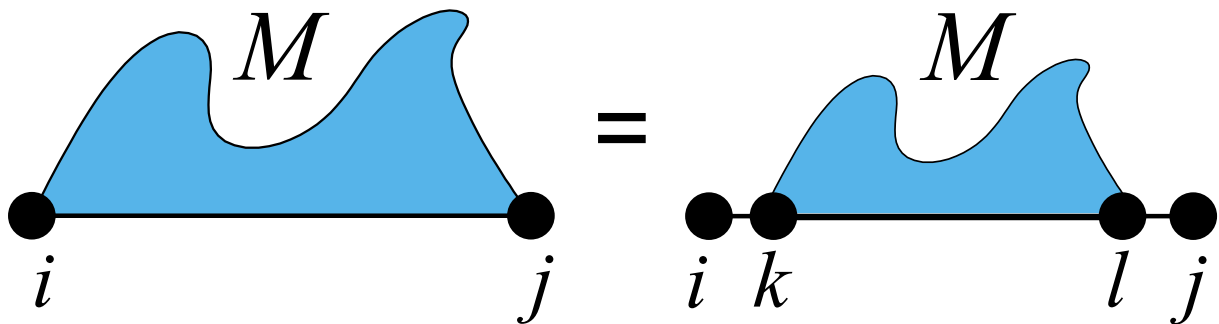
## 15.11.2.9 VRNA\_DECOMP\_ML\_ML

```
#define VRNA_DECOMP_ML_ML (unsigned char)7
```

```
#include <ViennaRNA/constraints/basic.h>
```

Indicator for decomposition of multibranch loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multibranch loop part in the interval  $[i : j]$ , which will be decomposed into a (usually) smaller multibranch loop part  $[k : l]$ .



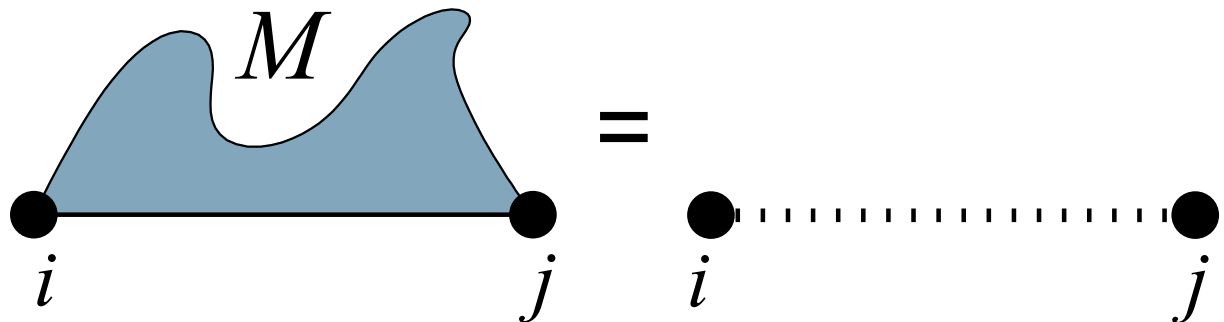
## 15.11.2.10 VRNA\_DECOMP\_ML\_UP

```
#define VRNA_DECOMP_ML_UP (unsigned char)8
```

```
#include <ViennaRNA/constraints/basic.h>
```

Indicator for decomposition of multibranch loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multibranch loop part in the interval  $[i : j]$ , which will be considered a multibranch loop part that only consists of unpaired nucleotides.



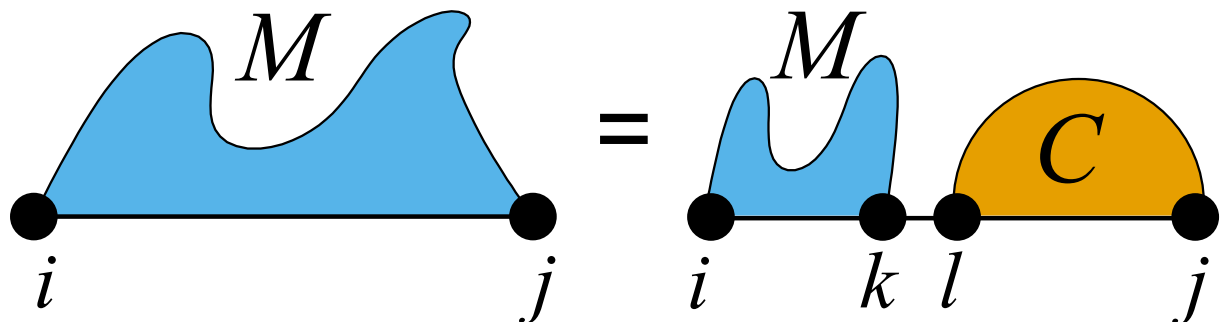
## 15.11.2.11 VRNA\_DECOMP\_ML\_ML\_STEM

```
#define VRNA_DECOMP_ML_ML_STEM (unsigned char)9
```

```
#include <ViennaRNA/constraints/basic.h>
```

Indicator for decomposition of multibranch loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multibranch loop part in the interval  $[i : j]$ , which will be decomposed into a multibranch loop part  $[i : k]$ , and a stem with enclosing base pair  $(l, j)$ .



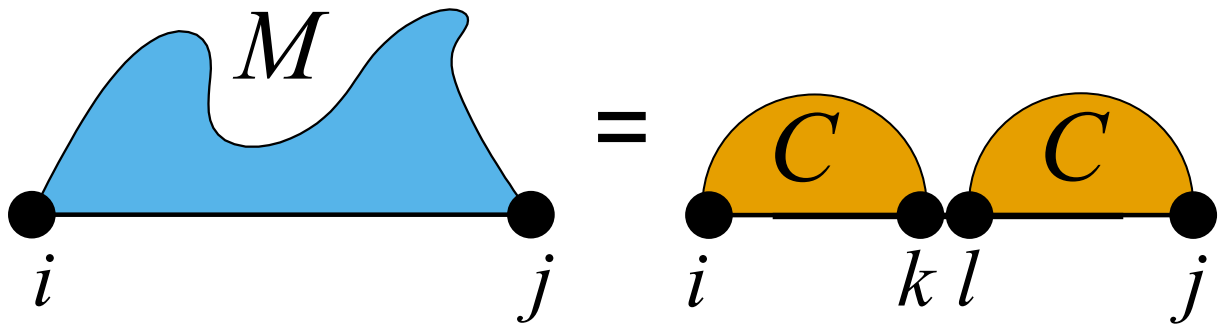
## 15.11.2.12 VRNA\_DECOMP\_ML\_COAXIAL

```
#define VRNA_DECOMP_ML_COAXIAL (unsigned char)10
```

```
#include <ViennaRNA/constraints/basic.h>
```

Indicator for decomposition of multibranch loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multibranch loop part in the interval  $[i : j]$ , where two stems with enclosing pairs  $(i, k)$  and  $(l, j)$  are coaxially stacking onto each other.



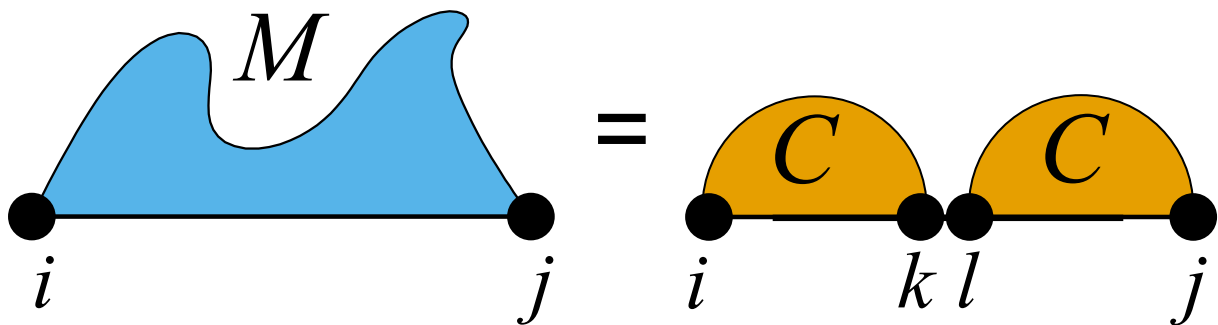
## 15.11.2.13 VRNA\_DECOMP\_ML\_COAXIAL\_ENC

```
#define VRNA_DECOMP_ML_COAXIAL_ENC (unsigned char)11
```

```
#include <ViennaRNA/constraints/basic.h>
```

Indicator for decomposition of multibranch loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multibranch loop part in the interval  $[i : j]$ , where two stems with enclosing pairs  $(i, k)$  and  $(l, j)$  are coaxially stacking onto each other.



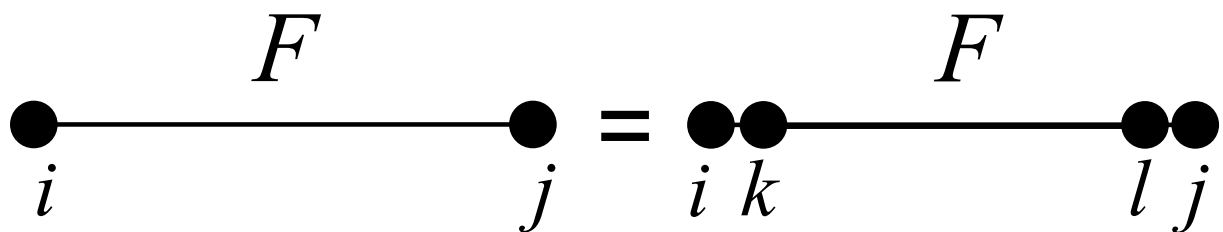
## 15.11.2.14 VRNA\_DECOMP\_EXT\_EXT

```
#define VRNA_DECOMP_EXT_EXT (unsigned char)12
```

```
#include <ViennaRNA/constraints/basic.h>
```

Indicator for decomposition of exterior loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval  $[i : j]$ , which will be decomposed into a (usually) smaller exterior loop part  $[k : l]$ .



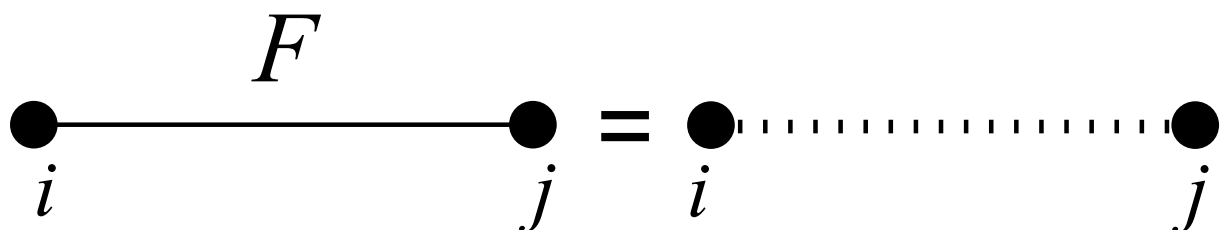
## 15.11.2.15 VRNA\_DECOMP\_EXT\_UP

```
#define VRNA_DECOMP_EXT_UP (unsigned char)13
```

```
#include <ViennaRNA/constraints/basic.h>
```

Indicator for decomposition of exterior loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval  $[i : j]$ , which will be considered as an exterior loop component consisting of only unpaired nucleotides.



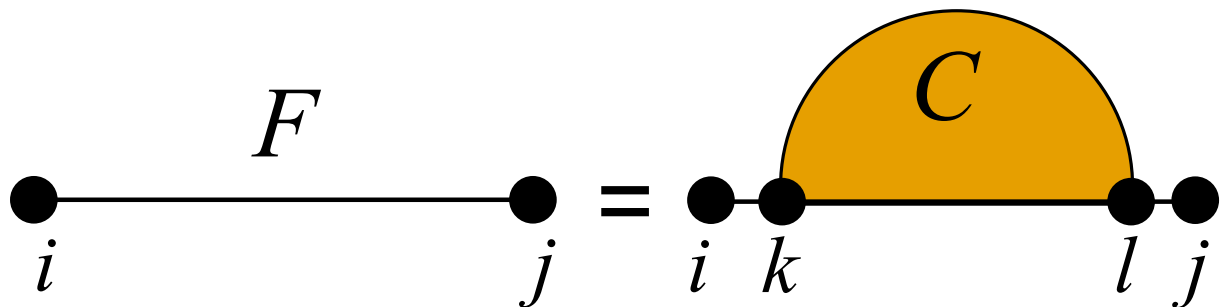
## 15.11.2.16 VRNA\_DECOMP\_EXT\_STEM

```
#define VRNA_DECOMP_EXT_STEM (unsigned char)14
```

```
#include <ViennaRNA/constraints/basic.h>
```

Indicator for decomposition of exterior loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval  $[i : j]$ , which will be considered a stem with enclosing pair  $(k, l)$ .



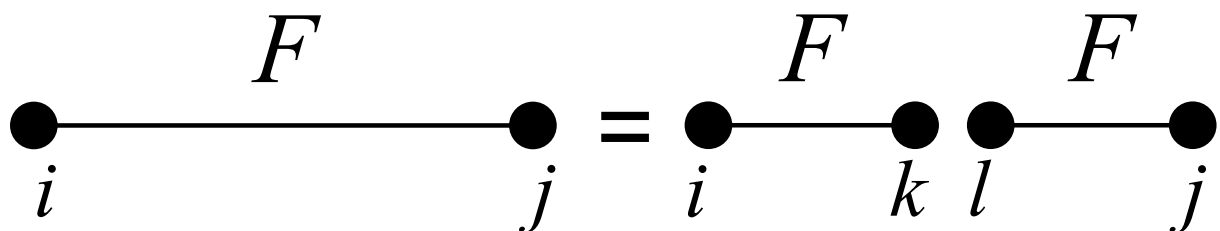
## 15.11.2.17 VRNA\_DECOMP\_EXT\_EXT\_EXT

```
#define VRNA_DECOMP_EXT_EXT_EXT (unsigned char)15
```

```
#include <ViennaRNA/constraints/basic.h>
```

Indicator for decomposition of exterior loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval  $[i : j]$ , which will be decomposed into two exterior loop parts  $[i : k]$  and  $[l : j]$ .



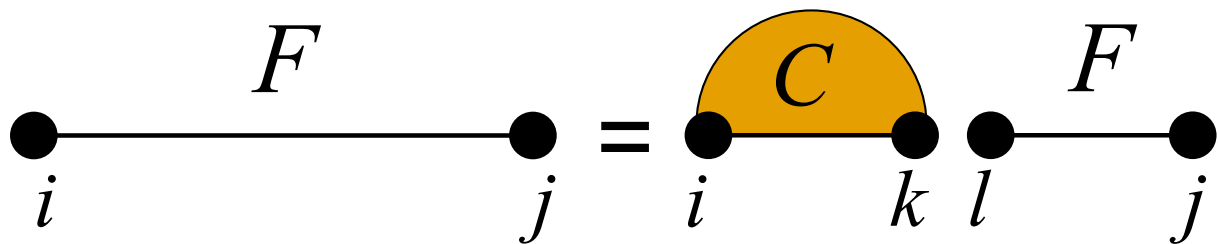
## 15.11.2.18 VRNA\_DECOMP\_EXT\_STEM\_EXT

```
#define VRNA_DECOMP_EXT_STEM_EXT (unsigned char)16
```

```
#include <ViennaRNA/constraints/basic.h>
```

Indicator for decomposition of exterior loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval  $[i : j]$ , which will be decomposed into a stem branching off with base pair  $(i, k)$ , and an exterior loop part  $[l : j]$ .



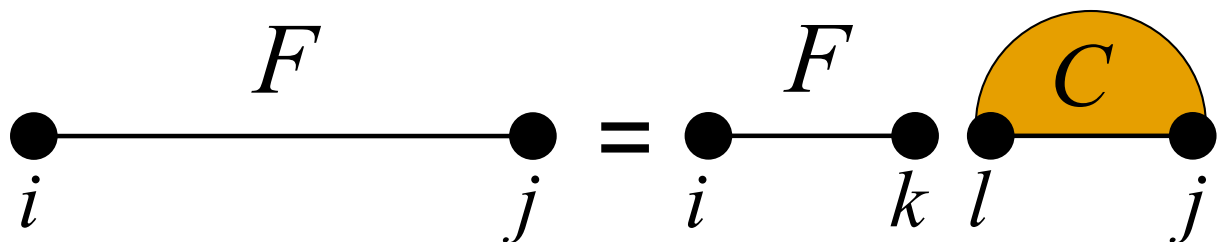
## 15.11.2.19 VRNA\_DECOMP\_EXT\_EXT\_STEM

```
#define VRNA_DECOMP_EXT_EXT_STEM (unsigned char)18
```

```
#include <ViennaRNA/constraints/basic.h>
```

Indicator for decomposition of exterior loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval  $[i : j]$ , which will be decomposed into an exterior loop part  $[i : k]$ , and a stem branching off with base pair  $(l, j)$ .





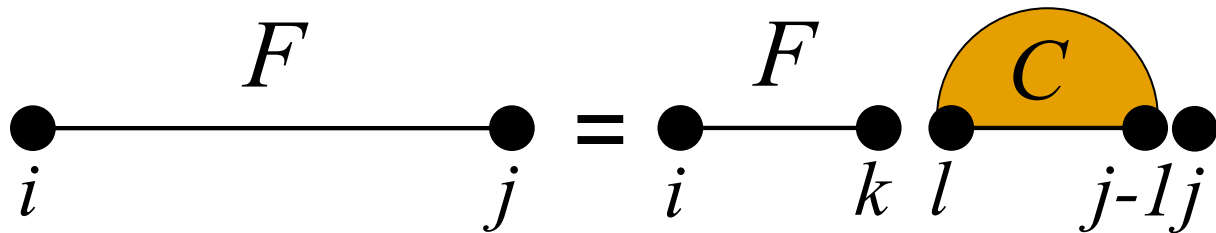
## 15.11.2.20 VRNA\_DECOMP\_EXT\_EXT\_STEM1

```
#define VRNA_DECOMP_EXT_EXT_STEM1 (unsigned char)19
```

```
#include <ViennaRNA/constraints/basic.h>
```

Indicator for decomposition of exterior loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval  $[i : j]$ , which will be decomposed into an exterior loop part  $[i : k]$ , and a stem branching off with base pair  $(l, j - 1)$ .



## 15.11.3 Function Documentation

## 15.11.3.1 vrna\_constraints\_add()

```
void vrna_constraints_add (
    vrna_fold_compound_t * vc,
    const char * constraint,
    unsigned int options )
```

```
#include <ViennaRNA/constraints/basic.h>
```

Add constraints to a `vrna_fold_compound_t` data structure.

Use this function to add/update the hard/soft constraints. The function allows for passing a string 'constraint' that can either be a filename that points to a constraints definition file or it may be a pseudo dot-bracket notation indicating hard constraints. For the latter, the user has to pass the `VRNA_CONSTRAINT_DB` option. Also, the user has to specify, which characters are allowed to be interpreted as constraints by passing the corresponding options via the third parameter.

See also

```
vrna_hc_init(), vrna_hc_add_up(), vrna_hc_add_up_batch(), vrna_hc_add_bp(), vrna_sc_init(), vrna_sc_set_up(),
vrna_sc_set_bp(), vrna_sc_add_SHAPE_deigan(), vrna_sc_add_SHAPE_zarringhalam(), vrna_hc_free(),
vrna_sc_free(), VRNA_CONSTRAINT_DB, VRNA_CONSTRAINT_DB_DEFAULT, VRNA_CONSTRAINT_DB_PIPE,
VRNA_CONSTRAINT_DB_DOT, VRNA_CONSTRAINT_DB_X, VRNA_CONSTRAINT_DB_ANG_BRACK,
VRNA_CONSTRAINT_DB_RND_BRACK, VRNA_CONSTRAINT_DB_INTRAMOL, VRNA_CONSTRAINT_DB_INTERMOL,
VRNA_CONSTRAINT_DB_GQUAD
```

The following is an example for adding hard constraints given in pseudo dot-bracket notation. Here, `vc` is the `vrna_fold_compound_t` object, `structure` is a char array with the hard constraint in dot-bracket notation, and `enforceConstraints` is a flag indicating whether or not constraints for base pairs should be enforced instead of just doing a removal of base pair that conflict with the constraint.

```

unsigned int constraint_options = VRNA_CONSTRAINT_DB_DEFAULT;

if (enforceConstraints)
    constraint_options |= VRNA_CONSTRAINT_DB_ENFORCE_BP;

if (canonicalBOnly)
    constraint_options |= VRNA_CONSTRAINT_DB_CANONICAL_BP;

vrna_constraints_add(fc, (const char *)cstruc, constraint_options);

```

In constrat to the above, constraints may also be read from file:

```

vrna_constraints_add(fc, constraints_file,
    VRNA_OPTION_DEFAULT);

```

#### See also

[vrna\\_hc\\_add\\_from\\_db\(\)](#), [vrna\\_hc\\_add\\_up\(\)](#), [vrna\\_hc\\_add\\_up\\_batch\(\)](#) [vrna\\_hc\\_add\\_bp\\_unspecific\(\)](#),  
[vrna\\_hc\\_add\\_bp\(\)](#)

#### Parameters

<i>vc</i>	The fold compound
<i>constraint</i>	A string with either the filename of the constraint definitions or a pseudo dot-bracket notation of the hard constraint. May be NULL.
<i>options</i>	The option flags

#### 15.11.3.2 vrna\_message\_constraint\_options()

```

void vrna_message_constraint_options (
    unsigned int option )

```

```
#include <ViennaRNA/constraints/hard.h>
```

Print a help message for pseudo dot-bracket structure constraint characters to stdout. (constraint support is specified by option parameter)

Currently available options are:

[VRNA\\_CONSTRAINT\\_DB\\_PIPE](#) (paired with another base)  
[VRNA\\_CONSTRAINT\\_DB\\_DOT](#) (no constraint at all)  
[VRNA\\_CONSTRAINT\\_DB\\_X](#) (base must not pair)  
[VRNA\\_CONSTRAINT\\_DB\\_ANG\\_BRACK](#) (paired downstream/upstream)  
[VRNA\\_CONSTRAINT\\_DB\\_RND\\_BRACK](#) (base i pairs base j)

pass a collection of options as one value like this:

```
vrna_message_constraints(option_1 | option_2 | option_n)
```

#### See also

[vrna\\_message\\_constraint\\_options\\_all\(\)](#), [vrna\\_constraints\\_add\(\)](#), [VRNA\\_CONSTRAINT\\_DB](#), [VRNA\\_CONSTRAINT\\_DB\\_PIPE](#),  
[VRNA\\_CONSTRAINT\\_DB\\_DOT](#), [VRNA\\_CONSTRAINT\\_DB\\_X](#), [VRNA\\_CONSTRAINT\\_DB\\_ANG\\_BRACK](#),  
[VRNA\\_CONSTRAINT\\_DB\\_RND\\_BRACK](#), [VRNA\\_CONSTRAINT\\_DB\\_INTERMOL](#), [VRNA\\_CONSTRAINT\\_DB\\_INTRAMOL](#)

## Parameters

<i>option</i>	Option switch that tells which constraint help will be printed
---------------	--

15.11.3.3 `vrna_message_constraint_options_all()`

```
void vrna_message_constraint_options_all (  
    void )
```

```
#include <ViennaRNA/constraints/hard.h>
```

Print structure constraint characters to stdout (full constraint support)

## See also

[vrna\\_message\\_constraint\\_options\(\)](#), [vrna\\_constraints\\_add\(\)](#), [VRNA\\_CONSTRAINT\\_DB](#), [VRNA\\_CONSTRAINT\\_DB\\_PIPE](#), [VRNA\\_CONSTRAINT\\_DB\\_DOT](#), [VRNA\\_CONSTRAINT\\_DB\\_X](#), [VRNA\\_CONSTRAINT\\_DB\\_ANG\\_BRACK](#), [VRNA\\_CONSTRAINT\\_DB\\_RND\\_BRACK](#), [VRNA\\_CONSTRAINT\\_DB\\_INTERMOL](#), [VRNA\\_CONSTRAINT\\_DB\\_INTRAMOL](#)

## 15.12 Hard Constraints

This module covers all functionality for hard constraints in secondary structure prediction.

### 15.12.1 Detailed Description

This module covers all functionality for hard constraints in secondary structure prediction.

Collaboration diagram for Hard Constraints:

#### Files

- file [hard.h](#)  
*Functions and data structures for handling of secondary structure hard constraints.*

#### Data Structures

- struct [vrna\\_hc\\_s](#)  
*The hard constraints data structure. [More...](#)*
- struct [vrna\\_hc\\_up\\_s](#)  
*A single hard constraint for a single nucleotide. [More...](#)*

#### Macros

- `#define VRNA_CONSTRAINT_DB 16384U`  
*Flag for [vrna\\_constraints\\_add\(\)](#) to indicate that constraint is passed in pseudo dot-bracket notation.*
- `#define VRNA_CONSTRAINT_DB_ENFORCE_BP 32768U`  
*Switch for dot-bracket structure constraint to enforce base pairs.*
- `#define VRNA_CONSTRAINT_DB_PIPE 65536U`  
*Flag that is used to indicate the pipe '|' sign in pseudo dot-bracket notation of hard constraints.*
- `#define VRNA_CONSTRAINT_DB_DOT 131072U`  
*dot '.' switch for structure constraints (no constraint at all)*
- `#define VRNA_CONSTRAINT_DB_X 262144U`  
*'x' switch for structure constraint (base must not pair)*
- `#define VRNA_CONSTRAINT_DB_RND_BRACK 1048576U`  
*round brackets '(',')' switch for structure constraint (base i pairs base j)*
- `#define VRNA_CONSTRAINT_DB_INTRAMOL 2097152U`  
*Flag that is used to indicate the character 'I' in pseudo dot-bracket notation of hard constraints.*
- `#define VRNA_CONSTRAINT_DB_INTERMOL 4194304U`  
*Flag that is used to indicate the character 'e' in pseudo dot-bracket notation of hard constraints.*
- `#define VRNA_CONSTRAINT_DB_GQUAD 8388608U`  
*'+' switch for structure constraint (base is involved in a gquad)*
- `#define VRNA_CONSTRAINT_DB_WUSS 33554432U`  
*Flag to indicate Washington University Secondary Structure (WUSS) notation of the hard constraint string.*
- `#define VRNA_CONSTRAINT_DB_DEFAULT`

*Switch for dot-bracket structure constraint with default symbols.*

- `#define VRNA_CONSTRAINT_CONTEXT_EXT_LOOP` (unsigned char)0x01  
*Hard constraints flag, base pair in the exterior loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_HP_LOOP` (unsigned char)0x02  
*Hard constraints flag, base pair encloses hairpin loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_INT_LOOP` (unsigned char)0x04  
*Hard constraints flag, base pair encloses an interior loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_INT_LOOP_ENC` (unsigned char)0x08  
*Hard constraints flag, base pair encloses a multi branch loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_MB_LOOP` (unsigned char)0x10  
*Hard constraints flag, base pair is enclosed in an interior loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_MB_LOOP_ENC` (unsigned char)0x20  
*Hard constraints flag, base pair is enclosed in a multi branch loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_ALL_LOOPS`  
*Constraint context flag indicating any loop context.*

## Typedefs

- `typedef struct vrna_hc_s vrna_hc_t`  
*Typename for the hard constraints data structure `vrna_hc_s`.*
- `typedef struct vrna_hc_up_s vrna_hc_up_t`  
*Typename for the single nucleotide hard constraint data structure `vrna_hc_up_s`.*
- `typedef unsigned char() vrna_callback_hc_evaluate(int i, int j, int k, int l, unsigned char d, void *data)`  
*Callback to evaluate whether or not a particular decomposition step is contributing to the solution space.*

## Functions

- `void vrna_hc_init(vrna_fold_compound_t *vc)`  
*Initialize/Reset hard constraints to default values.*
- `void vrna_hc_add_up(vrna_fold_compound_t *vc, int i, unsigned char option)`  
*Make a certain nucleotide unpaired.*
- `int vrna_hc_add_up_batch(vrna_fold_compound_t *vc, vrna_hc_up_t *constraints)`  
*Apply a list of hard constraints for single nucleotides.*
- `void vrna_hc_add_bp(vrna_fold_compound_t *vc, int i, int j, unsigned char option)`  
*Favorize/Enforce a certain base pair (i,j)*
- `void vrna_hc_add_bp_nonspecific(vrna_fold_compound_t *vc, int i, int d, unsigned char option)`  
*Enforce a nucleotide to be paired (upstream/downstream)*
- `void vrna_hc_free(vrna_hc_t *hc)`  
*Free the memory allocated by a `vrna_hc_t` data structure.*
- `int vrna_hc_add_from_db(vrna_fold_compound_t *vc, const char *constraint, unsigned int options)`  
*Add hard constraints from pseudo dot-bracket notation.*

## 15.12.2 Data Structure Documentation

### 15.12.2.1 struct vrna\_hc\_s

The hard constraints data structure.

The content of this data structure determines the decomposition pattern used in the folding recursions. Attribute 'matrix' is used as source for the branching pattern of the decompositions during all folding recursions. Any entry in matrix[i,j] consists of the 6 LSB that allows one to distinguish the following types of base pairs:

- in the exterior loop ([VRNA\\_CONSTRAINT\\_CONTEXT\\_EXT\\_LOOP](#))
- enclosing a hairpin ([VRNA\\_CONSTRAINT\\_CONTEXT\\_HP\\_LOOP](#))
- enclosing an interior loop ([VRNA\\_CONSTRAINT\\_CONTEXT\\_INT\\_LOOP](#))
- enclosed by an exterior loop ([VRNA\\_CONSTRAINT\\_CONTEXT\\_INT\\_LOOP\\_ENC](#))
- enclosing a multi branch loop ([VRNA\\_CONSTRAINT\\_CONTEXT\\_MB\\_LOOP](#))
- enclosed by a multi branch loop ([VRNA\\_CONSTRAINT\\_CONTEXT\\_MB\\_LOOP\\_ENC](#))

The four linear arrays 'up\_XXX' provide the number of available unpaired nucleotides (including position i) 3' of each position in the sequence.

See also

[vrna\\_hc\\_init\(\)](#), [vrna\\_hc\\_free\(\)](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_EXT\\_LOOP](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_HP\\_LOOP](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_INT\\_LOOP](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_MB\\_LOOP](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_MB\\_LOOP\\_ENC](#)

Collaboration diagram for vrna\_hc\_s:

#### Data Fields

- int \* [up\\_ext](#)  
A linear array that holds the number of allowed unpaired nucleotides in an exterior loop.
- int \* [up\\_hp](#)  
A linear array that holds the number of allowed unpaired nucleotides in a hairpin loop.
- int \* [up\\_int](#)  
A linear array that holds the number of allowed unpaired nucleotides in an interior loop.
- int \* [up\\_ml](#)  
A linear array that holds the number of allowed unpaired nucleotides in a multi branched loop.
- [vrna\\_callback\\_hc\\_evaluate](#) \* f  
A function pointer that returns whether or not a certain decomposition may be evaluated.
- void \* [data](#)  
A pointer to some structure where the user may store necessary data to evaluate its generic hard constraint function.
- [vrna\\_callback\\_free\\_auxdata](#) \* [free\\_data](#)  
A pointer to a function to free memory occupied by auxiliary data.
- unsigned char \* [matrix](#)  
Upper triangular matrix that encodes where a base pair or unpaired nucleotide is allowed.

### 15.12.2.1.1 Field Documentation

#### 15.12.2.1.1.1 free\_data

```
vrna_callback_free_auxdata* vrna_hc_s::free_data
```

A pointer to a function to free memory occupied by auxiliary data.

The function this pointer is pointing to will be called upon destruction of the [vrna\\_hc\\_s](#), and provided with the [vrna\\_hc\\_s.data](#) pointer that may hold auxiliary data. Hence, to avoid leaking memory, the user may use this pointer to free memory occupied by auxiliary data.

### 15.12.2.2 struct vrna\_hc\_up\_s

A single hard constraint for a single nucleotide.

#### Data Fields

- int [position](#)  
*The sequence position (1-based)*
- unsigned char [options](#)  
*The hard constraint option.*

## 15.12.3 Macro Definition Documentation

### 15.12.3.1 VRNA\_CONSTRAINT\_DB

```
#define VRNA_CONSTRAINT_DB 16384U  
  
#include <ViennaRNA/constraints/hard.h>
```

Flag for [vrna\\_constraints\\_add\(\)](#) to indicate that constraint is passed in pseudo dot-bracket notation.

#### See also

[vrna\\_constraints\\_add\(\)](#), [vrna\\_message\\_constraint\\_options\(\)](#), [vrna\\_message\\_constraint\\_options\\_all\(\)](#)

### 15.12.3.2 VRNA\_CONSTRAINT\_DB\_ENFORCE\_BP

```
#define VRNA_CONSTRAINT_DB_ENFORCE_BP 32768U  
  
#include <ViennaRNA/constraints/hard.h>
```

Switch for dot-bracket structure constraint to enforce base pairs.

This flag should be used to really enforce base pairs given in dot-bracket constraint rather than just weakly-enforcing them.

See also

[vrna\\_hc\\_add\\_from\\_db\(\)](#), [vrna\\_constraints\\_add\(\)](#), [vrna\\_message\\_constraint\\_options\(\)](#), [vrna\\_message\\_constraint\\_options\\_all\(\)](#)

### 15.12.3.3 VRNA\_CONSTRAINT\_DB\_PIPE

```
#define VRNA_CONSTRAINT_DB_PIPE 65536U  
  
#include <ViennaRNA/constraints/hard.h>
```

Flag that is used to indicate the pipe '|' sign in pseudo dot-bracket notation of hard constraints.

Use this definition to indicate the pipe sign '|' (paired with another base)

See also

[vrna\\_hc\\_add\\_from\\_db\(\)](#), [vrna\\_constraints\\_add\(\)](#), [vrna\\_message\\_constraint\\_options\(\)](#), [vrna\\_message\\_constraint\\_options\\_all\(\)](#)

### 15.12.3.4 VRNA\_CONSTRAINT\_DB\_DOT

```
#define VRNA_CONSTRAINT_DB_DOT 131072U  
  
#include <ViennaRNA/constraints/hard.h>
```

dot '.' switch for structure constraints (no constraint at all)

See also

[vrna\\_hc\\_add\\_from\\_db\(\)](#), [vrna\\_constraints\\_add\(\)](#), [vrna\\_message\\_constraint\\_options\(\)](#), [vrna\\_message\\_constraint\\_options\\_all\(\)](#)



### 15.12.3.5 VRNA\_CONSTRAINT\_DB\_X

```
#define VRNA_CONSTRAINT_DB_X 262144U
```

```
#include <ViennaRNA/constraints/hard.h>
```

'x' switch for structure constraint (base must not pair)

See also

[vrna\\_hc\\_add\\_from\\_db\(\)](#), [vrna\\_constraints\\_add\(\)](#), [vrna\\_message\\_constraint\\_options\(\)](#), [vrna\\_message\\_constraint\\_options\\_all\(\)](#)

### 15.12.3.6 VRNA\_CONSTRAINT\_DB\_RND\_BRACK

```
#define VRNA_CONSTRAINT_DB_RND_BRACK 1048576U
```

```
#include <ViennaRNA/constraints/hard.h>
```

round brackets '(',')' switch for structure constraint (base i pairs base j)

See also

[vrna\\_hc\\_add\\_from\\_db\(\)](#), [vrna\\_constraints\\_add\(\)](#), [vrna\\_message\\_constraint\\_options\(\)](#), [vrna\\_message\\_constraint\\_options\\_all\(\)](#)

### 15.12.3.7 VRNA\_CONSTRAINT\_DB\_INTRAMOL

```
#define VRNA_CONSTRAINT_DB_INTRAMOL 2097152U
```

```
#include <ViennaRNA/constraints/hard.h>
```

Flag that is used to indicate the character 'I' in pseudo dot-bracket notation of hard constraints.

Use this definition to indicate the usage of 'I' character (intramolecular pairs only)

See also

[vrna\\_hc\\_add\\_from\\_db\(\)](#), [vrna\\_constraints\\_add\(\)](#), [vrna\\_message\\_constraint\\_options\(\)](#), [vrna\\_message\\_constraint\\_options\\_all\(\)](#)

### 15.12.3.8 VRNA\_CONSTRAINT\_DB\_INTERMOL

```
#define VRNA_CONSTRAINT_DB_INTERMOL 4194304U
```

```
#include <ViennaRNA/constraints/hard.h>
```

Flag that is used to indicate the character 'e' in pseudo dot-bracket notation of hard constraints.

Use this definition to indicate the usage of 'e' character (intermolecular pairs only)

#### See also

[vrna\\_hc\\_add\\_from\\_db\(\)](#), [vrna\\_constraints\\_add\(\)](#), [vrna\\_message\\_constraint\\_options\(\)](#), [vrna\\_message\\_constraint\\_options\\_all\(\)](#)

### 15.12.3.9 VRNA\_CONSTRAINT\_DB\_GQUAD

```
#define VRNA_CONSTRAINT_DB_GQUAD 8388608U
```

```
#include <ViennaRNA/constraints/hard.h>
```

'+' switch for structure constraint (base is involved in a gquad)

#### See also

[vrna\\_hc\\_add\\_from\\_db\(\)](#), [vrna\\_constraints\\_add\(\)](#), [vrna\\_message\\_constraint\\_options\(\)](#), [vrna\\_message\\_constraint\\_options\\_all\(\)](#)

#### Warning

This flag is for future purposes only! No implementation recognizes it yet.

### 15.12.3.10 VRNA\_CONSTRAINT\_DB\_WUSS

```
#define VRNA_CONSTRAINT_DB_WUSS 33554432U
```

```
#include <ViennaRNA/constraints/hard.h>
```

Flag to indicate Washington University Secondary Structure (WUSS) notation of the hard constraint string.

This secondary structure notation for RNAs is usually used as consensus secondary structure (SS\_cons) entry in Stockholm formatted files

## 15.12.3.11 VRNA\_CONSTRAINT\_DB\_DEFAULT

```
#define VRNA_CONSTRAINT_DB_DEFAULT
```

```
#include <ViennaRNA/constraints/hard.h>
```

**Value:**

```
(VRNA_CONSTRAINT_DB \
 | VRNA_CONSTRAINT_DB_PIPE \
 | VRNA_CONSTRAINT_DB_DOT \
 | VRNA_CONSTRAINT_DB_X \
 | VRNA_CONSTRAINT_DB_ANG_BRACK \
 | VRNA_CONSTRAINT_DB_RND_BRACK \
 | VRNA_CONSTRAINT_DB_INTRAMOL \
 | VRNA_CONSTRAINT_DB_INTERMOL \
 | VRNA_CONSTRAINT_DB_GQUAD \
)
```

Switch for dot-bracket structure constraint with default symbols.

This flag conveniently combines all possible symbols in dot-bracket notation for hard constraints and [VRNA\\_CONSTRAINT\\_DB](#)

See also

[vrna\\_hc\\_add\\_from\\_db\(\)](#), [vrna\\_constraints\\_add\(\)](#), [vrna\\_message\\_constraint\\_options\(\)](#), [vrna\\_message\\_constraint\\_options\\_all\(\)](#)

## 15.12.4 Typedef Documentation

## 15.12.4.1 vrna\_callback\_hc\_evaluate

```
typedef unsigned char() vrna_callback_hc_evaluate(int i, int j, int k, int l, unsigned char d,
void *data)
```

```
#include <ViennaRNA/constraints/hard.h>
```

Callback to evaluate whether or not a particular decomposition step is contributing to the solution space.

This is the prototype for callback functions used by the folding recursions to evaluate generic hard constraints. The first four parameters passed indicate the delimiting nucleotide positions of the decomposition, and the parameter `denotes` the decomposition step. The last parameter `data` is the auxiliary data structure associated to the hard constraints via [vrna\\_hc\\_add\\_data\(\)](#), or NULL if no auxiliary data was added.

**Notes on Callback Functions** This callback enables one to over-rule default hard constraints in secondary structure decompositions.

See also

[VRNA\\_DECOMP\\_PAIR\\_HP](#), [VRNA\\_DECOMP\\_PAIR\\_IL](#), [VRNA\\_DECOMP\\_PAIR\\_ML](#), [VRNA\\_DECOMP\\_ML\\_ML\\_ML](#), [VRNA\\_DECOMP\\_ML\\_STEM](#), [VRNA\\_DECOMP\\_ML\\_ML](#), [VRNA\\_DECOMP\\_ML\\_UP](#), [VRNA\\_DECOMP\\_ML\\_ML\\_STEM](#), [VRNA\\_DECOMP\\_ML\\_COAXIAL](#), [VRNA\\_DECOMP\\_EXT\\_EXT](#), [VRNA\\_DECOMP\\_EXT\\_UP](#), [VRNA\\_DECOMP\\_EXT\\_STEM](#), [VRNA\\_DECOMP\\_EXT\\_EXT\\_EXT](#), [VRNA\\_DECOMP\\_EXT\\_STEM\\_EXT](#), [VRNA\\_DECOMP\\_EXT\\_EXT\\_STEM](#), [VRNA\\_DECOMP\\_EXT\\_EXT\\_STEM1](#), [vrna\\_hc\\_add\\_f\(\)](#), [vrna\\_hc\\_add\\_data\(\)](#)

**Parameters**

<i>i</i>	Left (5') delimiter position of substructure
<i>j</i>	Right (3') delimiter position of substructure
<i>k</i>	Left delimiter of decomposition
<i>l</i>	Right delimiter of decomposition
<i>d</i>	Decomposition step indicator
<i>data</i>	Auxiliary data

**Returns**

A non-zero value if the decomposition is valid, 0 otherwise

**15.12.5 Function Documentation****15.12.5.1 vrna\_hc\_init()**

```
void vrna_hc_init (
    vrna_fold_compound_t * vc )

#include <ViennaRNA/constraints/hard.h>
```

Initialize/Reset hard constraints to default values.

This function resets the hard constraints to their default values, i.e. all positions may be unpaired in all contexts, and base pairs are allowed in all contexts, if they resemble canonical pairs. Previously set hard constraints will be removed before initialization.

**See also**

[vrna\\_hc\\_add\\_bp\(\)](#), [vrna\\_hc\\_add\\_bp\\_nonspecific\(\)](#), [vrna\\_hc\\_add\\_up\(\)](#)

**Parameters**

<i>vc</i>	The fold compound
-----------	-------------------

**SWIG Wrapper Notes** This function is attached as method **hc\_init()** to objects of type *fold\_compound*

**15.12.5.2 vrna\_hc\_add\_up()**

```
void vrna_hc_add_up (
    vrna_fold_compound_t * vc,
```

```

        int i,
        unsigned char option )

#include <ViennaRNA/constraints/hard.h>

```

Make a certain nucleotide unpaired.

See also

[vrna\\_hc\\_add\\_bp\(\)](#), [vrna\\_hc\\_add\\_bp\\_nonspecific\(\)](#), [vrna\\_hc\\_init\(\)](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_EXT\\_LOOP](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_HP\\_LOOP](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_INT\\_LOOP](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_ALL\\_LOOPS](#)

#### Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> the hard constraints are associated with
<i>i</i>	The position that needs to stay unpaired (1-based)
<i>option</i>	The options flag indicating how/where to store the hard constraints

#### 15.12.5.3 vrna\_hc\_add\_up\_batch()

```

int vrna_hc_add_up_batch (
    vrna_fold_compound_t * vc,
    vrna_hc_up_t * constraints )

#include <ViennaRNA/constraints/hard.h>

```

Apply a list of hard constraints for single nucleotides.

#### Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> the hard constraints are associated with
<i>constraints</i>	The list off constraints to apply, last entry must have position attribute set to 0

#### 15.12.5.4 vrna\_hc\_add\_bp()

```

void vrna_hc_add_bp (
    vrna_fold_compound_t * vc,
    int i,
    int j,
    unsigned char option )

#include <ViennaRNA/constraints/hard.h>

```

Favorize/Enforce a certain base pair (i,j)

## See also

[vrna\\_hc\\_add\\_bp\\_nonspecific\(\)](#), [vrna\\_hc\\_add\\_up\(\)](#), [vrna\\_hc\\_init\(\)](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_EXT\\_LOOP](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_HP\\_LOOP](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_INT\\_LOOP](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_MB\\_LOOP](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_MB\\_LOOP\\_ENC](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_ENFORCE](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_ALL\\_LOOPS](#)

## Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> the hard constraints are associated with
<i>i</i>	The 5' located nucleotide position of the base pair (1-based)
<i>j</i>	The 3' located nucleotide position of the base pair (1-based)
<i>option</i>	The options flag indicating how/where to store the hard constraints

15.12.5.5 `vrna_hc_add_bp_nonspecific()`

```
void vrna_hc_add_bp_nonspecific (
    vrna_fold_compound_t * vc,
    int i,
    int d,
    unsigned char option )
```

```
#include <ViennaRNA/constraints/hard.h>
```

Enforce a nucleotide to be paired (upstream/downstream)

## See also

[vrna\\_hc\\_add\\_bp\(\)](#), [vrna\\_hc\\_add\\_up\(\)](#), [vrna\\_hc\\_init\(\)](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_EXT\\_LOOP](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_HP\\_LOOP](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_INT\\_LOOP](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_MB\\_LOOP](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_MB\\_LOOP\\_ENC](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_ALL\\_LOOPS](#)

## Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> the hard constraints are associated with
<i>i</i>	The position that needs to stay unpaired (1-based)
<i>d</i>	The direction of base pairing ( $d < 0$ : pairs upstream, $d > 0$ : pairs downstream, $d == 0$ : no direction)
<i>option</i>	The options flag indicating in which loop type context the pairs may appear

15.12.5.6 `vrna_hc_free()`

```
void vrna_hc_free (
    vrna_hc_t * hc )
```

```
#include <ViennaRNA/constraints/hard.h>
```

Free the memory allocated by a `vrna_hc_t` data structure.

Use this function to free all memory that was allocated for a data structure of type `vrna_hc_t`.

See also

`get_hard_constraints()`, `vrna_hc_t`

#### 15.12.5.7 `vrna_hc_add_from_db()`

```
int vrna_hc_add_from_db (
    vrna_fold_compound_t * vc,
    const char * constraint,
    unsigned int options )

#include <ViennaRNA/constraints/hard.h>
```

Add hard constraints from pseudo dot-bracket notation.

This function allows one to apply hard constraints from a pseudo dot-bracket notation. The `options` parameter controls, which characters are recognized by the parser. Use the `VRNA_CONSTRAINT_DB_DEFAULT` convenience macro, if you want to allow all known characters

See also

`VRNA_CONSTRAINT_DB_PIPE`, `VRNA_CONSTRAINT_DB_DOT`, `VRNA_CONSTRAINT_DB_X`, `VRNA_CONSTRAINT_DB_`  
`VRNA_CONSTRAINT_DB_RND_BRACK`, `VRNA_CONSTRAINT_DB_INTRAMOL`, `VRNA_CONSTRAINT_DB_INTERMOL`,  
`VRNA_CONSTRAINT_DB_GQUAD`

#### Parameters

<code>vc</code>	The fold compound
<code>constraint</code>	A pseudo dot-bracket notation of the hard constraint.
<code>options</code>	The option flags

**SWIG Wrapper Notes** This function is attached as method `hc_add_from_db()` to objects of type `fold_compound`

## 15.13 Soft Constraints

Functions and data structures for secondary structure soft constraints.

### 15.13.1 Detailed Description

Functions and data structures for secondary structure soft constraints.

Soft-constraints are used to change position specific contributions in the recursions by adding bonuses/penalties in form of pseudo free energies to certain loop configurations. Collaboration diagram for Soft Constraints:

#### Files

- file [soft.h](#)

*Functions and data structures for secondary structure soft constraints.*

#### Data Structures

- struct [vrna\\_sc\\_s](#)

*The soft constraints data structure. [More...](#)*

#### Typedefs

- typedef struct [vrna\\_sc\\_s](#) [vrna\\_sc\\_t](#)  
*Typename for the soft constraints data structure [vrna\\_sc\\_s](#).*
- typedef int() [vrna\\_callback\\_sc\\_energy](#)(int i, int j, int k, int l, unsigned char d, void \*data)  
*Callback to retrieve pseudo energy contribution for soft constraint feature.*
- typedef [FLT\\_OR\\_DBL](#)() [vrna\\_callback\\_sc\\_exp\\_energy](#)(int i, int j, int k, int l, unsigned char d, void \*data)  
*Callback to retrieve pseudo energy contribution as Boltzmann Factors for soft constraint feature.*
- typedef [vrna\\_basepair\\_t](#) \*() [vrna\\_callback\\_sc\\_backtrack](#)(int i, int j, int k, int l, unsigned char d, void \*data)  
*Callback to retrieve auxiliary base pairs for soft constraint feature.*

#### Functions

- void [vrna\\_sc\\_init](#) ([vrna\\_fold\\_compound\\_t](#) \*vc)  
*Initialize an empty soft constraints data structure within a [vrna\\_fold\\_compound\\_t](#).*
- void [vrna\\_sc\\_set\\_bp](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const [FLT\\_OR\\_DBL](#) \*\*constraints, unsigned int options)  
*Set soft constraints for paired nucleotides.*
- void [vrna\\_sc\\_add\\_bp](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int i, int j, [FLT\\_OR\\_DBL](#) energy, unsigned int options)  
*Add soft constraints for paired nucleotides.*
- void [vrna\\_sc\\_set\\_up](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const [FLT\\_OR\\_DBL](#) \*constraints, unsigned int options)  
*Set soft constraints for unpaired nucleotides.*
- void [vrna\\_sc\\_add\\_up](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int i, [FLT\\_OR\\_DBL](#) energy, unsigned int options)  
*Add soft constraints for unpaired nucleotides.*
- void [vrna\\_sc\\_remove](#) ([vrna\\_fold\\_compound\\_t](#) \*vc)



- Remove soft constraints from [vrna\\_fold\\_compound\\_t](#).
- void [vrna\\_sc\\_free](#) ([vrna\\_sc\\_t](#) \*sc)  
Free memory occupied by a [vrna\\_sc\\_t](#) data structure.
- void [vrna\\_sc\\_add\\_data](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, void \*data, [vrna\\_callback\\_free\\_auxdata](#) \*free\_data)  
Add an auxiliary data structure for the generic soft constraints callback function.
- void [vrna\\_sc\\_add\\_f](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_callback\\_sc\\_energy](#) \*f)  
Bind a function pointer for generic soft constraint feature (MFE version)
- void [vrna\\_sc\\_add\\_bt](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_callback\\_sc\\_backtrack](#) \*f)  
Bind a backtracking function pointer for generic soft constraint feature.
- void [vrna\\_sc\\_add\\_exp\\_f](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_callback\\_sc\\_exp\\_energy](#) \*exp\_f)  
Bind a function pointer for generic soft constraint feature (PF version)

## 15.13.2 Data Structure Documentation

### 15.13.2.1 struct [vrna\\_sc\\_s](#)

The soft constraints data structure.

Collaboration diagram for [vrna\\_sc\\_s](#):

#### Data Fields

- int \*\* [energy\\_up](#)  
Energy contribution for stretches of unpaired nucleotides.
- [FLT\\_OR\\_DBL](#) \*\* [exp\\_energy\\_up](#)  
Boltzmann Factors of the energy contributions for unpaired sequence stretches.
- int \* [up\\_storage](#)  
Storage container for energy contributions per unpaired nucleotide.
- [vrna\\_sc\\_bp\\_storage\\_t](#) \*\* [bp\\_storage](#)  
Storage container for energy contributions per base pair.
- int \* [energy\\_stack](#)  
Pseudo Energy contribution per base pair involved in a stack.
- [FLT\\_OR\\_DBL](#) \* [exp\\_energy\\_stack](#)  
Boltzmann weighted pseudo energy contribution per nucleotide involved in a stack.
- [vrna\\_callback\\_sc\\_energy](#) \* f  
A function pointer used for pseudo energy contribution in MFE calculations.
- [vrna\\_callback\\_sc\\_backtrack](#) \* bt  
A function pointer used to obtain backtraced base pairs in loop regions that were altered by soft constrained pseudo energy contributions.
- [vrna\\_callback\\_sc\\_exp\\_energy](#) \* exp\_f  
A function pointer used for pseudo energy contribution boltzmann factors in PF calculations.
- void \* [data](#)  
A pointer to the data object provided for for pseudo energy contribution functions of the generic soft constraints feature.
- int \* [energy\\_bp](#)  
Energy contribution for base pairs.
- [FLT\\_OR\\_DBL](#) \* [exp\\_energy\\_bp](#)  
Boltzmann Factors of the energy contribution for base pairs.
- int \*\* [energy\\_bp\\_local](#)  
Energy contribution for base pairs (sliding window approach)
- [FLT\\_OR\\_DBL](#) \*\* [exp\\_energy\\_bp\\_local](#)  
Boltzmann Factors of the energy contribution for base pairs (sliding window approach)

### 15.13.2.1.1 Field Documentation

#### 15.13.2.1.1.1 f

`vrna_callback_sc_energy* vrna_sc_s::f`

A function pointer used for pseudo energy contribution in MFE calculations.

See also

[vrna\\_sc\\_add\\_f\(\)](#)

#### 15.13.2.1.1.2 bt

`vrna_callback_sc_backtrack* vrna_sc_s::bt`

A function pointer used to obtain backtraced base pairs in loop regions that were altered by soft constrained pseudo energy contributions.

See also

[vrna\\_sc\\_add\\_bt\(\)](#)

#### 15.13.2.1.1.3 exp\_f

`vrna_callback_sc_exp_energy* vrna_sc_s::exp_f`

A function pointer used for pseudo energy contribution boltzmann factors in PF calculations.

See also

[vrna\\_sc\\_add\\_exp\\_f\(\)](#)

## 15.13.3 Typedef Documentation

## 15.13.3.1 vrna\_callback\_sc\_energy

```
typedef int() vrna_callback_sc_energy(int i, int j, int k, int l, unsigned char d, void *data)

#include <ViennaRNA/constraints/soft.h>
```

Callback to retrieve pseudo energy contribution for soft constraint feature.

This is the prototype for callback functions used by the folding recursions to evaluate generic soft constraints. The first four parameters passed indicate the delimiting nucleotide positions of the decomposition, and the parameter `denotes` the decomposition step. The last parameter `data` is the auxiliary data structure associated to the hard constraints via [vrna\\_sc\\_add\\_data\(\)](#), or NULL if no auxiliary data was added.

**Notes on Callback Functions** This callback enables one to add (pseudo-)energy contributions to individual decompositions of the secondary structure.

See also

[VRNA\\_DECOMP\\_PAIR\\_HP](#), [VRNA\\_DECOMP\\_PAIR\\_IL](#), [VRNA\\_DECOMP\\_PAIR\\_ML](#), [VRNA\\_DECOMP\\_ML\\_ML\\_ML](#), [VRNA\\_DECOMP\\_ML\\_STEM](#), [VRNA\\_DECOMP\\_ML\\_ML](#), [VRNA\\_DECOMP\\_ML\\_UP](#), [VRNA\\_DECOMP\\_ML\\_ML\\_STEM](#), [VRNA\\_DECOMP\\_ML\\_COAXIAL](#), [VRNA\\_DECOMP\\_EXT\\_EXT](#), [VRNA\\_DECOMP\\_EXT\\_UP](#), [VRNA\\_DECOMP\\_EXT\\_STEM](#), [VRNA\\_DECOMP\\_EXT\\_EXT\\_EXT](#), [VRNA\\_DECOMP\\_EXT\\_STEM\\_EXT](#), [VRNA\\_DECOMP\\_EXT\\_EXT\\_STEM](#), [VRNA\\_DECOMP\\_EXT\\_EXT\\_STEM1](#), [vrna\\_sc\\_add\\_f\(\)](#), [vrna\\_sc\\_add\\_exp\\_f\(\)](#), [vrna\\_sc\\_add\\_bt\(\)](#), [vrna\\_sc\\_add\\_data\(\)](#)

**Parameters**

<i>i</i>	Left (5') delimiter position of substructure
<i>j</i>	Right (3') delimiter position of substructure
<i>k</i>	Left delimiter of decomposition
<i>l</i>	Right delimiter of decomposition
<i>d</i>	Decomposition step indicator
<i>data</i>	Auxiliary data

**Returns**

Pseudo energy contribution in deka-kalories per mol

**15.13.3.2 vrna\_callback\_sc\_exp\_energy**

```
typedef FLT_OR_DBL() vrna_callback_sc_exp_energy(int i, int j, int k, int l, unsigned char d,
void *data)
```

```
#include <ViennaRNA/constraints/soft.h>
```

Callback to retrieve pseudo energy contribution as Boltzmann Factors for soft constraint feature.

This is the prototype for callback functions used by the partition function recursions to evaluate generic soft constraints. The first four parameters passed indicate the delimiting nucleotide positions of the decomposition, and the parameter *denotes* the decomposition step. The last parameter *data* is the auxiliary data structure associated to the hard constraints via [vrna\\_sc\\_add\\_data\(\)](#), or NULL if no auxiliary data was added.

**Notes on Callback Functions** This callback enables one to add (pseudo-)energy contributions to individual decompositions of the secondary structure (Partition function variant, i.e. contributions must be returned as Boltzmann factors).

**See also**

[VRNA\\_DECOMP\\_PAIR\\_HP](#), [VRNA\\_DECOMP\\_PAIR\\_IL](#), [VRNA\\_DECOMP\\_PAIR\\_ML](#), [VRNA\\_DECOMP\\_ML\\_ML\\_ML](#), [VRNA\\_DECOMP\\_ML\\_STEM](#), [VRNA\\_DECOMP\\_ML\\_ML](#), [VRNA\\_DECOMP\\_ML\\_UP](#), [VRNA\\_DECOMP\\_ML\\_ML\\_STEM](#), [VRNA\\_DECOMP\\_ML\\_COAXIAL](#), [VRNA\\_DECOMP\\_EXT\\_EXT](#), [VRNA\\_DECOMP\\_EXT\\_UP](#), [VRNA\\_DECOMP\\_EXT\\_STEM](#), [VRNA\\_DECOMP\\_EXT\\_EXT\\_EXT](#), [VRNA\\_DECOMP\\_EXT\\_STEM\\_EXT](#), [VRNA\\_DECOMP\\_EXT\\_EXT\\_STEM](#), [VRNA\\_DECOMP\\_EXT\\_EXT\\_STEM1](#), [vrna\\_sc\\_add\\_exp\\_f\(\)](#), [vrna\\_sc\\_add\\_f\(\)](#), [vrna\\_sc\\_add\\_bt\(\)](#), [vrna\\_sc\\_add\\_data\(\)](#)

**Parameters**

<i>i</i>	Left (5') delimiter position of substructure
<i>j</i>	Right (3') delimiter position of substructure
<i>k</i>	Left delimiter of decomposition
<i>l</i>	Right delimiter of decomposition
<i>d</i>	Decomposition step indicator
<i>data</i>	Auxiliary data

**Returns**

Pseudo energy contribution in deka-kalories per mol

**15.13.3.3 vrna\_callback\_sc\_backtrack**

```
typedef vrna_basepair_t*() vrna_callback_sc_backtrack(int i, int j, int k, int l, unsigned
char d, void *data)
```

```
#include <ViennaRNA/constraints/soft.h>
```

Callback to retrieve auxiliary base pairs for soft constraint feature.

**Notes on Callback Functions** This callback enables one to add auxiliary base pairs in the backtracking steps of hairpin- and interior loops.

**See also**

VRNA\_DECOMP\_PAIR\_HP, VRNA\_DECOMP\_PAIR\_IL, VRNA\_DECOMP\_PAIR\_ML, VRNA\_DECOMP\_ML\_ML\_ML,  
 VRNA\_DECOMP\_ML\_STEM, VRNA\_DECOMP\_ML\_ML, VRNA\_DECOMP\_ML\_UP, VRNA\_DECOMP\_ML\_ML\_STEM,  
 VRNA\_DECOMP\_ML\_COAXIAL, VRNA\_DECOMP\_EXT\_EXT, VRNA\_DECOMP\_EXT\_UP, VRNA\_DECOMP\_EXT\_STEM,  
 VRNA\_DECOMP\_EXT\_EXT\_EXT, VRNA\_DECOMP\_EXT\_STEM\_EXT, VRNA\_DECOMP\_EXT\_EXT\_STEM,  
 VRNA\_DECOMP\_EXT\_EXT\_STEM1, vrna\_sc\_add\_bt(), vrna\_sc\_add\_f(), vrna\_sc\_add\_exp\_f(), vrna\_sc\_add\_data()

**Parameters**

<i>i</i>	Left (5') delimiter position of substructure
<i>j</i>	Right (3') delimiter position of substructure
<i>k</i>	Left delimiter of decomposition
<i>l</i>	Right delimiter of decomposition
<i>d</i>	Decomposition step indicator
<i>data</i>	Auxiliary data

**Returns**

List of additional base pairs

**15.13.4 Function Documentation****15.13.4.1 vrna\_sc\_init()**

```
void vrna_sc_init (
    vrna_fold_compound_t * vc )
```

```
#include <ViennaRNA/constraints/soft.h>
```

Initialize an empty soft constraints data structure within a `vrna_fold_compound_t`.

This function adds a proper soft constraints data structure to the `vrna_fold_compound_t` data structure. If soft constraints already exist within the fold compound, they are removed.

#### Note

Accepts `vrna_fold_compound_t` of type `VRNA_FC_TYPE_SINGLE` and `VRNA_FC_TYPE_COMPARATIVE`

#### See also

`vrna_sc_set_bp()`, `vrna_sc_set_up()`, `vrna_sc_add_SHAPE_deigan()`, `vrna_sc_add_SHAPE_zarringham()`, `vrna_sc_remove()`, `vrna_sc_add_f()`, `vrna_sc_add_exp_f()`, `vrna_sc_add_pre()`, `vrna_sc_add_post()`

#### Parameters

<code>vc</code>	The <code>vrna_fold_compound_t</code> where an empty soft constraint feature is to be added to
-----------------	--

**SWIG Wrapper Notes** This function is attached as method `sc_init()` to objects of type `fold_compound`

#### 15.13.4.2 `vrna_sc_set_bp()`

```
void vrna_sc_set_bp (
    vrna_fold_compound_t * vc,
    const FLT_OR_DBL ** constraints,
    unsigned int options )
```

```
#include <ViennaRNA/constraints/soft.h>
```

Set soft constraints for paired nucleotides.

#### Note

This function replaces any pre-existing soft constraints with the ones supplied in `constraints`.

#### See also

`vrna_sc_add_bp()`, `vrna_sc_set_up()`, `vrna_sc_add_up()`

#### Parameters

<code>vc</code>	The <code>vrna_fold_compound_t</code> the soft constraints are associated with
<code>constraints</code>	A two-dimensional array of pseudo free energies in <i>kcal/mol</i>
<code>options</code>	The options flag indicating how/where to store the soft constraints

**SWIG Wrapper Notes** This function is attached as method **sc\_set\_bp()** to objects of type *fold\_compound*

#### 15.13.4.3 vrna\_sc\_add\_bp()

```
void vrna_sc_add_bp (
    vrna_fold_compound_t * vc,
    int i,
    int j,
    FLT_OR_DBL energy,
    unsigned int options )
```

```
#include <ViennaRNA/constraints/soft.h>
```

Add soft constraints for paired nucleotides.

See also

[vrna\\_sc\\_set\\_bp\(\)](#), [vrna\\_sc\\_set\\_up\(\)](#), [vrna\\_sc\\_add\\_up\(\)](#)

#### Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> the soft constraints are associated with
<i>i</i>	The 5' position of the base pair the soft constraint is added for
<i>j</i>	The 3' position of the base pair the soft constraint is added for
<i>energy</i>	The free energy (soft-constraint) in <i>kcal/mol</i>
<i>options</i>	The options flag indicating how/where to store the soft constraints

**SWIG Wrapper Notes** This function is attached as an overloaded method **sc\_add\_bp()** to objects of type *fold\_compound*. The method either takes arguments for a single base pair (i,j) with the corresponding energy value:

```
fold_compound.sc_add_bp(i, j, energy, options)
```

or an entire 2-dimensional matrix with dimensions  $n \times n$  that stores free energy contributions for any base pair (i,j) with  $1 \leq i < j \leq n$ :

```
fold_compound.sc_add_bp(matrix, options)
```

In both variants, the *options* argument is optional can may be omitted.

#### 15.13.4.4 vrna\_sc\_set\_up()

```
void vrna_sc_set_up (
    vrna_fold_compound_t * vc,
    const FLT_OR_DBL * constraints,
    unsigned int options )
```

```
#include <ViennaRNA/constraints/soft.h>
```

Set soft constraints for unpaired nucleotides.

**Note**

This function replaces any pre-existing soft constraints with the ones supplied in `constraints`.

**See also**

[vrna\\_sc\\_add\\_up\(\)](#), [vrna\\_sc\\_set\\_bp\(\)](#), [vrna\\_sc\\_add\\_bp\(\)](#)

**Parameters**

<code>vc</code>	The <a href="#">vrna_fold_compound_t</a> the soft constraints are associated with
<code>constraints</code>	A vector of pseudo free energies in <i>kcal/mol</i>
<code>options</code>	The options flag indicating how/where to store the soft constraints

**SWIG Wrapper Notes** This function is attached as method `sc_set_up()` to objects of type *fold\_compound*

**15.13.4.5 vrna\_sc\_add\_up()**

```
void vrna_sc_add_up (
    vrna_fold_compound_t * vc,
    int i,
    FLT_OR_DBL energy,
    unsigned int options )
```

```
#include <ViennaRNA/constraints/soft.h>
```

Add soft constraints for unpaired nucleotides.

**See also**

[vrna\\_sc\\_set\\_up\(\)](#), [vrna\\_sc\\_add\\_bp\(\)](#), [vrna\\_sc\\_set\\_bp\(\)](#)

**Parameters**

<code>vc</code>	The <a href="#">vrna_fold_compound_t</a> the soft constraints are associated with
<code>i</code>	The nucleotide position the soft constraint is added for
<code>energy</code>	The free energy (soft-constraint) in <i>kcal/mol</i>
<code>options</code>	The options flag indicating how/where to store the soft constraints

**SWIG Wrapper Notes** This function is attached as an overloaded method `sc_add_up()` to objects of type *fold\_compound*. The method either takes arguments for a single nucleotide *i* with the corresponding energy value:

```
fold_compound.sc_add_up(i, energy, options)
```

or an entire vector that stores free energy contributions for each nucleotide *i* with  $1 \leq i \leq n$ :



```
fold_compound.sc_add_bp(vector, options)
```

In both variants, the `options` argument is optional can may be omitted.

#### 15.13.4.6 `vrna_sc_remove()`

```
void vrna_sc_remove (
    vrna_fold_compound_t * vc )

#include <ViennaRNA/constraints/soft.h>
```

Remove soft constraints from `vrna_fold_compound_t`.

##### Note

Accepts `vrna_fold_compound_t` of type `VRNA_FC_TYPE_SINGLE` and `VRNA_FC_TYPE_COMPARATIVE`

##### Parameters

<code>vc</code>	The <code>vrna_fold_compound_t</code> possibly containing soft constraints
-----------------	--

**SWIG Wrapper Notes** This function is attached as method `sc_remove()` to objects of type `fold_compound`

#### 15.13.4.7 `vrna_sc_free()`

```
void vrna_sc_free (
    vrna_sc_t * sc )

#include <ViennaRNA/constraints/soft.h>
```

Free memory occupied by a `vrna_sc_t` data structure.

##### Parameters

<code>sc</code>	The data structure to free from memory
-----------------	--

#### 15.13.4.8 `vrna_sc_add_data()`

```
void vrna_sc_add_data (
    vrna_fold_compound_t * vc,
```

```
void * data,
vrna_callback_free_auxdata * free_data )
```

```
#include <ViennaRNA/constraints/soft.h>
```

Add an auxiliary data structure for the generic soft constraints callback function.

See also

[vrna\\_sc\\_add\\_f\(\)](#), [vrna\\_sc\\_add\\_exp\\_f\(\)](#), [vrna\\_sc\\_add\\_bt\(\)](#)

#### Parameters

<i>vc</i>	The fold compound the generic soft constraint function should be bound to
<i>data</i>	A pointer to the data structure that holds required data for function 'f'
<i>free_data</i>	A pointer to a function that free's the memory occupied by <i>data</i> (Maybe NULL)

**SWIG Wrapper Notes** This function is attached as method **sc\_add\_data()** to objects of type *fold\_compound*

#### 15.13.4.9 vrna\_sc\_add\_f()

```
void vrna_sc_add_f (
    vrna_fold_compound_t * vc,
    vrna_callback_sc_energy * f )
```

```
#include <ViennaRNA/constraints/soft.h>
```

Bind a function pointer for generic soft constraint feature (MFE version)

This function allows one to easily bind a function pointer and corresponding data structure to the soft constraint part [vrna\\_sc\\_t](#) of the [vrna\\_fold\\_compound\\_t](#). The function for evaluating the generic soft constraint feature has to return a pseudo free energy  $\hat{E}$  in *dacal/mol*, where  $1\text{dacal/mol} = 10\text{cal/mol}$ .

See also

[vrna\\_sc\\_add\\_data\(\)](#), [vrna\\_sc\\_add\\_bt\(\)](#), [vrna\\_sc\\_add\\_exp\\_f\(\)](#)

#### Parameters

<i>vc</i>	The fold compound the generic soft constraint function should be bound to
<i>f</i>	A pointer to the function that evaluates the generic soft constraint feature

**SWIG Wrapper Notes** This function is attached as method **sc\_add\_f()** to objects of type *fold\_compound*

15.13.4.10 `vrna_sc_add_bt()`

```
void vrna_sc_add_bt (
    vrna_fold_compound_t * vc,
    vrna_callback_sc_backtrack * f )

#include <ViennaRNA/constraints/soft.h>
```

Bind a backtracking function pointer for generic soft constraint feature.

This function allows one to easily bind a function pointer to the soft constraint part `vrna_sc_t` of the `vrna_fold_compound_t`. The provided function should be used for backtracking purposes in loop regions that were altered via the generic soft constraint feature. It has to return an array of `vrna_basepair_t` data structures, where the last element in the list is indicated by a value of -1 in its `i` position.

See also

[vrna\\_sc\\_add\\_data\(\)](#), [vrna\\_sc\\_add\\_f\(\)](#), [vrna\\_sc\\_add\\_exp\\_f\(\)](#)

## Parameters

<code>vc</code>	The fold compound the generic soft constraint function should be bound to
<code>f</code>	A pointer to the function that returns additional base pairs

**SWIG Wrapper Notes** This function is attached as method `sc_add_bt()` to objects of type `fold_compound`

15.13.4.11 `vrna_sc_add_exp_f()`

```
void vrna_sc_add_exp_f (
    vrna_fold_compound_t * vc,
    vrna_callback_sc_exp_energy * exp_f )

#include <ViennaRNA/constraints/soft.h>
```

Bind a function pointer for generic soft constraint feature (PF version)

This function allows one to easily bind a function pointer and corresponding data structure to the soft constraint part `vrna_sc_t` of the `vrna_fold_compound_t`. The function for evaluating the generic soft constraint feature has to return a pseudo free energy  $\hat{E}$  as Boltzmann factor, i.e.  $\exp(-\hat{E}/kT)$ . The required unit for  $E$  is *cal/mol*.

See also

[vrna\\_sc\\_add\\_bt\(\)](#), [vrna\\_sc\\_add\\_f\(\)](#), [vrna\\_sc\\_add\\_data\(\)](#)

## Parameters

<code>vc</code>	The fold compound the generic soft constraint function should be bound to
<code>exp_f</code>	A pointer to the function that evaluates the generic soft constraint feature

**SWIG Wrapper Notes** This function is attached as method **sc\_add\_exp\_f()** to objects of type *fold\_compound*

## 15.14 The RNA Secondary Structure Landscape

### 15.14.1 Detailed Description

Collaboration diagram for The RNA Secondary Structure Landscape:

#### Modules

- [Neighborhood Relation and Move Sets for Secondary Structures](#)  
*Different functions to generate structural neighbors of a secondary structure according to a particular Move Set.*
- [Refolding Paths of Secondary Structures](#)

## 15.15 Minimum Free Energy (MFE) Algorithms

Predicting the Minimum Free Energy (MFE) and a corresponding (consensus) secondary structure.

### 15.15.1 Detailed Description

Predicting the Minimum Free Energy (MFE) and a corresponding (consensus) secondary structure.

In a nutshell we provide two different flavors for MFE prediction:

- [Global MFE Prediction](#) - to compute the MFE for the entire sequence
- [Local \(sliding window\) MFE Prediction](#) - to compute MFEs for each window using a sliding window approach

Each of these flavors, again, provides two implementations to either compute the MFE based on

- single RNA (DNA) sequence(s), or
- a comparative approach using multiple sequence alignments (MSA).

For the latter, a consensus secondary structure is predicted and our implementations compute an average of free energies for each sequence in the MSA plus an additional covariance pseudo-energy term.

The implementations for [Backtracking MFE structures](#) are generally agnostic with respect to whether local or global structure prediction is in place. Collaboration diagram for Minimum Free Energy (MFE) Algorithms:

### Modules

- [Global MFE Prediction](#)  
*Variations of the global Minimum Free Energy (MFE) prediction algorithm.*
- [Local \(sliding window\) MFE Prediction](#)  
*Variations of the local (sliding window) Minimum Free Energy (MFE) prediction algorithm.*
- [Backtracking MFE structures](#)  
*Backtracking related interfaces.*

### Files

- file [mfe.h](#)  
*Compute Minimum Free energy (MFE) and backtrace corresponding secondary structures from RNA sequence data.*
- file [mfe\\_window.h](#)  
*Compute local Minimum Free Energy (MFE) using a sliding window approach and backtrace corresponding secondary structures.*

## 15.16 Partition Function and Equilibrium Properties

Compute the partition function to assess various equilibrium properties.

### 15.16.1 Detailed Description

Compute the partition function to assess various equilibrium properties.

Similar to our [Minimum Free Energy \(MFE\) Algorithms](#), we provide two different flavors for partition function computations:

- [Global Partition Function and Equilibrium Probabilities](#) - to compute the partition function for a full length sequence
- [Local \(sliding window\) Partition Function and Equilibrium Probabilities](#) - to compute the partition function of each window using a sliding window approach

While the global partition function approach supports predictions using single sequences as well as consensus partition functions for multiple sequence alignments (MSA), we currently do not support MSA input for the local variant.

Comparative prediction computes an average of the free energy contributions plus an additional covariance pseudo-energy term, exactly as we do for the [Minimum Free Energy \(MFE\) Algorithms](#) implementation.

Boltzmann weights for the free energy contributions of individual loops can be found in [Energy Evaluation for Individual Loops](#).

Our implementations also provide a stochastic backtracking procedure to draw [Random Structure Samples from the Ensemble](#) according to their equilibrium probability. Collaboration diagram for Partition Function and Equilibrium Properties:

### Modules

- [Global Partition Function and Equilibrium Probabilities](#)  
*Variations of the global partition function algorithm.*
- [Local \(sliding window\) Partition Function and Equilibrium Probabilities](#)  
*Scanning version using a sliding window approach to compute equilibrium probabilities.*

### Files

- file [concentrations.h](#)  
*Concentration computations for RNA-RNA interactions.*
- file [equilibrium\\_probs.h](#)  
*Equilibrium Probability implementations.*
- file [part\\_func.h](#)  
*Partition function implementations.*
- file [part\\_func\\_window.h](#)  
*Partition function and equilibrium probability implementation for the sliding window algorithm.*

## Functions

- `int vrna_pf_float_precision (void)`

*Find out whether partition function computations are using single precision floating points.*

### 15.16.2 Function Documentation

#### 15.16.2.1 `vrna_pf_float_precision()`

```
int vrna_pf_float_precision (  
    void )
```

```
#include <ViennaRNA/part_func.h>
```

Find out whether partition function computations are using single precision floating points.

See also

[FLT\\_OR\\_DBL](#)

Returns

1 if single precision is used, 0 otherwise



## 15.17 Global MFE Prediction

Variations of the global Minimum Free Energy (MFE) prediction algorithm.

### 15.17.1 Detailed Description

Variations of the global Minimum Free Energy (MFE) prediction algorithm.

We provide implementations of the global MFE prediction algorithm for

- Single sequences,
- Multiple sequence alignments (MSA), and
- RNA-RNA hybrids

Collaboration diagram for Global MFE Prediction:

### Modules

- [Computing MFE representatives of a Distance Based Partitioning](#)  
*Compute the minimum free energy (MFE) and secondary structures for a partitioning of the secondary structure space according to the base pair distance to two fixed reference structures basepair distance to two fixed reference structures.*
- [Deprecated Interface for Global MFE Prediction](#)

### Files

- file [mfe.h](#)  
*Compute Minimum Free energy (MFE) and backtrace corresponding secondary structures from RNA sequence data.*

### Basic global MFE prediction interface

- float [vrna\\_mfe](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, char \*structure)  
*Compute minimum free energy and an appropriate secondary structure of an RNA sequence, or RNA sequence alignment.*
- float [vrna\\_mfe\\_dimer](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, char \*structure)  
*Compute the minimum free energy of two interacting RNA molecules.*

### Simplified global MFE prediction using sequence(s) or multiple sequence alignment(s)

- float [vrna\\_fold](#) (const char \*sequence, char \*structure)  
*Compute Minimum Free Energy (MFE), and a corresponding secondary structure for an RNA sequence.*
- float [vrna\\_circfold](#) (const char \*sequence, char \*structure)  
*Compute Minimum Free Energy (MFE), and a corresponding secondary structure for a circular RNA sequence.*
- float [vrna\\_alifold](#) (const char \*\*sequences, char \*structure)  
*Compute Minimum Free Energy (MFE), and a corresponding consensus secondary structure for an RNA sequence alignment using a comparative method.*
- float [vrna\\_circalifold](#) (const char \*\*sequences, char \*structure)  
*Compute Minimum Free Energy (MFE), and a corresponding consensus secondary structure for a sequence alignment of circular RNAs using a comparative method.*
- float [vrna\\_cofold](#) (const char \*sequence, char \*structure)  
*Compute Minimum Free Energy (MFE), and a corresponding secondary structure for two dimerized RNA sequences.*

## 15.17.2 Function Documentation

### 15.17.2.1 `vrna_mfe()`

```
float vrna_mfe (
    vrna_fold_compound_t * vc,
    char * structure )
```

```
#include <ViennaRNA/mfe.h>
```

Compute minimum free energy and an appropriate secondary structure of an RNA sequence, or RNA sequence alignment.

Depending on the type of the provided `vrna_fold_compound_t`, this function predicts the MFE for a single sequence, or a corresponding averaged MFE for a sequence alignment. If backtracking is activated, it also constructs the corresponding secondary structure, or consensus structure. Therefore, the second parameter, *structure*, has to point to an allocated block of memory with a size of at least `strlen(sequence) + 1` to store the backtracked MFE structure. (For consensus structures, this is the length of the alignment + 1. If `NULL` is passed, no backtracking will be performed.

#### Note

This function is polymorphic. It accepts `vrna_fold_compound_t` of type `VRNA_FC_TYPE_SINGLE`, and `VRNA_FC_TYPE_COMPARATIVE`.

#### See also

`vrna_fold_compound_t`, `vrna_fold_compound()`, `vrna_fold()`, `vrna_circfold()`, `vrna_fold_compound_comparative()`, `vrna_alifold()`, `vrna_circalifold()`

#### Parameters

<i>vc</i>	fold compound
<i>structure</i>	A pointer to the character array where the secondary structure in dot-bracket notation will be written to (Maybe <code>NULL</code> )

#### Returns

the minimum free energy (MFE) in kcal/mol

**SWIG Wrapper Notes** This function is attached as method `mfe()` to objects of type `fold_compound`

### 15.17.2.2 `vrna_mfe_dimer()`

```
float vrna_mfe_dimer (
    vrna_fold_compound_t * vc,
    char * structure )
```

```
#include <ViennaRNA/mfe.h>
```

Compute the minimum free energy of two interacting RNA molecules.

The code is analog to the [vrna\\_mfe\(\)](#) function.

#### Parameters

<i>vc</i>	fold compound
<i>structure</i>	Will hold the barcket dot structure of the dimer molecule

#### Returns

minimum free energy of the structure

**SWIG Wrapper Notes** This function is attached as method **mfe\_dimer()** to objects of type *fold\_compound*

#### 15.17.2.3 vrna\_fold()

```
float vrna_fold (
    const char * sequence,
    char * structure )
```

```
#include <ViennaRNA/mfe.h>
```

Compute Minimum Free Energy (MFE), and a corresponding secondary structure for an RNA sequence.

This simplified interface to [vrna\\_mfe\(\)](#) computes the MFE and, if required, a secondary structure for an RNA sequence using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing, e.g. suboptimal backtracking, etc.

#### Note

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use [vrna\\_mfe\(\)](#), and the data structure [vrna\\_fold\\_compound\\_t](#) instead.

#### See also

[vrna\\_circfold\(\)](#), [vrna\\_mfe\(\)](#)

#### Parameters

<i>sequence</i>	RNA sequence
<i>structure</i>	A pointer to the character array where the secondary structure in dot-bracket notation will be written to

**Returns**

the minimum free energy (MFE) in kcal/mol

**15.17.2.4 vrna\_circfold()**

```
float vrna_circfold (
    const char * sequence,
    char * structure )
```

```
#include <ViennaRNA/mfe.h>
```

Compute Minimum Free Energy (MFE), and a corresponding secondary structure for a circular RNA sequence.

This simplified interface to [vrna\\_mfe\(\)](#) computes the MFE and, if required, a secondary structure for a circular RNA sequence using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing, e.g. suboptimal backtracking, etc.

Folding of circular RNA sequences is handled as a post-processing step of the forward recursions. See [12] for further details.

**Note**

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use [vrna\\_mfe\(\)](#), and the data structure [vrna\\_fold\\_compound\\_t](#) instead.

**See also**

[vrna\\_fold\(\)](#), [vrna\\_mfe\(\)](#)

**Parameters**

<i>sequence</i>	RNA sequence
<i>structure</i>	A pointer to the character array where the secondary structure in dot-bracket notation will be written to

**Returns**

the minimum free energy (MFE) in kcal/mol

**15.17.2.5 vrna\_alifold()**

```
float vrna_alifold (
    const char ** sequences,
    char * structure )
```

```
#include <ViennaRNA/mfe.h>
```

Compute Minimum Free Energy (MFE), and a corresponding consensus secondary structure for an RNA sequence alignment using a comparative method.

This simplified interface to [vrna\\_mfe\(\)](#) computes the MFE and, if required, a consensus secondary structure for an RNA sequence alignment using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing, e.g. suboptimal backtracking, etc.

#### Note

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use [vrna\\_mfe\(\)](#), and the data structure [vrna\\_fold\\_compound\\_t](#) instead.

#### See also

[vrna\\_circalifold\(\)](#), [vrna\\_mfe\(\)](#)

#### Parameters

<i>sequences</i>	RNA sequence alignment
<i>structure</i>	A pointer to the character array where the secondary structure in dot-bracket notation will be written to

#### Returns

the minimum free energy (MFE) in kcal/mol

#### 15.17.2.6 vrna\_circalifold()

```
float vrna_circalifold (
    const char ** sequences,
    char * structure )
```

```
#include <ViennaRNA/mfe.h>
```

Compute Minimum Free Energy (MFE), and a corresponding consensus secondary structure for a sequence alignment of circular RNAs using a comparative method.

This simplified interface to [vrna\\_mfe\(\)](#) computes the MFE and, if required, a consensus secondary structure for an RNA sequence alignment using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing, e.g. suboptimal backtracking, etc.

Folding of circular RNA sequences is handled as a post-processing step of the forward recursions. See [12] for further details.

**Note**

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use [vrna\\_mfe\(\)](#), and the data structure [vrna\\_fold\\_compound\\_t](#) instead.

**See also**

[vrna\\_alifold\(\)](#), [vrna\\_mfe\(\)](#)

**Parameters**

<i>sequences</i>	Sequence alignment of circular RNAs
<i>structure</i>	A pointer to the character array where the secondary structure in dot-bracket notation will be written to

**Returns**

the minimum free energy (MFE) in kcal/mol

**15.17.2.7 vrna\_cofold()**

```
float vrna_cofold (
    const char * sequence,
    char * structure )
```

```
#include <ViennaRNA/mfe.h>
```

Compute Minimum Free Energy (MFE), and a corresponding secondary structure for two dimerized RNA sequences.

This simplified interface to [vrna\\_mfe\(\)](#) computes the MFE and, if required, a secondary structure for two RNA sequences upon dimerization using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing, e.g. suboptimal backtracking, etc.

**Note**

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use [vrna\\_mfe\(\)](#), and the data structure [vrna\\_fold\\_compound\\_t](#) instead.

**See also**

[vrna\\_mfe\\_dimer\(\)](#), [vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_t](#), [vrna\\_cut\\_point\\_insert\(\)](#)

**Parameters**

<i>sequence</i>	two RNA sequences separated by the '&' character
<i>structure</i>	A pointer to the character array where the secondary structure in dot-bracket notation will be written to

Returns

the minimum free energy (MFE) in kcal/mol

## 15.18 Local (sliding window) MFE Prediction

Variations of the local (sliding window) Minimum Free Energy (MFE) prediction algorithm.

### 15.18.1 Detailed Description

Variations of the local (sliding window) Minimum Free Energy (MFE) prediction algorithm.

We provide implementations for the local (sliding window) MFE prediction algorithm for

- Single sequences,
- Multiple sequence alignments (MSA), and

Note, that our implementation scans an RNA sequence (or MSA) from the 3' to the 5' end, and reports back locally optimal (consensus) structures, the corresponding free energy, and the position of the sliding window in global coordinates.

For any particular RNA sequence (or MSA) multiple locally optimal (consensus) secondary structures may be predicted. Thus, we tried to implement an interface that allows for an effortless conversion of the corresponding hits into any target data structure. As a consequence, we provide two distinct ways to retrieve the corresponding predictions, either

- through directly writing to an open `FILE` stream on-the-fly, or
- through a callback function mechanism.

The latter allows one to store the results in any possible target data structure. Our implementations then pass the results through the user-implemented callback as soon as the prediction for a particular window is finished. Collaboration diagram for Local (sliding window) MFE Prediction:

### Modules

- [Deprecated Interface for Local \(Sliding Window\) MFE Prediction](#)

### Files

- file [mfe\\_window.h](#)

*Compute local Minimum Free Energy (MFE) using a sliding window approach and backtrace corresponding secondary structures.*

### Typedefs

- typedef void() [vrna\\_mfe\\_window\\_callback](#)(int start, int end, const char \*structure, float en, void \*data)  
*The default callback for sliding window MFE structure predictions.*



### Basic local (sliding window) MFE prediction interface

- float [vrna\\_mfe\\_window](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, FILE \*file)  
*Local MFE prediction using a sliding window approach.*
- float [vrna\\_mfe\\_window\\_cb](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_mfe\\_window\\_callback](#) \*cb, void \*data)
- float [vrna\\_mfe\\_window\\_zscore](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, double min\_z, FILE \*file)  
*Local MFE prediction using a sliding window approach (with z-score cut-off)*
- float [vrna\\_mfe\\_window\\_zscore\\_cb](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, double min\_z, [vrna\\_mfe\\_window\\_zscore\\_callback](#) \*cb, void \*data)

### Simplified local MFE prediction using sequence(s) or multiple sequence alignment(s)

- float [vrna\\_Lfold](#) (const char \*string, int window\_size, FILE \*file)  
*Local MFE prediction using a sliding window approach (simplified interface)*
- float [vrna\\_Lfold\\_cb](#) (const char \*string, int window\_size, [vrna\\_mfe\\_window\\_callback](#) \*cb, void \*data)
- float [vrna\\_Lfoldz](#) (const char \*string, int window\_size, double min\_z, FILE \*file)  
*Local MFE prediction using a sliding window approach with z-score cut-off (simplified interface)*
- float [vrna\\_Lfoldz\\_cb](#) (const char \*string, int window\_size, double min\_z, [vrna\\_mfe\\_window\\_zscore\\_callback](#) \*cb, void \*data)
- float [vrna\\_alifold](#) (const char \*\*alignment, int maxdist, FILE \*fp)
- float [vrna\\_alifold\\_cb](#) (const char \*\*alignment, int maxdist, [vrna\\_mfe\\_window\\_callback](#) \*cb, void \*data)

## 15.18.2 Typedef Documentation

### 15.18.2.1 vrna\_mfe\_window\_callback

```
typedef void() vrna_mfe_window_callback(int start, int end, const char *structure, float en, void *data)
```

```
#include <ViennaRNA/mfe_window.h>
```

The default callback for sliding window MFE structure predictions.

**Notes on Callback Functions** This function will be called for each hit in a sliding window MFE prediction.

See also

[vrna\\_mfe\\_window\(\)](#)

#### Parameters

<i>start</i>	provides the first position of the hit (1-based, relative to entire sequence/alignment)
<i>end</i>	provides the last position of the hit (1-based, relative to the entire sequence/alignment)
<i>structure</i>	provides the (sub)structure in dot-bracket notation
<i>en</i>	is the free energy of the structure hit in kcal/mol
<i>data</i>	is some arbitrary data pointer passed through by the function executing the callback

### 15.18.3 Function Documentation

#### 15.18.3.1 vrna\_mfe\_window()

```
float vrna_mfe_window (
    vrna_fold_compound_t * vc,
    FILE * file )

#include <ViennaRNA/mfe_window.h>
```

Local MFE prediction using a sliding window approach.

Computes minimum free energy structures using a sliding window approach, where base pairs may not span outside the window. In contrast to [vrna\\_mfe\(\)](#), where a maximum base pair span may be set using the [vrna\\_md\\_t.max\\_bp\\_span](#) attribute and one globally optimal structure is predicted, this function uses a sliding window to retrieve all locally optimal structures within each window. The size of the sliding window is set in the [vrna\\_md\\_t.window\\_size](#) attribute, prior to the retrieval of the [vrna\\_fold\\_compound\\_t](#) using [vrna\\_fold\\_compound\(\)](#) with option [VRNA\\_OPTION\\_WINDOW](#)

The predicted structures are written on-the-fly, either to stdout, if a NULL pointer is passed as file parameter, or to the corresponding filehandle.

See also

[vrna\\_fold\\_compound\(\)](#), [vrna\\_mfe\\_window\\_zscore\(\)](#), [vrna\\_mfe\(\)](#), [vrna\\_Lfold\(\)](#), [vrna\\_Lfoldz\(\)](#), [VRNA\\_OPTION\\_WINDOW](#), [vrna\\_md\\_t.max\\_bp\\_span](#), [vrna\\_md\\_t.window\\_size](#)

#### Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> with preallocated memory for the DP matrices
<i>file</i>	The output file handle where predictions are written to (maybe NULL)

**SWIG Wrapper Notes** This function is attached as method **mfe\_window()** to objects of type *fold\_compound*

#### 15.18.3.2 vrna\_mfe\_window\_zscore()

```
float vrna_mfe_window_zscore (
    vrna_fold_compound_t * vc,
    double min_z,
    FILE * file )

#include <ViennaRNA/mfe_window.h>
```

Local MFE prediction using a sliding window approach (with z-score cut-off)

Computes minimum free energy structures using a sliding window approach, where base pairs may not span outside the window. This function is the z-score version of [vrna\\_mfe\\_window\(\)](#), i.e. only predictions above a certain z-score cut-off value are printed. As for [vrna\\_mfe\\_window\(\)](#), the size of the sliding window is set in the [vrna\\_md\\_t.window\\_size](#) attribute, prior to the retrieval of the [vrna\\_fold\\_compound\\_t](#) using [vrna\\_fold\\_compound\(\)](#) with option [VRNA\\_OPTION\\_WINDOW](#).

The predicted structures are written on-the-fly, either to stdout, if a NULL pointer is passed as file parameter, or to the corresponding filehandle.

See also

[vrna\\_fold\\_compound\(\)](#), [vrna\\_mfe\\_window\\_zscore\(\)](#), [vrna\\_mfe\(\)](#), [vrna\\_Lfold\(\)](#), [vrna\\_Lfoldz\(\)](#), [VRNA\\_OPTION\\_WINDOW](#), [vrna\\_md\\_t.max\\_bp\\_span](#), [vrna\\_md\\_t.window\\_size](#)

#### Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> with preallocated memory for the DP matrices
<i>min<sub>z</sub></i>	The minimal z-score for a predicted structure to appear in the output
<i>file</i>	The output file handle where predictions are written to (maybe NULL)

#### 15.18.3.3 vrna\_Lfold()

```
float vrna_Lfold (
    const char * string,
    int window_size,
    FILE * file )

#include <ViennaRNA/mfe_window.h>
```

Local MFE prediction using a sliding window approach (simplified interface)

This simplified interface to [vrna\\_mfe\\_window\(\)](#) computes the MFE and locally optimal secondary structure using default options. Structures are predicted using a sliding window approach, where base pairs may not span outside the window. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing.

#### Note

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use [vrna\\_mfe\\_window\(\)](#), and the data structure [vrna\\_fold\\_compound\\_t](#) instead.

See also

[vrna\\_mfe\\_window\(\)](#), [vrna\\_Lfoldz\(\)](#), [vrna\\_mfe\\_window\\_zscore\(\)](#)

#### Parameters

<i>string</i>	The nucleic acid sequence
<i>window_size</i>	The window size for locally optimal structures
<i>file</i>	The output file handle where predictions are written to (if NULL, output is written to stdout)

### 15.18.3.4 vrna\_Lfoldz()

```
float vrna_Lfoldz (
    const char * string,
    int window_size,
    double min_z,
    FILE * file )

#include <ViennaRNA/mfe_window.h>
```

Local MFE prediction using a sliding window approach with z-score cut-off (simplified interface)

This simplified interface to [vrna\\_mfe\\_window\\_zscore\(\)](#) computes the MFE and locally optimal secondary structure using default options. Structures are predicted using a sliding window approach, where base pairs may not span outside the window. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing. This function is the z-score version of [vrna\\_Lfold\(\)](#), i.e. only predictions above a certain z-score cut-off value are printed.

#### Note

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use [vrna\\_mfe\\_window\(\)](#), and the data structure [vrna\\_fold\\_compound\\_t](#) instead.

#### See also

[vrna\\_mfe\\_window\\_zscore\(\)](#), [vrna\\_Lfold\(\)](#), [vrna\\_mfe\\_window\(\)](#)

#### Parameters

<i>string</i>	The nucleic acid sequence
<i>window_size</i>	The window size for locally optimal structures
<i>min_z</i>	The minimal z-score for a predicted structure to appear in the output
<i>file</i>	The output file handle where predictions are written to (if NULL, output is written to stdout)

## 15.19 Backtracking MFE structures

Backtracking related interfaces.

### 15.19.1 Detailed Description

Backtracking related interfaces.

Collaboration diagram for Backtracking MFE structures:

#### Functions

- `int vrna_BT_hp_loop (vrna_fold_compound_t *fc, int i, int j, int en, vrna_bp_stack_t *bp_stack, int *stack_count)`  
*Backtrack a hairpin loop closed by  $(i, j)$ .*
- `int vrna_BT_stack (vrna_fold_compound_t *fc, int *i, int *j, int *en, vrna_bp_stack_t *bp_stack, int *stack_count)`  
*Backtrack a stacked pair closed by  $(i, j)$ .*
- `int vrna_BT_int_loop (vrna_fold_compound_t *fc, int *i, int *j, int en, vrna_bp_stack_t *bp_stack, int *stack_count)`  
*Backtrack an interior loop closed by  $(i, j)$ .*
- `int vrna_BT_mb_loop (vrna_fold_compound_t *fc, int *i, int *j, int *k, int en, int *component1, int *component2)`  
*Backtrack the decomposition of a multi branch loop closed by  $(i, j)$ .*

### 15.19.2 Function Documentation

#### 15.19.2.1 vrna\_BT\_hp\_loop()

```
int vrna_BT_hp_loop (
    vrna_fold_compound_t * fc,
    int i,
    int j,
    int en,
    vrna_bp_stack_t * bp_stack,
    int * stack_count )
```

```
#include <ViennaRNA/loops/hairpin.h>
```

Backtrack a hairpin loop closed by  $(i, j)$ .

#### Note

This function is polymorphic! The provided `vrna_fold_compound_t` may be of type `VRNA_FC_TYPE_SINGLE` or `VRNA_FC_TYPE_COMPARATIVE`

## 15.19.2.2 vrna\_BT\_mb\_loop()

```
int vrna_BT_mb_loop (
    vrna_fold_compound_t * fc,
    int * i,
    int * j,
    int * k,
    int * en,
    int * component1,
    int * component2 )
```

```
#include <ViennaRNA/loops/multibranch.h>
```

Backtrack the decomposition of a multi branch loop closed by  $(i, j)$ .

## Parameters

<i>fc</i>	The <code>vrna_fold_compound_t</code> filled with all relevant data for backtracking
<i>i</i>	5' position of base pair closing the loop (will be set to 5' position of leftmost decomposed block upon successful backtracking)
<i>j</i>	3' position of base pair closing the loop (will be set to 3' position of rightmost decomposed block upon successful backtracking)
<i>k</i>	Split position that delimits leftmost from rightmost block, $[i, k]$ and $[k+1, j]$ , respectively. (Will be set upon successful backtracking)
<i>en</i>	The energy contribution of the substructure enclosed by $(i, j)$
<i>component1</i>	Type of leftmost block (1 = ML, 2 = C)
<i>component2</i>	Type of rightmost block (1 = ML, 2 = C)

## Returns

1, if backtracking succeeded, 0 otherwise.

## 15.20 Global Partition Function and Equilibrium Probabilities

Variations of the global partition function algorithm.

### 15.20.1 Detailed Description

Variations of the global partition function algorithm.

We provide implementations of the global partition function algorithm for

- Single sequences,
- Multiple sequence alignments (MSA), and
- RNA-RNA hybrids

Collaboration diagram for Global Partition Function and Equilibrium Probabilities:

### Modules

- [Computing Partition Functions of a Distance Based Partitioning](#)  
*Compute the partition function and stochastically sample secondary structures for a partitioning of the secondary structure space according to the base pair distance to two fixed reference structures.*
- [Deprecated Interface for Global Partition Function Computation](#)

### Files

- file [part\\_func.h](#)  
*Partition function implementations.*

### Data Structures

- struct [vrna\\_dimer\\_pf\\_s](#)  
*Data structure returned by [vrna\\_pf\\_dimer\(\)](#) [More...](#)*

### Functions

- void [vrna\\_pf\\_dimer\\_probs](#) (double FAB, double FA, double FB, [vrna\\_ep\\_t](#) \*prAB, const [vrna\\_ep\\_t](#) \*prA, const [vrna\\_ep\\_t](#) \*prB, int Alength, const [vrna\\_exp\\_param\\_t](#) \*exp\_params)  
*Compute Boltzmann probabilities of dimerization without homodimers.*
- double [vrna\\_pr\\_structure](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, const char \*structure)  
*Compute the equilibrium probability of a particular secondary structure.*
- [vrna\\_ep\\_t](#) \* [vrna\\_plist\\_from\\_probs](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, double cut\_off)  
*Create a [vrna\\_ep\\_t](#) from base pair probability matrix.*

## Base pair related probability computations

- double [vrna\\_mean\\_bp\\_distance\\_pr](#) (int length, [FLT\\_OR\\_DBL](#) \*pr)  
*Get the mean base pair distance in the thermodynamic ensemble from a probability matrix.*
- double [vrna\\_mean\\_bp\\_distance](#) ([vrna\\_fold\\_compound\\_t](#) \*vc)  
*Get the mean base pair distance in the thermodynamic ensemble.*
- double [vrna\\_ensemble\\_defect](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, const char \*structure)  
*Compute the Ensemble Defect for a given target structure.*
- [vrna\\_ep\\_t](#) \* [vrna\\_stack\\_prob](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, double cutoff)  
*Compute stacking probabilities.*

## Basic global partition function interface

- float [vrna\\_pf](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, char \*structure)  
*Compute the partition function  $Q$  for a given RNA sequence, or sequence alignment.*
- [vrna\\_dimer\\_pf\\_t](#) [vrna\\_pf\\_dimer](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, char \*structure)  
*Calculate partition function and base pair probabilities of nucleic acid/nucleic acid dimers.*

## Simplified global partition function computation using sequence(s) or multiple sequence alignment(s)

- float [vrna\\_pf\\_fold](#) (const char \*sequence, char \*structure, [vrna\\_ep\\_t](#) \*\*pl)  
*Compute Partition function  $Q$  (and base pair probabilities) for an RNA sequence using a comparative method.*
- float [vrna\\_pf\\_circfold](#) (const char \*sequence, char \*structure, [vrna\\_ep\\_t](#) \*\*pl)  
*Compute Partition function  $Q$  (and base pair probabilities) for a circular RNA sequences using a comparative method.*
- float [vrna\\_pf\\_alifold](#) (const char \*\*sequences, char \*structure, [vrna\\_ep\\_t](#) \*\*pl)  
*Compute Partition function  $Q$  (and base pair probabilities) for an RNA sequence alignment using a comparative method.*
- float [vrna\\_pf\\_circalifold](#) (const char \*\*sequences, char \*structure, [vrna\\_ep\\_t](#) \*\*pl)  
*Compute Partition function  $Q$  (and base pair probabilities) for an alignment of circular RNA sequences using a comparative method.*
- [vrna\\_dimer\\_pf\\_t](#) [vrna\\_pf\\_co\\_fold](#) (const char \*seq, char \*structure, [vrna\\_ep\\_t](#) \*\*pl)  
*Calculate partition function and base pair probabilities of nucleic acid/nucleic acid dimers.*

## 15.20.2 Data Structure Documentation

### 15.20.2.1 struct [vrna\\_dimer\\_pf\\_s](#)

Data structure returned by [vrna\\_pf\\_dimer\(\)](#)

#### Data Fields

- double [F0AB](#)  
*Null model without DuplexInit.*
- double [FAB](#)  
*all states with DuplexInit correction*
- double [FcAB](#)  
*true hybrid states only*
- double [FA](#)  
*monomer A*
- double [FB](#)  
*monomer B*



## 15.20.3 Function Documentation

## 15.20.3.1 vrna\_mean\_bp\_distance\_pr()

```
double vrna_mean_bp_distance_pr (
    int length,
    FLT_OR_DBL * pr )
```

```
#include <ViennaRNA/equilibrium_probs.h>
```

Get the mean base pair distance in the thermodynamic ensemble from a probability matrix.

$$\langle d \rangle = \sum_{a,b} p_a p_b d(S_a, S_b)$$

this can be computed from the pair probs  $p_{ij}$  as

$$\langle d \rangle = \sum_{ij} p_{ij} (1 - p_{ij})$$

**Parameters**

<i>length</i>	The length of the sequence
<i>pr</i>	The matrix containing the base pair probabilities

**Returns**

The mean pair distance of the structure ensemble

## 15.20.3.2 vrna\_mean\_bp\_distance()

```
double vrna_mean_bp_distance (
    vrna_fold_compound_t * vc )
```

```
#include <ViennaRNA/equilibrium_probs.h>
```

Get the mean base pair distance in the thermodynamic ensemble.

$$\langle d \rangle = \sum_{a,b} p_a p_b d(S_a, S_b)$$

this can be computed from the pair probs  $p_{ij}$  as

$$\langle d \rangle = \sum_{ij} p_{ij} (1 - p_{ij})$$

**Parameters**

<i>vc</i>	The fold compound data structure
-----------	----------------------------------

**Returns**

The mean pair distance of the structure ensemble

**SWIG Wrapper Notes** This function is attached as method `mean_bp_distance()` to objects of type `fold_compound`

### 15.20.3.3 `vrna_ensemble_defect()`

```
double vrna_ensemble_defect (
    vrna_fold_compound_t * fc,
    const char * structure )

#include <ViennaRNA/equilibrium_probs.h>
```

Compute the Ensemble Defect for a given target structure.

Given a target structure  $s$ , compute the average dissimilarity of a randomly drawn structure from the ensemble, i.e.:

$$ED(s) = 1 - \frac{1}{n} \sum_{ij, (i,j) \in s} p_{ij} - \frac{1}{n} \sum_i (1 - s_i) q_i$$

with sequence length  $n$ , the probability  $p_{ij}$  of a base pair  $(i, j)$ , the probability  $q_i = 1 - \sum_j p_{ij}$  of nucleotide  $i$  being unpaired, and the indicator variable  $s_i = 1$  if  $\exists (i, j) \in s$ , and  $s_i = 0$  otherwise.

#### Precondition

The `vrna_fold_compound_t` input parameter `fc` must contain a valid base pair probability matrix. This means that partition function and base pair probabilities must have been computed using `fc` before execution of this function!

#### See also

`vrna_pf()`, `vrna_pairing_probs()`

#### Parameters

<code>fc</code>	A <code>fold_compound</code> with pre-computed base pair probabilities
<code>structure</code>	A target structure in dot-bracket notation

#### Returns

The ensemble defect with respect to the target structure, or -1. upon failure, e.g. pre-conditions are not met

### 15.20.3.4 `vrna_stack_prob()`

```
vrna_ep_t* vrna_stack_prob (
    vrna_fold_compound_t * vc,
    double cutoff )

#include <ViennaRNA/equilibrium_probs.h>
```

Compute stacking probabilities.

For each possible base pair  $(i, j)$ , compute the probability of a stack  $(i, j)$ ,  $(i + 1, j - 1)$ .

## Parameters

<i>vc</i>	The fold compound data structure with precomputed base pair probabilities
<i>cutoff</i>	A cutoff value that limits the output to stacks with $p > \text{cutoff}$ .

## Returns

A list of stacks with enclosing base pair  $(i, j)$  and probability  $p$

15.20.3.5 `vrna_pf_dimer_probs()`

```
void vrna_pf_dimer_probs (
    double FAB,
    double FA,
    double FB,
    vrna_ep_t * prAB,
    const vrna_ep_t * prA,
    const vrna_ep_t * prB,
    int Alength,
    const vrna_exp_param_t * exp_params )
```

```
#include <ViennaRNA/equilibrium_probs.h>
```

Compute Boltzmann probabilities of dimerization without homodimers.

Given the pair probabilities and free energies (in the null model) for a dimer AB and the two constituent monomers A and B, compute the conditional pair probabilities given that a dimer AB actually forms. Null model pair probabilities are given as a list as produced by `vrna_plist_from_probs()`, the dimer probabilities 'prAB' are modified in place.

## Parameters

<i>FAB</i>	free energy of dimer AB
<i>FA</i>	free energy of monomer A
<i>FB</i>	free energy of monomer B
<i>prAB</i>	pair probabilities for dimer
<i>prA</i>	pair probabilities monomer
<i>prB</i>	pair probabilities monomer
<i>Alength</i>	Length of molecule A
<i>exp_params</i>	The precomputed Boltzmann factors

15.20.3.6 `vrna_pr_structure()`

```
double vrna_pr_structure (
    vrna_fold_compound_t * fc,
    const char * structure )
```

```
#include <ViennaRNA/equilibrium_probs.h>
```

Compute the equilibrium probability of a particular secondary structure.

The probability  $p(s)$  of a particular secondary structure  $s$  can be computed as

$$p(s) = \frac{\exp(-\beta E(s))}{Z}$$

from the structures free energy  $E(s)$  and the partition function

$$Z = \sum_s \exp(-\beta E(s)), \quad \text{with } \beta = \frac{1}{RT}$$

where  $R$  is the gas constant and  $T$  the thermodynamic temperature.

#### Precondition

The fold compound `fc` must have went through a call to `vrna_pf()` to fill the dynamic programming matrices with the corresponding partition function.

#### Parameters

<code>fc</code>	The fold compound data structure with precomputed partition function
<code>structure</code>	The secondary structure to compute the probability for in dot-bracket notation

#### Returns

The probability of the input structure (range  $[0 : 1]$ )

#### 15.20.3.7 `vrna_pf()`

```
float vrna_pf (
    vrna_fold_compound_t * vc,
    char * structure )

#include <ViennaRNA/part_func.h>
```

Compute the partition function  $Q$  for a given RNA sequence, or sequence alignment.

If `structure` is not a NULL pointer on input, it contains on return a string consisting of the letters ". , | { } ( )" denoting bases that are essentially unpaired, weakly paired, strongly paired without preference, weakly upstream (downstream) paired, or strongly up- (down-)stream paired bases, respectively. If the model's `compute_bpp` is set to 0 base pairing probabilities will not be computed (saving CPU time), otherwise after calculations took place `pr` will contain the probability that bases  $i$  and  $j$  pair.

#### Note

This function is polymorphic. It accepts `vrna_fold_compound_t` of type `VRNA_FC_TYPE_SINGLE`, and `VRNA_FC_TYPE_COMPARATIVE`.

This function may return `INF` / 100. in case of contradicting constraints or numerical over-/underflow. In the latter case, a corresponding warning will be issued to `stdout`.

#### See also

`vrna_fold_compound_t`, `vrna_fold_compound()`, `vrna_pf_fold()`, `vrna_pf_circfold()`, `vrna_fold_compound_comparative()`, `vrna_pf_alifold()`, `vrna_pf_circalifold()`, `vrna_db_from_probs()`, `vrna_exp_params()`, `vrna_aln_pinfo()`

## Parameters

<code>in, out</code>	<code>vc</code>	The fold compound data structure
<code>in, out</code>	<code>structure</code>	A pointer to the character array where position-wise pairing propensity will be stored. (Maybe NULL)

## Returns

The Gibbs free energy of the ensemble (  $G = -RT \cdot \log(Q)$  ) in kcal/mol

**SWIG Wrapper Notes** This function is attached as method **pf()** to objects of type *fold\_compound*

15.20.3.8 `vrna_pf_dimer()`

```
vrna_dimer_pf_t vrna_pf_dimer (
    vrna_fold_compound_t * vc,
    char * structure )
```

```
#include <ViennaRNA/part_func.h>
```

Calculate partition function and base pair probabilities of nucleic acid/nucleic acid dimers.

This is the cofold partition function folding.

## Note

This function may return `INF` / 100. for the `FA`, `FB`, `FAB`, `F0AB` members of the output data structure in case of contradicting constraints or numerical over-/underflow. In the latter case, a corresponding warning will be issued to `stdout`.

## See also

[`vrna\_fold\_compound\(\)`](#) for how to retrieve the necessary data structure

## Parameters

<code>vc</code>	the fold compound data structure
<code>structure</code>	Will hold the structure or constraints

## Returns

`vrna_dimer_pf_t` structure containing a set of energies needed for concentration computations.

**SWIG Wrapper Notes** This function is attached as method **pf\_dimer()** to objects of type *fold\_compound*

15.20.3.9 `vrna_pf_fold()`

```
float vrna_pf_fold (
    const char * sequence,
    char * structure,
    vrna_ep_t ** pl )

#include <ViennaRNA/part_func.h>
```

Compute Partition function  $Q$  (and base pair probabilities) for an RNA sequence using a comparative method.

This simplified interface to `vrna_pf()` computes the partition function and, if required, base pair probabilities for an RNA sequence using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing.

**Note**

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use `vrna_pf()`, and the data structure `vrna_fold_compound_t` instead.

**See also**

[vrna\\_pf\\_circfold\(\)](#), [vrna\\_pf\(\)](#), [vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_t](#)

**Parameters**

<i>sequence</i>	RNA sequence
<i>structure</i>	A pointer to the character array where position-wise pairing propensity will be stored. (Maybe NULL)
<i>pl</i>	A pointer to a list of <a href="#">vrna_ep_t</a> to store pairing probabilities (Maybe NULL)

**Returns**

The Gibbs free energy of the ensemble (  $G = -RT \cdot \log(Q)$  ) in kcal/mol

15.20.3.10 `vrna_pf_circfold()`

```
float vrna_pf_circfold (
    const char * sequence,
    char * structure,
    vrna_ep_t ** pl )

#include <ViennaRNA/part_func.h>
```

Compute Partition function  $Q$  (and base pair probabilities) for a circular RNA sequences using a comparative method.

This simplified interface to `vrna_pf()` computes the partition function and, if required, base pair probabilities for a circular RNA sequence using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing.

**Note**

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use [vrna\\_pf\(\)](#), and the data structure [vrna\\_fold\\_compound\\_t](#) instead.

Folding of circular RNA sequences is handled as a post-processing step of the forward recursions. See [12] for further details.

**See also**

[vrna\\_pf\\_fold\(\)](#), [vrna\\_pf\(\)](#), [vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_t](#)

**Parameters**

<i>sequence</i>	A circular RNA sequence
<i>structure</i>	A pointer to the character array where position-wise pairing propensity will be stored. (Maybe NULL)
<i>pl</i>	A pointer to a list of <a href="#">vrna_ep_t</a> to store pairing probabilities (Maybe NULL)

**Returns**

The Gibbs free energy of the ensemble (  $G = -RT \cdot \log(Q)$  ) in kcal/mol

**15.20.3.11 vrna\_pf\_alifold()**

```
float vrna_pf_alifold (
    const char ** sequences,
    char * structure,
    vrna_ep_t ** pl )

#include <ViennaRNA/part_func.h>
```

Compute Partition function  $Q$  (and base pair probabilities) for an RNA sequence alignment using a comparative method.

This simplified interface to [vrna\\_pf\(\)](#) computes the partition function and, if required, base pair probabilities for an RNA sequence alignment using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing.

**Note**

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use [vrna\\_pf\(\)](#), and the data structure [vrna\\_fold\\_compound\\_t](#) instead.

**See also**

[vrna\\_pf\\_circularfold\(\)](#), [vrna\\_pf\(\)](#), [vrna\\_fold\\_compound\\_comparative\(\)](#), [vrna\\_fold\\_compound\\_t](#)

## Parameters

<i>sequences</i>	RNA sequence alignment
<i>structure</i>	A pointer to the character array where position-wise pairing propensity will be stored. (Maybe NULL)
<i>pl</i>	A pointer to a list of <a href="#">vrna_ep_t</a> to store pairing probabilities (Maybe NULL)

## Returns

The Gibbs free energy of the ensemble (  $G = -RT \cdot \log(Q)$ ) in kcal/mol

## 15.20.3.12 vrna\_pf\_circalifold()

```
float vrna_pf_circalifold (
    const char ** sequences,
    char * structure,
    vrna_ep_t ** pl )
```

```
#include <ViennaRNA/part_func.h>
```

Compute Partition function  $Q$  (and base pair probabilities) for an alignment of circular RNA sequences using a comparative method.

This simplified interface to [vrna\\_pf\(\)](#) computes the partition function and, if required, base pair probabilities for an RNA sequence alignment using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing.

## Note

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use [vrna\\_pf\(\)](#), and the data structure [vrna\\_fold\\_compound\\_t](#) instead.

Folding of circular RNA sequences is handled as a post-processing step of the forward recursions. See [12] for further details.

## See also

[vrna\\_pf\\_alifold\(\)](#), [vrna\\_pf\(\)](#), [vrna\\_fold\\_compound\\_comparative\(\)](#), [vrna\\_fold\\_compound\\_t](#)

## Parameters

<i>sequences</i>	Sequence alignment of circular RNAs
<i>structure</i>	A pointer to the character array where position-wise pairing propensity will be stored. (Maybe NULL)
<i>pl</i>	A pointer to a list of <a href="#">vrna_ep_t</a> to store pairing probabilities (Maybe NULL)



**Returns**

The Gibbs free energy of the ensemble (  $G = -RT \cdot \log(Q)$ ) in kcal/mol

**15.20.3.13 vrna\_plist\_from\_probs()**

```
vrna_ep_t* vrna_plist_from_probs (
    vrna_fold_compound_t * vc,
    double cut_off )

#include <ViennaRNA/utils/structures.h>
```

Create a [vrna\\_ep\\_t](#) from base pair probability matrix.

The probability matrix provided via the [vrna\\_fold\\_compound\\_t](#) is parsed and all pair probabilities above the given threshold are used to create an entry in the plist

The end of the plist is marked by sequence positions i as well as j equal to 0. This condition should be used to stop looping over its entries

**Parameters**

in	<i>vc</i>	The fold compound
in	<i>cut_off</i>	The cutoff value

**Returns**

A pointer to the plist that is to be created

**15.20.3.14 vrna\_pf\_co\_fold()**

```
vrna_dimer_pf_t vrna_pf_co_fold (
    const char * seq,
    char * structure,
    vrna_ep_t ** pl )

#include <ViennaRNA/part_func.h>
```

Calculate partition function and base pair probabilities of nucleic acid/nucleic acid dimers.

This simplified interface to [vrna\\_pf\\_dimer\(\)](#) computes the partition function and, if required, base pair probabilities for an RNA-RNA interaction using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing.

**Note**

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use [vrna\\_pf\\_dimer\(\)](#), and the data structure [vrna\\_fold\\_compound\\_t](#) instead.

**See also**

[vrna\\_pf\\_dimer\(\)](#)

**Parameters**

<i>seq</i>	Two concatenated RNA sequences with a delimiting '&' in between
<i>structure</i>	A pointer to the character array where position-wise pairing propensity will be stored. (Maybe NULL)
<i>pl</i>	A pointer to a list of <a href="#">vrna_ep_t</a> to store pairing probabilities (Maybe NULL)

**Returns**

vrna\_dimer\_pf\_t structure containing a set of energies needed for concentration computations.

## 15.21 Local (sliding window) Partition Function and Equilibrium Probabilities

Scanning version using a sliding window approach to compute equilibrium probabilities.

### 15.21.1 Detailed Description

Scanning version using a sliding window approach to compute equilibrium probabilities.

Collaboration diagram for Local (sliding window) Partition Function and Equilibrium Probabilities:

### Modules

- [Deprecated Interface for Local \(Sliding Window\) Partition Function Computation](#)

### Files

- file [part\\_func\\_window.h](#)  
*Partition function and equilibrium probability implementation for the sliding window algorithm.*

### Macros

- `#define VRNA_EXT_LOOP 1U`  
*Exterior loop.*
- `#define VRNA_HP_LOOP 2U`  
*Hairpin loop.*
- `#define VRNA_INT_LOOP 4U`  
*Internal loop.*
- `#define VRNA_MB_LOOP 8U`  
*Multibranch loop.*
- `#define VRNA_ANY_LOOP (VRNA_EXT_LOOP | VRNA_HP_LOOP | VRNA_INT_LOOP | VRNA_MB_LOOP)`  
*Any loop.*
- `#define VRNA_PROBS_WINDOW_BPP 4096U`  
*Trigger base pairing probabilities.*
- `#define VRNA_PROBS_WINDOW_UP 8192U`  
*Trigger unpaired probabilities.*
- `#define VRNA_PROBS_WINDOW_STACKP 16384U`  
*Trigger base pair stack probabilities.*
- `#define VRNA_PROBS_WINDOW_UP_SPLIT 32768U`  
*Trigger detailed unpaired probabilities split up into different loop type contexts.*
- `#define VRNA_PROBS_WINDOW_PF 65536U`  
*Trigger partition function.*

### Typedefs

- typedef void() [vrna\\_probs\\_window\\_callback](#)([FLT\\_OR\\_DBL](#) \*pr, int pr\_size, int i, int max, unsigned int type, void \*data)  
*Sliding window probability computation callback.*

## Basic local partition function interface

- `int vrna_probs_window (vrna_fold_compound_t *fc, int ulength, unsigned int options, vrna_probs_window_callback *cb, void *data)`

*Compute various equilibrium probabilities under a sliding window approach.*

## Simplified global partition function computation using sequence(s) or multiple sequence alignment(s)

- `vrna_ep_t * vrna_pfl_fold (const char *sequence, int window_size, int max_bp_span, float cutoff)`  
*Compute base pair probabilities using a sliding-window approach.*
- `int vrna_pfl_fold_cb (const char *sequence, int window_size, int max_bp_span, vrna_probs_window_callback *cb, void *data)`  
*Compute base pair probabilities using a sliding-window approach (callback version)*

- `double ** vrna_pfl_fold_up (const char *sequence, int ulength, int window_size, int max_bp_span)`  
*Compute probability of contiguous unpaired segments.*
- `int vrna_pfl_fold_up_cb (const char *sequence, int ulength, int window_size, int max_bp_span, vrna_probs_window_callback *cb, void *data)`  
*Compute probability of contiguous unpaired segments.*

## 15.21.2 Macro Definition Documentation

### 15.21.2.1 VRNA\_PROBS\_WINDOW\_BPP

```
#define VRNA_PROBS_WINDOW_BPP 4096U

#include <ViennaRNA/part_func_window.h>
```

Trigger base pairing probabilities.

Passing this flag to `vrna_probs_window()` activates callback execution for base pairing probabilities. In turn, the corresponding callback receives this flag through the `type` argument whenever base pairing probabilities are provided.

Detailed information for the algorithm to compute unpaired probabilities can be taken from [3].

See also

`vrna_probs_window()`

### 15.21.2.2 VRNA\_PROBS\_WINDOW\_UP

```
#define VRNA_PROBS_WINDOW_UP 8192U
#include <ViennaRNA/part_func_window.h>
```

Trigger unpaired probabilities.

Passing this flag to [vrna\\_probs\\_window\(\)](#) activates callback execution for unpaired probabilities. In turn, the corresponding callback receives this flag through the `type` argument whenever unpaired probabilities are provided.

Detailed information for the algorithm to compute unpaired probabilities can be taken from [4].

See also

[vrna\\_probs\\_window\(\)](#)

### 15.21.2.3 VRNA\_PROBS\_WINDOW\_STACKP

```
#define VRNA_PROBS_WINDOW_STACKP 16384U
#include <ViennaRNA/part_func_window.h>
```

Trigger base pair stack probabilities.

Passing this flag to [vrna\\_probs\\_window\(\)](#) activates callback execution for stacking probabilities. In turn, the corresponding callback receives this flag through the `type` argument whenever stack probabilities are provided.

**Bug** Currently, this flag is a placeholder doing nothing as the corresponding implementation for stack probability computation is missing.

See also

[vrna\\_probs\\_window\(\)](#)

### 15.21.2.4 VRNA\_PROBS\_WINDOW\_UP\_SPLIT

```
#define VRNA_PROBS_WINDOW_UP_SPLIT 32768U
#include <ViennaRNA/part_func_window.h>
```

Trigger detailed unpaired probabilities split up into different loop type contexts.

Passing this flag to [vrna\\_probs\\_window\(\)](#) activates callback execution for unpaired probabilities. In contrast to [VRNA\\_PROBS\\_WINDOW\\_UP](#) this flag requests unpaired probabilities to be split up into different loop type contexts. In turn, the corresponding callback receives the [VRNA\\_PROBS\\_WINDOW\\_UP](#) flag OR-ed together with the corresponding loop type, i.e.:

- [VRNA\\_EXT\\_LOOP](#) - Exterior loop.
- [VRNA\\_HP\\_LOOP](#) - Hairpin loop.
- [VRNA\\_INT\\_LOOP](#) - Internal loop.
- [VRNA\\_MB\\_LOOP](#) - Multibranch loop.
- [VRNA\\_ANY\\_LOOP](#) - Any loop.

See also

[vrna\\_probs\\_window\(\)](#), [VRNA\\_PROBS\\_WINDOW\\_UP](#)

### 15.21.2.5 VRNA\_PROBS\_WINDOW\_PF

```
#define VRNA_PROBS_WINDOW_PF 65536U
#include <ViennaRNA/part_func_window.h>
```

Trigger partition function.

Passing this flag to [vrna\\_probs\\_window\(\)](#) activates callback execution for partition function. In turn, the corresponding callback receives this flag through its `type` argument whenever partition function data is provided.

#### Note

Instead of actually providing the partition function  $Z$ , the callback is always provided with the corresponding ensemble free energy  $\Delta G = -RT \ln Z$ .

#### See also

[vrna\\_probs\\_window\(\)](#)

## 15.21.3 Typedef Documentation

### 15.21.3.1 vrna\_probs\_window\_callback

```
typedef void() vrna_probs_window_callback(FLT_OR_DBL *pr, int pr_size, int i, int max, unsigned
int type, void *data)
#include <ViennaRNA/part_func_window.h>
```

Sliding window probability computation callback.

**Notes on Callback Functions** This function will be called for each probability data set in the sliding window probability computation implementation of [vrna\\_probs\\_window\(\)](#). The argument *type* specifies the type of probability that is passed to this function.

#### Types:

- [VRNA\\_PROBS\\_WINDOW\\_BPP](#) - Trigger base pairing probabilities.
- [VRNA\\_PROBS\\_WINDOW\\_UP](#) - Trigger unpaired probabilities.
- [VRNA\\_PROBS\\_WINDOW\\_PF](#) - Trigger partition function.

The above types usually come exclusively. However, for unpaired probabilities, the [VRNA\\_PROBS\\_WINDOW\\_UP](#) flag is OR-ed together with one of the loop type contexts

- [VRNA\\_EXT\\_LOOP](#) - Exterior loop.
- [VRNA\\_HP\\_LOOP](#) - Hairpin loop.
- [VRNA\\_INT\\_LOOP](#) - Internal loop.
- [VRNA\\_MB\\_LOOP](#) - Multibranch loop.
- [VRNA\\_ANY\\_LOOP](#) - Any loop.

to indicate the particular type of data available through the `pr` pointer.

#### See also

[vrna\\_probs\\_window\(\)](#), [vrna\\_pfl\\_fold\\_up\\_cb\(\)](#)

## Parameters

<i>pr</i>	An array of probabilities
<i>pr_size</i>	The length of the probability array
<i>i</i>	The i-position (5') of the probabilities
<i>max</i>	The (theoretical) maximum length of the probability array
<i>type</i>	The type of data that is provided
<i>data</i>	Auxiliary data

## 15.21.4 Function Documentation

15.21.4.1 `vrna_probs_window()`

```
int vrna_probs_window (
    vrna_fold_compound_t * fc,
    int ulength,
    unsigned int options,
    vrna_probs_window_callback * cb,
    void * data )
```

```
#include <ViennaRNA/part_func_window.h>
```

Compute various equilibrium probabilities under a sliding window approach.

This function applies a sliding window scan for the sequence provided with the argument `fc` and reports back equilibrium probabilities through the callback function `cb`. The data reported to the callback depends on the `options` flag.

## Note

The parameter `ulength` only affects computation and resulting data if unpaired probability computations are requested through the `options` flag.

## Options:

- `VRNA_PROBS_WINDOW_BPP` - Trigger base pairing probabilities.
- `VRNA_PROBS_WINDOW_UP` - Trigger unpaired probabilities.
- `VRNA_PROBS_WINDOW_UP_SPLIT` - Trigger detailed unpaired probabilities split up into different loop type contexts.

Options may be OR-ed together

## See also

`vrna_pfl_fold_cb()`, `vrna_pfl_fold_up_cb()`

## Parameters

<i>fc</i>	The fold compound with sequence data, model settings and precomputed energy parameters
<i>ulength</i>	The maximal length of an unpaired segment (only for unpaired probability computations)
<i>cb</i>	The callback function which collects the pair probability data for further processing
<i>data</i>	Some arbitrary data structure that is passed to the callback <i>cb</i>
<i>options</i>	Option flags to control the behavior of this function

## Returns

0 on failure, non-zero on success

15.21.4.2 `vrna_pfl_fold()`

```
vrna_ep_t* vrna_pfl_fold (
    const char * sequence,
    int window_size,
    int max_bp_span,
    float cutoff )
```

```
#include <ViennaRNA/part_func_window.h>
```

Compute base pair probabilities using a sliding-window approach.

This is a simplified wrapper to [vrna\\_probs\\_window\(\)](#) that given a nucleic acid sequence, a window size, a maximum base pair span, and a cutoff value computes the pair probabilities for any base pair in any window. The pair probabilities are returned as a list and the user has to take care to `free()` the memory occupied by the list.

## Note

This function uses default model settings! For custom model settings, we refer to the function [vrna\\_probs\\_window\(\)](#).

In case of any computation errors, this function returns `NULL`

## See also

[vrna\\_probs\\_window\(\)](#), [vrna\\_pfl\\_fold\\_cb\(\)](#), [vrna\\_pfl\\_fold\\_up\(\)](#)

## Parameters

<i>sequence</i>	The nucleic acid input sequence
<i>window_size</i>	The size of the sliding window
<i>max_bp_span</i>	The maximum distance along the backbone between two nucleotides that form a base pairs
<i>cutoff</i>	A cutoff value that omits all pairs with lower probability



**Returns**

A list of base pair probabilities, terminated by an entry with [vrna\\_ep\\_t.i](#) and [vrna\\_ep\\_t.j](#) set to 0

**15.21.4.3 vrna\_pfl\_fold\_cb()**

```
int vrna_pfl_fold_cb (
    const char * sequence,
    int window_size,
    int max_bp_span,
    vrna_probs_window_callback * cb,
    void * data )
```

```
#include <ViennaRNA/part_func_window.h>
```

Compute base pair probabilities using a sliding-window approach (callback version)

This is a simplified wrapper to [vrna\\_probs\\_window\(\)](#) that given a nucleic acid sequence, a window size, a maximum base pair span, and a cutoff value computes the pair probabilities for any base pair in any window. It is similar to [vrna\\_pfl\\_fold\(\)](#) but uses a callback mechanism to return the pair probabilities.

Read the details for [vrna\\_probs\\_window\(\)](#) for details on the callback implementation!

**Note**

This function uses default model settings! For custom model settings, we refer to the function [vrna\\_probs\\_window\(\)](#).

**See also**

[vrna\\_probs\\_window\(\)](#), [vrna\\_pfl\\_fold\(\)](#), [vrna\\_pfl\\_fold\\_up\\_cb\(\)](#)

**Parameters**

<i>sequence</i>	The nucleic acid input sequence
<i>window_size</i>	The size of the sliding window
<i>max_bp_span</i>	The maximum distance along the backbone between two nucleotides that form a base pairs
<i>cb</i>	The callback function which collects the pair probability data for further processing
<i>data</i>	Some arbitrary data structure that is passed to the callback <i>cb</i>

**Returns**

0 on failure, non-zero on success

**15.21.4.4 vrna\_pfl\_fold\_up()**

```
double** vrna_pfl_fold_up (
    const char * sequence,
```

```
int ulength,
int window_size,
int max_bp_span )
```

```
#include <ViennaRNA/part_func_window.h>
```

Compute probability of contiguous unpaired segments.

This is a simplified wrapper to [vrna\\_probs\\_window\(\)](#) that given a nucleic acid sequence, a maximum length of unpaired segments (`ulength`), a window size, and a maximum base pair span computes the equilibrium probability of any segment not exceeding `ulength`. The probabilities to be unpaired are returned as a 1-based, 2-dimensional matrix with dimensions  $N \times M$ , where  $N$  is the length of the sequence and  $M$  is the maximum segment length. As an example, the probability of a segment of size 5 starting at position 100 is stored in the matrix entry  $X[100][5]$ .

It is the users responsibility to free the memory occupied by this matrix.

#### Note

This function uses default model settings! For custom model settings, we refer to the function [vrna\\_probs\\_window\(\)](#).

#### Parameters

<i>sequence</i>	The nucleic acid input sequence
<i>ulength</i>	The maximal length of an unpaired segment
<i>window_size</i>	The size of the sliding window
<i>max_bp_span</i>	The maximum distance along the backbone between two nucleotides that form a base pairs

#### Returns

The probabilities to be unpaired for any segment not exceeding `ulength`

#### 15.21.4.5 vrna\_pfl\_fold\_up\_cb()

```
int vrna_pfl_fold_up_cb (
    const char * sequence,
    int ulength,
    int window_size,
    int max_bp_span,
    vrna_probs_window_callback * cb,
    void * data )
```

```
#include <ViennaRNA/part_func_window.h>
```

Compute probability of contiguous unpaired segments.

This is a simplified wrapper to [vrna\\_probs\\_window\(\)](#) that given a nucleic acid sequence, a maximum length of unpaired segments (`ulength`), a window size, and a maximum base pair span computes the equilibrium probability of any segment not exceeding `ulength`. It is similar to [vrna\\_pfl\\_fold\\_up\(\)](#) but uses a callback mechanism to return the unpaired probabilities.

Read the details for [vrna\\_probs\\_window\(\)](#) for details on the callback implementation!

**Note**

This function uses default model settings! For custom model settings, we refer to the function [vrna\\_probs\\_window\(\)](#).

**Parameters**

<i>sequence</i>	The nucleic acid input sequence
<i>ulength</i>	The maximal length of an unpaired segment
<i>window_size</i>	The size of the sliding window
<i>max_bp_span</i>	The maximum distance along the backbone between two nucleotides that form a base pairs
<i>cb</i>	The callback function which collects the pair probability data for further processing
<i>data</i>	Some arbitrary data structure that is passed to the callback <code>cb</code>

**Returns**

0 on failure, non-zero on success

## 15.22 Suboptimals and Representative Structures

Sample and enumerate suboptimal secondary structures from RNA sequence data.

### 15.22.1 Detailed Description

Sample and enumerate suboptimal secondary structures from RNA sequence data.

Collaboration diagram for Suboptimals and Representative Structures:

#### Modules

- [Suboptimal Structures sensu Stiegler et al. 1984 / Zuker et al. 1989](#)
- [Suboptimal Structures within an Energy Band around the MFE](#)
- [Random Structure Samples from the Ensemble](#)  
*Functions to draw random structure samples from the ensemble according to their equilibrium probability.*
- [Compute the Structure with Maximum Expected Accuracy \(MEA\)](#)
- [Compute the Centroid Structure](#)

#### Files

- file [boltzmann\\_sampling.h](#)  
*Boltzmann Sampling of secondary structures from the ensemble.*
- file [centroid.h](#)  
*Centroid structure computation.*
- file [MEA.h](#)  
*Computes a MEA (maximum expected accuracy) structure.*
- file [mm.h](#)  
*Several Maximum Matching implementations.*
- file [subopt.h](#)  
*RNAsubopt and density of states declarations.*

## 15.23 Suboptimal Structures sensu Stiegler et al. 1984 / Zuker et al. 1989

### 15.23.1 Detailed Description

Collaboration diagram for Suboptimal Structures sensu Stiegler et al. 1984 / Zuker et al. 1989:

#### Functions

- `vrna_subopt_solution_t * vrna_subopt_zuker (vrna_fold_compound_t *vc)`  
*Compute Zuker type suboptimal structures.*
- `SOLUTION * zukersubopt (const char *string)`  
*Compute Zuker type suboptimal structures.*
- `SOLUTION * zukersubopt_par (const char *string, vrna_param_t *parameters)`  
*Compute Zuker type suboptimal structures.*

### 15.23.2 Function Documentation

#### 15.23.2.1 `vrna_subopt_zuker()`

```
vrna_subopt_solution_t * vrna_subopt_zuker (
    vrna_fold_compound_t * vc )
```

```
#include <ViennaRNA/subopt.h>
```

Compute Zuker type suboptimal structures.

Compute Suboptimal structures according to M. Zuker [26] , i.e. for every possible base pair the minimum energy structure containing the resp. base pair. Returns a list of these structures and their energies.

#### Note

This function internally uses the cofold implementation to compute the suboptimal structures. For that purpose, the function doubles the sequence and enlarges the DP matrices, which in fact will grow by a factor of 4 during the computation! At the end of the structure prediction, everything will be re-set to its original requirements, i.e. normal sequence, normal (empty) DP matrices.

**Bug** Due to resizing, any pre-existing constraints will be lost!

#### See also

`vrna_subopt()`, `zukersubopt()`, `zukersubopt_par()`

## Parameters

vc	fold compound
----	---------------

## Returns

List of zuker suboptimal structures

**SWIG Wrapper Notes** This function is attached as method **subopt\_zuker()** to objects of type *fold\_compound*

## 15.23.2.2 zuckersubopt()

```
SOLUTION* zuckersubopt (
    const char * string )
```

```
#include <ViennaRNA/subopt.h>
```

Compute Zuker type suboptimal structures.

Compute Suboptimal structures according to M. Zuker, i.e. for every possible base pair the minimum energy structure containing the resp. base pair. Returns a list of these structures and their energies.

**Deprecated** use `vrna_zuckersubopt()` instead

## Parameters

<i>string</i>	RNA sequence
---------------	--------------

## Returns

List of zuker suboptimal structures

## 15.23.2.3 zuckersubopt\_par()

```
SOLUTION* zuckersubopt_par (
    const char * string,
    vrna_param_t * parameters )
```

```
#include <ViennaRNA/subopt.h>
```

Compute Zuker type suboptimal structures.

**Deprecated** use `vrna_zuckersubopt()` instead

## 15.24 Suboptimal Structures within an Energy Band around the MFE

### 15.24.1 Detailed Description

Collaboration diagram for Suboptimal Structures within an Energy Band around the MFE:

#### Typedefs

- typedef void() [vrna\\_subopt\\_callback](#)(const char \*structure, float energy, void \*data)  
Callback for [vrna\\_subopt\\_cb\(\)](#)

#### Functions

- [vrna\\_subopt\\_solution\\_t](#) \* [vrna\\_subopt](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int delta, int sorted, FILE \*fp)  
Returns list of subopt structures or writes to fp.
- void [vrna\\_subopt\\_cb](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int delta, [vrna\\_subopt\\_callback](#) \*cb, void \*data)  
Generate suboptimal structures within an energy band around the MFE.
- [SOLUTION](#) \* [subopt](#) (char \*seq, char \*structure, int delta, FILE \*fp)  
Returns list of subopt structures or writes to fp.
- [SOLUTION](#) \* [subopt\\_par](#) (char \*seq, char \*structure, [vrna\\_param\\_t](#) \*parameters, int delta, int is\_[↔](#)constrained, int is\_circular, FILE \*fp)  
Returns list of subopt structures or writes to fp.
- [SOLUTION](#) \* [subopt\\_circ](#) (char \*seq, char \*sequence, int delta, FILE \*fp)  
Returns list of circular subopt structures or writes to fp.

#### Variables

- double [print\\_energy](#)  
printing threshold for use with logML
- int [subopt\\_sorted](#)  
Sort output by energy.

### 15.24.2 Typedef Documentation

#### 15.24.2.1 vrna\_subopt\_callback

```
typedef void() vrna_subopt_callback(const char *structure, float energy, void *data)
```

```
#include <ViennaRNA/subopt.h>
```

Callback for [vrna\\_subopt\\_cb\(\)](#)

**Notes on Callback Functions** This function will be called for each suboptimal secondary structure that is successfully backtraced.

See also

[vrna\\_subopt\\_cb\(\)](#)



## Parameters

<i>structure</i>	The suboptimal secondary structure in dot-bracket notation
<i>energy</i>	The free energy of the secondary structure in kcal/mol
<i>data</i>	Some arbitrary, auxiliary data address as passed to <a href="#">vrna_subopt_cb()</a>

## 15.24.3 Function Documentation

15.24.3.1 [vrna\\_subopt\(\)](#)

```
vrna_subopt_solution_t * vrna_subopt (
    vrna_fold_compound_t * vc,
    int delta,
    int sorted,
    FILE * fp )
```

```
#include <ViennaRNA/subopt.h>
```

Returns list of subopt structures or writes to fp.

This function produces **all** suboptimal secondary structures within 'delta' \* 0.01 kcal/mol of the optimum, see [24]. The results are either directly written to a 'fp' (if 'fp' is not NULL), or (fp==NULL) returned in a [vrna\\_subopt\\_solution\\_t](#) \* list terminated by an entry where the 'structure' member is NULL.

## Note

This function requires all multibranch loop DP matrices for unique multibranch loop backtracing. Therefore, the supplied [vrna\\_fold\\_compound\\_t](#) *vc* (argument 1) must be initialized with [vrna\\_md\\_t.uniq\\_ML](#) = 1, for instance like this:

```
vrna_md_t md;
vrna_md_set_default(&md);
md.uniq_ML = 1;

vrna_fold_compound_t *vc=vrna_fold_compound("GGGGGAAAAACCCCC", &md
    , VRNA_OPTION_DEFAULT);
```

## See also

[vrna\\_subopt\\_cb\(\)](#), [vrna\\_subopt\\_zuker\(\)](#)

## Parameters

<i>vc</i>	
<i>delta</i>	
<i>sorted</i>	Sort results by energy in ascending order
<i>fp</i>	

## Returns

**SWIG Wrapper Notes** This function is attached as method **subopt()** to objects of type *fold\_compound*

## 15.24.3.2 vrna\_subopt\_cb()

```
void vrna_subopt_cb (
    vrna_fold_compound_t * vc,
    int delta,
    vrna_subopt_callback * cb,
    void * data )
```

```
#include <ViennaRNA/subopt.h>
```

Generate suboptimal structures within an energy band around the MFE.

This is the most generic implementation of the suboptimal structure generator according to Wuchty et al. 1999 [24]. Identical to **vrna\_subopt()**, it computes all secondary structures within an energy band *delta* around the MFE. However, this function does not print the resulting structures and their corresponding free energies to a file pointer, or returns them as a list. Instead, it calls a user-provided callback function which it passes the structure in dot-bracket format, the corresponding free energy in kcal/mol, and a user-provided data structure each time a structure was backtracked successfully. This function indicates the final output, i.e. the end of the backtracking procedure by passing NULL instead of an actual dot-bracket string to the callback.

## Note

This function requires all multibranch loop DP matrices for unique multibranch loop backtracing. Therefore, the supplied **vrna\_fold\_compound\_t** *vc* (argument 1) must be initialized with **vrna\_md\_t.uniq\_ML** = 1, for instance like this:

```
vrna_md_t md;
vrna_md_set_default (&md);
md.uniq_ML = 1;

vrna_fold_compound_t *vc=vrna_fold_compound("GGGGGAAAAACCCCC", &md
    , VRNA_OPTION_DEFAULT);
```

## See also

**vrna\_subopt\_callback**, **vrna\_subopt()**, **vrna\_subopt\_zuker()**

## Parameters

<i>vc</i>	fold compound with the sequence data
<i>delta</i>	Energy band around the MFE in 10cal/mol, i.e. deka-calories
<i>cb</i>	Pointer to a callback function that handles the backtracked structure and its free energy in kcal/mol
<i>data</i>	Pointer to some data structure that is passed along to the callback

**SWIG Wrapper Notes** This function is attached as method **subopt\_cb()** to objects of type *fold\_compound*

### 15.24.3.3 subopt()

```
SOLUTION* subopt (
    char * seq,
    char * structure,
    int delta,
    FILE * fp )

#include <ViennaRNA/subopt.h>
```

Returns list of subopt structures or writes to fp.

This function produces **all** suboptimal secondary structures within 'delta' \* 0.01 kcal/mol of the optimum. The results are either directly written to a 'fp' (if 'fp' is not NULL), or (fp==NULL) returned in a **SOLUTION** \* list terminated by an entry where the 'structure' pointer is NULL.

#### Parameters

<i>seq</i>	
<i>structure</i>	
<i>delta</i>	
<i>fp</i>	

#### Returns

### 15.24.3.4 subopt\_circ()

```
SOLUTION* subopt_circ (
    char * seq,
    char * sequence,
    int delta,
    FILE * fp )

#include <ViennaRNA/subopt.h>
```

Returns list of circular subopt structures or writes to fp.

This function is similar to [subopt\(\)](#) but calculates secondary structures assuming the RNA sequence to be circular instead of linear

#### Parameters

<i>seq</i>	
<i>sequence</i>	
<i>delta</i>	
<i>fp</i>	

Returns

## 15.25 Random Structure Samples from the Ensemble

Functions to draw random structure samples from the ensemble according to their equilibrium probability.

### 15.25.1 Detailed Description

Functions to draw random structure samples from the ensemble according to their equilibrium probability.

Collaboration diagram for Random Structure Samples from the Ensemble:

#### Modules

- [Stochastic Backtracking of Structures from Distance Based Partitioning](#)  
*Contains functions related to stochastic backtracking from a specified distance class.*

#### Functions

- `char * vrna_pbacktrack5 (vrna_fold_compound_t *vc, int length)`  
*Sample a secondary structure of a subsequence from the Boltzmann ensemble according its probability.*
- `char ** vrna_pbacktrack_nr (vrna_fold_compound_t *vc, int num_samples)`  
*Samples multiple secondary structures non-redundantly from the Boltzmann ensemble according its probability.*
- `char * vrna_pbacktrack (vrna_fold_compound_t *vc)`  
*Sample a secondary structure (consensus structure) from the Boltzmann ensemble according its probability.*
- `char * pbacktrack (char *sequence)`  
*Sample a secondary structure from the Boltzmann ensemble according its probability.*
- `char * pbacktrack_circ (char *sequence)`  
*Sample a secondary structure of a circular RNA from the Boltzmann ensemble according its probability.*

#### Variables

- `int st_back`  
*Flag indicating that auxiliary arrays are needed throughout the computations. This is essential for stochastic backtracking.*

### 15.25.2 Function Documentation

#### 15.25.2.1 vrna\_pbacktrack5()

```
char * vrna_pbacktrack5 (
    vrna_fold_compound_t * vc,
    int length )

#include <ViennaRNA/boltzmann_sampling.h>
```

Sample a secondary structure of a subsequence from the Boltzmann ensemble according its probability.

#### Precondition

Unique multiloop decomposition has to be active upon creation of `vc` with `vrna_fold_compound()` or similar. This can be done easily by passing `vrna_fold_compound()` a model details parameter with `vrna_md_t.uniq_ML = 1`.

`vrna_pf()` has to be called first to fill the partition function matrices

## Parameters

<i>vc</i>	The fold compound data structure
<i>length</i>	The length of the subsequence to consider (starting with 5' end)

## Returns

A sampled secondary structure in dot-bracket notation (or NULL on error)

**SWIG Wrapper Notes** This function is attached as overloaded method **pbacktrack()** to objects of type *fold\_compound*

15.25.2.2 `vrna_pbacktrack_nr()`

```
char * vrna_pbacktrack_nr (
    vrna_fold_compound_t * vc,
    int num_samples )
```

```
#include <ViennaRNA/boltzmann_sampling.h>
```

Samples multiple secondary structures non-redundantly from the Boltzmann ensemble according its probability.

## Precondition

The fold compound has to be obtained using the [VRNA\\_OPTION\\_HYBRID](#) option in `vrna_fold_compound()` `vrna_pf()` has to be called first to fill the partition function matrices

## Note

In some cases, this function does not return the number of requested samples but a smaller number. This may happen if a) the number of requested structures is larger than the total number of structures in the ensemble, or b) numeric instabilities prevent the backtracking function to enumerate structures with very high free energies.

## Parameters

<i>vc</i>	The fold compound data structure
<i>num_samples</i>	The number of desired non-redundant samples

## Returns

A list of sampled secondary structures in dot-bracket notation, terminated by *NULL*

**SWIG Wrapper Notes** This function is attached as method **pbacktrack\_nr()** to objects of type *fold\_compound*.

15.25.2.3 `vrna_pbacktrack()`

```
char * vrna_pbacktrack (
    vrna_fold_compound_t * vc )

#include <ViennaRNA/boltzmann_sampling.h>
```

Sample a secondary structure (consensus structure) from the Boltzmann ensemble according its probability.

**Precondition**

Unique multiloop decomposition has to be active upon creation of `vc` with `vrna_fold_compound()` or similar. This can be done easily by passing `vrna_fold_compound()` a model details parameter with `vrna_md_t.uniq_ML = 1`. `vrna_pf()` has to be called first to fill the partition function matrices

**Note**

This function is polymorphic. It accepts `vrna_fold_compound_t` of type `VRNA_FC_TYPE_SINGLE`, and `VRNA_FC_TYPE_COMPARATIVE`. The function will automagically detect cicular RNAs based on the model\_details in `exp_params` as provided via the `vrna_fold_compound_t`

**Parameters**

<code>vc</code>	The fold compound data structure
-----------------	----------------------------------

**Returns**

A sampled secondary structure in dot-bracket notation (or NULL on error)

**SWIG Wrapper Notes** This function is attached as overloaded method `pbacktrack()` to objects of type `fold_compound` that accepts an optional `length` argument. Hence, it serves as a replacement for `vrna_pbacktrack()`.

15.25.2.4 `pbacktrack()`

```
char* pbacktrack (
    char * sequence )

#include <ViennaRNA/part_func.h>
```

Sample a secondary structure from the Boltzmann ensemble according its probability.

**Precondition**

`st_back` has to be set to 1 before calling `pf_fold()` or `pf_fold_par()`  
`pf_fold_par()` or `pf_fold()` have to be called first to fill the partition function matrices

**Parameters**

<i>sequence</i>	The RNA sequence
-----------------	------------------

**Returns**

A sampled secondary structure in dot-bracket notation

**15.25.2.5 pbacktrack\_circ()**

```
char* pbacktrack_circ (
    char * sequence )
```

```
#include <ViennaRNA/part_func.h>
```

Sample a secondary structure of a circular RNA from the Boltzmann ensemble according its probability.

This function does the same as [pbacktrack\(\)](#) but assumes the RNA molecule to be circular

**Precondition**

[st\\_back](#) has to be set to 1 before calling [pf\\_fold\(\)](#) or [pf\\_fold\\_par\(\)](#)  
[pf\\_fold\\_par\(\)](#) or [pf\\_circ\\_fold\(\)](#) have to be called first to fill the partition function matrices

**Deprecated** Use [vrna\\_pbacktrack\(\)](#) instead.

**Parameters**

<i>sequence</i>	The RNA sequence
-----------------	------------------

**Returns**

A sampled secondary structure in dot-bracket notation

**15.25.3 Variable Documentation****15.25.3.1 st\_back**

```
int st_back
```

```
#include <ViennaRNA/part_func.h>
```

Flag indicating that auxiliary arrays are needed throughout the computations. This is essential for stochastic backtracking.

Set this variable to 1 prior to a call of [pf\\_fold\(\)](#) to ensure that all matrices needed for stochastic backtracking are filled in the forward recursions



**Deprecated** set the *uniq\_ML* flag in `vrna_md_t` before passing it to `vrna_fold_compound()`.

See also

`pbacktrack()`, `pbacktrack_circ`

## 15.26 Compute the Structure with Maximum Expected Accuracy (MEA)

### 15.26.1 Detailed Description

Collaboration diagram for Compute the Structure with Maximum Expected Accuracy (MEA):

### Functions

- float [MEA](#) ([plist](#) \*p, char \*structure, double gamma)  
*Computes a MEA (maximum expected accuracy) structure.*

### 15.26.2 Function Documentation

#### 15.26.2.1 MEA()

```
float MEA (
    plist * p,
    char * structure,
    double gamma )

#include <ViennaRNA/MEA.h>
```

Computes a MEA (maximum expected accuracy) structure.

The algorithm maximizes the expected accuracy

$$A(S) = \sum_{(i,j) \in S} 2\gamma p_{ij} + \sum_{i \notin S} p_i^u$$

Higher values of  $\gamma$  result in more base pairs of lower probability and thus higher sensitivity. Low values of  $\gamma$  result in structures containing only highly likely pairs (high specificity). The code of the MEA function also demonstrates the use of sparse dynamic programming scheme to reduce the time and memory complexity of folding.

## 15.27 Compute the Centroid Structure

### 15.27.1 Detailed Description

Collaboration diagram for Compute the Centroid Structure:

#### Functions

- char \* [vrna\\_centroid](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, double \*dist)  
*Get the centroid structure of the ensemble.*
- char \* [vrna\\_centroid\\_from\\_plist](#) (int length, double \*dist, [vrna\\_ep\\_t](#) \*pl)  
*Get the centroid structure of the ensemble.*
- char \* [vrna\\_centroid\\_from\\_probs](#) (int length, double \*dist, [FLT\\_OR\\_DBL](#) \*probs)  
*Get the centroid structure of the ensemble.*

### 15.27.2 Function Documentation

#### 15.27.2.1 vrna\_centroid()

```
char* vrna_centroid (
    vrna_fold_compound_t * vc,
    double * dist )
```

```
#include <ViennaRNA/centroid.h>
```

Get the centroid structure of the ensemble.

The centroid is the structure with the minimal average distance to all other structures

$$< d(S) > = \sum_{(i,j) \in S} (1 - p_{ij}) + \sum_{(i,j) \notin S} p_{ij}$$

Thus, the centroid is simply the structure containing all pairs with  $p_{ij} > 0.5$ . The distance of the centroid to the ensemble is written to the memory addressed by *dist*.

#### Parameters

in	<i>vc</i>	The fold compound data structure
out	<i>dist</i>	A pointer to the distance variable where the centroid distance will be written to

#### Returns

The centroid structure of the ensemble in dot-bracket notation (NULL on error)

## 15.27.2.2 vrna\_centroid\_from\_plist()

```
char* vrna_centroid_from_plist (
    int length,
    double * dist,
    vrna_ep_t * pl )

#include <ViennaRNA/centroid.h>
```

Get the centroid structure of the ensemble.

This function is a threadsafe replacement for [centroid\(\)](#) with a [vrna\\_ep\\_t](#) input

The centroid is the structure with the minimal average distance to all other structures

$$\langle d(S) \rangle = \sum_{(i,j) \in S} (1 - p_{ij}) + \sum_{(i,j) \notin S} p_{ij}$$

Thus, the centroid is simply the structure containing all pairs with  $p_{ij} > 0.5$ . The distance of the centroid to the ensemble is written to the memory addressed by *dist*.

## Parameters

in	<i>length</i>	The length of the sequence
out	<i>dist</i>	A pointer to the distance variable where the centroid distance will be written to
in	<i>pl</i>	A pair list containing base pair probability information about the ensemble

## Returns

The centroid structure of the ensemble in dot-bracket notation (NULL on error)

## 15.27.2.3 vrna\_centroid\_from\_probs()

```
char* vrna_centroid_from_probs (
    int length,
    double * dist,
    FLT_OR_DBL * probs )

#include <ViennaRNA/centroid.h>
```

Get the centroid structure of the ensemble.

This function is a threadsafe replacement for [centroid\(\)](#) with a probability array input

The centroid is the structure with the minimal average distance to all other structures

$$\langle d(S) \rangle = \sum_{(i,j) \in S} (1 - p_{ij}) + \sum_{(i,j) \notin S} p_{ij}$$

Thus, the centroid is simply the structure containing all pairs with  $p_{ij} > 0.5$ . The distance of the centroid to the ensemble is written to the memory addressed by *dist*.

## Parameters

in	<i>length</i>	The length of the sequence
out	<i>dist</i>	A pointer to the distance variable where the centroid distance will be written to
in	<i>probs</i>	An upper triangular matrix containing base pair probabilities (access via <a href="#">iindx_vrna_idx_row_wise()</a> )

**Returns**

The centroid structure of the ensemble in dot-bracket notation (`NULL` on error)

## 15.28 RNA-RNA Interaction

### 15.28.1 Detailed Description

Collaboration diagram for RNA-RNA Interaction:

#### Modules

- [Partition Function for Two Hybridized Sequences](#)  
*Partition Function Cofolding.*
- [Partition Function for two Hybridized Sequences as a Stepwise Process](#)  
*RNA-RNA interaction as a stepwise process.*

#### Files

- file [concentrations.h](#)  
*Concentration computations for RNA-RNA interactions.*
- file [duplex.h](#)  
*Functions for simple RNA-RNA duplex interactions.*
- file [part\\_func\\_up.h](#)  
*Implementations for accessibility and RNA-RNA interaction as a stepwise process.*

## 15.29 Classified Dynamic Programming Variants

### 15.29.1 Detailed Description

Collaboration diagram for Classified Dynamic Programming Variants:

#### Modules

- [Distance Based Partitioning of the Secondary Structure Space](#)
- [Compute the Density of States](#)

## 15.30 Distance Based Partitioning of the Secondary Structure Space

### 15.30.1 Detailed Description

Collaboration diagram for Distance Based Partitioning of the Secondary Structure Space:

#### Modules

- [Computing MFE representatives of a Distance Based Partitioning](#)  
*Compute the minimum free energy (MFE) and secondary structures for a partitioning of the secondary structure space according to the base pair distance to two fixed reference structures basepair distance to two fixed reference structures.*
- [Computing Partition Functions of a Distance Based Partitioning](#)  
*Compute the partition function and stochastically sample secondary structures for a partitioning of the secondary structure space according to the base pair distance to two fixed reference structures.*
- [Stochastic Backtracking of Structures from Distance Based Partitioning](#)  
*Contains functions related to stochastic backtracking from a specified distance class.*

#### Files

- file [2Dfold.h](#)  
*MFE structures for base pair distance classes.*
- file [2Dpfold.h](#)  
*Partition function implementations for base pair distance classes.*



## 15.31 Computing MFE representatives of a Distance Based Partitioning

Compute the minimum free energy (MFE) and secondary structures for a partitioning of the secondary structure space according to the base pair distance to two fixed reference structures basepair distance to two fixed reference structures.

### 15.31.1 Detailed Description

Compute the minimum free energy (MFE) and secondary structures for a partitioning of the secondary structure space according to the base pair distance to two fixed reference structures basepair distance to two fixed reference structures.

See also

For further details, we refer to Lorenz et al. 2009 [[14](#)]

Collaboration diagram for Computing MFE representatives of a Distance Based Partitioning:

### Data Structures

- struct [vrna\\_sol\\_TwoD\\_t](#)  
*Solution element returned from [vrna\\_mfe\\_TwoD\(\)](#) [More...](#)*
- struct [TwoDfold\\_vars](#)  
*Variables compound for 2Dfold MFE folding. [More...](#)*

### Typedefs

- typedef struct [vrna\\_sol\\_TwoD\\_t](#) [vrna\\_sol\\_TwoD\\_t](#)  
*Solution element returned from [vrna\\_mfe\\_TwoD\(\)](#)*
- typedef struct [TwoDfold\\_vars](#) [TwoDfold\\_vars](#)  
*Variables compound for 2Dfold MFE folding.*

### Functions

- [vrna\\_sol\\_TwoD\\_t](#) \* [vrna\\_mfe\\_TwoD](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int distance1, int distance2)  
*Compute MFE's and representative for distance partitioning.*
- char \* [vrna\\_backtrack5\\_TwoD](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int k, int l, unsigned int j)  
*Backtrack a minimum free energy structure from a 5' section of specified length.*
- [TwoDfold\\_vars](#) \* [get\\_TwoDfold\\_variables](#) (const char \*seq, const char \*structure1, const char \*structure2, int circ)  
*Get a structure of type [TwoDfold\\_vars](#) prefilled with current global settings.*
- void [destroy\\_TwoDfold\\_variables](#) ([TwoDfold\\_vars](#) \*our\_variables)  
*Destroy a [TwoDfold\\_vars](#) datastructure without memory loss.*
- [vrna\\_sol\\_TwoD\\_t](#) \* [TwoDfoldList](#) ([TwoDfold\\_vars](#) \*vars, int distance1, int distance2)  
*Compute MFE's and representative for distance partitioning.*
- char \* [TwoDfold\\_backtrack\\_f5](#) (unsigned int j, int k, int l, [TwoDfold\\_vars](#) \*vars)  
*Backtrack a minimum free energy structure from a 5' section of specified length.*

## 15.31.2 Data Structure Documentation

### 15.31.2.1 struct vrna\_sol\_TwoD\_t

Solution element returned from [vrna\\_mfe\\_TwoD\(\)](#)

This element contains free energy and structure for the appropriate kappa (k), lambda (l) neighborhood. The data structure contains two integer attributes 'k' and 'l' as well as an attribute 'en' of type float representing the free energy in kcal/mol and an attribute 's' of type char\* containing the secondary structure representative,

A value of [INF](#) in k denotes the end of a list

See also

[vrna\\_mfe\\_TwoD\(\)](#)

#### Data Fields

- int [k](#)  
*Distance to first reference.*
- int [l](#)  
*Distance to second reference.*
- float [en](#)  
*Free energy in kcal/mol.*
- char \* [s](#)  
*MFE representative structure in dot-bracket notation.*

### 15.31.2.2 struct TwoDfold\_vars

Variables compound for 2Dfold MFE folding.

**Deprecated** This data structure will be removed from the library soon! Use [vrna\\_fold\\_compound\\_t](#) and the corresponding functions [vrna\\_fold\\_compound\\_TwoD\(\)](#), [vrna\\_mfe\\_TwoD\(\)](#), and [vrna\\_fold\\_compound\\_free\(\)](#) instead!

Collaboration diagram for TwoDfold\_vars:

## Data Fields

- `vrna_param_t * P`  
*Precomputed energy parameters and model details.*
- `int do_backtrack`  
*Flag whether to do backtracing of the structure(s) or not.*
- `char * ptype`  
*Precomputed array of pair types.*
- `char * sequence`  
*The input sequence.*
- `short * S1`  
*The input sequences in numeric form.*
- `unsigned int maxD1`  
*Maximum allowed base pair distance to first reference.*
- `unsigned int maxD2`  
*Maximum allowed base pair distance to second reference.*
- `unsigned int * mm1`  
*Maximum matching matrix, reference struct 1 disallowed.*
- `unsigned int * mm2`  
*Maximum matching matrix, reference struct 2 disallowed.*
- `int * my_iindx`  
*Index for moving in quadratic distance dimensions.*
- `unsigned int * referenceBPs1`  
*Matrix containing number of basepairs of reference structure1 in interval [i,j].*
- `unsigned int * referenceBPs2`  
*Matrix containing number of basepairs of reference structure2 in interval [i,j].*
- `unsigned int * bpdist`  
*Matrix containing base pair distance of reference structure 1 and 2 on interval [i,j].*

## 15.31.3 Typedef Documentation

15.31.3.1 `vrna_sol_TwoD_t`

```
typedef struct vrna_sol_TwoD_t vrna_sol_TwoD_t

#include <ViennaRNA/2Dfold.h>
```

Solution element returned from `vrna_mfe_TwoD()`

This element contains free energy and structure for the appropriate kappa (k), lambda (l) neighborhood. The data-structure contains two integer attributes 'k' and 'l' as well as an attribute 'en' of type float representing the free energy in kcal/mol and an attribute 's' of type char\* containing the secondary structure representative,

A value of `INF` in k denotes the end of a list

See also

`vrna_mfe_TwoD()`

### 15.31.3.2 TwoDfold\_vars

```
typedef struct TwoDfold_vars TwoDfold_vars
```

```
#include <ViennaRNA/2Dfold.h>
```

Variables compound for 2Dfold MFE folding.

**Deprecated** This data structure will be removed from the library soon! Use [vrna\\_fold\\_compound\\_t](#) and the corresponding functions [vrna\\_fold\\_compound\\_TwoD\(\)](#), [vrna\\_mfe\\_TwoD\(\)](#), and [vrna\\_fold\\_compound\\_free\(\)](#) instead!

## 15.31.4 Function Documentation

### 15.31.4.1 vrna\_mfe\_TwoD()

```
vrna_sol_TwoD_t* vrna_mfe_TwoD (
    vrna_fold_compound_t * vc,
    int distance1,
    int distance2 )
```

```
#include <ViennaRNA/2Dfold.h>
```

Compute MFE's and representative for distance partitioning.

This function computes the minimum free energies and a representative secondary structure for each distance class according to the two references specified in the datastructure 'vars'. The maximum basepair distance to each of both references may be set by the arguments 'distance1' and 'distance2', respectively. If both distance arguments are set to '-1', no restriction is assumed and the calculation is performed for each distance class possible.

The returned list contains an entry for each distance class. If a maximum basepair distance to either of the references was passed, an entry with  $k=-1$  will be appended in the list, denoting the class where all structures exceeding the maximum will be thrown into. The end of the list is denoted by an attribute value of [INF](#) in the k-attribute of the list entry.

See also

[vrna\\_fold\\_compound\\_TwoD\(\)](#), [vrna\\_fold\\_compound\\_free\(\)](#), [vrna\\_pf\\_TwoD\(\)](#) [vrna\\_backtrack5\\_TwoD\(\)](#),  
[vrna\\_sol\\_TwoD\\_t](#), [vrna\\_fold\\_compound\\_t](#)

#### Parameters

<i>vc</i>	The datastructure containing all precomputed folding attributes
<i>distance1</i>	maximum distance to reference1 (-1 means no restriction)
<i>distance2</i>	maximum distance to reference2 (-1 means no restriction)

**Returns**

A list of minimum free energies (and corresponding structures) for each distance class

**15.31.4.2 vrna\_backtrack5\_TwoD()**

```
char* vrna_backtrack5_TwoD (
    vrna_fold_compound_t * vc,
    int k,
    int l,
    unsigned int j )
```

```
#include <ViennaRNA/2Dfold.h>
```

Backtrack a minimum free energy structure from a 5' section of specified length.

This function allows one to backtrack a secondary structure beginning at the 5' end, a specified length and residing in a specific distance class. If the argument 'k' gets a value of -1, the structure that is backtracked is assumed to reside in the distance class where all structures exceeding the maximum basepair distance specified in [vrna\\_mfe\\_TwoD\(\)](#) belong to.

**Note**

The argument 'vars' must contain precalculated energy values in the energy matrices, i.e. a call to [vrna\\_mfe\\_TwoD\(\)](#) preceding this function is mandatory!

**See also**

[vrna\\_mfe\\_TwoD\(\)](#)

**Parameters**

<i>vc</i>	The datastructure containing all precomputed folding attributes
<i>j</i>	The length in nucleotides beginning from the 5' end
<i>k</i>	distance to reference1 (may be -1)
<i>l</i>	distance to reference2

**15.31.4.3 get\_TwoDfold\_variables()**

```
TwoDfold_vars* get_TwoDfold_variables (
    const char * seq,
    const char * structure1,
    const char * structure2,
    int circ )
```

```
#include <ViennaRNA/2Dfold.h>
```

Get a structure of type [TwoDfold\\_vars](#) prefilled with current global settings.

This function returns a datastructure of type [TwoDfold\\_vars](#). The data fields inside the [TwoDfold\\_vars](#) are prefilled by global settings and all memory allocations necessary to start a computation are already done for the convenience of the user

#### Note

Make sure that the reference structures are compatible with the sequence according to Watson-Crick- and Wobble-base pairing

**Deprecated** Use the new API that relies on [vrna\\_fold\\_compound\\_t](#) and the corresponding functions [vrna\\_fold\\_compound\\_TwoD\(\)](#), [vrna\\_mfe\\_TwoD\(\)](#), and [vrna\\_fold\\_compound\\_free\(\)](#) instead!

#### Parameters

<i>seq</i>	The RNA sequence
<i>structure1</i>	The first reference structure in dot-bracket notation
<i>structure2</i>	The second reference structure in dot-bracket notation
<i>circ</i>	A switch to indicate the assumption to fold a circular instead of linear RNA (0=OFF, 1=ON)

#### Returns

A datastructure prefilled with folding options and allocated memory

#### 15.31.4.4 `destroy_TwoDfold_variables()`

```
void destroy_TwoDfold_variables (
    TwoDfold\_vars * our_variables )
```

```
#include <ViennaRNA/2Dfold.h>
```

Destroy a [TwoDfold\\_vars](#) datastructure without memory loss.

This function free's all allocated memory that depends on the datastructure given.

**Deprecated** Use the new API that relies on [vrna\\_fold\\_compound\\_t](#) and the corresponding functions [vrna\\_fold\\_compound\\_TwoD\(\)](#), [vrna\\_mfe\\_TwoD\(\)](#), and [vrna\\_fold\\_compound\\_free\(\)](#) instead!

#### Parameters

<i>our_variables</i>	A pointer to the datastructure to be destroyed
----------------------	--

#### 15.31.4.5 `TwoDfoldList()`

```
vrna_sol_TwoD_t* TwoDfoldList (
    TwoDfold_vars * vars,
    int distance1,
    int distance2 )

#include <ViennaRNA/2Dfold.h>
```

Compute MFE's and representative for distance partitioning.

This function computes the minimum free energies and a representative secondary structure for each distance class according to the two references specified in the datastructure 'vars'. The maximum basepair distance to each of both references may be set by the arguments 'distance1' and 'distance2', respectively. If both distance arguments are set to '-1', no restriction is assumed and the calculation is performed for each distance class possible.

The returned list contains an entry for each distance class. If a maximum basepair distance to either of the references was passed, an entry with  $k=-1$  will be appended in the list, denoting the class where all structures exceeding the maximum will be thrown into. The end of the list is denoted by an attribute value of `INF` in the  $k$ -attribute of the list entry.

**Deprecated** Use the new API that relies on `vrna_fold_compound_t` and the corresponding functions `vrna_fold_compound_TwoD()`, `vrna_mfe_TwoD()`, and `vrna_fold_compound_free()` instead!

#### Parameters

<i>vars</i>	the datastructure containing all predefined folding attributes
<i>distance1</i>	maximum distance to reference1 (-1 means no restriction)
<i>distance2</i>	maximum distance to reference2 (-1 means no restriction)

#### 15.31.4.6 TwoDfold\_backtrack\_f5()

```
char* TwoDfold_backtrack_f5 (
    unsigned int j,
    int k,
    int l,
    TwoDfold_vars * vars )

#include <ViennaRNA/2Dfold.h>
```

Backtrack a minimum free energy structure from a 5' section of specified length.

This function allows one to backtrack a secondary structure beginning at the 5' end, a specified length and residing in a specific distance class. If the argument 'k' gets a value of -1, the structure that is backtracked is assumed to reside in the distance class where all structures exceeding the maximum basepair distance specified in `TwoDfold()` belong to.

#### Note

The argument 'vars' must contain precalculated energy values in the energy matrices, i.e. a call to `TwoDfold()` preceding this function is mandatory!

**Deprecated** Use the new API that relies on `vrna_fold_compound_t` and the corresponding functions `vrna_fold_compound_TwoD()`, `vrna_mfe_TwoD()`, `vrna_backtrack5_TwoD()`, and `vrna_fold_compound_free()` instead!

**Parameters**

<i>j</i>	The length in nucleotides beginning from the 5' end
<i>k</i>	distance to reference1 (may be -1)
<i>l</i>	distance to reference2
<i>vars</i>	the datastructure containing all predefined folding attributes



## 15.32 Computing Partition Functions of a Distance Based Partitioning

Compute the partition function and stochastically sample secondary structures for a partitioning of the secondary structure space according to the base pair distance to two fixed reference structures.

### 15.32.1 Detailed Description

Compute the partition function and stochastically sample secondary structures for a partitioning of the secondary structure space according to the base pair distance to two fixed reference structures.

Collaboration diagram for Computing Partition Functions of a Distance Based Partitioning:

### Data Structures

- struct [vrna\\_sol\\_TwoD\\_pf\\_t](#)  
Solution element returned from [vrna\\_pf\\_TwoD\(\)](#) [More...](#)

### Typedefs

- typedef struct [vrna\\_sol\\_TwoD\\_pf\\_t](#) [vrna\\_sol\\_TwoD\\_pf\\_t](#)  
Solution element returned from [vrna\\_pf\\_TwoD\(\)](#)

### Functions

- [vrna\\_sol\\_TwoD\\_pf\\_t](#) \* [vrna\\_pf\\_TwoD](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int maxDistance1, int maxDistance2)  
Compute the partition function for all distance classes.

### 15.32.2 Data Structure Documentation

#### 15.32.2.1 struct [vrna\\_sol\\_TwoD\\_pf\\_t](#)

Solution element returned from [vrna\\_pf\\_TwoD\(\)](#)

This element contains the partition function for the appropriate kappa (k), lambda (l) neighborhood. The datastructure contains two integer attributes 'k' and 'l' as well as an attribute 'q' of type [FLT\\_OR\\_DBL](#).

A value of [INF](#) in k denotes the end of a list.

See also

[vrna\\_pf\\_TwoD\(\)](#)

## Data Fields

- `int k`  
*Distance to first reference.*
- `int l`  
*Distance to second reference.*
- `FLT_OR_DBL q`  
*partition function*

## 15.32.3 Typedef Documentation

15.32.3.1 `vrna_sol_TwoD_pf_t`

```
typedef struct vrna_sol_TwoD_pf_t vrna_sol_TwoD_pf_t
#include <ViennaRNA/2Dpfold.h>
```

Solution element returned from `vrna_pf_TwoD()`

This element contains the partition function for the appropriate kappa (k), lambda (l) neighborhood. The datastructure contains two integer attributes 'k' and 'l' as well as an attribute 'q' of type `FLT_OR_DBL`.

A value of `INF` in k denotes the end of a list.

See also

`vrna_pf_TwoD()`

## 15.32.4 Function Documentation

15.32.4.1 `vrna_pf_TwoD()`

```
vrna_sol_TwoD_pf_t* vrna_pf_TwoD (
    vrna_fold_compound_t * vc,
    int maxDistance1,
    int maxDistance2 )
#include <ViennaRNA/2Dpfold.h>
```

Compute the partition function for all distance classes.

This function computes the partition functions for all distance classes according to the two reference structures specified in the datastructure 'vars'. Similar to `vrna_mfe_TwoD()` the arguments `maxDistance1` and `maxDistance2` specify the maximum distance to both reference structures. A value of '-1' in either of them makes the appropriate distance restrictionless, i.e. all basepair distances to the reference are taken into account during computation. In case there is a restriction, the returned solution contains an entry where the attribute `k=l=-1` contains the partition function for all structures exceeding the restriction. A value of `INF` in the attribute 'k' of the returned list denotes the end of the list.

See also

`vrna_fold_compound_TwoD()`, `vrna_fold_compound_free()`, `vrna_fold_compound`, `vrna_sol_TwoD_pf_t`

**Parameters**

<i>vc</i>	The datastructure containing all necessary folding attributes and matrices
<i>maxDistance1</i>	The maximum basepair distance to reference1 (may be -1)
<i>maxDistance2</i>	The maximum basepair distance to reference2 (may be -1)

**Returns**

A list of partition funtions for the corresponding distance classes

## 15.33 Stochastic Backtracking of Structures from Distance Based Partitioning

Contains functions related to stochastic backtracking from a specified distance class.

### 15.33.1 Detailed Description

Contains functions related to stochastic backtracking from a specified distance class.

Collaboration diagram for Stochastic Backtracking of Structures from Distance Based Partitioning:

### Functions

- char \* [vrna\\_pbacktrack\\_TwoD](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int d1, int d2)  
*Sample secondary structure representatives from a set of distance classes according to their Boltzmann probability.*
- char \* [vrna\\_pbacktrack5\\_TwoD](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int d1, int d2, unsigned int length)  
*Sample secondary structure representatives with a specified length from a set of distance classes according to their Boltzmann probability.*

### 15.33.2 Function Documentation

#### 15.33.2.1 vrna\_pbacktrack\_TwoD()

```
char* vrna_pbacktrack_TwoD (
    vrna\_fold\_compound\_t * vc,
    int d1,
    int d2 )
```

```
#include <ViennaRNA/2Dpfold.h>
```

Sample secondary structure representatives from a set of distance classes according to their Boltzmann probability.

If the argument 'd1' is set to '-1', the structure will be backtracked in the distance class where all structures exceeding the maximum basepair distance to either of the references reside.

#### Precondition

The argument 'vars' must contain precalculated partition function matrices, i.e. a call to [vrna\\_pf\\_TwoD\(\)](#) preceding this function is mandatory!

#### See also

[vrna\\_pf\\_TwoD\(\)](#)

**Parameters**

in, out	vc	The <a href="#">vrna_fold_compound_t</a> datastructure containing all necessary folding attributes and matrices
in	d1	The distance to reference1 (may be -1)
in	d2	The distance to reference2

**Returns**

A sampled secondary structure in dot-bracket notation

**15.33.2.2 vrna\_pbacktrack5\_TwoD()**

```
char* vrna_pbacktrack5_TwoD (
    vrna_fold_compound_t * vc,
    int d1,
    int d2,
    unsigned int length )
```

```
#include <ViennaRNA/2Dpfold.h>
```

Sample secondary structure representatives with a specified length from a set of distance classes according to their Boltzmann probability.

This function does essentially the same as [vrna\\_pbacktrack\\_TwoD\(\)](#) with the only difference that partial structures, i.e. structures beginning from the 5' end with a specified length of the sequence, are backtracked

**Note**

This function does not work (since it makes no sense) for circular RNA sequences!

**Precondition**

The argument 'vars' must contain precalculated partition function matrices, i.e. a call to [vrna\\_pf\\_TwoD\(\)](#) preceding this function is mandatory!

**See also**

[vrna\\_pbacktrack\\_TwoD\(\)](#), [vrna\\_pf\\_TwoD\(\)](#)

**Parameters**

in, out	vc	The <a href="#">vrna_fold_compound_t</a> datastructure containing all necessary folding attributes and matrices
in	d1	The distance to reference1 (may be -1)
in	d2	The distance to reference2
in	length	The length of the structure beginning from the 5' end

**Returns**

A sampled secondary structure in dot-bracket notation

## 15.34 Compute the Density of States

### 15.34.1 Detailed Description

Collaboration diagram for Compute the Density of States:

#### Variables

- int [density\\_of\\_states](#) [MAXDOS+1]  
*The Density of States.*

### 15.34.2 Variable Documentation

#### 15.34.2.1 density\_of\_states

```
int density_of_states[MAXDOS+1]
```

```
#include <ViennaRNA/subopt.h>
```

The Density of States.

This array contains the density of states for an RNA sequences after a call to [subopt\\_par\(\)](#), [subopt\(\)](#) or [subopt\\_circ\(\)](#).

#### Precondition

Call one of the functions [subopt\\_par\(\)](#), [subopt\(\)](#) or [subopt\\_circ\(\)](#) prior accessing the contents of this array

#### See also

[subopt\\_par\(\)](#), [subopt\(\)](#), [subopt\\_circ\(\)](#)

## 15.35 Inverse Folding (Design)

RNA sequence design.

### 15.35.1 Detailed Description

RNA sequence design.

#### Files

- file [inverse.h](#)  
*Inverse folding routines.*

#### Functions

- float [inverse\\_fold](#) (char \*start, const char \*target)  
*Find sequences with predefined structure.*
- float [inverse\\_pf\\_fold](#) (char \*start, const char \*target)  
*Find sequence that maximizes probability of a predefined structure.*

#### Variables

- char \* [symbolset](#)  
*This global variable points to the allowed bases, initially "AUGC". It can be used to design sequences from reduced alphabets.*
- float [final\\_cost](#)
- int [give\\_up](#)
- int [inv\\_verbose](#)

### 15.35.2 Function Documentation

#### 15.35.2.1 [inverse\\_fold\(\)](#)

```
float inverse_fold (  
    char * start,  
    const char * target )  
  
#include <ViennaRNA/inverse.h>
```

Find sequences with predefined structure.

This function searches for a sequence with minimum free energy structure provided in the parameter 'target', starting with sequence 'start'. It returns 0 if the search was successful, otherwise a structure distance in terms of the energy difference between the search result and the actual target 'target' is returned. The found sequence is returned in 'start'. If [give\\_up](#) is set to 1, the function will return as soon as it is clear that the search will be unsuccessful, this speeds up the algorithm if you are only interested in exact solutions.



## Parameters

<i>start</i>	The start sequence
<i>target</i>	The target secondary structure in dot-bracket notation

## Returns

The distance to the target in case a search was unsuccessful, 0 otherwise

15.35.2.2 `inverse_pf_fold()`

```
float inverse_pf_fold (
    char * start,
    const char * target )

#include <ViennaRNA/inverse.h>
```

Find sequence that maximizes probability of a predefined structure.

This function searches for a sequence with maximum probability to fold into the provided structure 'target' using the partition function algorithm. It returns  $-kT \cdot \log(p)$  where  $p$  is the frequency of 'target' in the ensemble of possible structures. This is usually much slower than `inverse_fold()`.

## Parameters

<i>start</i>	The start sequence
<i>target</i>	The target secondary structure in dot-bracket notation

## Returns

The distance to the target in case a search was unsuccessful, 0 otherwise

## 15.35.3 Variable Documentation

15.35.3.1 `final_cost`

```
float final_cost

#include <ViennaRNA/inverse.h>

when to stop inverse_pf_fold()
```

### 15.35.3.2 give\_up

```
int give_up
```

```
#include <ViennaRNA/inverse.h>
```

default 0: try to minimize structure distance even if no exact solution can be found

### 15.35.3.3 inv\_verbose

```
int inv_verbose
```

```
#include <ViennaRNA/inverse.h>
```

print out substructure on which `inverse_fold()` fails

## 15.36 Neighborhood Relation and Move Sets for Secondary Structures

Different functions to generate structural neighbors of a secondary structure according to a particular Move Set.

### 15.36.1 Detailed Description

Different functions to generate structural neighbors of a secondary structure according to a particular Move Set.

This module contains methods to compute the neighbors of an RNA secondary structure. Neighbors of a given structure are all structures that differ in exactly one base pair. That means one can insert or delete base pairs in the given structure. These insertions and deletions of base pairs are usually called moves. A third move which is considered in these methods is a shift move. A shifted base pair has one stable position and one position that changes. These moves are encoded as follows:

- insertion:  $(i, j)$  where  $i, j > 0$
  - deletion:  $(i, j)$  where  $i, j < 0$
  - shift:  $(i, j)$  where either  $i > 0, j < 0$  or  $i < 0, j > 0$
- The negative position of a shift indicates the position that has changed.

Example:

```
We have given a sequence and a structure.
Sequence  AAGGAAACC
Structure ..(.....)
Indices   123456789
```

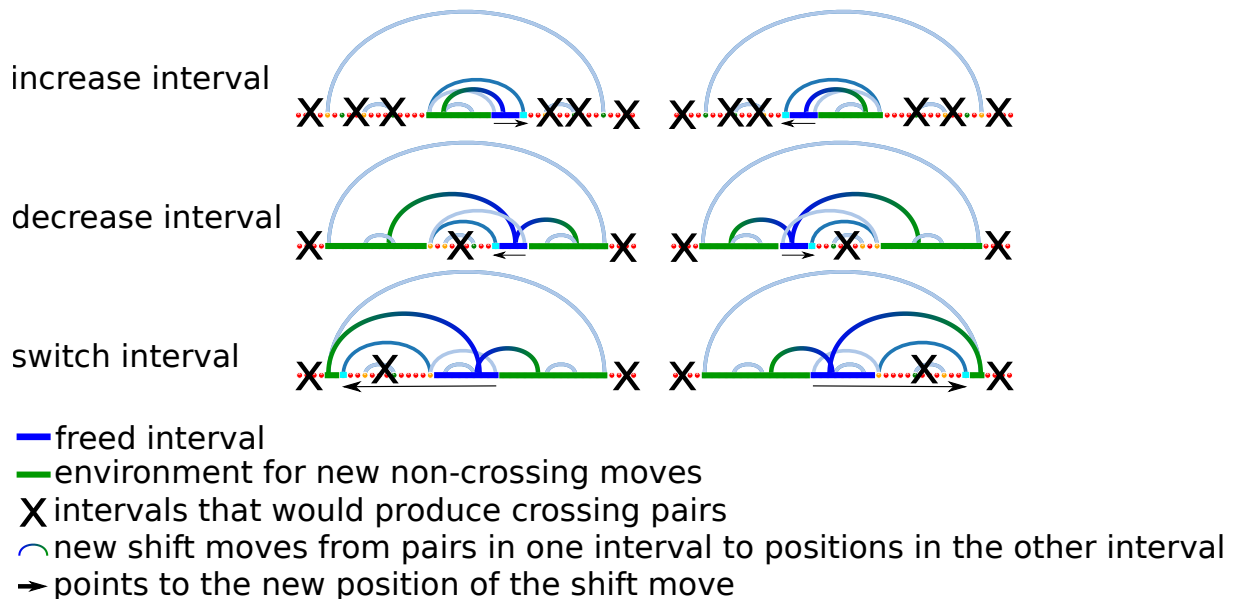
```
The given base pair is (3,9) and the neighbors are the insertion (4, 8), the deletion (-3,-9), the
shift (3,-8)
and the shift (-4, 9).
This leads to the neighbored structures:
... (....)
.....
... (....)
.... (....)
```

A simple method to construct all insertions is to iterate over the positions of a sequence twice. The first iteration has the index  $i$  in  $[1, \text{sequence length}]$ , the second iteration has the index  $j$  in  $[i+1, \text{sequence length}]$ . All pairs  $(i, j)$  with compatible letters and which are non-crossing with present base pairs are valid neighbored insertion moves. Valid deletion moves are all present base pairs with negative sign. Valid shift moves are constructed by taking all paired positions as fix position of a shift move and iterating over all positions of the sequence. If the letters of a position are compatible and if it the move is non-crossing with existing base pairs, we have a valid shift move. The method of generating shift moves can be accelerated by skipping neighbored base pairs.

If we need to construct all neighbors several times for subsequent moves, we can speed up the task by using the move set of the previous structure. The previous move set has to be filtered, such that all moves that would cross the next selected move are non-crossing. Next, the selected move has to be removed. Then one has to only to generate all moves that were not possible before. One move is the inverted selected move (if it was an insertion, simply make the indices negative). The generation of all other new moves is different and depends on the selected move. It is easy for an insertion move, because we have only to include all non-crossing shift moves, that are possible with the new base pair. For that we can either iterate over the sequence or we can select all crossing shift moves in the filter procedure and convert them into shifts.

The generation of new moves given a deletion is a little bit more complex, because we can create more moves. At first we can insert the deleted pair as insertion move. Then we generate all insertions that would have crossed the deleted base pair. Finally we construct all crossing shift moves.

If the given move is a shift, we can save much time by specifying the intervals for the generation of new moves. The interval which was enclosed by the positive position of the shift move and the previous paired position is the freed interval after applying the move. This freed interval includes all positions and base pairs that we need to construct new insertions and shifts. All these new moves have one position in the freed interval and the other position in the environment of the freed interval. The environment are all position which are outside the freed interval, but within the same enclosing loop of the shift move. The environment for valid base pairs can be divided into one or more intervals, depending on the shift move. The following examples describe a few scenarios to specify the intervals of the environment.



Given the intervals of the environment and the freed interval, the new shift moves can be constructed quickly. One has to take all positions of pairs from the environment in order to create valid pairs with positions in the freed interval. The same procedure can be applied for the other direction. This is taking all paired positions within the freed interval in order to look for pairs with valid positions in the intervals of the environment. Collaboration diagram for Neighborhood Relation and Move Sets for Secondary Structures:

## Files

- file [neighbor.h](#)

*Methods to compute the neighbors of an RNA secondary structure.*

## Data Structures

- struct [vrna\\_move\\_s](#)

*An atomic representation of the transition / move from one structure to its neighbor. [More...](#)*

## Macros

- `#define VRNA_MOVESET_INSERTION 4`  
*Option flag indicating insertion move.*
- `#define VRNA_MOVESET_DELETION 8`  
*Option flag indicating deletion move.*
- `#define VRNA_MOVESET_SHIFT 16`  
*Option flag indicating shift move.*
- `#define VRNA_MOVESET_NO_LP 32`  
*Option flag indicating moves without lonely base pairs.*
- `#define VRNA_MOVESET_DEFAULT (VRNA_MOVESET_INSERTION | VRNA_MOVESET_DELETION)`  
*Option flag indicating default move set, i.e. insertions/deletion of a base pair.*

## Functions

- `void vrna_move_list_free (vrna_move_t *moves)`
- `void vrna_move_apply (short *pt, const vrna_move_t *m)`  
*Apply a particular move / transition to a secondary structure, i.e. transform a structure.*
- `void vrna_loopidx_update (int *loopidx, const short *pt, int length, const vrna_move_t *m)`  
*Alters the loopIndices array that was constructed with `vrna_loopidx_from_ptable()`.*
- `vrna_move_t * vrna_neighbors (vrna_fold_compound_t *vc, const short *pt, unsigned int options)`  
*Generate neighbors of a secondary structure.*
- `vrna_move_t * vrna_neighbors_successive (const vrna_fold_compound_t *vc, const vrna_move_t *curr, const short *prev_pt, const vrna_move_t *prev_neighbors, int size_prev_neighbors, int *size_neighbors, unsigned int options)`  
*Generate neighbors of a secondary structure (the fast way)*

### 15.36.2 Data Structure Documentation

#### 15.36.2.1 struct vrna\_move\_s

An atomic representation of the transition / move from one structure to its neighbor.

An atomic transition / move may be (a) the insertion of a base pair (both fields are positive), (b) the deletion of a base pair (both fields are negative), or (c) a base pair shift where one position stays constant while the other is allowed to shift along the same loop it resides in (one field position and the other negative, where the positive field indicates the constant position and the absolute value of the negative field is the new position of the pairing partner).

A value of 0 in either field is typically used to indicate the list's last element.

Collaboration diagram for `vrna_move_s`:

#### Data Fields

- `int pos_5`
- `int pos_3`
- `vrna_move_t * next`

### 15.36.2.1.1 Field Documentation

#### 15.36.2.1.1.1 pos\_5

```
int vrna_move_s::pos_5
```

The 5' position of a base pair, or any position of a shifted pair

#### 15.36.2.1.1.2 pos\_3

```
int vrna_move_s::pos_3
```

The 3' position of a base pair, or any position of a shifted pair

#### 15.36.2.1.1.3 next

```
vrna_move_t* vrna_move_s::next
```

The next base pair (if an elementary move changes more than one base pair) Has to be terminated with move 0,0

## 15.36.3 Macro Definition Documentation

### 15.36.3.1 VRNA\_MOVESET\_INSERTION

```
#define VRNA_MOVESET_INSERTION 4  
  
#include <ViennaRNA/neighbor.h>
```

Option flag indicating insertion move.

See also

[vrna\\_neighbors\(\)](#), [vrna\\_neighbors\\_successive](#), [vrna\\_path\(\)](#)

### 15.36.3.2 VRNA\_MOVESET\_DELETION

```
#define VRNA_MOVESET_DELETION 8  
  
#include <ViennaRNA/neighbor.h>
```

Option flag indicating deletion move.

See also

[vrna\\_neighbors\(\)](#), [vrna\\_neighbors\\_successive](#), [vrna\\_path\(\)](#)

### 15.36.3.3 VRNA\_MOVESET\_SHIFT

```
#define VRNA_MOVESET_SHIFT 16
#include <ViennaRNA/neighbor.h>
```

Option flag indicating shift move.

See also

[vrna\\_neighbors\(\)](#), [vrna\\_neighbors\\_successive](#), [vrna\\_path\(\)](#)

### 15.36.3.4 VRNA\_MOVESET\_NO\_LP

```
#define VRNA_MOVESET_NO_LP 32
#include <ViennaRNA/neighbor.h>
```

Option flag indicating moves without lonely base pairs.

See also

[vrna\\_neighbors\(\)](#), [vrna\\_neighbors\\_successive](#), [vrna\\_path\(\)](#)

### 15.36.3.5 VRNA\_MOVESET\_DEFAULT

```
#define VRNA_MOVESET_DEFAULT (VRNA_MOVESET_INSERTION | VRNA_MOVESET_DELETION)
#include <ViennaRNA/neighbor.h>
```

Option flag indicating default move set, i.e. insertions/deletion of a base pair.

See also

[vrna\\_neighbors\(\)](#), [vrna\\_neighbors\\_successive](#), [vrna\\_path\(\)](#)

## 15.36.4 Function Documentation

### 15.36.4.1 vrna\_move\_list\_free()

```
void vrna_move_list_free (
    vrna_move_t * moves )
#include <ViennaRNA/neighbor.h>
```

delete all moves in a zero terminated list.

### 15.36.4.2 vrna\_move\_apply()

```
void vrna_move_apply (
    short * pt,
    const vrna_move_t * m )
#include <ViennaRNA/neighbor.h>
```

Apply a particular move / transition to a secondary structure, i.e. transform a structure.

## Parameters

<i>in, out</i>	<i>pt</i>	The pair table representation of the secondary structure
<i>in</i>	<i>m</i>	The move to apply

15.36.4.3 `vrna_loopidx_update()`

```
void vrna_loopidx_update (
    int * loopidx,
    const short * pt,
    int length,
    const vrna_move_t * m )
```

```
#include <ViennaRNA/neighbor.h>
```

Alters the `loopIndices` array that was constructed with `vrna_loopidx_from_ptable()`.

The `loopIndex` of the current move will be inserted. The correctness of the input will not be checked because the speed should be optimized.

## Parameters

<i>in, out</i>	<i>loopidx</i>	The loop index data structure that needs an update
<i>in</i>	<i>pt</i>	A pair table on which the move will be executed
	<i>length</i>	The length of the structure
<i>in</i>	<i>m</i>	The move that is applied to the current structure

15.36.4.4 `vrna_neighbors()`

```
vrna_neighbors (
    vrna_fold_compound_t * vc,
    const short * pt,
    unsigned int options )
```

```
#include <ViennaRNA/neighbor.h>
```

Generate neighbors of a secondary structure.

This function allows one to generate all structural neighbors (according to a particular move set) of an RNA secondary structure. The neighborhood is then returned as a list of transitions / moves required to transform the current structure into the actual neighbor.

## See also

[vrna\\_neighbors\\_successive\(\)](#), [vrna\\_move\\_apply\(\)](#), [VRNA\\_MOVESET\\_INSERTION](#), [VRNA\\_MOVESET\\_DELETION](#), [VRNA\\_MOVESET\\_SHIFT](#), [VRNA\\_MOVESET\\_DEFAULT](#)



## Parameters

in	<i>vc</i>	A <code>vrna_fold_compound_t</code> containing the energy parameters and model details
in	<i>pt</i>	The pair table representation of the structure
	<i>options</i>	Options to modify the behavior of this function, e.g. available move set

## Returns

Neighbors as a list of moves / transitions (the last element in the list has both of its fields set to 0)

**SWIG Wrapper Notes** This function is attached as an overloaded method *neighbors()* to objects of type *fold\_compound*. The optional parameter *options* defaults to `VRNA_MOVESET_DEFAULT` if it is omitted.

15.36.4.5 `vrna_neighbors_successive()`

```
vrna_move_t* vrna_neighbors_successive (
    const vrna_fold_compound_t * vc,
    const vrna_move_t * curr_move,
    const short * prev_pt,
    const vrna_move_t * prev_neighbors,
    int size_prev_neighbors,
    int * size_neighbors,
    unsigned int options )
```

```
#include <ViennaRNA/neighbor.h>
```

Generate neighbors of a secondary structure (the fast way)

This function implements a fast way to generate all neighbors of a secondary structure that results from successive applications of individual moves. The speed-up results from updating an already known list of valid neighbors before the individual move towards the current structure took place. In essence, this function removes neighbors that are not accessible anymore and inserts neighbors emerging after a move took place.

## See also

`vrna_neighbors()`, `vrna_move_apply()`, `VRNA_MOVESET_INSERTION`, `VRNA_MOVESET_DELETION`, `VRNA_MOVESET_SHIFT`, `VRNA_MOVESET_DEFAULT`

## Parameters

in	<i>vc</i>	A <code>vrna_fold_compound_t</code> containing the energy parameters and model details
in	<i>curr_move</i>	The move that was/will be applied to <i>prev_pt</i>
in	<i>prev_pt</i>	A pair table representation of the structure before <i>curr_move</i> is/was applied
in	<i>prev_neighbors</i>	The list of neighbors of <i>prev_pt</i>
	<i>size_prev_neighbors</i>	The size of <i>prev_neighbors</i> , i.e. the lists length
out	<i>size_neighbors</i>	A pointer to store the size / length of the new neighbor list
	<i>options</i>	Options to modify the behavior of this function, e.g. available move set

**Returns**

Neighbors as a list of moves / transitions (the last element in the list has both of its fields set to 0)

## 15.37 Refolding Paths of Secondary Structures

### 15.37.1 Detailed Description

Collaboration diagram for Refolding Paths of Secondary Structures:

#### Modules

- [Direct Refolding Paths between two Secondary Structures](#)

*Heuristics to explore direct, optimal (re-)folding paths between two secondary structures.*

#### Files

- file [findpath.h](#)

*A breadth-first search heuristic for optimal direct folding paths.*

- file [walk.h](#)

*Methods to generate particular paths such as gradient or random walks through the energy landscape of an RNA sequence.*

#### Macros

- `#define VRNA_PATH_STEEPEST_DESCENT 128`

*Option flag to request a steepest descent / gradient path.*

- `#define VRNA_PATH_RANDOM 256`

*Option flag to request a random walk path.*

- `#define VRNA_PATH_NO_TRANSITION_OUTPUT 512`

*Option flag to omit returning the transition path.*

- `#define VRNA_PATH_DEFAULT (VRNA_PATH_STEEPEST_DESCENT | VRNA_MOVESET_DEFAULT)`

*Option flag to request defaults (steepest descent / default move set)*

#### Functions

- `vrna_move_t * vrna_path (vrna_fold_compound_t *vc, short *pt, unsigned int steps, unsigned int options)`

*Compute a path, store the final structure, and return a list of transition moves from the start to the final structure.*

- `vrna_move_t * vrna_path_gradient (vrna_fold_compound_t *vc, short *pt, unsigned int options)`

*Compute a steepest descent / gradient path, store the final structure, and return a list of transition moves from the start to the final structure.*

- `vrna_move_t * vrna_path_random (vrna_fold_compound_t *vc, short *pt, unsigned int steps, unsigned int options)`

*Generate a random walk / path of a given length, store the final structure, and return a list of transition moves from the start to the final structure.*

### 15.37.2 Macro Definition Documentation

### 15.37.2.1 VRNA\_PATH\_STEEPEST\_DESCENT

```
#define VRNA_PATH_STEEPEST_DESCENT 128
```

```
#include <ViennaRNA/walk.h>
```

Option flag to request a steepest descent / gradient path.

See also

[vrna\\_path\(\)](#)

### 15.37.2.2 VRNA\_PATH\_RANDOM

```
#define VRNA_PATH_RANDOM 256
```

```
#include <ViennaRNA/walk.h>
```

Option flag to request a random walk path.

See also

[vrna\\_path\(\)](#)

### 15.37.2.3 VRNA\_PATH\_NO\_TRANSITION\_OUTPUT

```
#define VRNA_PATH_NO_TRANSITION_OUTPUT 512
```

```
#include <ViennaRNA/walk.h>
```

Option flag to omit returning the transition path.

See also

[vrna\\_path\(\)](#), [vrna\\_path\\_gradient\(\)](#), [vrna\\_path\\_random\(\)](#)

### 15.37.2.4 VRNA\_PATH\_DEFAULT

```
#define VRNA_PATH_DEFAULT (VRNA_PATH_STEEPEST_DESCENT | VRNA_MOVESET_DEFAULT)
```

```
#include <ViennaRNA/walk.h>
```

Option flag to request defaults (steepest descent / default move set)

See also

[vrna\\_path\(\)](#), [VRNA\\_PATH\\_STEEPEST\\_DESCENT](#), [VRNA\\_MOVESET\\_DEFAULT](#)

### 15.37.3 Function Documentation

#### 15.37.3.1 `vrna_path()`

```
vrna_path (
    vrna_fold_compound_t * vc,
    short * pt,
    unsigned int steps,
    unsigned int options )
```

```
#include <ViennaRNA/walk.h>
```

Compute a path, store the final structure, and return a list of transition moves from the start to the final structure.

This function computes, given a start structure in pair table format, a transition path, updates the pair table to the final structure of the path. Finally, if not requested otherwise by using the `VRNA_PATH_NO_TRANSITION_OUTPUT` flag in the `options` field, this function returns a list of individual transitions that lead from the start to the final structure if requested.

The currently available transition paths are

- Steepest Descent / Gradient walk (flag: `VRNA_PATH_STEEPEST_DESCENT`)
- Random walk (flag: `VRNA_PATH_RANDOM`)

The type of transitions must be set through the `options` parameter

#### Note

Since the result is written to the input structure you may want to use `vrna_ptable_copy()` before calling this function to keep the initial structure

#### See also

`vrna_path_gradient()`, `vrna_path_random()`, `vrna_ptable()`, `vrna_ptable_copy()`, `vrna_fold_compound()`, `VRNA_PATH_STEEPEST_DESCENT`, `VRNA_PATH_RANDOM`, `VRNA_MOVESET_DEFAULT`, `VRNA_MOVESET_SHIFT`, `VRNA_PATH_NO_TRANSITION_OUTPUT`

#### Parameters

in	<i>vc</i>	A <code>vrna_fold_compound_t</code> containing the energy parameters and model details
in, out	<i>pt</i>	The pair table containing the start structure. Used to update to the final structure after execution of this function
in	<i>options</i>	Options to modify the behavior of this function

#### Returns

A list of transition moves (default), or NULL (if options & `VRNA_PATH_NO_TRANSITION_OUTPUT`)

**SWIG Wrapper Notes** This function is attached as an overloaded method *path()* to objects of type *fold\_compound*. The optional parameter *options* defaults to [VRNA\\_PATH\\_DEFAULT](#) if it is omitted.

### 15.37.3.2 vrna\_path\_gradient()

```
vrna_path_gradient (
    vrna_fold_compound_t * vc,
    short * pt,
    unsigned int options )
```

```
#include <ViennaRNA/walk.h>
```

Compute a steepest descent / gradient path, store the final structure, and return a list of transition moves from the start to the final structure.

This function computes, given a start structure in pair table format, a steepest descent path, updates the pair table to the final structure of the path. Finally, if not requested otherwise by using the [VRNA\\_PATH\\_NO\\_TRANSITION\\_OUTPUT](#) flag in the *options* field, this function returns a list of individual transitions that lead from the start to the final structure if requested.

#### Note

Since the result is written to the input structure you may want to use [vrna\\_ptable\\_copy\(\)](#) before calling this function to keep the initial structure

#### See also

[vrna\\_path\\_random\(\)](#), [vrna\\_path\(\)](#), [vrna\\_ptable\(\)](#), [vrna\\_ptable\\_copy\(\)](#), [vrna\\_fold\\_compound\(\)](#) [VRNA\\_MOVESET\\_DEFAULT](#), [VRNA\\_MOVESET\\_SHIFT](#), [VRNA\\_PATH\\_NO\\_TRANSITION\\_OUTPUT](#)

#### Parameters

in	<i>vc</i>	A <i>vrna_fold_compound_t</i> containing the energy parameters and model details
in, out	<i>pt</i>	The pair table containing the start structure. Used to update to the final structure after execution of this function
in	<i>options</i>	Options to modify the behavior of this function

#### Returns

A list of transition moves (default), or NULL (if *options* & [VRNA\\_PATH\\_NO\\_TRANSITION\\_OUTPUT](#))

**SWIG Wrapper Notes** This function is attached as an overloaded method *path\_gradient()* to objects of type *fold\_compound*. The optional parameter *options* defaults to [VRNA\\_PATH\\_DEFAULT](#) if it is omitted.

15.37.3.3 `vrna_path_random()`

```
vrna_path_random (
    vrna_fold_compound_t * vc,
    short * pt,
    unsigned int steps,
    unsigned int options )
```

```
#include <ViennaRNA/walk.h>
```

Generate a random walk / path of a given length, store the final structure, and return a list of transition moves from the start to the final structure.

This function generates, given a start structure in pair table format, a random walk / path, updates the pair table to the final structure of the path. Finally, if not requested otherwise by using the `VRNA_PATH_NO_TRANSITION_OUTPUT` flag in the `options` field, this function returns a list of individual transitions that lead from the start to the final structure if requested.

**Note**

Since the result is written to the input structure you may want to use `vrna_ptable_copy()` before calling this function to keep the initial structure

**See also**

`vrna_path_gradient()`, `vrna_path()`, `vrna_ptable()`, `vrna_ptable_copy()`, `vrna_fold_compound()`, `VRNA_MOVESET_DEFAULT`, `VRNA_MOVESET_SHIFT`, `VRNA_PATH_NO_TRANSITION_OUTPUT`

**Parameters**

in	<i>vc</i>	A <code>vrna_fold_compound_t</code> containing the energy parameters and model details
in, out	<i>pt</i>	The pair table containing the start structure. Used to update to the final structure after execution of this function
in	<i>steps</i>	The length of the path, i.e. the total number of transitions / moves
in	<i>options</i>	Options to modify the behavior of this function

**Returns**

A list of transition moves (default), or NULL (if options & `VRNA_PATH_NO_TRANSITION_OUTPUT`)

**SWIG Wrapper Notes** This function is attached as an overloaded method `path_gradient()` to objects of type `fold↔_compound`. The optional parameter `options` defaults to `VRNA_PATH_DEFAULT` if it is omitted.

## 15.38 Experimental Structure Probing Data

Include Experimental Structure Probing Data to Guide Structure Predictions.

### 15.38.1 Detailed Description

Include Experimental Structure Probing Data to Guide Structure Predictions.

Collaboration diagram for Experimental Structure Probing Data:

#### Modules

- [SHAPE Reactivity Data](#)

*Incorporate SHAPE reactivity structure probing data into the folding recursions by means of soft constraints.*

- [Generate Soft Constraints from Data](#)

*Find a vector of perturbation energies that minimizes the discrepancies between predicted and observed pairing probabilities and the amount of necessary adjustments.*



## 15.39 SHAPE Reactivity Data

Incorporate SHAPE reactivity structure probing data into the folding recursions by means of soft constraints.

### 15.39.1 Detailed Description

Incorporate SHAPE reactivity structure probing data into the folding recursions by means of soft constraints.

Details for our implementation to incorporate SHAPE reactivity data to guide secondary structure prediction can be found in [16] Collaboration diagram for SHAPE Reactivity Data:

#### Files

- file [SHAPE.h](#)

*This module provides function to incorporate SHAPE reactivity data into the folding recursions by means of soft constraints.*

#### Functions

- int [vrna\\_sc\\_add\\_SHAPE\\_deigan](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const double \*reactivities, double m, double b, unsigned int options)

*Add SHAPE reactivity data as soft constraints (Deigan et al. method)*

- int [vrna\\_sc\\_add\\_SHAPE\\_deigan\\_al](#)i ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*\*shape\_files, const int \*shape\_file\_association, double m, double b, unsigned int options)

*Add SHAPE reactivity data from files as soft constraints for consensus structure prediction (Deigan et al. method)*

- int [vrna\\_sc\\_add\\_SHAPE\\_zarrin](#)ghalam ([vrna\\_fold\\_compound\\_t](#) \*vc, const double \*reactivities, double b, double default\_value, const char \*shape\_conversion, unsigned int options)

*Add SHAPE reactivity data as soft constraints (Zarringhalam et al. method)*

- int [vrna\\_sc\\_SHAPE\\_to\\_pr](#) (const char \*shape\_conversion, double \*values, int length, double default\_value)

*Convert SHAPE reactivity values to probabilities for being unpaired.*

### 15.39.2 Function Documentation

15.39.2.1 `vrna_sc_add_SHAPE_deigan()`

```
int vrna_sc_add_SHAPE_deigan (
    vrna_fold_compound_t * vc,
    const double * reactivities,
    double m,
    double b,
    unsigned int options )

#include <ViennaRNA/constraints/SHAPE.h>
```

Add SHAPE reactivity data as soft constraints (Deigan et al. method)

This approach of SHAPE directed RNA folding uses the simple linear ansatz

$$\Delta G_{\text{SHAPE}}(i) = m \ln(\text{SHAPE reactivity}(i) + 1) + b$$

to convert SHAPE reactivity values to pseudo energies whenever a nucleotide  $i$  contributes to a stacked pair. A positive slope  $m$  penalizes high reactivities in paired regions, while a negative intercept  $b$  results in a confirmatory "bonus" free energy for correctly predicted base pairs. Since the energy evaluation of a base pair stack involves two pairs, the pseudo energies are added for all four contributing nucleotides. Consequently, the energy term is applied twice for pairs inside a helix and only once for pairs adjacent to other structures. For all other loop types the energy model remains unchanged even when the experimental data highly disagrees with a certain motif.

## See also

For further details, we refer to [6].

`vrna_sc_remove()`, `vrna_sc_add_SHAPE_zarringhalam()`, `vrna_sc_minimize_pertubation()`

## Parameters

<i>vc</i>	The <code>vrna_fold_compound_t</code> the soft constraints are associated with
<i>reactivities</i>	A vector of normalized SHAPE reactivities
<i>m</i>	The slope of the conversion function
<i>b</i>	The intercept of the conversion function
<i>options</i>	The options flag indicating how/where to store the soft constraints

## Returns

1 on successful extraction of the method, 0 on errors

**SWIG Wrapper Notes** This function is attached as method `sc_add_SHAPE_deigan()` to objects of type `fold_compound`

15.39.2.2 `vrna_sc_add_SHAPE_deigan_al()`

```
int vrna_sc_add_SHAPE_deigan_al (
    vrna_fold_compound_t * vc,
    const char ** shape_files,
```

```

const int * shape_file_association,
double m,
double b,
unsigned int options )

```

```
#include <ViennaRNA/constraints/SHAPE.h>
```

Add SHAPE reactivity data from files as soft constraints for consensus structure prediction (Deigan et al. method)

#### Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> the soft constraints are associated with
<i>shape_files</i>	A set of filenames that contain normalized SHAPE reactivity data
<i>shape_file_association</i>	An array of integers that associate the files with sequences in the alignment
<i>m</i>	The slope of the conversion function
<i>b</i>	The intercept of the conversion function
<i>options</i>	The options flag indicating how/where to store the soft constraints

#### Returns

1 on successful extraction of the method, 0 on errors

**SWIG Wrapper Notes** This function is attached as method `sc_add_SHAPE_deigan.ali()` to objects of type `fold↔_compound`

#### 15.39.2.3 vrna\_sc\_add\_SHAPE\_zarringhalam()

```

int vrna_sc_add_SHAPE_zarringhalam (
    vrna_fold_compound_t * vc,
    const double * reactivities,
    double b,
    double default_value,
    const char * shape_conversion,
    unsigned int options )

```

```
#include <ViennaRNA/constraints/SHAPE.h>
```

Add SHAPE reactivity data as soft constraints (Zarringhalam et al. method)

This method first converts the observed SHAPE reactivity of nucleotide  $i$  into a probability  $q_i$  that position  $i$  is unpaired by means of a non-linear map. Then pseudo-energies of the form

$$\Delta G_{\text{SHAPE}}(x, i) = \beta |x_i - q_i|$$

are computed, where  $x_i = 0$  if position  $i$  is unpaired and  $x_i = 1$  if  $i$  is paired in a given secondary structure. The parameter  $\beta$  serves as scaling factor. The magnitude of discrepancy between prediction and experimental observation is represented by  $|x_i - q_i|$ .

#### See also

For further details, we refer to [25]

[vrna\\_sc\\_remove\(\)](#), [vrna\\_sc\\_add\\_SHAPE\\_deigan\(\)](#), [vrna\\_sc\\_minimize\\_pertubation\(\)](#)

## Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> the soft constraints are associated with
<i>reactivities</i>	A vector of normalized SHAPE reactivities
<i>b</i>	The scaling factor $\beta$ of the conversion function
<i>default_value</i>	The default value for a nucleotide where reactivity data is missing for
<i>shape_conversion</i>	A flag that specifies how to convert reactivities to probabilities
<i>options</i>	The options flag indicating how/where to store the soft constraints

## Returns

1 on successful extraction of the method, 0 on errors

**SWIG Wrapper Notes** This function is attached as method `sc_add_SHAPE_zarringhalam()` to objects of type *fold\_compound*

## 15.39.2.4 vrna\_sc\_SHAPE\_to\_pr()

```
int vrna_sc_SHAPE_to_pr (
    const char * shape_conversion,
    double * values,
    int length,
    double default_value )
```

```
#include <ViennaRNA/constraints/SHAPE.h>
```

Convert SHAPE reactivity values to probabilities for being unpaired.

This function parses the informations from a given file and stores the result in the preallocated string sequence and the [FLT\\_OR\\_DBL](#) array values.

## See also

[vrna\\_file\\_SHAPE\\_read\(\)](#)

## Parameters

<i>shape_conversion</i>	String defining the method used for the conversion process
<i>values</i>	Pointer to an array of SHAPE reactivities
<i>length</i>	Length of the array of SHAPE reactivities
<i>default_value</i>	Result used for position with invalid/missing reactivity values

## 15.40 Generate Soft Constraints from Data

Find a vector of perturbation energies that minimizes the discrepancies between predicted and observed pairing probabilities and the amount of necessary adjustments.

### 15.40.1 Detailed Description

Find a vector of perturbation energies that minimizes the discrepancies between predicted and observed pairing probabilities and the amount of necessary adjustments.

Collaboration diagram for Generate Soft Constraints from Data:

#### Files

- file [perturbation\\_fold.h](#)

*Find a vector of perturbation energies that minimizes the discrepancies between predicted and observed pairing probabilities and the amount of necessary adjustments.*

#### Macros

- `#define VRNA_OBJECTIVE_FUNCTION_QUADRATIC 0`  
*Use the sum of squared aberrations as objective function.*
- `#define VRNA_OBJECTIVE_FUNCTION_ABSOLUTE 1`  
*Use the sum of absolute aberrations as objective function.*
- `#define VRNA_MINIMIZER_DEFAULT 0`  
*Use a custom implementation of the gradient descent algorithm to minimize the objective function.*
- `#define VRNA_MINIMIZER_CONJUGATE_FR 1`  
*Use the GNU Scientific Library implementation of the Fletcher-Reeves conjugate gradient algorithm to minimize the objective function.*
- `#define VRNA_MINIMIZER_CONJUGATE_PR 2`  
*Use the GNU Scientific Library implementation of the Polak-Ribiere conjugate gradient algorithm to minimize the objective function.*
- `#define VRNA_MINIMIZER_VECTOR_BFGS 3`  
*Use the GNU Scientific Library implementation of the vector Broyden-Fletcher-Goldfarb-Shanno algorithm to minimize the objective function.*
- `#define VRNA_MINIMIZER_VECTOR_BFGS2 4`  
*Use the GNU Scientific Library implementation of the vector Broyden-Fletcher-Goldfarb-Shanno algorithm to minimize the objective function.*
- `#define VRNA_MINIMIZER_STEEPEST_DESCENT 5`  
*Use the GNU Scientific Library implementation of the steepest descent algorithm to minimize the objective function.*

#### Typedefs

- `typedef void(* progress_callback) (int iteration, double score, double *epsilon)`  
*Callback for following the progress of the minimization process.*

## Functions

- void `vrna_sc_minimize_perturbation` (`vrna_fold_compound_t` \*vc, const double \*q\_prob\_unpaired, int objective\_function, double sigma\_squared, double tau\_squared, int algorithm, int sample\_size, double \*epsilon, double initialStepSize, double minStepSize, double minImprovement, double minimizerTolerance, `progress_callback` callback)

*Find a vector of perturbation energies that minimizes the discrepancies between predicted and observed pairing probabilities and the amount of necessary adjustments.*

## 15.40.2 Macro Definition Documentation

### 15.40.2.1 VRNA\_OBJECTIVE\_FUNCTION\_QUADRATIC

```
#define VRNA_OBJECTIVE_FUNCTION_QUADRATIC 0
#include <ViennaRNA/perturbation_fold.h>
```

Use the sum of squared aberrations as objective function.

$$F(\vec{\epsilon}) = \sum_{i=1}^n \frac{\epsilon_i^2}{\tau^2} + \sum_{i=1}^n \frac{(p_i(\vec{\epsilon}) - q_i)^2}{\sigma^2} \rightarrow \min$$

### 15.40.2.2 VRNA\_OBJECTIVE\_FUNCTION\_ABSOLUTE

```
#define VRNA_OBJECTIVE_FUNCTION_ABSOLUTE 1
#include <ViennaRNA/perturbation_fold.h>
```

Use the sum of absolute aberrations as objective function.

$$F(\vec{\epsilon}) = \sum_{i=1}^n \frac{|\epsilon_i|}{\tau^2} + \sum_{i=1}^n \frac{|p_i(\vec{\epsilon}) - q_i|}{\sigma^2} \rightarrow \min$$

### 15.40.2.3 VRNA\_MINIMIZER\_CONJUGATE\_FR

```
#define VRNA_MINIMIZER_CONJUGATE_FR 1
#include <ViennaRNA/perturbation_fold.h>
```

Use the GNU Scientific Library implementation of the Fletcher-Reeves conjugate gradient algorithm to minimize the objective function.

Please note that this algorithm can only be used when the GNU Scientific Library is available on your system

### 15.40.2.4 VRNA\_MINIMIZER\_CONJUGATE\_PR

```
#define VRNA_MINIMIZER_CONJUGATE_PR 2
#include <ViennaRNA/perturbation_fold.h>
```

Use the GNU Scientific Library implementation of the Polak-Ribiere conjugate gradient algorithm to minimize the objective function.

Please note that this algorithm can only be used when the GNU Scientific Library is available on your system

#### 15.40.2.5 VRNA\_MINIMIZER\_VECTOR\_BFGS

```
#define VRNA_MINIMIZER_VECTOR_BFGS 3
```

```
#include <ViennaRNA/perturbation_fold.h>
```

Use the GNU Scientific Library implementation of the vector Broyden-Fletcher-Goldfarb-Shanno algorithm to minimize the objective function.

Please note that this algorithm can only be used when the GNU Scientific Library is available on your system

#### 15.40.2.6 VRNA\_MINIMIZER\_VECTOR\_BFGS2

```
#define VRNA_MINIMIZER_VECTOR_BFGS2 4
```

```
#include <ViennaRNA/perturbation_fold.h>
```

Use the GNU Scientific Library implementation of the vector Broyden-Fletcher-Goldfarb-Shanno algorithm to minimize the objective function.

Please note that this algorithm can only be used when the GNU Scientific Library is available on your system

#### 15.40.2.7 VRNA\_MINIMIZER\_STEEPEST\_DESCENT

```
#define VRNA_MINIMIZER_STEEPEST_DESCENT 5
```

```
#include <ViennaRNA/perturbation_fold.h>
```

Use the GNU Scientific Library implementation of the steepest descent algorithm to minimize the objective function.

Please note that this algorithm can only be used when the GNU Scientific Library is available on your system

### 15.40.3 Typedef Documentation

#### 15.40.3.1 progress\_callback

```
typedef void(* progress_callback) (int iteration, double score, double *epsilon)
```

```
#include <ViennaRNA/perturbation_fold.h>
```

Callback for following the progress of the minimization process.

##### Parameters

<i>iteration</i>	The number of the current iteration
<i>score</i>	The score of the objective function
<i>epsilon</i>	The perturbation vector yielding the reported score

## 15.40.4 Function Documentation

### 15.40.4.1 vrna\_sc\_minimize\_pertubation()

```
void vrna_sc_minimize_pertubation (
    vrna_fold_compound_t * vc,
    const double * q_prob_unpaired,
    int objective_function,
    double sigma_squared,
    double tau_squared,
    int algorithm,
    int sample_size,
    double * epsilon,
    double initialStepSize,
    double minStepSize,
    double minImprovement,
    double minimizerTolerance,
    progress_callback callback )
```

```
#include <ViennaRNA/perturbation_fold.h>
```

Find a vector of perturbation energies that minimizes the discrepancies between predicted and observed pairing probabilities and the amount of necessary adjustments.

Use an iterative minimization algorithm to find a vector of perturbation energies whose incorporation as soft constraints shifts the predicted pairing probabilities closer to the experimentally observed probabilities. The algorithm aims to minimize an objective function that penalizes discrepancies between predicted and observed pairing probabilities and energy model adjustments, i.e. an appropriate vector of perturbation energies satisfies

$$F(\vec{\epsilon}) = \sum_{\mu} \frac{\epsilon_{\mu}^2}{\tau^2} + \sum_{i=1}^n \frac{(p_i(\vec{\epsilon}) - q_i)^2}{\sigma^2} \rightarrow \min.$$

An initialized fold compound and an array containing the observed probability for each nucleotide to be unbound are required as input data. The parameters `objective_function`, `sigma_squared` and `tau_squared` are responsible for adjusting the aim of the objective function. Dependend on which type of objective function is selected, either squared or absolute aberrations are contributing to the objective function. The ratio of the parameters `sigma_squared` and `tau_squared` can be used to adjust the algorithm to find a solution either close to the thermodynamic prediction (`sigma_squared >> tau_squared`) or close to the experimental data (`tau_squared >> sigma_squared`). The minimization can be performed by makeing use of a custom gradient descent implementation or using one of the minimizing algorithms provided by the GNU Scientific Library. All algorithms require the evaluation of the gradient of the objective function, which includes the evaluation of conditional pairing probabilities. Since an exact evaluation is expensive, the probabilities can also be estimated from sampling by setting an appropriate sample size. The found vector of perturbation energies will be stored in the array `epsilon`. The progress of the minimization process can be tracked by implementing and passing a callback function.

See also

For further details we refere to [23].

#### Parameters

<code>vc</code>	Pointer to a fold compound
-----------------	----------------------------



## Parameters

<i>q_prob_unpaired</i>	Pointer to an array containing the probability to be unpaired for each nucleotide
<i>objective_function</i>	The type of objective function to be used (VRNA_OBJECTIVE_FUNCTION_QUADRATIC / VRNA_OBJECTIVE_FUNCTION_LINEAR)
<i>sigma_squared</i>	A factor used for weighting the objective function. More weight on this factor will lead to a solution close to the null vector.
<i>tau_squared</i>	A factor used for weighting the objective function. More weight on this factor will lead to a solution close to the data provided in <i>q_prob_unpaired</i> .
<i>algorithm</i>	The minimization algorithm (VRNA_MINIMIZER_*)
<i>sample_size</i>	The number of sampled sequences used for estimating the pairing probabilities. A value $\leq 0$ will lead to an exact evaluation.
<i>epsilon</i>	A pointer to an array used for storing the calculated vector of perturbation energies
<i>callback</i>	A pointer to a callback function used for reporting the current minimization progress

## 15.41 Ligands Binding to RNA Structures

Simple Extensions to Model Ligand Binding to RNA Structures.

### 15.41.1 Detailed Description

Simple Extensions to Model Ligand Binding to RNA Structures.

Collaboration diagram for Ligands Binding to RNA Structures:

#### Modules

- [Ligands Binding to Unstructured Domains](#)  
*Add ligand binding to loop regions using the [Unstructured Domains](#) feature.*
- [Incorporating Ligands Binding to Specific Sequence/Structure Motifs using Soft Constraints](#)  
*Ligand binding to specific hairpin/interior loop like motifs using the [Soft Constraints](#) feature.*

#### Files

- file [ligand.h](#)  
*Functions for incorporation of ligands binding to hairpin and interior loop motifs using the soft constraints framework.*

## 15.42 Ligands Binding to Unstructured Domains

Add ligand binding to loop regions using the [Unstructured Domains](#) feature.

Add ligand binding to loop regions using the [Unstructured Domains](#) feature.

Sometime, certain ligands, like single strand binding (SSB) proteins, compete with intramolecular base pairing of the RNA. In situations, where the dissociation constant of the ligand is known and the ligand binds to a consecutive stretch of single-stranded nucleotides we can use the [Unstructured Domains](#) functionality to extend the RNA folding grammar. This module provides a convenience default implementation that covers most of the application scenarios.

The function `vrna_ud_add_motif()` attaches a ligands sequence motif and corresponding binding free energy to the list of known ligand motifs within a `vrna_fold_compound_t.domains_up` attribute. The first call to this function initializes the [Unstructured Domains](#) feature with our default implementation. Subsequent calls of secondary structure prediction algorithms with the modified `vrna_fold_compound_t` then directly include the competition of the ligand with regules base pairing. Since we utilize the unstructured domain extension, The ligand binding model can be removed again using the `vrna_ud_remove()` function. Collaboration diagram for Ligands Binding to Unstructured Domains:

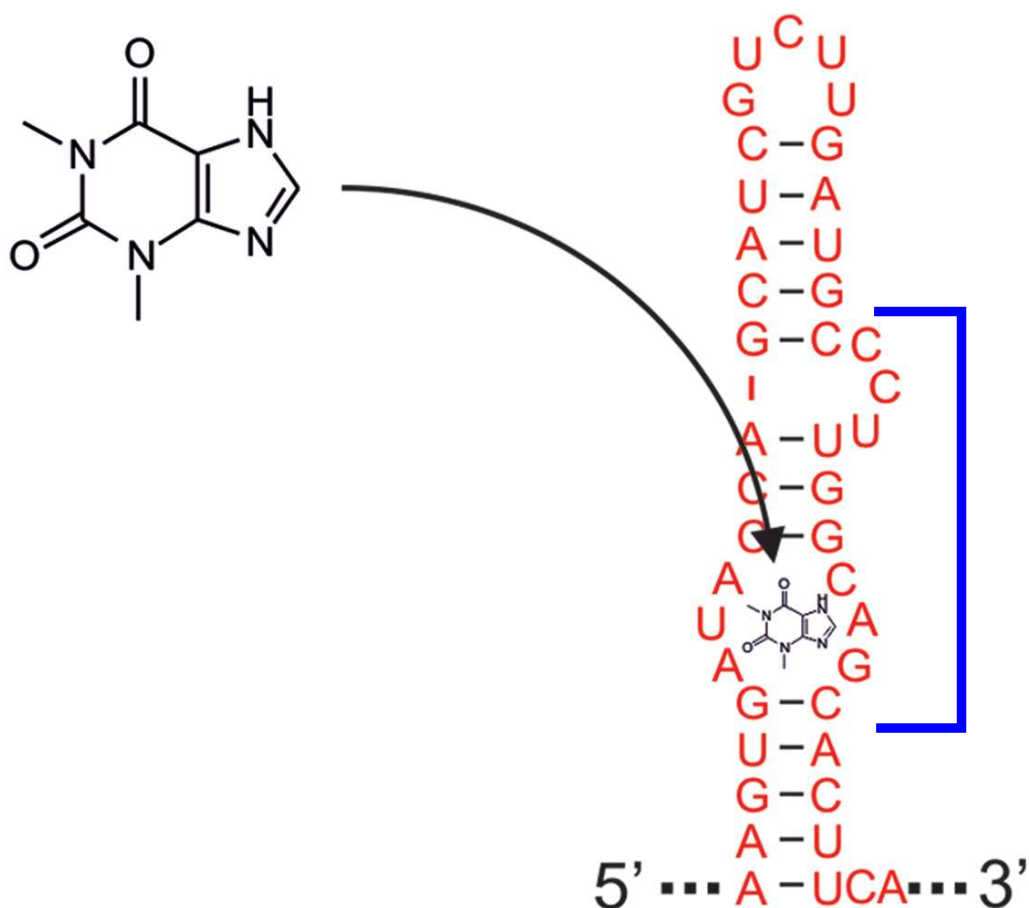
## 15.43 Incorporating Ligands Binding to Specific Sequence/Structure Motifs using Soft Constraints

Ligand binding to specific hairpin/interior loop like motifs using the [Soft Constraints](#) feature.

### 15.43.1 Detailed Description

Ligand binding to specific hairpin/interior loop like motifs using the [Soft Constraints](#) feature.

Here is an example that adds a theophylline binding motif. Free energy contribution is derived from  $k_d = 0.32 \mu\text{mol/l}$ , taken from Jenison et al. 1994



```
vrna_sc_add_hi_motif(vc,
    "GAUACCAG&CCCUUGGCAGC",
    "...((((&)...))...)",
    -9.22, VRNA_OPTION_DEFAULT);
```

Collaboration diagram for Incorporating Ligands Binding to Specific Sequence/Structure Motifs using Soft Constraints:

## Functions

- `int vrna_sc_add_hi_motif (vrna_fold_compound_t *vc, const char *seq, const char *structure, FLT_OR_DBL energy, unsigned int options)`

*Add soft constraints for hairpin or interior loop binding motif.*

### 15.43.2 Function Documentation

#### 15.43.2.1 vrna\_sc\_add\_hi\_motif()

```
int vrna_sc_add_hi_motif (
    vrna_fold_compound_t * vc,
    const char * seq,
    const char * structure,
    FLT_OR_DBL energy,
    unsigned int options )
```

```
#include <ViennaRNA/constraints/ligand.h>
```

Add soft constraints for hairpin or interior loop binding motif.

#### Parameters

<i>vc</i>	The <code>vrna_fold_compound_t</code> the motif is applied to
<i>seq</i>	The sequence motif (may be interspaced by '&' character)
<i>structure</i>	The structure motif (may be interspaced by '&' character)
<i>energy</i>	The free energy of the motif (e.g. binding free energy)
<i>options</i>	Options

#### Returns

non-zero value if application of the motif using soft constraints was successful

**SWIG Wrapper Notes** This function is attached as method `sc_add_hi_motif()` to objects of type `fold_compound`

## 15.44 Complex Structured Modules

### 15.44.1 Detailed Description

Collaboration diagram for Complex Structured Modules:

#### Modules

- [G-Quadruplexes](#)  
*Various functions related to G-quadruplex computations.*

#### Files

- file [gquad.h](#)  
*G-quadruplexes.*

## 15.45 G-Quadruplexes

Various functions related to G-quadruplex computations.

### 15.45.1 Detailed Description

Various functions related to G-quadruplex computations.

Collaboration diagram for G-Quadruplexes:

### Functions

- int \* [get\\_gquad\\_matrix](#) (short \*S, [vrna\\_param\\_t](#) \*P)  
*Get a triangular matrix prefilled with minimum free energy contributions of G-quadruplexes.*
- int [parse\\_gquad](#) (const char \*struc, int \*L, int l[3])
- PRIVATE int [backtrack\\_GQuad\\_IntLoop](#) (int c, int i, int j, int type, short \*S, int \*ggg, int \*index, int \*p, int \*q, [vrna\\_param\\_t](#) \*P)
- PRIVATE int [backtrack\\_GQuad\\_IntLoop\\_L](#) (int c, int i, int j, int type, short \*S, int \*\*ggg, int maxdist, int \*p, int \*q, [vrna\\_param\\_t](#) \*P)

### 15.45.2 Function Documentation

#### 15.45.2.1 [get\\_gquad\\_matrix\(\)](#)

```
int* get_gquad_matrix (
    short * S,
    vrna\_param\_t * P )
```

```
#include <ViennaRNA/gquad.h>
```

Get a triangular matrix prefilled with minimum free energy contributions of G-quadruplexes.

At each position ij in the matrix, the minimum free energy of any G-quadruplex delimited by i and j is stored. If no G-quadruplex formation is possible, the matrix element is set to INF. Access the elements in the matrix via `matrix[indx[j]+i]`. To get the integer array `indx` see [get\\_jindx\(\)](#).

See also

[get\\_jindx\(\)](#), [encode\\_sequence\(\)](#)

#### Parameters

<i>S</i>	The encoded sequence
<i>P</i>	A pointer to the data structure containing the precomputed energy contributions

**Returns**

A pointer to the G-quadruplex contribution matrix

**15.45.2.2 parse\_gquad()**

```
int parse_gquad (
    const char * struc,
    int * L,
    int l[3] )
```

```
#include <ViennaRNA/gquad.h>
```

given a dot-bracket structure (possibly) containing gquads encoded by '+' signs, find first gquad, return end position or 0 if none found Upon return L and l[] contain the number of stacked layers, as well as the lengths of the linker regions. To parse a string with many gquads, call parse\_gquad repeatedly e.g. end1 = parse\_gquad(struc, &L, l); ... ; end2 = parse\_gquad(struc+end1, &L, l); end2+=end1; ... ; end3 = parse\_gquad(struc+end2, &L, l); end3+=end2; ... ;

**15.45.2.3 backtrack\_GQuad\_IntLoop()**

```
PRIVATE int backtrack_GQuad_IntLoop (
    int c,
    int i,
    int j,
    int type,
    short * S,
    int * ggg,
    int * index,
    int * p,
    int * q,
    vrna_param_t * P )
```

```
#include <ViennaRNA/gquad.h>
```

backtrack an interior loop like enclosed g-quadruplex with closing pair (i,j)

**Parameters**

<i>c</i>	The total contribution the loop should resemble
<i>i</i>	position i of enclosing pair
<i>j</i>	position j of enclosing pair
<i>type</i>	base pair type of enclosing pair (must be reverse type)
<i>S</i>	integer encoded sequence
<i>ggg</i>	triangular matrix containing g-quadruplex contributions
<i>index</i>	the index for accessing the triangular matrix
<i>p</i>	here the 5' position of the gquad is stored
<i>q</i>	here the 3' position of the gquad is stored
<i>P</i>	the datastructure containing the precalculated contributions



**Returns**

1 on success, 0 if no gquad found

**15.45.2.4 backtrack\_GQuad\_IntLoop\_L()**

```
PRIVATE int backtrack_GQuad_IntLoop_L (
    int c,
    int i,
    int j,
    int type,
    short * S,
    int ** ggg,
    int maxdist,
    int * p,
    int * q,
    vrna_param_t * P )
```

```
#include <ViennaRNA/gquad.h>
```

backtrack an interior loop like enclosed g-quadruplex with closing pair (i,j) with underlying Lfold matrix

**Parameters**

<i>c</i>	The total contribution the loop should resemble
<i>i</i>	position i of enclosing pair
<i>j</i>	position j of enclosing pair
<i>type</i>	base pair type of enclosing pair (must be reverse type)
<i>S</i>	integer encoded sequence
<i>ggg</i>	triangular matrix containing g-quadruplex contributions
<i>p</i>	here the 5' position of the gquad is stored
<i>q</i>	here the 3' position of the gquad is stored
<i>P</i>	the datastructure containing the precalculated contibutions

**Returns**

1 on success, 0 if no gquad found

## 15.46 Utilities

### 15.46.1 Detailed Description

Collaboration diagram for Utilities:

#### Modules

- [Utilities to deal with Nucleotide Alphabets](#)  
*Functions to cope with various aspects related to the nucleotide sequence alphabet.*
- [\(Nucleic Acid Sequence\) String Utilitites](#)  
*Functions to parse, convert, manipulate, create, and compare (nucleic acid sequence) strings.*
- [Secondary Structure Utilities](#)  
*Functions to create, parse, convert, manipulate, and compare secondary structure representations.*
- [Multiple Sequence Alignment Utilities](#)  
*Functions to extract features from and to manipulate multiple sequence alignments.*
- [Files and I/O](#)  
*Functions to parse, write, and convert various file formats and to deal with file system related issues.*
- [Plotting](#)  
*Functions for Creating Secondary Structure Plots, Dot-Plots, and More.*
- [Search Algorithms](#)  
*Implementations of various search algorithms to detect strings of objects within other strings of objects.*
- [Combinatorics Algorithms](#)  
*Implementations to solve various combinatorial aspects for strings of objects.*
- [\(Abstract\) Data Structures](#)  
*All datastructures and typedefs shared among the ViennaRNA Package can be found here.*
- [Messages](#)  
*Functions to print various kind of messages.*
- [Unit Conversion](#)  
*Functions to convert between various physical units.*

#### Files

- file [alphabet.h](#)  
*Functions to process, convert, and generally handle different nucleotide and/or base pair alphabets.*
- file [combinatorics.h](#)  
*Various implementations that deal with combinatorial aspects of objects.*
- file [commands.h](#)  
*Parse and apply different commands that alter the behavior of secondary structure prediction and evaluation.*
- file [sequence.h](#)  
*Functions and data structures related to sequence representations ,.*
- file [units.h](#)  
*Physical Units and Functions to convert them into each other.*
- file [file\\_formats\\_msa.h](#)  
*Functions dealing with file formats for Multiple Sequence Alignments (MSA)*
- file [utils.h](#)  
*Several utilities for file handling.*

- file [utils.h](#)  
*Various utilities to assist in plotting secondary structures and consensus structures.*
- file [alignments.h](#)  
*Various utility- and helper-functions for sequence alignments and comparative structure prediction.*
- file [basic.h](#)  
*General utility- and helper-functions used throughout the ViennaRNA Package.*
- file [strings.h](#)  
*General utility- and helper-functions for RNA sequence and structure strings used throughout the ViennaRNA Package.*
- file [BoyerMoore.h](#)  
*Variants of the Boyer-Moore string search algorithm.*
- file [char\\_stream.h](#)  
*Implementation of a dynamic, buffered character stream.*
- file [stream\\_output.h](#)  
*An implementation of a buffered, ordered stream output data structure.*

## Macros

- `#define VRNA_INPUT_ERROR 1U`  
*Output flag of `get_input_line()`: "An ERROR has occurred, maybe EOF".*
- `#define VRNA_INPUT_QUIT 2U`  
*Output flag of `get_input_line()`: "the user requested quitting the program".*
- `#define VRNA_INPUT_MISC 4U`  
*Output flag of `get_input_line()`: "something was read".*
- `#define VRNA_INPUT_FASTA_HEADER 8U`  
*Input/Output flag of `get_input_line()`:  
if used as input option this tells `get_input_line()` that the data to be read should comply with the FASTA format.*
- `#define VRNA_INPUT_CONSTRAINT 32U`  
*Input flag for `get_input_line()`:  
Tell `get_input_line()` that we assume to read a structure constraint.*
- `#define VRNA_INPUT_NO_TRUNCATION 256U`  
*Input switch for `get_input_line()`: "do not truncate the line by eliminating white spaces at end of line".*
- `#define VRNA_INPUT_NO_REST 512U`  
*Input switch for `vrna_file_fasta_read_record()`: "do fill rest array".*
- `#define VRNA_INPUT_NO_SPAN 1024U`  
*Input switch for `vrna_file_fasta_read_record()`: "never allow data to span more than one line".*
- `#define VRNA_INPUT_NOSKIP_BLANK_LINES 2048U`  
*Input switch for `vrna_file_fasta_read_record()`: "do not skip empty lines".*
- `#define VRNA_INPUT_BLANK_LINE 4096U`  
*Output flag for `vrna_file_fasta_read_record()`: "read an empty line".*
- `#define VRNA_INPUT_NOSKIP_COMMENTS 128U`  
*Input switch for `get_input_line()`: "do not skip comment lines".*
- `#define VRNA_INPUT_COMMENT 8192U`  
*Output flag for `vrna_file_fasta_read_record()`: "read a comment".*
- `#define MIN2(A, B) ((A) < (B) ? (A) : (B))`  
*Get the minimum of two comparable values.*
- `#define MAX2(A, B) ((A) > (B) ? (A) : (B))`  
*Get the maximum of two comparable values.*
- `#define MIN3(A, B, C) (MIN2((MIN2((A), (B))), (C)))`  
*Get the minimum of three comparable values.*
- `#define MAX3(A, B, C) (MAX2((MAX2((A), (B))), (C)))`  
*Get the maximum of three comparable values.*

## Functions

- void \* [vrna\\_alloc](#) (unsigned size)  
*Allocate space safely.*
- void \* [vrna\\_realloc](#) (void \*p, unsigned size)  
*Reallocate space safely.*
- void [vrna\\_init\\_rand](#) (void)  
*Initialize seed for random number generator.*
- double [vrna\\_urn](#) (void)  
*get a random number from [0..1]*
- int [vrna\\_int\\_urn](#) (int from, int to)  
*Generates a pseudo random integer in a specified range.*
- char \* [vrna\\_time\\_stamp](#) (void)  
*Get a timestamp.*
- unsigned int [get\\_input\\_line](#) (char \*\*string, unsigned int options)
- int \* [vrna\\_idx\\_row\\_wise](#) (unsigned int length)  
*Get an index mapper array (iidx) for accessing the energy matrices, e.g. in partition function related functions.*
- int \* [vrna\\_idx\\_col\\_wise](#) (unsigned int length)  
*Get an index mapper array (indx) for accessing the energy matrices, e.g. in MFE related functions.*

## Variables

- unsigned short [xsubi](#) [3]  
*Current 48 bit random number.*

## 15.46.2 Macro Definition Documentation

### 15.46.2.1 VRNA\_INPUT\_FASTA\_HEADER

```
#define VRNA_INPUT_FASTA_HEADER 8U
```

```
#include <ViennaRNA/utils/basic.h>
```

Input/Output flag of [get\\_input\\_line\(\)](#):

if used as input option this tells [get\\_input\\_line\(\)](#) that the data to be read should comply with the FASTA format.

the function will return this flag if a fasta header was read

### 15.46.2.2 VRNA\_INPUT\_CONSTRAINT

```
#define VRNA_INPUT_CONSTRAINT 32U
```

```
#include <ViennaRNA/utils/basic.h>
```

Input flag for [get\\_input\\_line\(\)](#):

Tell [get\\_input\\_line\(\)](#) that we assume to read a structure constraint.

### 15.46.3 Function Documentation

#### 15.46.3.1 `vrna_alloc()`

```
void* vrna_alloc (
    unsigned size )
```

```
#include <ViennaRNA/utils/basic.h>
```

Allocate space safely.

##### Parameters

<i>size</i>	The size of the memory to be allocated in bytes
-------------	---

##### Returns

A pointer to the allocated memory

#### 15.46.3.2 `vrna_realloc()`

```
void* vrna_realloc (
    void * p,
    unsigned size )
```

```
#include <ViennaRNA/utils/basic.h>
```

Reallocate space safely.

##### Parameters

<i>p</i>	A pointer to the memory region to be reallocated
<i>size</i>	The size of the memory to be allocated in bytes

##### Returns

A pointer to the newly allocated memory

#### 15.46.3.3 `vrna_urn()`

```
double vrna_urn (
    void )
```

```
#include <ViennaRNA/utils/basic.h>
```

get a random number from [0..1]

See also

[vrna\\_int\\_urn\(\)](#), [vrna\\_init\\_rand\(\)](#)

Note

Usually implemented by calling *erand48()*.

Returns

A random number in range [0..1]

#### 15.46.3.4 vrna\_int\_urn()

```
int vrna_int_urn (
    int from,
    int to )
```

```
#include <ViennaRNA/utils/basic.h>
```

Generates a pseudo random integer in a specified range.

See also

[vrna\\_urn\(\)](#), [vrna\\_init\\_rand\(\)](#)

Parameters

<i>from</i>	The first number in range
<i>to</i>	The last number in range

Returns

A pseudo random number in range [from, to]

#### 15.46.3.5 vrna\_time\_stamp()

```
char* vrna_time_stamp (
    void )
```

```
#include <ViennaRNA/utils/basic.h>
```

Get a timestamp.

Returns a string containing the current date in the format

```
Fri Mar 19 21:10:57 1993
```

**Returns**

A string containing the timestamp

**15.46.3.6 get\_input\_line()**

```
unsigned int get_input_line (
    char ** string,
    unsigned int options )
```

```
#include <ViennaRNA/utils/basic.h>
```

Retrieve a line from 'stdin' safely while skipping comment characters and other features This function returns the type of input it has read if recognized. An option argument allows one to switch between different reading modes. Currently available options are:

#VRNA\_INPUT\_NOPRINT\_COMMENTS, [VRNA\\_INPUT\\_NOSKIP\\_COMMENTS](#), #VRNA\_INPUT\_NOELIM\_WS\_SUFFIX

pass a collection of options as one value like this:

```
get_input_line(string, option_1 | option_2 | option_n)
```

If the function recognizes the type of input, it will report it in the return value. It also reports if a user defined 'quit' command (-sign on 'stdin') was given. Possible return values are:

[VRNA\\_INPUT\\_FASTA\\_HEADER](#), [VRNA\\_INPUT\\_ERROR](#), [VRNA\\_INPUT\\_MISC](#), [VRNA\\_INPUT\\_QUIT](#)

**Parameters**

<i>string</i>	A pointer to the character array that contains the line read
<i>options</i>	A collection of options for switching the functions behavior

**Returns**

A flag with information about what has been read

**15.46.3.7 vrna\_idx\_row\_wise()**

```
int* vrna_idx_row_wise (
    unsigned int length )
```

```
#include <ViennaRNA/utils/basic.h>
```

Get an index mapper array (iindx) for accessing the energy matrices, e.g. in partition function related functions.

Access of a position "(i,j)" is then accomplished by using

```
(i,j) ~ iindx[i]-j
```

This function is necessary as most of the two-dimensional energy matrices are actually one-dimensional arrays throughout the ViennaRNA Package

Consult the implemented code to find out about the mapping formula ;)

See also

[vrna\\_idx\\_col\\_wise\(\)](#)

#### Parameters

<i>length</i>	The length of the RNA sequence
---------------	--------------------------------

#### Returns

The mapper array

#### 15.46.3.8 vrna\_idx\_col\_wise()

```
int* vrna_idx_col_wise (
    unsigned int length )
```

```
#include <ViennaRNA/utils/basic.h>
```

Get an index mapper array (indx) for accessing the energy matrices, e.g. in MFE related functions.

Access of a position "(i,j)" is then accomplished by using

```
(i,j) ~ indx[j]+i
```

This function is necessary as most of the two-dimensional energy matrices are actually one-dimensional arrays throughout the ViennaRNAPackage

Consult the implemented code to find out about the mapping formula ;)

See also

[vrna\\_idx\\_row\\_wise\(\)](#)

#### Parameters

<i>length</i>	The length of the RNA sequence
---------------	--------------------------------

#### Returns

The mapper array

#### 15.46.4 Variable Documentation



#### 15.46.4.1 xsubi

```
unsigned short xsubi[3]
```

```
#include <ViennaRNA/utils/basic.h>
```

Current 48 bit random number.

This variable is used by [vrna\\_urn\(\)](#). These should be set to some random number seeds before the first call to [vrna\\_urn\(\)](#).

See also

[vrna\\_urn\(\)](#)

## 15.47 Exterior Loops

Functions to evaluate the free energy contributions for exterior loops.

### 15.47.1 Detailed Description

Functions to evaluate the free energy contributions for exterior loops.

Collaboration diagram for Exterior Loops:

#### Files

- file [external.h](#)  
*Energy evaluation of exterior loops for MFE and partition function calculations.*

#### Basic free energy interface

- int [vrna\\_E\\_ext\\_stem](#) (unsigned int type, int n5d, int n3d, [vrna\\_param\\_t](#) \*p)  
*Evaluate a stem branching off the exterior loop.*
- int [vrna\\_E\\_ext\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j)  
*Evaluate the free energy of a base pair in the exterior loop.*
- int [vrna\\_E\\_ext\\_loop\\_5](#) ([vrna\\_fold\\_compound\\_t](#) \*fc)
- int [vrna\\_E\\_ext\\_loop\\_3](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i)

#### Boltzmann weight (partition function) interface

- typedef struct vrna\_mx\_pf\_aux\_el\_s \* [vrna\\_mx\\_pf\\_aux\\_el\\_t](#)  
*Auxiliary helper arrays for fast exterior loop computations.*
- [FLT\\_OR\\_DBL vrna\\_exp\\_E\\_ext\\_stem](#) (unsigned int type, int n5d, int n3d, [vrna\\_exp\\_param\\_t](#) \*p)  
*Evaluate a stem branching off the exterior loop (Boltzmann factor version)*
- struct vrna\_mx\_pf\_aux\_el\_s \* [vrna\\_exp\\_E\\_ext\\_fast\\_init](#) ([vrna\\_fold\\_compound\\_t](#) \*fc)
- void [vrna\\_exp\\_E\\_ext\\_fast\\_rotate](#) (struct vrna\_mx\_pf\_aux\_el\_s \*aux\_mx)
- void [vrna\\_exp\\_E\\_ext\\_fast\\_free](#) (struct vrna\_mx\_pf\_aux\_el\_s \*aux\_mx)
- [FLT\\_OR\\_DBL vrna\\_exp\\_E\\_ext\\_fast](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j, struct vrna\_mx\_pf\_aux\_el\_s \*aux\_mx)
- void [vrna\\_exp\\_E\\_ext\\_fast\\_update](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int j, struct vrna\_mx\_pf\_aux\_el\_s \*aux\_mx ↵  
mx)

### 15.47.2 Typedef Documentation

15.47.2.1 `vrna_mx_pf_aux_el_t`

```
typedef struct vrna_mx_pf_aux_el_s* vrna_mx_pf_aux_el_t

#include <ViennaRNA/loops/external.h>
```

Auxiliary helper arrays for fast exterior loop computations.

## See also

`vrna_exp_E_ext_fast_init()`, `vrna_exp_E_ext_fast_rotate()`, `vrna_exp_E_ext_fast_free()`, `vrna_exp_E_ext_fast()`

## 15.47.3 Function Documentation

15.47.3.1 `vrna_E_ext_stem()`

```
int vrna_E_ext_stem (
    unsigned int type,
    int n5d,
    int n3d,
    vrna_param_t * p )

#include <ViennaRNA/loops/external.h>
```

Evaluate a stem branching off the exterior loop.

Given a base pair  $(i, j)$  encoded by *type*, compute the energy contribution including dangling-end/terminal-mismatch contributions. Instead of returning the energy contribution per-se, this function returns the corresponding Boltzmann factor. If either of the adjacent nucleotides  $(i - 1)$  and  $(j + 1)$  must not contribute stacking energy, the corresponding encoding must be  $-1$ .

## See also

`vrna_E_exp_stem()`

## Parameters

<i>type</i>	The base pair encoding
<i>n5d</i>	The encoded nucleotide directly adjacent at the 5' side of the base pair (may be -1)
<i>n3d</i>	The encoded nucleotide directly adjacent at the 3' side of the base pair (may be -1)
<i>p</i>	The pre-computed energy parameters

## Returns

The energy contribution of the introduced exterior-loop stem

### 15.47.3.2 vrna\_E\_ext\_loop()

```
int vrna_E_ext_loop (
    vrna_fold_compound_t * fc,
    int i,
    int j )

#include <ViennaRNA/loops/external.h>
```

Evaluate the free energy of a base pair in the exterior loop.

Evaluate the free energy of a base pair connecting two nucleotides in the exterior loop and take hard constraints into account.

Typically, this is simply dangling end contributions of the adjacent nucleotides, potentially a terminal A-U mismatch penalty, and maybe some generic soft constraint contribution for that decomposition.

#### Note

For dangles == 1 || 3 this function also evaluates the three additional pairs  $(i + 1, j)$ ,  $(i, j - 1)$ , and  $(i + 1, j - 1)$  and returns the minimum for all four possibilities in total.

#### Parameters

<i>fc</i>	Fold compound to work on (defines the model and parameters)
<i>i</i>	5' position of the base pair
<i>j</i>	3' position of the base pair

#### Returns

Free energy contribution that arises when this pair is formed in the exterior loop

### 15.47.3.3 vrna\_exp\_E\_ext\_stem()

```
FLT_OR_DBL vrna_exp_E_ext_stem (
    unsigned int type,
    int n5d,
    int n3d,
    vrna_exp_param_t * p )

#include <ViennaRNA/loops/external.h>
```

Evaluate a stem branching off the exterior loop (Boltzmann factor version)

Given a base pair  $(i, j)$  encoded by *type*, compute the energy contribution including dangling-end/terminal-mismatch contributions. Instead of returning the energy contribution per-se, this function returns the corresponding Boltzmann factor. If either of the adjacent nucleotides  $(i - 1)$  and  $(j + 1)$  must not contribute stacking energy, the corresponding encoding must be  $-1$ .

#### See also

[vrna\\_E\\_ext\\_stem\(\)](#)

## Parameters

<i>type</i>	The base pair encoding
<i>n5d</i>	The encoded nucleotide directly adjacent at the 5' side of the base pair (may be -1)
<i>n3d</i>	The encoded nucleotide directly adjacent at the 3' side of the base pair (may be -1)
<i>p</i>	The pre-computed energy parameters (Boltzmann factor version)

## Returns

The Boltzmann weighted energy contribution of the introduced exterior-loop stem

## 15.48 Hairpin Loops

Functions to evaluate the free energy contributions for hairpin loops.

### 15.48.1 Detailed Description

Functions to evaluate the free energy contributions for hairpin loops.

Collaboration diagram for Hairpin Loops:

#### Files

- file [hairpin.h](#)  
*Energy evaluation of hairpin loops for MFE and partition function calculations.*

#### Basic free energy interface

- int [vrna\\_E\\_hp\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j)  
*Evaluate the free energy of a hairpin loop and consider hard constraints if they apply.*
- int [vrna\\_E\\_ext\\_hp\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j)  
*Evaluate the free energy of an exterior hairpin loop and consider possible hard constraints.*
- int [vrna\\_eval\\_ext\\_hp\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j)  
*Evaluate free energy of an exterior hairpin loop.*
- int [vrna\\_eval\\_hp\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j)  
*Evaluate free energy of a hairpin loop.*
- PRIVATE int [E\\_Hairpin](#) (int size, int type, int si1, int sj1, const char \*string, [vrna\\_param\\_t](#) \*P)  
*Compute the Energy of a hairpin-loop.*

#### Boltzmann weight (partition function) interface

- PRIVATE [FLT\\_OR\\_DBL](#) [exp\\_E\\_Hairpin](#) (int u, int type, short si1, short sj1, const char \*string, [vrna\\_exp\\_param\\_t](#) \*P)  
*Compute Boltzmann weight  $e^{-\Delta G/kT}$  of a hairpin loop.*
- [FLT\\_OR\\_DBL](#) [vrna\\_exp\\_E\\_hp\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j)  
*High-Level function for hairpin loop energy evaluation (partition function variant)*

### 15.48.2 Function Documentation

15.48.2.1 `vrna_E_hp_loop()`

```
int vrna_E_hp_loop (
    vrna_fold_compound_t * fc,
    int i,
    int j )

#include <ViennaRNA/loops/hairpin.h>
```

Evaluate the free energy of a hairpin loop and consider hard constraints if they apply.

This function evaluates the free energy of a hairpin loop

In case the base pair is not allowed due to a constraint conflict, this function returns `INF`.

**Note**

This function is polymorphic! The provided `vrna_fold_compound_t` may be of type `VRNA_FC_TYPE_SINGLE` or `VRNA_FC_TYPE_COMPARATIVE`

**Parameters**

<i>fc</i>	The <code>vrna_fold_compound_t</code> that stores all relevant model settings
<i>i</i>	The 5' nucleotide of the base pair (3' to evaluate the pair as exterior hairpin loop)
<i>j</i>	The 3' nucleotide of the base pair (5' to evaluate the pair as exterior hairpin loop)

**Returns**

The free energy of the hairpin loop in 10cal/mol

15.48.2.2 `vrna_E_ext_hp_loop()`

```
int vrna_E_ext_hp_loop (
    vrna_fold_compound_t * fc,
    int i,
    int j )

#include <ViennaRNA/loops/hairpin.h>
```

Evaluate the free energy of an exterior hairpin loop and consider possible hard constraints.

**Note**

This function is polymorphic! The provided `vrna_fold_compound_t` may be of type `VRNA_FC_TYPE_SINGLE` or `VRNA_FC_TYPE_COMPARATIVE`

15.48.2.3 `vrna_eval_hp_loop()`

```
int vrna_eval_hp_loop (
    vrna_fold_compound_t * fc,
    int i,
    int j )

#include <ViennaRNA/loops/hairpin.h>
```

Evaluate free energy of a hairpin loop.

**Note**

This function is polymorphic! The provided `vrna_fold_compound_t` may be of type `VRNA_FC_TYPE_SINGLE` or `VRNA_FC_TYPE_COMPARATIVE`

**Parameters**

<i>fc</i>	The <code>vrna_fold_compound_t</code> for the particular energy evaluation
<i>i</i>	5'-position of the base pair
<i>j</i>	3'-position of the base pair

**Returns**

Free energy of the hairpin loop closed by  $(i, j)$  in deka-kal/mol

**SWIG Wrapper Notes** This function is attached as method `eval_hp_loop()` to objects of type `fold_compound`

15.48.2.4 `E_Hairpin()`

```
PRIVATE int E_Hairpin (
    int size,
    int type,
    int sil,
    int sjl,
    const char * string,
    vrna_param_t * P )

#include <ViennaRNA/loops/hairpin.h>
```

Compute the Energy of a hairpin-loop.

To evaluate the free energy of a hairpin-loop, several parameters have to be known. A general hairpin-loop has this structure:

```

      a3 a4
a2      a5
a1      a6
  X - Y
  |   |
  5'  3'
```



where X-Y marks the closing pair [e.g. a (**G,C**) pair]. The length of this loop is 6 as there are six unpaired nucleotides (a1-a6) enclosed by (X,Y). The 5' mismatching nucleotide is a1 while the 3' mismatch is a6. The nucleotide sequence of this loop is "a1.a2.a3.a4.a5.a6"

#### Note

The parameter sequence should contain the sequence of the loop in capital letters of the nucleic acid alphabet if the loop size is below 7. This is useful for unusually stable tri-, tetra- and hexa-loops which are treated differently (based on experimental data) if they are tabulated.

#### See also

[scale\\_parameters\(\)](#)  
[vrna\\_param\\_t](#)

#### Warning

Not (really) thread safe! A threadsafe implementation will replace this function in a future release!  
 Energy evaluation may change due to updates in global variable "tetra\_loop"

#### Parameters

<i>size</i>	The size of the loop (number of unpaired nucleotides)
<i>type</i>	The pair type of the base pair closing the hairpin
<i>si1</i>	The 5'-mismatching nucleotide
<i>sj1</i>	The 3'-mismatching nucleotide
<i>string</i>	The sequence of the loop (May be NULL, otherwise mst be at least <i>size</i> + 2 long)
<i>P</i>	The datastructure containing scaled energy parameters

#### Returns

The Free energy of the Hairpin-loop in dcal/mol

#### 15.48.2.5 exp\_E\_Hairpin()

```
PRIVATE FLT_OR_DBL exp_E_Hairpin (
    int u,
    int type,
    short sil,
    short sjl,
    const char * string,
    vrna_exp_param_t * P )
```

```
#include <ViennaRNA/loops/hairpin.h>
```

Compute Boltzmann weight  $e^{-\Delta G/kT}$  of a hairpin loop.

multiply by scale[u+2]

## See also

[get\\_scaled\\_pf\\_parameters\(\)](#)  
[vrna\\_exp\\_param\\_t](#)  
[E\\_Hairpin\(\)](#)

## Warning

Not (really) thread safe! A threadsafe implementation will replace this function in a future release!  
 Energy evaluation may change due to updates in global variable "tetra\_loop"

## Parameters

<i>u</i>	The size of the loop (number of unpaired nucleotides)
<i>type</i>	The pair type of the base pair closing the hairpin
<i>si1</i>	The 5'-mismatching nucleotide
<i>sj1</i>	The 3'-mismatching nucleotide
<i>string</i>	The sequence of the loop (May be NULL, otherwise mst be at least <i>size</i> + 2 long)
<i>P</i>	The datastructure containing scaled Boltzmann weights of the energy parameters

## Returns

The Boltzmann weight of the Hairpin-loop

## 15.48.2.6 vrna\_exp\_E\_hp\_loop()

```
FLT_OR_DBL vrna_exp_E_hp_loop (
    vrna_fold_compound_t * fc,
    int i,
    int j )
```

```
#include <ViennaRNA/loops/hairpin.h>
```

High-Level function for hairpin loop energy evaluation (partition function variant)

## See also

[vrna\\_E\\_hp\\_loop\(\)](#) for it's free energy counterpart

## Note

This function is polymorphic! The provided [vrna\\_fold\\_compound\\_t](#) may be of type [VRNA\\_FC\\_TYPE\\_SINGLE](#) or [VRNA\\_FC\\_TYPE\\_COMPARATIVE](#)

## 15.49 Internal Loops

Functions to evaluate the free energy contributions for internal loops.

### 15.49.1 Detailed Description

Functions to evaluate the free energy contributions for internal loops.

Collaboration diagram for Internal Loops:

#### Files

- file [internal.h](#)

*Energy evaluation of interior loops for MFE and partition function calculations.*

#### Basic free energy interface

- int [vrna\\_E\\_int\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j)
- int [vrna\\_eval\\_int\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j, int k, int l)  
*Evaluate the free energy contribution of an interior loop with delimiting base pairs  $(i, j)$  and  $(k, l)$ .*
- int [vrna\\_E\\_ext\\_int\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j, int \*ip, int \*iq)
- int [vrna\\_E\\_stack](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j)

#### Boltzmann weight (partition function) interface

- [FLT\\_OR\\_DBL vrna\\_exp\\_E\\_int\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j)
- [FLT\\_OR\\_DBL vrna\\_exp\\_E\\_interior\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j, int k, int l)

### 15.49.2 Function Documentation

#### 15.49.2.1 vrna\_eval\_int\_loop()

```
int vrna_eval_int_loop (
    vrna\_fold\_compound\_t * vc,
    int i,
    int j,
    int k,
    int l )

#include <ViennaRNA/loops/internal.h>
```

Evaluate the free energy contribution of an interior loop with delimiting base pairs  $(i, j)$  and  $(k, l)$ .

#### Note

This function is polymorphic, i.e. it accepts [vrna\\_fold\\_compound\\_t](#) of type [VRNA\\_FC\\_TYPE\\_SINGLE](#) as well as [VRNA\\_FC\\_TYPE\\_COMPARATIVE](#)

**SWIG Wrapper Notes** This function is attached as method [eval\\_int\\_loop\(\)](#) to objects of type *fold\_compound*

## 15.50 Multibranch Loops

Functions to evaluate the free energy contributions for multibranch loops.

### 15.50.1 Detailed Description

Functions to evaluate the free energy contributions for multibranch loops.

Collaboration diagram for Multibranch Loops:

#### Files

- file [multibranch.h](#)  
*Energy evaluation of multibranch loops for MFE and partition function calculations.*

#### Basic free energy interface

- int [vrna\\_E\\_mb\\_loop\\_stack](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j)  
*Evaluate energy of a multi branch helices stacking onto closing pair (i,j)*
- int [vrna\\_E\\_mb\\_loop\\_fast](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j, int \*dmli1, int \*dmli2)
- int [E\\_ml\\_rightmost\\_stem](#) (int i, int j, [vrna\\_fold\\_compound\\_t](#) \*fc)
- int [vrna\\_E\\_ml\\_stems\\_fast](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j, int \*fmi, int \*dmli)

#### Boltzmann weight (partition function) interface

- typedef struct [vrna\\_mx\\_pf\\_aux\\_ml\\_s](#) \* [vrna\\_mx\\_pf\\_aux\\_ml\\_t](#)  
*Auxiliary helper arrays for fast exterior loop computations.*
- [FLT\\_OR\\_DBL](#) [vrna\\_exp\\_E\\_mb\\_loop\\_fast](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j, [vrna\\_mx\\_pf\\_aux\\_ml\\_t](#) aux\_mx)
- [vrna\\_mx\\_pf\\_aux\\_ml\\_t](#) [vrna\\_exp\\_E\\_ml\\_fast\\_init](#) ([vrna\\_fold\\_compound\\_t](#) \*fc)
- void [vrna\\_exp\\_E\\_ml\\_fast\\_rotate](#) ([vrna\\_mx\\_pf\\_aux\\_ml\\_t](#) aux\_mx)
- void [vrna\\_exp\\_E\\_ml\\_fast\\_free](#) ([vrna\\_mx\\_pf\\_aux\\_ml\\_t](#) aux\_mx)
- const [FLT\\_OR\\_DBL](#) \* [vrna\\_exp\\_E\\_ml\\_fast\\_qqm](#) (struct [vrna\\_mx\\_pf\\_aux\\_ml\\_s](#) \*aux\_mx)
- const [FLT\\_OR\\_DBL](#) \* [vrna\\_exp\\_E\\_ml\\_fast\\_qqm1](#) (struct [vrna\\_mx\\_pf\\_aux\\_ml\\_s](#) \*aux\_mx)
- [FLT\\_OR\\_DBL](#) [vrna\\_exp\\_E\\_ml\\_fast](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j, [vrna\\_mx\\_pf\\_aux\\_ml\\_t](#) aux\_mx)

### 15.50.2 Typedef Documentation

### 15.50.2.1 vrna\_mx\_pf\_aux\_ml\_t

```
typedef struct vrna_mx_pf_aux_ml_s* vrna_mx_pf_aux_ml_t  
  
#include <ViennaRNA/loops/multibranch.h>
```

Auxiliary helper arrays for fast exterior loop computations.

See also

`vrna_exp_E_ml_fast_init()`, `vrna_exp_E_ml_fast_rotate()`, `vrna_exp_E_ml_fast_free()`, `vrna_exp_E_ml_fast()`

## 15.50.3 Function Documentation

### 15.50.3.1 vrna\_E\_mb\_loop\_stack()

```
int vrna_E_mb_loop_stack (  
    vrna_fold_compound_t * fc,  
    int i,  
    int j )
```

```
#include <ViennaRNA/loops/multibranch.h>
```

Evaluate energy of a multi branch helices stacking onto closing pair (i,j)

Computes total free energy for coaxial stacking of (i,j) with (i+1.k) or (k+1.j-1)

## 15.51 Deprecated Interface for Global MFE Prediction

### 15.51.1 Detailed Description

Collaboration diagram for Deprecated Interface for Global MFE Prediction:

#### Files

- file [alifold.h](#)  
*Functions for comparative structure prediction using RNA sequence alignments.*
- file [cofold.h](#)  
*MFE implementations for RNA-RNA interaction.*
- file [fold.h](#)  
*MFE calculations for single RNA sequences.*

#### Functions

- float [cofold](#) (const char \*sequence, char \*structure)  
*Compute the minimum free energy of two interacting RNA molecules.*
- float [cofold\\_par](#) (const char \*string, char \*structure, [vrna\\_param\\_t](#) \*parameters, int is\_constrained)  
*Compute the minimum free energy of two interacting RNA molecules.*
- void [free\\_co\\_arrays](#) (void)  
*Free memory occupied by [cofold\(\)](#)*
- void [update\\_cofold\\_params](#) (void)  
*Recalculate parameters.*
- void [update\\_cofold\\_params\\_par](#) ([vrna\\_param\\_t](#) \*parameters)  
*Recalculate parameters.*
- void [export\\_cofold\\_arrays\\_gq](#) (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*fc\_p, int \*\*ggg\_p, int \*\*indx\_p, char \*\*ptype\_p)  
*Export the arrays of partition function cofold (with gquadruplex support)*
- void [export\\_cofold\\_arrays](#) (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*fc\_p, int \*\*indx\_p, char \*\*ptype\_p)  
*Export the arrays of partition function cofold.*
- void [get\\_monomere\\_mfes](#) (float \*e1, float \*e2)  
*get\_monomer\_free\_energies*
- void [initialize\\_cofold](#) (int length)
- float [fold\\_par](#) (const char \*sequence, char \*structure, [vrna\\_param\\_t](#) \*parameters, int is\_constrained, int is\_circular)  
*Compute minimum free energy and an appropriate secondary structure of an RNA sequence.*
- float [fold](#) (const char \*sequence, char \*structure)  
*Compute minimum free energy and an appropriate secondary structure of an RNA sequence.*
- float [circfold](#) (const char \*sequence, char \*structure)  
*Compute minimum free energy and an appropriate secondary structure of a circular RNA sequence.*
- void [free\\_arrays](#) (void)  
*Free arrays for mfe folding.*
- void [update\\_fold\\_params](#) (void)  
*Recalculate energy parameters.*

- void [update\\_fold\\_params\\_par](#) ([vrna\\_param\\_t](#) \*parameters)  
*Recalculate energy parameters.*
- void [export\\_fold\\_arrays](#) (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*indx\_p, char \*\*ptype\_p)
- void [export\\_fold\\_arrays\\_par](#) (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*indx\_p, char \*\*ptype\_p, [vrna\\_param\\_t](#) \*\*P\_p)
- void [export\\_circfold\\_arrays](#) (int \*Fc\_p, int \*FcH\_p, int \*FcI\_p, int \*FcM\_p, int \*\*fM2\_p, int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*indx\_p, char \*\*ptype\_p)
- void [export\\_circfold\\_arrays\\_par](#) (int \*Fc\_p, int \*FcH\_p, int \*FcI\_p, int \*FcM\_p, int \*\*fM2\_p, int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*indx\_p, char \*\*ptype\_p, [vrna\\_param\\_t](#) \*\*P\_p)
- int [LoopEnergy](#) (int n1, int n2, int type, int type\_2, int si1, int sj1, int sp1, int sq1)
- int [HairpinE](#) (int size, int type, int si1, int sj1, const char \*string)
- void [initialize\\_fold](#) (int length)
  
- float [alifold](#) (const char \*\*strings, char \*structure)  
*Compute MFE and according consensus structure of an alignment of sequences.*
- float [circularifold](#) (const char \*\*strings, char \*structure)  
*Compute MFE and according structure of an alignment of sequences assuming the sequences are circular instead of linear.*
- void [free\\_alifold\\_arrays](#) (void)  
*Free the memory occupied by MFE alifold functions.*

## 15.51.2 Function Documentation

### 15.51.2.1 alifold()

```
float alifold (
    const char ** strings,
    char * structure )

#include <ViennaRNA/alifold.h>
```

Compute MFE and according consensus structure of an alignment of sequences.

This function predicts the consensus structure for the aligned 'sequences' and returns the minimum free energy; the mfe structure in bracket notation is returned in 'structure'.

Sufficient space must be allocated for 'structure' before calling [alifold\(\)](#).

**Deprecated** Usage of this function is discouraged! Use [vrna\\_alifold\(\)](#), or [vrna\\_mfe\(\)](#) instead!

See also

[vrna\\_alifold\(\)](#), [vrna\\_mfe\(\)](#)

#### Parameters

<i>strings</i>	A pointer to a NULL terminated array of character arrays
<i>structure</i>	A pointer to a character array that may contain a constraining consensus structure (will be overwritten by a consensus structure that exhibits the MFE)

**Returns**

The free energy score in kcal/mol

**15.51.2.2 cofold()**

```
float cofold (
    const char * sequence,
    char * structure )
```

```
#include <ViennaRNA/cofold.h>
```

Compute the minimum free energy of two interacting RNA molecules.

The code is analog to the [fold\(\)](#) function. If `cut_point == -1` results should be the same as with [fold\(\)](#).

**Deprecated** use [vrna\\_mfe\\_dimer\(\)](#) instead

**Parameters**

<i>sequence</i>	The two sequences concatenated
<i>structure</i>	Will hold the barcket dot structure of the dimer molecule

**Returns**

minimum free energy of the structure

**15.51.2.3 cofold\_par()**

```
float cofold_par (
    const char * string,
    char * structure,
    vrna_param_t * parameters,
    int is_constrained )
```

```
#include <ViennaRNA/cofold.h>
```

Compute the minimum free energy of two interacting RNA molecules.

**Deprecated** use [vrna\\_mfe\\_dimer\(\)](#) instead



#### 15.51.2.4 free\_co\_arrays()

```
void free_co_arrays (
    void )
```

```
#include <ViennaRNA/cofold.h>
```

Free memory occupied by [cofold\(\)](#)

**Deprecated** This function will only free memory allocated by a prior call of [cofold\(\)](#) or [cofold\\_par\(\)](#). See [vrna\\_mfe\\_dimer\(\)](#) for how to use the new API

#### Note

folding matrices now reside in the fold compound, and should be free'd there

#### See also

[vrna\\_fc\\_destroy\(\)](#), [vrna\\_mfe\\_dimer\(\)](#)

#### 15.51.2.5 update\_cofold\_params()

```
void update_cofold_params (
    void )
```

```
#include <ViennaRNA/cofold.h>
```

Recalculate parameters.

**Deprecated** See [vrna\\_params\\_subst\(\)](#) for an alternative using the new API

#### 15.51.2.6 update\_cofold\_params\_par()

```
void update_cofold_params_par (
    vrna_param_t * parameters )
```

```
#include <ViennaRNA/cofold.h>
```

Recalculate parameters.

**Deprecated** See [vrna\\_params\\_subst\(\)](#) for an alternative using the new API

### 15.51.2.7 export\_cofold\_arrays\_gq()

```
void export_cofold_arrays_gq (
    int ** f5_p,
    int ** c_p,
    int ** fML_p,
    int ** fM1_p,
    int ** fc_p,
    int ** ggg_p,
    int ** indx_p,
    char ** ptype_p )
```

```
#include <ViennaRNA/cofold.h>
```

Export the arrays of partition function cofold (with gquadruplex support)

Export the cofold arrays for use e.g. in the concentration Computations or suboptimal secondary structure backtracking

**Deprecated** folding matrices now reside within the fold compound. Thus, this function will only work in conjunction with a prior call to [cofold\(\)](#) or [cofold\\_par\(\)](#)

See also

[vrna\\_mfe\\_dimer\(\)](#) for the new API

#### Parameters

<i>f5_p</i>	A pointer to the 'f5' array, i.e. array containing best free energy in interval [1..j]
<i>c_p</i>	A pointer to the 'c' array, i.e. array containing best free energy in interval [i..j] given that i pairs with j
<i>fML_p</i>	A pointer to the 'M' array, i.e. array containing best free energy in interval [i..j] for any multiloop segment with at least one stem
<i>fM1_p</i>	A pointer to the 'M1' array, i.e. array containing best free energy in interval [i..j] for multiloop segment with exactly one stem
<i>fc_p</i>	A pointer to the 'fc' array, i.e. array ...
<i>ggg_p</i>	A pointer to the 'ggg' array, i.e. array containing best free energy of a gquadruplex delimited by [i..j]
<i>indx_p</i>	A pointer to the indexing array used for accessing the energy matrices
<i>ptype↔_p</i>	A pointer to the ptype array containing the base pair types for each possibility (i,j)

### 15.51.2.8 export\_cofold\_arrays()

```
void export_cofold_arrays (
    int ** f5_p,
    int ** c_p,
    int ** fML_p,
    int ** fM1_p,
    int ** fc_p,
```

```
int ** indx_p,
char ** ptype_p )
```

```
#include <ViennaRNA/cofold.h>
```

Export the arrays of partition function cofold.

Export the cofold arrays for use e.g. in the concentration Computations or suboptimal secondary structure backtracking

**Deprecated** folding matrices now reside within the [vrna\\_fold\\_compound\\_t](#). Thus, this function will only work in conjunction with a prior call to the deprecated functions [cofold\(\)](#) or [cofold\\_par\(\)](#)

See also

[vrna\\_mfe\\_dimer\(\)](#) for the new API

#### Parameters

<i>f5_p</i>	A pointer to the 'f5' array, i.e. array containing best free energy in interval [1..j]
<i>c_p</i>	A pointer to the 'c' array, i.e. array containing best free energy in interval [i..j] given that i pairs with j
<i>fML_p</i>	A pointer to the 'M' array, i.e. array containing best free energy in interval [i..j] for any multiloop segment with at least one stem
<i>fM1_p</i>	A pointer to the 'M1' array, i.e. array containing best free energy in interval [i..j] for multiloop segment with exactly one stem
<i>fc_p</i>	A pointer to the 'fc' array, i.e. array ...
<i>indx_p</i>	A pointer to the indexing array used for accessing the energy matrices
<i>ptype_p</i>	A pointer to the ptype array containing the base pair types for each possibility (i,j)

#### 15.51.2.9 [get\\_monomere\\_mfes\(\)](#)

```
void get_monomere_mfes (
    float * e1,
    float * e2 )
```

```
#include <ViennaRNA/cofold.h>
```

[get\\_monomer\\_free\\_energies](#)

Export monomer free energies out of cofold arrays

**Deprecated** {This function is obsolete and will be removed soon!}

#### Parameters

<i>e1</i>	A pointer to a variable where the energy of molecule A will be written to
<i>e2</i>	A pointer to a variable where the energy of molecule B will be written to

**15.51.2.10 initialize\_cofold()**

```
void initialize_cofold (
    int length )

#include <ViennaRNA/cofold.h>

allocate arrays for folding
```

**Deprecated** {This function is obsolete and will be removed soon!}

**15.51.2.11 fold\_par()**

```
float fold_par (
    const char * sequence,
    char * structure,
    vrna_param_t * parameters,
    int is_constrained,
    int is_circular )

#include <ViennaRNA/fold.h>
```

Compute minimum free energy and an appropriate secondary structure of an RNA sequence.

The first parameter given, the RNA sequence, must be *uppercase* and should only contain an alphabet  $\Sigma$  that is understood by the RNAlib (e.g.  $\Sigma = \{A, U, C, G\}$ )

The second parameter, *structure*, must always point to an allocated block of memory with a size of at least `strlen(sequence) + 1`

If the third parameter is NULL, global model detail settings are assumed for the folding recursions. Otherwise, the provided parameters are used.

The fourth parameter indicates whether a secondary structure constraint in enhanced dot-bracket notation is passed through the structure parameter or not. If so, the characters "`| x < >`" are recognized to mark bases that are paired, unpaired, paired upstream, or downstream, respectively. Matching brackets "`( )`" denote base pairs, dots "`.`" are used for unconstrained bases.

To indicate that the RNA sequence is circular and thus has to be post-processed, set the last parameter to non-zero

After a successful call of `fold_par()`, a backtracked secondary structure (in dot-bracket notation) that exhibits the minimum of free energy will be written to the memory *structure* is pointing to. The function returns the minimum of free energy for any fold of the sequence given.

**Note**

OpenMP: Passing NULL to the 'parameters' argument involves access to several global model detail variables and thus is not to be considered threadsafe

**Deprecated** use `vrna_mfe()` instead!

**See also**

`vrna_mfe()`, `fold()`, `circfold()`, `vrna_md_t`, `set_energy_model()`, `get_scaled_parameters()`

## Parameters

<i>sequence</i>	RNA sequence
<i>structure</i>	A pointer to the character array where the secondary structure in dot-bracket notation will be written to
<i>parameters</i>	A data structure containing the pre-scaled energy contributions and the model details. (NULL may be passed, see OpenMP notes above)
<i>is_constrained</i>	Switch to indicate that a structure constraint is passed via the structure argument (0==off)
<i>is_circular</i>	Switch to (de-)activate post-processing steps in case RNA sequence is circular (0==off)

## Returns

the minimum free energy (MFE) in kcal/mol

## 15.51.2.12 fold()

```
float fold (
    const char * sequence,
    char * structure )
```

```
#include <ViennaRNA/fold.h>
```

Compute minimum free energy and an appropriate secondary structure of an RNA sequence.

This function essentially does the same thing as [fold\\_par\(\)](#). However, it takes its model details, i.e. [temperature](#), [dangles](#), [tetra\\_loop](#), [noGU](#), [no\\_closingGU](#), [fold\\_constrained](#), [noLonelyPairs](#) from the current global settings within the library

**Deprecated** use [vrna\\_fold\(\)](#), or [vrna\\_mfe\(\)](#) instead!

## See also

[fold\\_par\(\)](#), [circfold\(\)](#)

## Parameters

<i>sequence</i>	RNA sequence
<i>structure</i>	A pointer to the character array where the secondary structure in dot-bracket notation will be written to

## Returns

the minimum free energy (MFE) in kcal/mol

### 15.51.2.13 circfold()

```
float circfold (
    const char * sequence,
    char * structure )
```

```
#include <ViennaRNA/fold.h>
```

Compute minimum free energy and an appropriate secondary structure of a circular RNA sequence.

This function essentially does the same thing as [fold\\_par\(\)](#). However, it takes its model details, i.e. [temperature](#), [dangles](#), [tetra\\_loop](#), [noGU](#), [no\\_closingGU](#), [fold\\_constrained](#), [noLonelyPairs](#) from the current global settings within the library

**Deprecated** Use [vrna\\_circfold\(\)](#), or [vrna\\_mfe\(\)](#) instead!

See also

[fold\\_par\(\)](#), [circfold\(\)](#)

#### Parameters

<i>sequence</i>	RNA sequence
<i>structure</i>	A pointer to the character array where the secondary structure in dot-bracket notation will be written to

#### Returns

the minimum free energy (MFE) in kcal/mol

### 15.51.2.14 free\_arrays()

```
void free_arrays (
    void )
```

```
#include <ViennaRNA/fold.h>
```

Free arrays for mfe folding.

**Deprecated** See [vrna\\_fold\(\)](#), [vrna\\_circfold\(\)](#), or [vrna\\_mfe\(\)](#) and [vrna\\_fold\\_compound\\_t](#) for the usage of the new API!

**15.51.2.15** `update_fold_params()`

```
void update_fold_params (
    void )
```

```
#include <ViennaRNA/fold.h>
```

Recalculate energy parameters.

**Deprecated** For non-default model settings use the new API with [vrna\\_params\\_subst\(\)](#) and [vrna\\_mfe\(\)](#) instead!

**15.51.2.16** `update_fold_params_par()`

```
void update_fold_params_par (
    vrna_param_t * parameters )
```

```
#include <ViennaRNA/fold.h>
```

Recalculate energy parameters.

**Deprecated** For non-default model settings use the new API with [vrna\\_params\\_subst\(\)](#) and [vrna\\_mfe\(\)](#) instead!

**15.51.2.17** `export_fold_arrays()`

```
void export_fold_arrays (
    int ** f5_p,
    int ** c_p,
    int ** fML_p,
    int ** fM1_p,
    int ** indx_p,
    char ** ptype_p )
```

```
#include <ViennaRNA/fold.h>
```

**Deprecated** See [vrna\\_mfe\(\)](#) and [vrna\\_fold\\_compound\\_t](#) for the usage of the new API!

**15.51.2.18 export\_fold\_arrays\_par()**

```
void export_fold_arrays_par (
    int ** f5_p,
    int ** c_p,
    int ** fML_p,
    int ** fM1_p,
    int ** indx_p,
    char ** ptype_p,
    vrna_param_t ** P_p )

#include <ViennaRNA/fold.h>
```

**Deprecated** See [vrna\\_mfe\(\)](#) and [vrna\\_fold\\_compound\\_t](#) for the usage of the new API!

**15.51.2.19 export\_circfold\_arrays()**

```
void export_circfold_arrays (
    int * Fc_p,
    int * FcH_p,
    int * FcI_p,
    int * FcM_p,
    int ** fM2_p,
    int ** f5_p,
    int ** c_p,
    int ** fML_p,
    int ** fM1_p,
    int ** indx_p,
    char ** ptype_p )

#include <ViennaRNA/fold.h>
```

**Deprecated** See [vrna\\_mfe\(\)](#) and [vrna\\_fold\\_compound\\_t](#) for the usage of the new API!

**15.51.2.20 export\_circfold\_arrays\_par()**

```
void export_circfold_arrays_par (
    int * Fc_p,
    int * FcH_p,
    int * FcI_p,
    int * FcM_p,
    int ** fM2_p,
    int ** f5_p,
    int ** c_p,
    int ** fML_p,
    int ** fM1_p,
    int ** indx_p,
    char ** ptype_p,
    vrna_param_t ** P_p )

#include <ViennaRNA/fold.h>
```

**Deprecated** See [vrna\\_mfe\(\)](#) and [vrna\\_fold\\_compound\\_t](#) for the usage of the new API!



### 15.51.2.21 LoopEnergy()

```
int LoopEnergy (
    int n1,
    int n2,
    int type,
    int type_2,
    int sil,
    int sj1,
    int spl,
    int sq1 )

#include <ViennaRNA/fold.h>
```

**Deprecated** {This function is deprecated and will be removed soon. Use [E\\_IntLoop\(\)](#) instead!}

### 15.51.2.22 HairpinE()

```
int HairpinE (
    int size,
    int type,
    int sil,
    int sj1,
    const char * string )

#include <ViennaRNA/fold.h>
```

**Deprecated** {This function is deprecated and will be removed soon. Use [E\\_Hairpin\(\)](#) instead!}

### 15.51.2.23 initialize\_fold()

```
void initialize_fold (
    int length )

#include <ViennaRNA/fold.h>
```

Allocate arrays for folding

**Deprecated** See [vrna\\_mfe\(\)](#) and [vrna\\_fold\\_compound\\_t](#) for the usage of the new API!

#### 15.51.2.24 `circalifold()`

```
float circalifold (
    const char ** strings,
    char * structure )

#include <ViennaRNA/alifold.h>
```

Compute MFE and according structure of an alignment of sequences assuming the sequences are circular instead of linear.

**Deprecated** Usage of this function is discouraged! Use `vrna_alicircfold()`, and `vrna_mfe()` instead!

See also

`vrna_alicircfold()`, `vrna_alifold()`, `vrna_mfe()`

##### Parameters

<i>strings</i>	A pointer to a NULL terminated array of character arrays
<i>structure</i>	A pointer to a character array that may contain a constraining consensus structure (will be overwritten by a consensus structure that exhibits the MFE)

##### Returns

The free energy score in kcal/mol

#### 15.51.2.25 `free_alifold_arrays()`

```
void free_alifold_arrays (
    void )

#include <ViennaRNA/alifold.h>
```

Free the memory occupied by MFE alifold functions.

**Deprecated** Usage of this function is discouraged! It only affects memory being free'd that was allocated by an old API function before. Release of memory occupied by the newly introduced `vrna_fold_compound_t` is handled by `vrna_fold_compound_free()`

See also

`vrna_fold_compound_free()`

## 15.52 Deprecated Interface for Local (Sliding Window) MFE Prediction

### 15.52.1 Detailed Description

Collaboration diagram for Deprecated Interface for Local (Sliding Window) MFE Prediction:

#### Files

- file [Lfold.h](#)  
*Functions for locally optimal MFE structure prediction.*

#### Functions

- float [Lfold](#) (const char \*string, const char \*structure, int maxdist)  
*The local analog to [fold\(\)](#).*
- float [Lfoldz](#) (const char \*string, const char \*structure, int maxdist, int zsc, double min\_z)

### 15.52.2 Function Documentation

#### 15.52.2.1 Lfold()

```
float Lfold (
    const char * string,
    const char * structure,
    int maxdist )
```

```
#include <ViennaRNA/Lfold.h>
```

The local analog to [fold\(\)](#).

Computes the minimum free energy structure including only base pairs with a span smaller than 'maxdist'

**Deprecated** Use [vrna\\_mfe\\_window\(\)](#) instead!

#### 15.52.2.2 Lfoldz()

```
float Lfoldz (
    const char * string,
    const char * structure,
    int maxdist,
    int zsc,
    double min_z )
```

```
#include <ViennaRNA/Lfold.h>
```

**Deprecated** Use [vrna\\_mfe\\_window\\_zscore\(\)](#) instead!

## 15.53 Deprecated Interface for Global Partition Function Computation

### 15.53.1 Detailed Description

Collaboration diagram for Deprecated Interface for Global Partition Function Computation:

#### Files

- file [part\\_func\\_co.h](#)  
*Partition function for two RNA sequences.*

#### Functions

- float [pf\\_fold\\_par](#) (const char \*sequence, char \*structure, [vrna\\_exp\\_param\\_t](#) \*parameters, int calculate\_↔  
bppm, int is\_constrained, int is\_circular)  
*Compute the partition function  $Q$  for a given RNA sequence.*
- float [pf\\_fold](#) (const char \*sequence, char \*structure)  
*Compute the partition function  $Q$  of an RNA sequence.*
- float [pf\\_circ\\_fold](#) (const char \*sequence, char \*structure)  
*Compute the partition function of a circular RNA sequence.*
- void [free\\_pf\\_arrays](#) (void)  
*Free arrays for the partition function recursions.*
- void [update\\_pf\\_params](#) (int length)  
*Recalculate energy parameters.*
- void [update\\_pf\\_params\\_par](#) (int length, [vrna\\_exp\\_param\\_t](#) \*parameters)  
*Recalculate energy parameters.*
- [FLT\\_OR\\_DBL](#) \* [export\\_bppm](#) (void)  
*Get a pointer to the base pair probability array.*
- int [get\\_pf\\_arrays](#) (short \*\*S\_p, short \*\*S1\_p, char \*\*ptype\_p, [FLT\\_OR\\_DBL](#) \*\*qb\_p, [FLT\\_OR\\_DBL](#) \*\*qm↔  
\_p, [FLT\\_OR\\_DBL](#) \*\*q1k\_p, [FLT\\_OR\\_DBL](#) \*\*qln\_p)  
*Get the pointers to (almost) all relevant computation arrays used in partition function computation.*
- double [get\\_subseq\\_F](#) (int i, int j)  
*Get the free energy of a subsequence from the  $q[]$  array.*
- double [mean\\_bp\\_distance](#) (int length)  
*Get the mean base pair distance of the last partition function computation.*
- double [mean\\_bp\\_distance\\_pr](#) (int length, [FLT\\_OR\\_DBL](#) \*pr)  
*Get the mean base pair distance in the thermodynamic ensemble.*
- [vrna\\_ep\\_t](#) \* [stackProb](#) (double cutoff)  
*Get the probability of stacks.*
- void [init\\_pf\\_fold](#) (int length)  
*Allocate space for [pf\\_fold\(\)](#)*
- [vrna\\_dimer\\_pf\\_t](#) co [pf\\_fold](#) (char \*sequence, char \*structure)  
*Calculate partition function and base pair probabilities.*
- [vrna\\_dimer\\_pf\\_t](#) co [pf\\_fold\\_par](#) (char \*sequence, char \*structure, [vrna\\_exp\\_param\\_t](#) \*parameters, int  
calculate\_bppm, int is\_constrained)  
*Calculate partition function and base pair probabilities.*

- void `compute_probabilities` (double FAB, double FEA, double FEB, `vrna_ep_t` \*prAB, `vrna_ep_t` \*prA, `vrna_ep_t` \*prB, int Alength)  
*Compute Boltzmann probabilities of dimerization without homodimers.*
- void `init_co_pf_fold` (int length)
- `FLT_OR_DBL` \* `export_co_bppm` (void)  
*Get a pointer to the base pair probability array.*
- void `free_co_pf_arrays` (void)  
*Free the memory occupied by `co_pf_fold()`*
- void `update_co_pf_params` (int length)  
*Recalculate energy parameters.*
- void `update_co_pf_params_par` (int length, `vrna_exp_param_t` \*parameters)  
*Recalculate energy parameters.*
- void `assign_plist_from_db` (`vrna_ep_t` \*\*pl, const char \*struc, float pr)  
*Create a `vrna_ep_t` from a dot-bracket string.*
- void `assign_plist_from_pr` (`vrna_ep_t` \*\*pl, `FLT_OR_DBL` \*probs, int length, double cutoff)  
*Create a `vrna_ep_t` from a probability matrix.*
  
- float `alipf_fold_par` (const char \*\*sequences, char \*structure, `vrna_ep_t` \*\*pl, `vrna_exp_param_t` \*parameters, int calculate\_bppm, int is\_constrained, int is\_circular)
- float `alipf_fold` (const char \*\*sequences, char \*structure, `vrna_ep_t` \*\*pl)  
*The partition function version of `alifold()` works in analogy to `pf_fold()`. Pair probabilities and information about sequence covariations are returned via the 'pi' variable as a list of `vrna_pinfo_t` structs. The list is terminated by the first entry with `pi.i = 0`.*
- float `alipf_circ_fold` (const char \*\*sequences, char \*structure, `vrna_ep_t` \*\*pl)
- `FLT_OR_DBL` \* `export_ali_bppm` (void)  
*Get a pointer to the base pair probability array.*
- void `free_alipf_arrays` (void)  
*Free the memory occupied by folding matrices allocated by `alipf_fold`, `alipf_circ_fold`, etc.*
- char \* `alipbacktrack` (double \*prob)  
*Sample a consensus secondary structure from the Boltzmann ensemble according its probability.*
- int `get_alipf_arrays` (short \*\*\*S\_p, short \*\*\*S5\_p, short \*\*\*S3\_p, unsigned short \*\*\*a2s\_p, char \*\*\*Ss←\_p, `FLT_OR_DBL` \*\*qb\_p, `FLT_OR_DBL` \*\*qm\_p, `FLT_OR_DBL` \*\*q1k\_p, `FLT_OR_DBL` \*\*qln\_p, short \*\*pscore)  
*Get pointers to (almost) all relevant arrays used in `alifold`'s partition function computation.*

## 15.53.2 Function Documentation

### 15.53.2.1 `alipf_fold_par()`

```
float alipf_fold_par (
    const char ** sequences,
    char * structure,
    vrna_ep_t ** pl,
    vrna_exp_param_t * parameters,
    int calculate_bppm,
    int is_constrained,
    int is_circular )
```

```
#include <ViennaRNA/alifold.h>
```

**Deprecated** Use `vrna_pf()` instead

## Parameters

<i>sequences</i>	
<i>structure</i>	
<i>pl</i>	
<i>parameters</i>	
<i>calculate_bppm</i>	
<i>is_constrained</i>	
<i>is_circular</i>	

## Returns

## 15.53.2.2 pf\_fold\_par()

```
float pf_fold_par (
    const char * sequence,
    char * structure,
    vrna_exp_param_t * parameters,
    int calculate_bppm,
    int is_constrained,
    int is_circular )
```

```
#include <ViennaRNA/part_func.h>
```

Compute the partition function  $Q$  for a given RNA sequence.

If *structure* is not a NULL pointer on input, it contains on return a string consisting of the letters ". , | { } ( ) " denoting bases that are essentially unpaired, weakly paired, strongly paired without preference, weakly upstream (downstream) paired, or strongly up- (down-)stream paired bases, respectively. If *fold\_constrained* is not 0, the *structure* string is interpreted on input as a list of constraints for the folding. The character "x" marks bases that must be unpaired, matching brackets "( )" denote base pairs, all other characters are ignored. Any pairs conflicting with the constraint will be forbidden. This is usually sufficient to ensure the constraints are honored. If the parameter *calculate\_bppm* is set to 0 base pairing probabilities will not be computed (saving CPU time), otherwise after calculations took place *pr* will contain the probability that bases *i* and *j* pair.

**Deprecated** Use [vrna\\_pf\(\)](#) instead

## Note

The global array *pr* is deprecated and the user who wants the calculated base pair probabilities for further computations is advised to use the function [export\\_bppm\(\)](#)

## Postcondition

After successful run the hidden folding matrices are filled with the appropriate Boltzmann factors. Depending on whether the global variable [do\\_backtrack](#) was set the base pair probabilities are already computed and may be accessed for further usage via the [export\\_bppm\(\)](#) function. A call of [free\\_pf\\_arrays\(\)](#) will free all memory allocated by this function. Successive calls will first free previously allocated memory before starting the computation.

## See also

[vrna\\_pf\(\)](#), [bppm\\_to\\_structure\(\)](#), [export\\_bppm\(\)](#), [vrna\\_exp\\_params\(\)](#), [free\\_pf\\_arrays\(\)](#)

## Parameters

in	<i>sequence</i>	The RNA sequence input
in, out	<i>structure</i>	A pointer to a char array where a base pair probability information can be stored in a pseudo-dot-bracket notation (may be NULL, too)
in	<i>parameters</i>	Data structure containing the precalculated Boltzmann factors
in	<i>calculate_bppm</i>	Switch to Base pair probability calculations on/off (0==off)
in	<i>is_constrained</i>	Switch to indicate that a structure constraint is passed via the structure argument (0==off)
in	<i>is_circular</i>	Switch to (de-)activate postprocessing steps in case RNA sequence is circular (0==off)

## Returns

The Gibbs free energy of the ensemble ( $G = -RT \cdot \log(Q)$ ) in kcal/mol

## 15.53.2.3 pf\_fold()

```
float pf_fold (
    const char * sequence,
    char * structure )

#include <ViennaRNA/part_func.h>
```

Compute the partition function  $Q$  of an RNA sequence.

If *structure* is not a NULL pointer on input, it contains on return a string consisting of the letters ". , | { } ( ) " denoting bases that are essentially unpaired, weakly paired, strongly paired without preference, weakly upstream (downstream) paired, or strongly up- (down-)stream paired bases, respectively. If *fold\_constrained* is not 0, the *structure* string is interpreted on input as a list of constraints for the folding. The character "x" marks bases that must be unpaired, matching brackets " ( ) " denote base pairs, all other characters are ignored. Any pairs conflicting with the constraint will be forbidden. This is usually sufficient to ensure the constraints are honored. If *do\_backtrack* has been set to 0 base pairing probabilities will not be computed (saving CPU time), otherwise *pr* will contain the probability that bases  $i$  and  $j$  pair.

## Note

The global array *pr* is deprecated and the user who wants the calculated base pair probabilities for further computations is advised to use the function *export\_bppm()*.

**OpenMP:** This function is not entirely threadsafe. While the recursions are working on their own copies of data the model details for the recursions are determined from the global settings just before entering the recursions. Consider using *pf\_fold\_par()* for a really threadsafe implementation.

## Precondition

This function takes its model details from the global variables provided in *RNAlib*

## Postcondition

After successful run the hidden folding matrices are filled with the appropriate Boltzmann factors. Depending on whether the global variable *do\_backtrack* was set the base pair probabilities are already computed and may be accessed for further usage via the *export\_bppm()* function. A call of *free\_pf\_arrays()* will free all memory allocated by this function. Successive calls will first free previously allocated memory before starting the computation.

## See also

*pf\_fold\_par()*, *pf\_circ\_fold()*, *bppm\_to\_structure()*, *export\_bppm()*

## Parameters

<i>sequence</i>	The RNA sequence input
<i>structure</i>	A pointer to a char array where a base pair probability information can be stored in a pseudo-dot-bracket notation (may be NULL, too)

## Returns

The Gibbs free energy of the ensemble (  $G = -RT \cdot \log(Q)$  ) in kcal/mol

## 15.53.2.4 pf\_circ\_fold()

```
float pf_circ_fold (
    const char * sequence,
    char * structure )

#include <ViennaRNA/part_func.h>
```

Compute the partition function of a circular RNA sequence.

## Note

The global array [pr](#) is deprecated and the user who wants the calculated base pair probabilities for further computations is advised to use the function [export\\_bppm\(\)](#).

**OpenMP:** This function is not entirely threadsafe. While the recursions are working on their own copies of data the model details for the recursions are determined from the global settings just before entering the recursions. Consider using [pf\\_fold\\_par\(\)](#) for a really threadsafe implementation.

## Precondition

This function takes its model details from the global variables provided in *RNAlib*

## Postcondition

After successful run the hidden folding matrices are filled with the appropriate Boltzmann factors. Depending on whether the global variable [do\\_backtrack](#) was set the base pair probabilities are already computed and may be accessed for further usage via the [export\\_bppm\(\)](#) function. A call of [free\\_pf\\_arrays\(\)](#) will free all memory allocated by this function. Successive calls will first free previously allocated memory before starting the computation.

## See also

[vrna\\_pf\(\)](#)

**Deprecated** Use [vrna\\_pf\(\)](#) instead!



## Parameters

<code>in</code>	<i>sequence</i>	The RNA sequence input
<code>in, out</code>	<i>structure</i>	A pointer to a char array where a base pair probability information can be stored in a pseudo-dot-bracket notation (may be NULL, too)

## Returns

The Gibbs free energy of the ensemble (  $G = -RT \cdot \log(Q)$ ) in kcal/mol

## 15.53.2.5 free\_pf\_arrays()

```
void free_pf_arrays (
    void )
```

```
#include <ViennaRNA/part_func.h>
```

Free arrays for the partition function recursions.

Call this function if you want to free all allocated memory associated with the partition function forward recursion.

## Note

Successive calls of [pf\\_fold\(\)](#), [pf\\_circ\\_fold\(\)](#) already check if they should free any memory from a previous run.

**OpenMP notice:**

This function should be called before leaving a thread in order to avoid leaking memory

**Deprecated** See [vrna\\_fold\\_compound\\_t](#) and its related functions for how to free memory occupied by the dynamic programming matrices

## Postcondition

All memory allocated by [pf\\_fold\\_par\(\)](#), [pf\\_fold\(\)](#) or [pf\\_circ\\_fold\(\)](#) will be free'd

## See also

[pf\\_fold\\_par\(\)](#), [pf\\_fold\(\)](#), [pf\\_circ\\_fold\(\)](#)

## 15.53.2.6 update\_pf\_params()

```
void update_pf_params (
    int length )
```

```
#include <ViennaRNA/part_func.h>
```

Recalculate energy parameters.

Call this function to recalculate the pair matrix and energy parameters after a change in folding parameters like [temperature](#)

**Deprecated** Use [vrna\\_exp\\_params\\_subst\(\)](#) instead

### 15.53.2.7 update\_pf\_params\_par()

```
void update_pf_params_par (
    int length,
    vrna_exp_param_t * parameters )
```

```
#include <ViennaRNA/part_func.h>
```

Recalculate energy parameters.

**Deprecated** Use [vrna\\_exp\\_params\\_subst\(\)](#) instead

### 15.53.2.8 export\_bppm()

```
FLT_OR_DBL* export_bppm (
    void )
```

```
#include <ViennaRNA/part_func.h>
```

Get a pointer to the base pair probability array.

Accessing the base pair probabilities for a pair (i,j) is achieved by

```
FLT_OR_DBL *pr = export_bppm();
pr_ij          = pr[iindx[i]-j];
```

#### Precondition

Call [pf\\_fold\\_par\(\)](#), [pf\\_fold\(\)](#) or [pf\\_circ\\_fold\(\)](#) first to fill the base pair probability array

#### See also

[pf\\_fold\(\)](#), [pf\\_circ\\_fold\(\)](#), [vrna\\_idx\\_row\\_wise\(\)](#)

#### Returns

A pointer to the base pair probability array

### 15.53.2.9 get\_pf\_arrays()

```
int get_pf_arrays (
    short ** S_p,
    short ** S1_p,
    char ** ptype_p,
    FLT_OR_DBL ** qb_p,
    FLT_OR_DBL ** qm_p,
    FLT_OR_DBL ** qlk_p,
    FLT_OR_DBL ** qln_p )
```

```
#include <ViennaRNA/part_func.h>
```

Get the pointers to (almost) all relevant computation arrays used in partition function computation.

#### Precondition

In order to assign meaningful pointers, you have to call [pf\\_fold\\_par\(\)](#) or [pf\\_fold\(\)](#) first!

#### See also

[pf\\_fold\\_par\(\)](#), [pf\\_fold\(\)](#), [pf\\_circ\\_fold\(\)](#)

## Parameters

out	<i>S_p</i>	A pointer to the 'S' array (integer representation of nucleotides)
out	<i>S1_p</i>	A pointer to the 'S1' array (2nd integer representation of nucleotides)
out	<i>p<sub>type</sub>↔<sub>p</sub></i>	A pointer to the pair type matrix
out	<i>qb_p</i>	A pointer to the $Q^B$ matrix
out	<i>qm_p</i>	A pointer to the $Q^M$ matrix
out	<i>q1k_p</i>	A pointer to the 5' slice of the Q matrix ( $q1k(k) = Q(1, k)$ )
out	<i>qln_p</i>	A pointer to the 3' slice of the Q matrix ( $qln(l) = Q(l, n)$ )

## Returns

Non Zero if everything went fine, 0 otherwise

## 15.53.2.10 mean\_bp\_distance()

```
double mean_bp_distance (
    int length )
```

```
#include <ViennaRNA/part_func.h>
```

Get the mean base pair distance of the last partition function computation.

**Deprecated** Use [vrna\\_mean\\_bp\\_distance\(\)](#) or [vrna\\_mean\\_bp\\_distance\\_pr\(\)](#) instead!

## See also

[vrna\\_mean\\_bp\\_distance\(\)](#), [vrna\\_mean\\_bp\\_distance\\_pr\(\)](#)

## Parameters

<i>length</i>	
---------------	--

## Returns

mean base pair distance in thermodynamic ensemble

## 15.53.2.11 mean\_bp\_distance\_pr()

```
double mean_bp_distance_pr (
    int length,
    FLT_OR_DBL * pr )
```

```
#include <ViennaRNA/part_func.h>
```

Get the mean base pair distance in the thermodynamic ensemble.

This is a threadsafe implementation of [mean\\_bp\\_dist\(\)](#) !

$$\langle d \rangle = \sum_{a,b} p_a p_b d(S_a, S_b)$$

this can be computed from the pair probs  $p_{ij}$  as

$$\langle d \rangle = \sum_{i,j} p_{ij} (1 - p_{ij})$$

**Deprecated** Use [vrna\\_mean\\_bp\\_distance\(\)](#) or [vrna\\_mean\\_bp\\_distance\\_pr\(\)](#) instead!

#### Parameters

<i>length</i>	The length of the sequence
<i>pr</i>	The matrix containing the base pair probabilities

#### Returns

The mean pair distance of the structure ensemble

#### 15.53.2.12 stackProb()

```
vrna_ep_t* stackProb (
    double cutoff )
```

```
#include <ViennaRNA/part_func.h>
```

Get the probability of stacks.

**Deprecated** Use [vrna\\_stack\\_prob\(\)](#) instead!

#### 15.53.2.13 init\_pf\_fold()

```
void init_pf_fold (
    int length )
```

```
#include <ViennaRNA/part_func.h>
```

Allocate space for [pf\\_fold\(\)](#)

**Deprecated** This function is obsolete and will be removed soon!

15.53.2.14 `co_pf_fold()`

```
vrna_dimer_pf_t co_pf_fold (
    char * sequence,
    char * structure )

#include <ViennaRNA/part_func_co.h>
```

Calculate partition function and base pair probabilities.

This is the cofold partition function folding. The second molecule starts at the [cut\\_point](#) nucleotide.

**Note**

OpenMP: Since this function relies on the global parameters [do\\_backtrack](#), [dangles](#), [temperature](#) and [pf\\_scale](#) it is not threadsafe according to concurrent changes in these variables! Use [co\\_pf\\_fold\\_par\(\)](#) instead to circumvent this issue.

**Deprecated** {Use [vrna\\_pf\\_dimer\(\)](#) instead!}

**Parameters**

<i>sequence</i>	Concatenated RNA sequences
<i>structure</i>	Will hold the structure or constraints

**Returns**

`vrna_dimer_pf_t` structure containing a set of energies needed for concentration computations.

15.53.2.15 `co_pf_fold_par()`

```
vrna_dimer_pf_t co_pf_fold_par (
    char * sequence,
    char * structure,
    vrna_exp_param_t * parameters,
    int calculate_bppm,
    int is_constrained )

#include <ViennaRNA/part_func_co.h>
```

Calculate partition function and base pair probabilities.

This is the cofold partition function folding. The second molecule starts at the [cut\\_point](#) nucleotide.

**Deprecated** Use [vrna\\_pf\\_dimer\(\)](#) instead!

**See also**

[get\\_boltzmann\\_factors\(\)](#), [co\\_pf\\_fold\(\)](#)

## Parameters

<i>sequence</i>	Concatenated RNA sequences
<i>structure</i>	Pointer to the structure constraint
<i>parameters</i>	Data structure containing the precalculated Boltzmann factors
<i>calculate_bppm</i>	Switch to turn Base pair probability calculations on/off (0==off)
<i>is_constrained</i>	Switch to indicate that a structure constraint is passed via the structure argument (0==off)

## Returns

`vrna_dimer_pf_t` structure containing a set of energies needed for concentration computations.

15.53.2.16 `compute_probabilities()`

```
void compute_probabilities (
    double FAB,
    double FEA,
    double FEB,
    vrna_ep_t * prAB,
    vrna_ep_t * prA,
    vrna_ep_t * prB,
    int Alength )
```

```
#include <ViennaRNA/part_func_co.h>
```

Compute Boltzmann probabilities of dimerization without homodimers.

Given the pair probabilities and free energies (in the null model) for a dimer AB and the two constituent monomers A and B, compute the conditional pair probabilities given that a dimer AB actually forms. Null model pair probabilities are given as a list as produced by [assign\\_plist\\_from\\_pr\(\)](#), the dimer probabilities 'prAB' are modified in place.

**Deprecated** { Use `vrna_pf_dimer_probs()` instead!}

## Parameters

<i>FAB</i>	free energy of dimer AB
<i>FEA</i>	free energy of monomer A
<i>FEB</i>	free energy of monomer B
<i>prAB</i>	pair probabilities for dimer
<i>prA</i>	pair probabilities monomer
<i>prB</i>	pair probabilities monomer
<i>Alength</i>	Length of molecule A

**15.53.2.17** `init_co_pf_fold()`

```
void init_co_pf_fold (
    int length )

#include <ViennaRNA/part_func_co.h>
```

DO NOT USE THIS FUNCTION ANYMORE

**Deprecated** { This function is deprecated and will be removed soon!}

**15.53.2.18** `export_co_bppm()`

```
FLT_OR_DBL* export_co_bppm (
    void )

#include <ViennaRNA/part_func_co.h>
```

Get a pointer to the base pair probability array.

Accessing the base pair probabilities for a pair (i,j) is achieved by

```
FLT_OR_DBL *pr = export_bppm(); pr_ij = pr[iindx[i]-j];
```

**Deprecated** This function is deprecated and will be removed soon! The base pair probability array is available through the `vrna_fold_compound_t` data structure, and its associated `vrna_mx_pf_t` member.

See also

[vrna\\_idx\\_row\\_wise\(\)](#)

Returns

A pointer to the base pair probability array

**15.53.2.19** `free_co_pf_arrays()`

```
void free_co_pf_arrays (
    void )

#include <ViennaRNA/part_func_co.h>
```

Free the memory occupied by `co_pf_fold()`

**Deprecated** This function will be removed for the new API soon! See [vrna\\_pf\\_dimer\(\)](#), [vrna\\_fold\\_compound\(\)](#), and [vrna\\_fold\\_compound\\_free\(\)](#) for an alternative

**15.53.2.20** `update_co_pf_params()`

```
void update_co_pf_params (
    int length )

#include <ViennaRNA/part_func_co.h>
```

Recalculate energy parameters.

This function recalculates all energy parameters given the current model settings.

**Deprecated** Use [vrna\\_exp\\_params\\_subst\(\)](#) instead!

## Parameters

<i>length</i>	Length of the current RNA sequence
---------------	------------------------------------

15.53.2.21 `update_co_pf_params_par()`

```
void update_co_pf_params_par (
    int length,
    vrna_exp_param_t * parameters )

#include <ViennaRNA/part_func_co.h>
```

Recalculate energy parameters.

This function recalculates all energy parameters given the current model settings. It's second argument can either be NULL or a data structure containing the precomputed Boltzmann factors. In the first scenario, the necessary data structure will be created automatically according to the current global model settings, i.e. this mode might not be threadsafe. However, if the provided data structure is not NULL, threadsafety for the model parameters [dangles](#), [pf\\_scale](#) and [temperature](#) is regained, since their values are taken from this data structure during subsequent calculations.

**Deprecated** Use `vrna_exp_params_subst()` instead!

## Parameters

<i>length</i>	Length of the current RNA sequence
<i>parameters</i>	data structure containing the precomputed Boltzmann factors

15.53.2.22 `assign_plist_from_db()`

```
void assign_plist_from_db (
    vrna_ep_t ** pl,
    const char * struc,
    float pr )

#include <ViennaRNA/utils/structures.h>
```

Create a `vrna_ep_t` from a dot-bracket string.

The dot-bracket string is parsed and for each base pair an entry in the plist is created. The probability of each pair in the list is set by a function parameter.

The end of the plist is marked by sequence positions *i* as well as *j* equal to 0. This condition should be used to stop looping over its entries

**Deprecated** Use `vrna_plist()` instead



## Parameters

<i>pl</i>	A pointer to the <a href="#">vrna_ep_t</a> that is to be created
<i>struc</i>	The secondary structure in dot-bracket notation
<i>pr</i>	The probability for each base pair

15.53.2.23 `assign_plist_from_pr()`

```
void assign_plist_from_pr (
    vrna_ep_t ** pl,
    FLT_OR_DBL * probs,
    int length,
    double cutoff )
```

```
#include <ViennaRNA/utils/structures.h>
```

Create a `vrna_ep_t` from a probability matrix.

The probability matrix given is parsed and all pair probabilities above the given threshold are used to create an entry in the plist

The end of the plist is marked by sequence positions *i* as well as *j* equal to 0. This condition should be used to stop looping over its entries

## Note

This function is threadsafe

**Deprecated** Use `vrna_plist_from_probs()` instead!

## Parameters

out	<i>pl</i>	A pointer to the <code>vrna_ep_t</code> that is to be created
in	<i>probs</i>	The probability matrix used for creating the plist
in	<i>length</i>	The length of the RNA sequence
in	<i>cutoff</i>	The cutoff value

15.53.2.24 `alipf_fold()`

```
float alipf_fold (
    const char ** sequences,
    char * structure,
    vrna_ep_t ** pl )
```

```
#include <ViennaRNA/alifold.h>
```

The partition function version of [alifold\(\)](#) works in analogy to [pf\\_fold\(\)](#). Pair probabilities and information about sequence covariations are returned via the 'pi' variable as a list of [vrna\\_pinfo\\_t](#) structs. The list is terminated by the first entry with pi.i = 0.

**Deprecated** Use [vrna\\_pf\(\)](#) instead

#### Parameters

<i>sequences</i>	
<i>structure</i>	
<i>pl</i>	

#### Returns

#### 15.53.2.25 alipf\_circ\_fold()

```
float alipf_circ_fold (
    const char ** sequences,
    char * structure,
    vrna_ep_t ** pl )

#include <ViennaRNA/alifold.h>
```

**Deprecated** Use [vrna\\_pf\(\)](#) instead

#### Parameters

<i>sequences</i>	
<i>structure</i>	
<i>pl</i>	

#### Returns

#### 15.53.2.26 export\_ali\_bppm()

```
FLT_OR_DBL* export_ali_bppm (
    void )

#include <ViennaRNA/alifold.h>
```

Get a pointer to the base pair probability array.

Accessing the base pair probabilities for a pair (i,j) is achieved by

```
FLT_OR_DBL *pr = export_bppm(); pr_ij = pr[iindx[i]-j];
```

**Deprecated** Usage of this function is discouraged! The new [vrna\\_fold\\_compound\\_t](#) allows direct access to the folding matrices, including the pair probabilities! The pair probability array returned here reflects the one of the latest call to [vrna\\_pf\(\)](#), or any of the old API calls for consensus structure partition function folding.

See also

[vrna\\_fold\\_compound\\_t](#), [vrna\\_fold\\_compound\\_comparative\(\)](#), and [vrna\\_pf\(\)](#)

Returns

A pointer to the base pair probability array

#### 15.53.2.27 free\_alipf\_arrays()

```
void free_alipf_arrays (
    void )
```

```
#include <ViennaRNA/alifold.h>
```

Free the memory occupied by folding matrices allocated by `alipf_fold`, `alipf_circ_fold`, etc.

**Deprecated** Usage of this function is discouraged! This function only free's memory allocated by old API function calls. Memory allocated by any of the new API calls (starting with `vrna_`) will be not affected!

See also

[vrna\\_fold\\_compound\\_t](#), [vrna\\_vrna\\_fold\\_compound\\_free\(\)](#)

#### 15.53.2.28 alipbacktrack()

```
char* alipbacktrack (
    double * prob )
```

```
#include <ViennaRNA/alifold.h>
```

Sample a consensus secondary structure from the Boltzmann ensemble according its probability.

**Deprecated** Use [vrna\\_pbacktrack\(\)](#) instead!

## Parameters

<i>prob</i>	to be described (berni)
-------------	-------------------------

## Returns

A sampled consensus secondary structure in dot-bracket notation

15.53.2.29 `get_alipf_arrays()`

```
int get_alipf_arrays (
    short *** S_p,
    short *** S5_p,
    short *** S3_p,
    unsigned short *** a2s_p,
    char *** Ss_p,
    FLT_OR_DBL ** qb_p,
    FLT_OR_DBL ** qm_p,
    FLT_OR_DBL ** q1k_p,
    FLT_OR_DBL ** qln_p,
    short ** pscore )
```

```
#include <ViennaRNA/alifold.h>
```

Get pointers to (almost) all relevant arrays used in alifold's partition function computation.

## Note

To obtain meaningful pointers, call `alipf_fold` first!

## See also

`pf_alifold()`, `alipf_circ_fold()`

**Deprecated** It is discouraged to use this function! The new `vrna_fold_compound_t` allows direct access to all necessary consensus structure prediction related variables!

## See also

`vrna_fold_compound_t`, `vrna_fold_compound_comparative()`, `vrna_pf()`

## Parameters

<i>S_p</i>	A pointer to the 'S' array (integer representation of nucleotides)
<i>S5_p</i>	A pointer to the 'S5' array
<i>S3_p</i>	A pointer to the 'S3' array
<i>a2s↔ _p</i>	A pointer to the alignment-column to sequence position mapping array
<i>Ss_p</i>	A pointer to the 'Ss' array
<i>qb_p</i>	A pointer to the $Q^B$ matrix
<i>qm_p</i>	A pointer to the $Q^M$ matrix
<i>q1k↔ _p</i>	A pointer to the 5' slice of the Q matrix ( $q1k(k) = Q(1, k)$ )

**Returns**

Non Zero if everything went fine, 0 otherwise

## 15.54 Deprecated Interface for Local (Sliding Window) Partition Function Computation

### 15.54.1 Detailed Description

Collaboration diagram for Deprecated Interface for Local (Sliding Window) Partition Function Computation:

#### Files

- file [LPfold.h](#)

*Partition function and equilibrium probability implementation for the sliding window algorithm.*

#### Functions

- void [update\\_pf\\_paramsLP](#) (int length)
- [vrna\\_ep\\_t](#) \* [pfl\\_fold](#) (char \*sequence, int winSize, int pairSize, float cutoffb, double \*\*pU, [vrna\\_ep\\_t](#) \*\*dpp2, FILE \*pUfp, FILE \*spup)  
*Compute partition functions for locally stable secondary structures.*
- [vrna\\_ep\\_t](#) \* [pfl\\_fold\\_par](#) (char \*sequence, int winSize, int pairSize, float cutoffb, double \*\*pU, [vrna\\_ep\\_t](#) \*\*dpp2, FILE \*pUfp, FILE \*spup, [vrna\\_exp\\_param\\_t](#) \*parameters)  
*Compute partition functions for locally stable secondary structures.*
- void [putoutpU\\_prob](#) (double \*\*pU, int length, int ulength, FILE \*fp, int energies)  
*Writes the unpaired probabilities (pU) or opening energies into a file.*
- void [putoutpU\\_prob\\_bin](#) (double \*\*pU, int length, int ulength, FILE \*fp, int energies)  
*Writes the unpaired probabilities (pU) or opening energies into a binary file.*

### 15.54.2 Function Documentation

#### 15.54.2.1 [update\\_pf\\_paramsLP\(\)](#)

```
void update_pf_paramsLP (
    int length )
```

```
#include <ViennaRNA/LPfold.h>
```

##### Parameters

<a href="#">length</a>	
------------------------	--

#### 15.54.2.2 [pfl\\_fold\(\)](#)

```

vrna_ep_t* pfl_fold (
    char * sequence,
    int winSize,
    int pairSize,
    float cutoffb,
    double ** pU,
    vrna_ep_t ** dpp2,
    FILE * pUfp,
    FILE * spup )

#include <ViennaRNA/LPfold.h>

```

Compute partition functions for locally stable secondary structures.

`pfl_fold` computes partition functions for every window of size 'winSize' possible in a RNA molecule, allowing only pairs with a span smaller than 'pairSize'. It returns the mean pair probabilities averaged over all windows containing the pair in 'pl'. 'winSize' should always be  $\geq$  'pairSize'. Note that in contrast to `Lfold()`, bases outside of the window do not influence the structure at all. Only probabilities higher than 'cutoffb' are kept.

If 'pU' is supplied (i.e. is not the NULL pointer), `pfl_fold()` will also compute the mean probability that regions of length 'u' and smaller are unpaired. The parameter 'u' is supplied in 'pup[0][0]'. On return the 'pup' array will contain these probabilities, with the entry on 'pup[x][y]' containing the mean probability that x and the y-1 preceding bases are unpaired. The 'pU' array needs to be large enough to hold  $n+1$  float\* entries, where n is the sequence length.

If an array dpp2 is supplied, the probability of base pair (i,j) given that there already exists a base pair (i+1,j-1) is also computed and saved in this array. If pUfp is given (i.e. not NULL), pU is not saved but put out immediately. If spup is given (i.e. is not NULL), the pair probabilities in pl are not saved but put out immediately.

#### Parameters

<i>sequence</i>	RNA sequence
<i>winSize</i>	size of the window
<i>pairSize</i>	maximum size of base pair
<i>cutoffb</i>	cutoffb for base pairs
<i>pU</i>	array holding all unpaired probabilities
<i>dpp2</i>	array of dependent pair probabilities
<i>pUfp</i>	file pointer for pU
<i>spup</i>	file pointer for pair probabilities

#### Returns

list of pair probabilities

#### 15.54.2.3 putoutpU\_prob()

```

void putoutpU_prob (
    double ** pU,
    int length,
    int ulength,
    FILE * fp,
    int energies )

```

```
#include <ViennaRNA/LPfold.h>
```

Writes the unpaired probabilities (pU) or opening energies into a file.

Can write either the unpaired probabilities (accessibilities) pU or the opening energies  $-\log(pU)kT$  into a file

#### Parameters

<i>pU</i>	pair probabilities
<i>length</i>	length of RNA sequence
<i>ulength</i>	maximum length of unpaired stretch
<i>fp</i>	file pointer of destination file
<i>energies</i>	switch to put out as opening energies

#### 15.54.2.4 putoutpU\_prob\_bin()

```
void putoutpU_prob_bin (
    double ** pU,
    int length,
    int ulength,
    FILE * fp,
    int energies )
```

```
#include <ViennaRNA/LPfold.h>
```

Writes the unpaired probabilities (pU) or opening energies into a binary file.

Can write either the unpaired probabilities (accessibilities) pU or the opening energies  $-\log(pU)kT$  into a file

#### Parameters

<i>pU</i>	pair probabilities
<i>length</i>	length of RNA sequence
<i>ulength</i>	maximum length of unpaired stretch
<i>fp</i>	file pointer of destination file
<i>energies</i>	switch to put out as opening energies



## 15.55 Partition Function for Two Hybridized Sequences

Partition Function Cofolding.

### 15.55.1 Detailed Description

Partition Function Cofolding.

To simplify the implementation the partition function computation is done internally in a null model that does not include the duplex initiation energy, i.e. the entropic penalty for producing a dimer from two monomers). The resulting free energies and pair probabilities are initially relative to that null model. In a second step the free energies can be corrected to include the dimerization penalty, and the pair probabilities can be divided into the conditional pair probabilities given that a re dimer is formed or not formed. See [2] for further details.

As for folding one RNA molecule, this computes the partition function of all possible structures and the base pair probabilities. Uses the same global `pf_scale` variable to avoid overflows.

After computing the partition functions of all possible dimers one can compute the probabilities of base pairs, the concentrations out of start concentrations and sofar and soaway.

Dimer formation is inherently concentration dependent. Given the free energies of the monomers A and B and dimers AB, AA, and BB one can compute the equilibrium concentrations, given input concentrations of A and B, see e.g. Dimitrov & Zuker (2004) Collaboration diagram for Partition Function for Two Hybridized Sequences:

### Files

- file `concentrations.h`  
*Concentration computations for RNA-RNA interactions.*
- file `part_func_up.h`  
*Implementations for accessibility and RNA-RNA interaction as a stepwise process.*

### Typedefs

- typedef struct `vrna_dimer_pf_s` `vrna_dimer_pf_t`  
*Typename for the data structure that stores the dimer partition functions, `vrna_dimer_pf_s`, as returned by `vrna_pf_dimer()`*
- typedef struct `vrna_dimer_pf_s` `cofoldF`  
*Backward compatibility typedef for `vrna_dimer_pf_s`.*

### Variables

- int `mirnatog`  
*Toggles no intrabp in 2nd mol.*
- double `F_monomer` [2]  
*Free energies of the two monomers.*
- typedef struct `vrna_dimer_conc_s` `vrna_dimer_conc_t`  
*Typename for the data structure that stores the dimer concentrations, `vrna_dimer_conc_s`, as required by `vrna_pf←_dimer_concentration()`*
- typedef struct `vrna_dimer_conc_s` `ConcEnt`  
*Backward compatibility typedef for `vrna_dimer_conc_s`.*
- `vrna_dimer_conc_t` \* `vrna_pf_dimer_concentrations` (double FcAB, double FcAA, double FcBB, double FEA, double FEB, const double \*startconc, const `vrna_exp_param_t` \*exp\_params)  
*Given two start monomer concentrations a and b, compute the concentrations in thermodynamic equilibrium of all dimers and the monomers.*

## Simplified global partition function computation using sequence(s) or multiple sequence alignment(s)

- [vrna\\_dimer\\_pf\\_t vrna\\_pf\\_co\\_fold](#) (const char \*seq, char \*structure, [vrna\\_ep\\_t](#) \*\*pl)

*Calculate partition function and base pair probabilities of nucleic acid/nucleic acid dimers.*

### 15.55.2 Function Documentation

#### 15.55.2.1 vrna\_pf\_co\_fold()

```
vrna_dimer_pf_t vrna_pf_co_fold (
    const char * seq,
    char * structure,
    vrna_ep_t ** pl )

#include <ViennaRNA/part_func.h>
```

Calculate partition function and base pair probabilities of nucleic acid/nucleic acid dimers.

This simplified interface to [vrna\\_pf\\_dimer\(\)](#) computes the partition function and, if required, base pair probabilities for an RNA-RNA interaction using default options. Memory required for dynamic programming (DP) matrices will be allocated and free'd on-the-fly. Hence, after return of this function, the recursively filled matrices are not available any more for any post-processing.

#### Note

In case you want to use the filled DP matrices for any subsequent post-processing step, or you require other conditions than specified by the default model details, use [vrna\\_pf\\_dimer\(\)](#), and the data structure [vrna\\_fold\\_compound\\_t](#) instead.

#### See also

[vrna\\_pf\\_dimer\(\)](#)

#### Parameters

<i>seq</i>	Two concatenated RNA sequences with a delimiting '&' in between
<i>structure</i>	A pointer to the character array where position-wise pairing propensity will be stored. (Maybe NULL)
<i>pl</i>	A pointer to a list of <a href="#">vrna_ep_t</a> to store pairing probabilities (Maybe NULL)

#### Returns

[vrna\\_dimer\\_pf\\_t](#) structure containing a set of energies needed for concentration computations.

15.55.2.2 `vrna_pf_dimer_concentrations()`

```
vrna_dimer_conc_t* vrna_pf_dimer_concentrations (
    double FcAB,
    double FcAA,
    double FcBB,
    double FEA,
    double FEB,
    const double * startconc,
    const vrna_exp_param_t * exp_params )

#include <ViennaRNA/concentrations.h>
```

Given two start monomer concentrations a and b, compute the concentrations in thermodynamic equilibrium of all dimers and the monomers.

This function takes an array 'startconc' of input concentrations with alternating entries for the initial concentrations of molecules A and B (terminated by two zeroes), then computes the resulting equilibrium concentrations from the free energies for the dimers. Dimer free energies should be the dimer-only free energies, i.e. the FcAB entries from the `vrna_dimer_pf_t` struct.

## Parameters

<i>FcAB</i>	Free energy of AB dimer (FcAB entry)
<i>FcAA</i>	Free energy of AA dimer (FcAB entry)
<i>FcBB</i>	Free energy of BB dimer (FcAB entry)
<i>FEA</i>	Free energy of monomer A
<i>FEB</i>	Free energy of monomer B
<i>startconc</i>	List of start concentrations [a0],[b0],[a1],[b1],...,[an],[bn],[0],[0]
<i>exp_params</i>	The precomputed Boltzmann factors

## Returns

`vrna_dimer_conc_t` array containing the equilibrium energies and start concentrations

## 15.56 Partition Function for two Hybridized Sequences as a Stepwise Process

RNA-RNA interaction as a stepwise process.

### 15.56.1 Detailed Description

RNA-RNA interaction as a stepwise process.

In this approach to cofolding the interaction between two RNA molecules is seen as a stepwise process. In a first step, the target molecule has to adopt a structure in which a binding site is accessible. In a second step, the ligand molecule will hybridize with a region accessible to an interaction. Consequently the algorithm is designed as a two step process: The first step is the calculation of the probability that a region within the target is unpaired, or equivalently, the calculation of the free energy needed to expose a region. In the second step we compute the free energy of an interaction for every possible binding site. Collaboration diagram for Partition Function for two Hybridized Sequences as a Stepwise Process:

### Functions

- `pu_contrib * pf_unstru` (char \*sequence, int max\_w)  
*Calculate the partition function over all unpaired regions of a maximal length.*
- `interact * pf_interact` (const char \*s1, const char \*s2, `pu_contrib *p_c`, `pu_contrib *p_c2`, int max\_w, char \*cstruc, int incr3, int incr5)  
*Calculates the probability of a local interaction between two sequences.*
- `void free_interact` (`interact *pin`)  
*Frees the output of function `pf_interact()`.*
- `void free_pu_contrib_struct` (`pu_contrib *pu`)  
*Frees the output of function `pf_unstru()`.*

### 15.56.2 Function Documentation

#### 15.56.2.1 pf\_unstru()

```
pu_contrib* pf_unstru (
    char * sequence,
    int max_w )

#include <ViennaRNA/part_func_up.h>
```

Calculate the partition function over all unpaired regions of a maximal length.

You have to call function `pf_fold()` providing the same sequence before calling `pf_unstru()`. If you want to calculate unpaired regions for a constrained structure, set variable 'structure' in function '`pf_fold()`' to the constrain string. It returns a `pu_contrib` struct containing four arrays of dimension  $[i = 1 \text{ to } \text{length}(\text{sequence})][j = 0 \text{ to } u-1]$  containing all possible contributions to the probabilities of unpaired regions of maximum length  $u$ . Each array in `pu_contrib` contains one of the contributions to the total probability of being unpaired: The probability of being unpaired within an exterior loop is in array `pu_contrib->E`, the probability of being unpaired within a hairpin loop is in array `pu_contrib->H`, the probability of being unpaired within an interior loop is in array `pu_contrib->I` and probability of being unpaired within a multi-loop is in array `pu_contrib->M`. The total probability of being unpaired is the sum of the four arrays of `pu_contrib`.

This function frees everything allocated automatically. To free the output structure call `free_pu_contrib()`.

## Parameters

<i>sequence</i>	
<i>max_w</i>	

## Returns

## 15.56.2.2 pf\_interact()

```

interact* pf_interact (
    const char * s1,
    const char * s2,
    pu_contrib * p_c,
    pu_contrib * p_c2,
    int max_w,
    char * cstruc,
    int incr3,
    int incr5 )

```

```
#include <ViennaRNA/part_func_up.h>
```

Calculates the probability of a local interaction between two sequences.

The function considers the probability that the region of interaction is unpaired within 's1' and 's2'. The longer sequence has to be given as 's1'. The shorter sequence has to be given as 's2'. Function [pf\\_unstru\(\)](#) has to be called for 's1' and 's2', where the probabilities of being unpaired have to be given in 'p\_c' and 'p\_c2', respectively. If you do not want to include the probabilities of being unpaired for 's2' set 'p\_c2' to NULL. If variable 'cstruc' is not NULL, constrained folding is done: The available constrains for intermolecular interaction are: '.' (no constrain), 'x' (the base has no intermolecular interaction) and '|' (the corresponding base has to be paired intermolecularly).

The parameter 'w' determines the maximal length of the interaction. The parameters 'incr5' and 'incr3' allows inclusion of unpaired residues left ('incr5') and right ('incr3') of the region of interaction in 's1'. If the 'incr' options are used, function [pf\\_unstru\(\)](#) has to be called with  $w=w+incr5+incr3$  for the longer sequence 's1'.

It returns a structure of type [interact](#) which contains the probability of the best local interaction including residue  $i$  in  $P_i$  and the minimum free energy in  $G_i$ , where  $i$  is the position in sequence 's1'. The member  $G_{ikjl}$  of structure [interact](#) is the best interaction between region  $[k,i]$   $k < i$  in longer sequence 's1' and region  $[j,l]$   $j < l$  in 's2'.  $G_{ikjl\_wo}$  is  $G_{ikjl}$  without the probability of being unpaired.

Use [free\\_interact\(\)](#) to free the returned structure, all other stuff is freed inside [pf\\_interact\(\)](#).

## Parameters

<i>s1</i>	
<i>s2</i>	
<i>p_c</i>	
<i>p_c2</i>	
<i>max<sub>w</sub></i>	
<i>cstruc</i>	
<i>incr3</i>	
<i>incr5</i>	

Returns

## 15.57 Reading/Writing Energy Parameter Sets from/to File

Read and Write energy parameter sets from and to text files.

### 15.57.1 Detailed Description

Read and Write energy parameter sets from and to text files.

A default set of parameters, identical to the one described in [17] and [22], is compiled into the library. Collaboration diagram for Reading/Writing Energy Parameter Sets from/to File:

### Modules

- [Converting Energy Parameter Files](#)  
*Convert energy parameter files into the latest format.*

### Functions

- `const char * last\_parameter\_file (void)`  
*Get the file name of the parameter file that was most recently loaded.*
- `void read\_parameter\_file (const char fname[ ])`  
*Read energy parameters from a file.*
- `void write\_parameter\_file (const char fname[ ])`  
*Write energy parameters to a file.*

### 15.57.2 Function Documentation

#### 15.57.2.1 `last_parameter_file()`

```
const char* last_parameter_file (  
    void )  
  
#include <ViennaRNA/params/io.h>
```

Get the file name of the parameter file that was most recently loaded.

#### Returns

The file name of the last parameter file, or NULL if parameters are still at defaults

#### 15.57.2.2 `read_parameter_file()`

```
void read_parameter_file (  
    const char fname[ ] )  
  
#include <ViennaRNA/params/io.h>
```

Read energy parameters from a file.

**Parameters**

<i>fname</i>	The path to the file containing the energy parameters
--------------	---

**15.57.2.3 write\_parameter\_file()**

```
void write_parameter_file (  
    const char fname[] )
```

```
#include <ViennaRNA/params/io.h>
```

Write energy parameters to a file.

**Parameters**

<i>fname</i>	A filename (path) for the file where the current energy parameters will be written to
--------------	---



## 15.58 Converting Energy Parameter Files

Convert energy parameter files into the latest format.

### 15.58.1 Detailed Description

Convert energy parameter files into the latest format.

To preserve some backward compatibility the RNAlib also provides functions to convert energy parameter files from the format used in version 1.4-1.8 into the new format used since version 2.0

Collaboration diagram for Converting Energy Parameter Files:

#### Files

- file [1.8.4\\_epars.h](#)  
*Free energy parameters for parameter file conversion.*
- file [1.8.4\\_intloops.h](#)  
*Free energy parameters for interior loop contributions needed by the parameter file conversion functions.*

#### Macros

- `#define VRNA_CONVERT_OUTPUT_ALL 1U`
- `#define VRNA_CONVERT_OUTPUT_HP 2U`
- `#define VRNA_CONVERT_OUTPUT_STACK 4U`
- `#define VRNA_CONVERT_OUTPUT_MM_HP 8U`
- `#define VRNA_CONVERT_OUTPUT_MM_INT 16U`
- `#define VRNA_CONVERT_OUTPUT_MM_INT_1N 32U`
- `#define VRNA_CONVERT_OUTPUT_MM_INT_23 64U`
- `#define VRNA_CONVERT_OUTPUT_MM_MULTI 128U`
- `#define VRNA_CONVERT_OUTPUT_MM_EXT 256U`
- `#define VRNA_CONVERT_OUTPUT_DANGLE5 512U`
- `#define VRNA_CONVERT_OUTPUT_DANGLE3 1024U`
- `#define VRNA_CONVERT_OUTPUT_INT_11 2048U`
- `#define VRNA_CONVERT_OUTPUT_INT_21 4096U`
- `#define VRNA_CONVERT_OUTPUT_INT_22 8192U`
- `#define VRNA_CONVERT_OUTPUT_BULGE 16384U`
- `#define VRNA_CONVERT_OUTPUT_INT 32768U`
- `#define VRNA_CONVERT_OUTPUT_ML 65536U`
- `#define VRNA_CONVERT_OUTPUT_MISC 131072U`
- `#define VRNA_CONVERT_OUTPUT_SPECIAL_HP 262144U`
- `#define VRNA_CONVERT_OUTPUT_VANILLA 524288U`
- `#define VRNA_CONVERT_OUTPUT_NINIO 1048576U`
- `#define VRNA_CONVERT_OUTPUT_DUMP 2097152U`

#### Functions

- void [convert\\_parameter\\_file](#) (const char \*iname, const char \*oname, unsigned int options)

## 15.58.2 Macro Definition Documentation

### 15.58.2.1 VRNA\_CONVERT\_OUTPUT\_ALL

```
#define VRNA_CONVERT_OUTPUT_ALL 1U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of a complete parameter set

### 15.58.2.2 VRNA\_CONVERT\_OUTPUT\_HP

```
#define VRNA_CONVERT_OUTPUT_HP 2U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of hairpin contributions

### 15.58.2.3 VRNA\_CONVERT\_OUTPUT\_STACK

```
#define VRNA_CONVERT_OUTPUT_STACK 4U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of base pair stack contributions

### 15.58.2.4 VRNA\_CONVERT\_OUTPUT\_MM\_HP

```
#define VRNA_CONVERT_OUTPUT_MM_HP 8U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of hairpin mismatch contribution

### 15.58.2.5 VRNA\_CONVERT\_OUTPUT\_MM\_INT

```
#define VRNA_CONVERT_OUTPUT_MM_INT 16U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of interior loop mismatch contribution

**15.58.2.6 VRNA\_CONVERT\_OUTPUT\_MM\_INT\_1N**

```
#define VRNA_CONVERT_OUTPUT_MM_INT_1N 32U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of 1:n interior loop mismatch contribution

**15.58.2.7 VRNA\_CONVERT\_OUTPUT\_MM\_INT\_23**

```
#define VRNA_CONVERT_OUTPUT_MM_INT_23 64U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of 2:3 interior loop mismatch contribution

**15.58.2.8 VRNA\_CONVERT\_OUTPUT\_MM\_MULTI**

```
#define VRNA_CONVERT_OUTPUT_MM_MULTI 128U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of multi loop mismatch contribution

**15.58.2.9 VRNA\_CONVERT\_OUTPUT\_MM\_EXT**

```
#define VRNA_CONVERT_OUTPUT_MM_EXT 256U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of exterior loop mismatch contribution

**15.58.2.10 VRNA\_CONVERT\_OUTPUT\_DANGLE5**

```
#define VRNA_CONVERT_OUTPUT_DANGLE5 512U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of 5' dangle contribution

**15.58.2.11 VRNA\_CONVERT\_OUTPUT\_DANGLE3**

```
#define VRNA_CONVERT_OUTPUT_DANGLE3 1024U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of 3' dangle contribution

**15.58.2.12 VRNA\_CONVERT\_OUTPUT\_INT\_11**

```
#define VRNA_CONVERT_OUTPUT_INT_11 2048U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of 1:1 interior loop contribution

**15.58.2.13 VRNA\_CONVERT\_OUTPUT\_INT\_21**

```
#define VRNA_CONVERT_OUTPUT_INT_21 4096U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of 2:1 interior loop contribution

**15.58.2.14 VRNA\_CONVERT\_OUTPUT\_INT\_22**

```
#define VRNA_CONVERT_OUTPUT_INT_22 8192U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of 2:2 interior loop contribution

**15.58.2.15 VRNA\_CONVERT\_OUTPUT\_BULGE**

```
#define VRNA_CONVERT_OUTPUT_BULGE 16384U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of bulge loop contribution

**15.58.2.16 VRNA\_CONVERT\_OUTPUT\_INT**

```
#define VRNA_CONVERT_OUTPUT_INT 32768U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of interior loop contribution

**15.58.2.17 VRNA\_CONVERT\_OUTPUT\_ML**

```
#define VRNA_CONVERT_OUTPUT_ML 65536U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of multi loop contribution

**15.58.2.18 VRNA\_CONVERT\_OUTPUT\_MISC**

```
#define VRNA_CONVERT_OUTPUT_MISC 131072U
```

```
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of misc contributions (such as terminalAU)

**15.58.2.19 VRNA\_CONVERT\_OUTPUT\_SPECIAL\_HP**

```
#define VRNA_CONVERT_OUTPUT_SPECIAL_HP 262144U
```

```
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of special hairpin contributions (tri-, tetra-, hexa-loops)

**15.58.2.20 VRNA\_CONVERT\_OUTPUT\_VANILLA**

```
#define VRNA_CONVERT_OUTPUT_VANILLA 524288U
```

```
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of given parameters only

**Note**

This option overrides all other output options, except [VRNA\\_CONVERT\\_OUTPUT\\_DUMP](#) !

**15.58.2.21 VRNA\_CONVERT\_OUTPUT\_NINIO**

```
#define VRNA_CONVERT_OUTPUT_NINIO 1048576U
```

```
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of interior loop asymmetry contribution

**15.58.2.22 VRNA\_CONVERT\_OUTPUT\_DUMP**

```
#define VRNA_CONVERT_OUTPUT_DUMP 2097152U
```

```
#include <ViennaRNA/params/convert.h>
```

Flag to indicate dumping the energy contributions from the library instead of an input file

### 15.58.3 Function Documentation

#### 15.58.3.1 `convert_parameter_file()`

```
void convert_parameter_file (
    const char * iname,
    const char * oname,
    unsigned int options )
```

```
#include <ViennaRNA/params/convert.h>
```

Convert/dump a Vienna 1.8.4 formatted energy parameter file

The options argument allows one to control the different output modes.

Currently available options are:

[VRNA\\_CONVERT\\_OUTPUT\\_ALL](#), [VRNA\\_CONVERT\\_OUTPUT\\_HP](#), [VRNA\\_CONVERT\\_OUTPUT\\_STACK](#)  
[VRNA\\_CONVERT\\_OUTPUT\\_MM\\_HP](#), [VRNA\\_CONVERT\\_OUTPUT\\_MM\\_INT](#), [VRNA\\_CONVERT\\_OUTPUT\\_MM\\_INT\\_1N](#)  
[VRNA\\_CONVERT\\_OUTPUT\\_MM\\_INT\\_23](#), [VRNA\\_CONVERT\\_OUTPUT\\_MM\\_MULTI](#), [VRNA\\_CONVERT\\_OUTPUT\\_MM\\_EXT](#)  
[VRNA\\_CONVERT\\_OUTPUT\\_DANGLE5](#), [VRNA\\_CONVERT\\_OUTPUT\\_DANGLE3](#), [VRNA\\_CONVERT\\_OUTPUT\\_INT\\_11](#)  
[VRNA\\_CONVERT\\_OUTPUT\\_INT\\_21](#), [VRNA\\_CONVERT\\_OUTPUT\\_INT\\_22](#), [VRNA\\_CONVERT\\_OUTPUT\\_BULGE](#)  
[VRNA\\_CONVERT\\_OUTPUT\\_INT](#), [VRNA\\_CONVERT\\_OUTPUT\\_ML](#), [VRNA\\_CONVERT\\_OUTPUT\\_MISC](#)  
[VRNA\\_CONVERT\\_OUTPUT\\_SPECIAL\\_HP](#), [VRNA\\_CONVERT\\_OUTPUT\\_VANILLA](#), [VRNA\\_CONVERT\\_OUTPUT\\_NINIO](#)  
[VRNA\\_CONVERT\\_OUTPUT\\_DUMP](#)

The defined options are fine for bitwise compare- and assignment-operations, e. g.: pass a collection of options as a single value like this:

```
convert_parameter_file(ifile, ofile, option_1 | option_2 | option_n)
```

#### Parameters

<i>iname</i>	The input file name (If NULL input is read from stdin)
<i>oname</i>	The output file name (If NULL output is written to stdout)
<i>options</i>	The options (as described above)

## 15.59 Direct Refolding Paths between two Secondary Structures

Heuristics to explore direct, optimal (re-)folding paths between two secondary structures.

### 15.59.1 Detailed Description

Heuristics to explore direct, optimal (re-)folding paths between two secondary structures.

Collaboration diagram for Direct Refolding Paths between two Secondary Structures:

### Data Structures

- struct [vrna\\_path\\_s](#)  
An element of a refolding path list. [More...](#)

### Typedefs

- typedef struct [vrna\\_path\\_s](#) [vrna\\_path\\_t](#)  
Typename for the refolding path data structure [vrna\\_path\\_s](#).
- typedef struct [vrna\\_path\\_s](#) [path\\_t](#)  
Old typename of [vrna\\_path\\_s](#).

### Functions

- int [vrna\\_path\\_findpath\\_saddle](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*s1, const char \*s2, int width)  
Find energy of a saddle point between 2 structures (search only direct path)
- int [vrna\\_path\\_findpath\\_saddle\\_ub](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*s1, const char \*s2, int width, int maxE)  
Find energy of a saddle point between 2 structures (search only direct path)
- [vrna\\_path\\_t](#) \* [vrna\\_path\\_findpath](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*s1, const char \*s2, int width)  
Find refolding path between 2 structures (search only direct path)
- [vrna\\_path\\_t](#) \* [vrna\\_path\\_findpath\\_ub](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*s1, const char \*s2, int width, int maxE)  
Find refolding path between 2 structures (search only direct path)
- int [find\\_saddle](#) (const char \*seq, const char \*s1, const char \*s2, int width)  
Find energy of a saddle point between 2 structures (search only direct path)
- void [free\\_path](#) ([vrna\\_path\\_t](#) \*path)  
Free memory allocated by [get\\_path\(\)](#) function.
- [vrna\\_path\\_t](#) \* [get\\_path](#) (const char \*seq, const char \*s1, const char \*s2, int width)  
Find refolding path between 2 structures (search only direct path)

## 15.59.2 Data Structure Documentation

### 15.59.2.1 struct vrna\_path\_s

An element of a refolding path list.

See also

[vrna\\_path\\_findpath\(\)](#)

#### Data Fields

- double [en](#)  
*Free energy of current structure.*
- char \* [s](#)  
*Secondary structure in dot-bracket notation.*

## 15.59.3 Typedef Documentation

### 15.59.3.1 path\_t

```
typedef struct vrna_path_s path_t
#include <ViennaRNA/findpath.h>
```

Old typename of [vrna\\_path\\_s](#).

**Deprecated** Use [vrna\\_path\\_t](#) instead!

## 15.59.4 Function Documentation

### 15.59.4.1 vrna\_path\_findpath\_saddle()

```
vrna_path_findpath_saddle (
    vrna_fold_compound_t * vc,
    const char * s1,
    const char * s2,
    int width )
#include <ViennaRNA/findpath.h>
```

Find energy of a saddle point between 2 structures (search only direct path)

This function uses an implementation of the *findpath* algorithm [7] for near-optimal direct refolding path prediction.

Model details, and energy parameters are used as provided via the parameter 'vc'. The [vrna\\_fold\\_compound\\_t](#) does not require memory for any DP matrices, but requires all most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_DEFAULT);
```

See also

[vrna\\_path\\_findpath\\_saddle\\_ub\(\)](#), [vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_t](#), [vrna\\_path\\_findpath\(\)](#)



## Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> with precomputed sequence encoding and model details
<i>s1</i>	The start structure in dot-bracket notation
<i>s2</i>	The target structure in dot-bracket notation
<i>width</i>	A number specifying how many strutures are being kept at each step during the search

## Returns

The saddle energy in 10cal/mol

**SWIG Wrapper Notes** This function is attached as an overloaded method *path\_findpath\_saddle()* to objects of type *fold\_compound*. The optional parameter *width* defaults to 1 if it is omitted.

## 15.59.4.2 vrna\_path\_findpath\_saddle\_ub()

```
vrna_path_findpath_saddle_ub (
    vrna_fold_compound_t * vc,
    const char * s1,
    const char * s2,
    int width,
    int maxE )
```

```
#include <ViennaRNA/findpath.h>
```

Find energy of a saddle point between 2 structures (search only direct path)

This function uses an implementation of the *findpath* algorithm [7] for near-optimal direct refolding path prediction.

Model details, and energy parameters are used as provided via the parameter 'vc'. The [vrna\\_fold\\_compound\\_t](#) does not require memory for any DP matrices, but requires all most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_DEFAULT);
```

## Warning

The argument *maxE* ( $E_{max}$ ) enables one to specify an upper bound, or maximum free energy for the saddle point between the two input structures. If no path with  $E_{saddle} < E_{max}$  is found, the function simply returns *maxE*

## See also

[vrna\\_path\\_findpath\\_saddle\(\)](#), [vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_t](#), [vrna\\_path\\_findpath\(\)](#)

## Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> with precomputed sequence encoding and model details
<i>s1</i>	The start structure in dot-bracket notation
<i>s2</i>	The target structure in dot-bracket notation
<i>width</i>	A number specifying how many strutures are being kept at each step during the search
<i>maxE</i>	An upper bound for the saddle point energy in 10cal/mol

**Returns**

The saddle energy in 10cal/mol

**SWIG Wrapper Notes** This function is attached as an overloaded method *path\_findpath\_saddle()* to objects of type *fold\_compound*. The optional parameter *width* defaults to 1 if it is omitted, while the optional parameter *maxE* defaults to *INF*. In case the function did not find a path with  $E_{\text{saddle}} < E_{\text{max}}$  the function returns a *NULL* object, i.e. *undef* for Perl and *None* for Python.

**15.59.4.3 vrna\_path\_findpath()**

```
vrna_path_findpath (
    vrna_fold_compound_t * vc,
    const char * s1,
    const char * s2,
    int width )
```

```
#include <ViennaRNA/findpath.h>
```

Find refolding path between 2 structures (search only direct path)

This function uses an implementation of the *findpath* algorithm [7] for near-optimal direct refolding path prediction.

Model details, and energy parameters are used as provided via the parameter 'vc'. The *vrna\_fold\_compound\_t* does not require memory for any DP matrices, but requires all most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_DEFAULT);
```

**See also**

[vrna\\_path\\_findpath\\_ub\(\)](#), [vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_t](#), [vrna\\_path\\_findpath\\_saddle\(\)](#)

**Parameters**

<i>vc</i>	The <i>vrna_fold_compound_t</i> with precomputed sequence encoding and model details
<i>s1</i>	The start structure in dot-bracket notation
<i>s2</i>	The target structure in dot-bracket notation
<i>width</i>	A number specifying how many structures are being kept at each step during the search

**Returns**

The saddle energy in 10cal/mol

**SWIG Wrapper Notes** This function is attached as an overloaded method *path\_findpath()* to objects of type *fold\_compound*. The optional parameter *width* defaults to 1 if it is omitted.

15.59.4.4 `vrna_path_findpath_ub()`

```
vrna_path_findpath_ub (
    vrna_fold_compound_t * vc,
    const char * s1,
    const char * s2,
    int width,
    int maxE )
```

```
#include <ViennaRNA/findpath.h>
```

Find refolding path between 2 structures (search only direct path)

This function uses an implementation of the *findpath* algorithm [7] for near-optimal direct refolding path prediction.

Model details, and energy parameters are used as provided via the parameter 'vc'. The `vrna_fold_compound_t` does not require memory for any DP matrices, but requires all most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_DEFAULT);
```

**Warning**

The argument `maxE` enables one to specify an upper bound, or maximum free energy for the saddle point between the two input structures. If no path with  $E_{saddle} < E_{max}$  is found, the function simply returns *NULL*

**See also**

[vrna\\_path\\_findpath\(\)](#), [vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_t](#), [vrna\\_path\\_findpath\\_saddle\(\)](#)

**Parameters**

<code>vc</code>	The <code>vrna_fold_compound_t</code> with precomputed sequence encoding and model details
<code>s1</code>	The start structure in dot-bracket notation
<code>s2</code>	The target structure in dot-bracket notation
<code>width</code>	A number specifying how many strutures are being kept at each step during the search
<code>maxE</code>	An upper bound for the saddle point energy in 10cal/mol

**Returns**

The saddle energy in 10cal/mol

**SWIG Wrapper Notes** This function is attached as an overloaded method `path_findpath()` to objects of type `fold_compound`. The optional parameter `width` defaults to 1 if it is omitted, while the optional parameter `maxE` defaults to *INF*. In case the function did not find a path with  $E_{saddle} < E_{max}$  the function returns an empty list.

#### 15.59.4.5 find\_saddle()

```
int find_saddle (
    const char * seq,
    const char * s1,
    const char * s2,
    int width )
```

```
#include <ViennaRNA/findpath.h>
```

Find energy of a saddle point between 2 structures (search only direct path)

##### Parameters

<i>seq</i>	RNA sequence
<i>s1</i>	A pointer to the character array where the first secondary structure in dot-bracket notation will be written to
<i>s2</i>	A pointer to the character array where the second secondary structure in dot-bracket notation will be written to
<i>width</i>	integer how many strutures are being kept during the search

##### Returns

the saddle energy in 10cal/mol

#### 15.59.4.6 free\_path()

```
void free_path (
    vrna_path_t * path )
```

```
#include <ViennaRNA/findpath.h>
```

Free memory allocated by [get\\_path\(\)](#) function.

##### Parameters

<i>path</i>	pointer to memory to be freed
-------------	-------------------------------

#### 15.59.4.7 get\_path()

```
vrna_path_t* get_path (
    const char * seq,
    const char * s1,
    const char * s2,
    int width )
```

```
#include <ViennaRNA/findpath.h>
```

Find refolding path between 2 structures (search only direct path)

**Parameters**

<i>seq</i>	RNA sequence
<i>s1</i>	A pointer to the character array where the first secondary structure in dot-bracket notation will be written to
<i>s2</i>	A pointer to the character array where the second secondary structure in dot-bracket notation will be written to
<i>width</i>	integer how many strutures are being kept during the search

**Returns**

direct refolding path between two structures

## 15.60 Utilities to deal with Nucleotide Alphabets

Functions to cope with various aspects related to the nucleotide sequence alphabet.

### 15.60.1 Detailed Description

Functions to cope with various aspects related to the nucleotide sequence alphabet.

Collaboration diagram for Utilities to deal with Nucleotide Alphabets:

#### Files

- file [alphabet.h](#)  
*Functions to process, convert, and generally handle different nucleotide and/or base pair alphabets.*
- file [sequence.h](#)  
*Functions and data structures related to sequence representations ,.*

#### Data Structures

- struct [vrna\\_sequence\\_s](#)  
*Data structure representing a nucleotide sequence. [More...](#)*

#### Typedefs

- typedef struct [vrna\\_sequence\\_s](#) [vrna\\_seq\\_t](#)  
*Typename for nucleotide sequence representation data structure [vrna\\_sequence\\_s](#).*

#### Enumerations

- enum [vrna\\_seq\\_type\\_e](#) { [VRNA\\_SEQ\\_UNKNOWN](#), [VRNA\\_SEQ\\_RNA](#), [VRNA\\_SEQ\\_DNA](#) }  
*A enumerator used in [vrna\\_sequence\\_s](#) to distinguish different nucleotide sequences.*

#### Functions

- char \* [vrna\\_ptypes](#) (const short \*S, [vrna\\_md\\_t](#) \*md)  
*Get an array of the numerical encoding for each possible base pair (i,j)*
- short \* [vrna\\_seq\\_encode](#) (const char \*sequence, [vrna\\_md\\_t](#) \*md)  
*Get a numerical representation of the nucleotide sequence.*
- short \* [vrna\\_seq\\_encode\\_simple](#) (const char \*sequence, [vrna\\_md\\_t](#) \*md)  
*Get a numerical representation of the nucleotide sequence (simple version)*
- int [vrna\\_nucleotide\\_encode](#) (char c, [vrna\\_md\\_t](#) \*md)  
*Encode a nucleotide character to numerical value.*
- char [vrna\\_nucleotide\\_decode](#) (int enc, [vrna\\_md\\_t](#) \*md)  
*Decode a numerical representation of a nucleotide back into nucleotide alphabet.*

## 15.60.2 Data Structure Documentation

### 15.60.2.1 struct vrna\_sequence\_s

Data structure representing a nucleotide sequence.

#### Data Fields

- [vrna\\_seq\\_type\\_e](#) type  
*The type of sequence.*
- char \* [string](#)  
*The string representation of the sequence.*
- short \* [encoding](#)  
*The integer representation of the sequence.*
- unsigned int [length](#)  
*The length of the sequence.*

## 15.60.3 Enumeration Type Documentation

### 15.60.3.1 vrna\_seq\_type\_e

```
enum vrna\_seq\_type\_e
```

```
#include <ViennaRNA/sequence.h>
```

A enumerator used in [vrna\\_sequence\\_s](#) to distinguish different nucleotide sequences.

#### Enumerator

VRNA_SEQ_UNKNOWN	Nucleotide sequence represents an Unkown type.
VRNA_SEQ_RNA	Nucleotide sequence represents an RNA type.
VRNA_SEQ_DNA	Nucleotide sequence represents a DNA type.

## 15.60.4 Function Documentation

### 15.60.4.1 vrna\_ptypes()

```
char* vrna_ptypes (
    const short * S,
    vrna\_md\_t * md )

#include <ViennaRNA/alphabet.h>
```

Get an array of the numerical encoding for each possible base pair (i,j)



**Note**

This array is always indexed in column-wise order, in contrast to previously different indexing between mfe and pf variants!

**See also**

[vrna\\_idx\\_col\\_wise\(\)](#), [vrna\\_fold\\_compound\\_t](#)

**15.60.4.2 vrna\_nucleotide\_encode()**

```
int vrna_nucleotide_encode (
    char c,
    vrna_md_t * md )

#include <ViennaRNA/alphabet.h>
```

Encode a nucleotide character to numerical value.

This function encodes a nucleotide character to its numerical representation as required by many functions in R<sub>N</sub>ALib.

**See also**

[vrna\\_nucleotide\\_decode\(\)](#), [vrna\\_seq\\_encode\(\)](#)

**Parameters**

<i>c</i>	The nucleotide character to encode
<i>md</i>	The model details that determine the kind of encoding

**Returns**

The encoded nucleotide

**15.60.4.3 vrna\_nucleotide\_decode()**

```
char vrna_nucleotide_decode (
    int enc,
    vrna_md_t * md )

#include <ViennaRNA/alphabet.h>
```

Decode a numerical representation of a nucleotide back into nucleotide alphabet.

This function decodes a numerical representation of a nucleotide character back into nucleotide alphabet

**See also**

[vrna\\_nucleotide\\_encode\(\)](#), [vrna\\_seq\\_encode\(\)](#)

**Parameters**

<i>enc</i>	The encoded nucleotide
<i>md</i>	The model details that determine the kind of decoding

**Returns**

The decoded nucleotide character

## 15.61 (Nucleic Acid Sequence) String Utilites

Functions to parse, convert, manipulate, create, and compare (nucleic acid sequence) strings.

### 15.61.1 Detailed Description

Functions to parse, convert, manipulate, create, and compare (nucleic acid sequence) strings.

Collaboration diagram for (Nucleic Acid Sequence) String Utilites:

### Files

- file [strings.h](#)

*General utility- and helper-functions for RNA sequence and structure strings used throughout the ViennaRNA Package.*

### Macros

- `#define XSTR(s) STR(s)`  
*Stringify a macro after expansion.*
- `#define STR(s) #s`  
*Stringify a macro argument.*
- `#define FILENAME_MAX_LENGTH 80`  
*Maximum length of filenames that are generated by our programs.*
- `#define FILENAME_ID_LENGTH 42`  
*Maximum length of id taken from fasta header for filename generation.*

### Functions

- `char * vrna_strdup_printf (const char *format,...)`  
*Safely create a formatted string.*
- `char * vrna_strdup_vprintf (const char *format, va_list argp)`  
*Safely create a formatted string.*
- `int vrna_strcat_printf (char **dest, const char *format,...)`  
*Safely append a formatted string to another string.*
- `int vrna_strcat_vprintf (char **dest, const char *format, va_list args)`  
*Safely append a formatted string to another string.*
- `char ** vrna_strsplit (const char *string, const char *delimiter)`  
*Split a string into tokens using a delimiting character.*
- `char * vrna_random_string (int l, const char symbols[ ])`  
*Create a random string using characters from a specified symbol set.*
- `int vrna_hamming_distance (const char *s1, const char *s2)`  
*Calculate hamming distance between two sequences.*
- `int vrna_hamming_distance_bound (const char *s1, const char *s2, int n)`  
*Calculate hamming distance between two sequences up to a specified length.*
- `void vrna_seq_toRNA (char *sequence)`

- Convert an input sequence (possibly containing DNA alphabet characters) to RNA alphabet.*

  - void [vrna\\_seq\\_toupper](#) (char \*sequence)

*Convert an input sequence to uppercase.*
- char \* [vrna\\_seq\\_ungapped](#) (const char \*seq)

*Remove gap characters from a nucleotide sequence.*
- char \* [vrna\\_cut\\_point\\_insert](#) (const char \*string, int cp)

*Add a separating '&' character into a string according to cut-point position.*
- char \* [vrna\\_cut\\_point\\_remove](#) (const char \*string, int \*cp)

*Remove a separating '&' character from a string.*

## 15.61.2 Macro Definition Documentation

### 15.61.2.1 FILENAME\_MAX\_LENGTH

```
#define FILENAME_MAX_LENGTH 80
#include <ViennaRNA/utils/strings.h>
```

Maximum length of filenames that are generated by our programs.

This definition should be used throughout the complete ViennaRNA package wherever a static array holding file-names of output files is declared.

### 15.61.2.2 FILENAME\_ID\_LENGTH

```
#define FILENAME_ID_LENGTH 42
#include <ViennaRNA/utils/strings.h>
```

Maximum length of id taken from fasta header for filename generation.

this has to be smaller than FILENAME\_MAX\_LENGTH since in most cases, some suffix will be appended to the ID

## 15.61.3 Function Documentation

### 15.61.3.1 vrna\_strdup\_printf()

```
char* vrna_strdup_printf (
    const char * format,
    ... )
#include <ViennaRNA/utils/strings.h>
```

Safely create a formatted string.

This function is a safe implementation for creating a formatted character array, similar to *sprintf*. Internally, it uses the *asprintf* function if available to dynamically allocate a large enough character array to store the supplied content. If *asprintf* is not available, mimic it's behavior using *vsprintf*.

#### Note

The returned pointer of this function should always be passed to *free()* to release the allocated memory

#### See also

[vrna\\_strdup\\_vprintf\(\)](#), [vrna\\_strcat\\_printf\(\)](#)

## Parameters

<i>format</i>	The format string (See also <code>asprintf</code> )
...	The list of variables used to fill the format string

## Returns

The formatted, null-terminated string, or NULL if something has gone wrong

15.61.3.2 `vrna_strdup_vprintf()`

```
char* vrna_strdup_vprintf (
    const char * format,
    va_list argp )
```

```
#include <ViennaRNA/utils/strings.h>
```

Safely create a formatted string.

This function is the *va\_list* version of [vrna\\_strdup\\_printf\(\)](#)

## Note

The returned pointer of this function should always be passed to `free()` to release the allocated memory

## See also

[vrna\\_strdup\\_printf\(\)](#), [vrna\\_strcat\\_printf\(\)](#), [vrna\\_strcat\\_vprintf\(\)](#)

## Parameters

<i>format</i>	The format string (See also <code>asprintf</code> )
<i>argp</i>	The list of arguments to fill the format string

## Returns

The formatted, null-terminated string, or NULL if something has gone wrong

15.61.3.3 `vrna_strcat_printf()`

```
int vrna_strcat_printf (
    char ** dest,
    const char * format,
    ... )
```

```
#include <ViennaRNA/utils/strings.h>
```

Safely append a formatted string to another string.

This function is a safe implementation for appending a formatted character array, similar to a combination of *strcat* and *sprintf*. The function automatically allocates enough memory to store both, the previous content stored at `dest` and the appended format string. If the `dest` pointer is NULL, the function allocate memory only for the format string. The function returns the number of characters in the resulting string or -1 in case of an error.

See also

[vrna\\_strcat\\_vprintf\(\)](#), [vrna\\_strdup\\_printf\(\)](#), [vrna\\_strdup\\_vprintf\(\)](#)

#### Parameters

<i>dest</i>	The address of a char *pointer where the formatted string is to be appended
<i>format</i>	The format string (See also <code>sprintf</code> )
...	The list of variables used to fill the format string

#### Returns

The number of characters in the final string, or -1 on error

#### 15.61.3.4 vrna\_strcat\_vprintf()

```
int vrna_strcat_vprintf (
    char ** dest,
    const char * format,
    va_list args )
```

```
#include <ViennaRNA/utils/strings.h>
```

Safely append a formatted string to another string.

This function is the *va\_list* version of [vrna\\_strcat\\_printf\(\)](#)

See also

[vrna\\_strcat\\_printf\(\)](#), [vrna\\_strdup\\_printf\(\)](#), [vrna\\_strdup\\_vprintf\(\)](#)

#### Parameters

<i>dest</i>	The address of a char *pointer where the formatted string is to be appended
<i>format</i>	The format string (See also <code>sprintf</code> )
<i>args</i>	The list of argument to fill the format string

**Returns**

The number of characters in the final string, or -1 on error

**15.61.3.5 vrna\_strsplit()**

```
char** vrna_strsplit (
    const char * string,
    const char * delimiter )
```

```
#include <ViennaRNA/utils/strings.h>
```

Split a string into tokens using a delimiting character.

This function splits a string into an array of strings using a single character that delimits the elements within the string. The default delimiter is the ampersand ' & ' and will be used when `NULL` is passed as a second argument. The returned list is `NULL` terminated, i.e. the last element is `NULL`. If the delimiter is not found, the returned list contains exactly one element: the input string.

For instance, the following code:

```
char **tok = vrna_strsplit ("GGGG&CCCC&AAAA", NULL);

for (char **ptr = tok; *ptr; ptr++) {
    printf("%s\n", *ptr);
    free(*ptr);
}
free(tok);
```

produces this output:

```
GGGG
CCCC
AAAAA
```

and properly free's the memory occupied by the returned element array.

**Note**

This function internally uses `strtok_r()` and is therefore considered to be thread-safe. Also note, that it is the users responsibility to free the memory of the array and that of the individual element strings!

**Parameters**

<i>string</i>	The input string that should be split into elements
<i>delimiter</i>	The delimiting character. If <code>NULL</code> , the delimiter is " & "

**Returns**

A `NULL` terminated list of the elements in the string

### 15.61.3.6 vrna\_random\_string()

```
char* vrna_random_string (
    int l,
    const char symbols[] )
```

```
#include <ViennaRNA/utils/strings.h>
```

Create a random string using characters from a specified symbol set.

#### Parameters

<i>l</i>	The length of the sequence
<i>symbols</i>	The symbol set

#### Returns

A random string of length 'l' containing characters from the symbolset

### 15.61.3.7 vrna\_hamming\_distance()

```
int vrna_hamming_distance (
    const char * s1,
    const char * s2 )
```

```
#include <ViennaRNA/utils/strings.h>
```

Calculate hamming distance between two sequences.

#### Parameters

<i>s1</i>	The first sequence
<i>s2</i>	The second sequence

#### Returns

The hamming distance between s1 and s2

### 15.61.3.8 vrna\_hamming\_distance\_bound()

```
int vrna_hamming_distance_bound (
    const char * s1,
    const char * s2,
    int n )
```



```
#include <ViennaRNA/utils/strings.h>
```

Calculate hamming distance between two sequences up to a specified length.

This function is similar to `vrna_hamming_distance()` but instead of comparing both sequences up to their actual length only the first `n` characters are taken into account

#### Parameters

<i>s1</i>	The first sequence
<i>s2</i>	The second sequence
<i>n</i>	The length of the subsequences to consider (starting from the 5' end)

#### Returns

The hamming distance between `s1` and `s2`

#### 15.61.3.9 `vrna_seq_toRNA()`

```
void vrna_seq_toRNA (  
    char * sequence )
```

```
#include <ViennaRNA/utils/strings.h>
```

Convert an input sequence (possibly containing DNA alphabet characters) to RNA alphabet.

This function substitutes *T* and *t* with *U* and *u*, respectively

#### Parameters

<i>sequence</i>	The sequence to be converted
-----------------	------------------------------

#### 15.61.3.10 `vrna_seq_toupper()`

```
void vrna_seq_toupper (  
    char * sequence )
```

```
#include <ViennaRNA/utils/strings.h>
```

Convert an input sequence to uppercase.

#### Parameters

<i>sequence</i>	The sequence to be converted
-----------------	------------------------------

### 15.61.3.11 `vrna_seq_ungapped()`

```
char* vrna_seq_ungapped (
    const char * seq )
```

```
#include <ViennaRNA/utils/strings.h>
```

Remove gap characters from a nucleotide sequence.

#### Parameters

<i>sequence</i>	The original, null-terminated nucleotide sequence
-----------------	---

#### Returns

A copy of the input sequence with all gap characters removed

### 15.61.3.12 `vrna_cut_point_insert()`

```
char* vrna_cut_point_insert (
    const char * string,
    int cp )
```

```
#include <ViennaRNA/utils/strings.h>
```

Add a separating '&' character into a string according to cut-point position.

If the cut-point position is less or equal to zero, this function just returns a copy of the provided string. Otherwise, the cut-point character is set at the corresponding position

#### Parameters

<i>string</i>	The original string
<i>cp</i>	The cut-point position

#### Returns

A copy of the provided string including the cut-point character

### 15.61.3.13 `vrna_cut_point_remove()`

```
char* vrna_cut_point_remove (
    const char * string,
    int * cp )
```

```
#include <ViennaRNA/Utils/strings.h>
```

Remove a separating '&' character from a string.

This function removes the cut-point indicating '&' character from a string and memorizes its position in a provided integer variable. If not '&' is found in the input, the integer variable is set to -1. The function returns a copy of the input string with the '&' being sliced out.

#### Parameters

<i>string</i>	The original string
<i>cp</i>	The cut-point position

#### Returns

A copy of the input string with the '&' being sliced out

## 15.62 Secondary Structure Utilities

Functions to create, parse, convert, manipulate, and compare secondary structure representations.

### 15.62.1 Detailed Description

Functions to create, parse, convert, manipulate, and compare secondary structure representations.

Collaboration diagram for Secondary Structure Utilities:

### Modules

- [Dot-Bracket Notation of Secondary Structures](#)
- [Pair Table Representation of Secondary Structures](#)
- [Pair List Representation of Secondary Structures](#)
- [Helix List Representation of Secondary Structures](#)
- [Tree Representation of Secondary Structures](#)
- [Deprecated Interface for Secondary Structure Utilities](#)

### Files

- file [structures.h](#)  
*Various utility- and helper-functions for secondary structure parsing, converting, etc.*

### Functions

- int \* [vrna\\_loopidx\\_from\\_ptable](#) (const short \*pt)  
*Get a loop index representation of a structure.*
- int [vrna\\_bp\\_distance](#) (const char \*str1, const char \*str2)  
*Compute the "base pair" distance between two secondary structures s1 and s2.*
- unsigned int \* [vrna\\_refBPcnt\\_matrix](#) (const short \*reference\_pt, unsigned int turn)  
*Make a reference base pair count matrix.*
- unsigned int \* [vrna\\_refBPdist\\_matrix](#) (const short \*pt1, const short \*pt2, unsigned int turn)  
*Make a reference base pair distance matrix.*
- char \* [vrna\\_db\\_from\\_probs](#) (const FLT\_OR\_DBL \*pr, unsigned int length)  
*Create a dot-bracket like structure string from base pair probability matrix.*
- char [vrna\\_bpp\\_symbol](#) (const float \*x)  
*Get a pseudo dot bracket notation for a given probability information.*
- char \* [vrna\\_db\\_from\\_bp\\_stack](#) ([vrna\\_bp\\_stack\\_t](#) \*bp, unsigned int length)  
*Create a dot-bracket/parenthesis structure from backtracking stack.*

### 15.62.2 Function Documentation

### 15.62.2.1 vrna\_bp\_distance()

```
int vrna_bp_distance (
    const char * str1,
    const char * str2 )
```

```
#include <ViennaRNA/utils/structures.h>
```

Compute the "base pair" distance between two secondary structures s1 and s2.

The sequences should have the same length. dist = number of base pairs in one structure but not in the other same as edit distance with open-pair close-pair as move-set

#### Parameters

<i>str1</i>	First structure in dot-bracket notation
<i>str2</i>	Second structure in dot-bracket notation

#### Returns

The base pair distance between str1 and str2

### 15.62.2.2 vrna\_refBPcnt\_matrix()

```
unsigned int* vrna_refBPcnt_matrix (
    const short * reference_pt,
    unsigned int turn )
```

```
#include <ViennaRNA/utils/structures.h>
```

Make a reference base pair count matrix.

Get an upper triangular matrix containing the number of basepairs of a reference structure for each interval [i,j] with i<j. Access it via iindx!!!

### 15.62.2.3 vrna\_refBPdist\_matrix()

```
unsigned int* vrna_refBPdist_matrix (
    const short * pt1,
    const short * pt2,
    unsigned int turn )
```

```
#include <ViennaRNA/utils/structures.h>
```

Make a reference base pair distance matrix.

Get an upper triangular matrix containing the base pair distance of two reference structures for each interval [i,j] with i<j. Access it via iindx!!!

#### 15.62.2.4 vrna\_db\_from\_bp\_stack()

```
char* vrna_db_from_bp_stack (
    vrna_bp_stack_t * bp,
    unsigned int length )
```

```
#include <ViennaRNA/utils/structures.h>
```

Create a dot-bracket/parenthesis structure from backtracking stack.

This function is capable to create dot-bracket structures from suboptimal structure prediction sensu M. Zuker

## Parameters

<i>bp</i>	Base pair stack containing the traced base pairs
<i>length</i>	The length of the structure

## Returns

The secondary structure in dot-bracket notation as provided in the input

## 15.63 Dot-Bracket Notation of Secondary Structures

### 15.63.1 Detailed Description

Collaboration diagram for Dot-Bracket Notation of Secondary Structures:

#### Macros

- `#define VRNA_BRACKETS_ALPHA 4U`  
*Bitflag to indicate secondary structure notations using uppercase/lowercase letters from the latin alphabet.*
- `#define VRNA_BRACKETS_RND 8U`  
*Bitflag to indicate secondary structure notations using round brackets (parenthesis), ( )*
- `#define VRNA_BRACKETS_CLY 16U`  
*Bitflag to indicate secondary structure notations using curly brackets, { }*
- `#define VRNA_BRACKETS_ANG 32U`  
*Bitflag to indicate secondary structure notations using angular brackets, < >*
- `#define VRNA_BRACKETS_SQR 64U`  
*Bitflag to indicate secondary structure notations using square brackets, [ ]*
- `#define VRNA_BRACKETS_DEFAULT`  
*Default bitmask to indicate secondary structure notation using any pair of brackets.*

#### Functions

- `char * vrna_db_pack (const char *struc)`  
*Pack secondary secondary structure, 5:1 compression using base 3 encoding.*
- `char * vrna_db_unpack (const char *packed)`  
*Unpack secondary structure previously packed with `vrna_db_pack()`*
- `void vrna_db_flatten (char *structure, unsigned int options)`  
*Substitute pairs of brackets in a string with parenthesis.*
- `void vrna_db_flatten_to (char *string, const char target[3], unsigned int options)`  
*Substitute pairs of brackets in a string with another type of pair characters.*
- `char * vrna_db_from_ptable (short *pt)`  
*Convert a pair table into dot-parenthesis notation.*
- `char * vrna_db_from_WUSS (const char *wuss)`  
*Convert a WUSS annotation string to dot-bracket format.*
- `char * vrna_db_from_plist (vrna_ep_t *pairs, unsigned int n)`  
*Convert a list of base pairs into dot-bracket notation.*
- `char * vrna_db_to_element_string (const char *structure)`  
*Convert a secondary structure in dot-bracket notation to a nucleotide annotation of loop contexts.*

### 15.63.2 Macro Definition Documentation



### 15.63.2.1 VRNA\_BRACKETS\_ALPHA

```
#define VRNA_BRACKETS_ALPHA 4U
```

```
#include <ViennaRNA/utils/structures.h>
```

Bitflag to indicate secondary structure notations using uppercase/lowercase letters from the latin alphabet.

See also

[vrna\\_ptable\\_from\\_string\(\)](#)

### 15.63.2.2 VRNA\_BRACKETS\_RND

```
#define VRNA_BRACKETS_RND 8U
```

```
#include <ViennaRNA/utils/structures.h>
```

Bitflag to indicate secondary structure notations using round brackets (parenthesis), ( )

See also

[vrna\\_ptable\\_from\\_string\(\)](#), [vrna\\_db\\_flatten\(\)](#), [vrna\\_db\\_flatten\\_to\(\)](#)

### 15.63.2.3 VRNA\_BRACKETS\_CLY

```
#define VRNA_BRACKETS_CLY 16U
```

```
#include <ViennaRNA/utils/structures.h>
```

Bitflag to indicate secondary structure notations using curly brackets, { }

See also

[vrna\\_ptable\\_from\\_string\(\)](#), [vrna\\_db\\_flatten\(\)](#), [vrna\\_db\\_flatten\\_to\(\)](#)

### 15.63.2.4 VRNA\_BRACKETS\_ANG

```
#define VRNA_BRACKETS_ANG 32U
```

```
#include <ViennaRNA/utils/structures.h>
```

Bitflag to indicate secondary structure notations using angular brackets, <>

See also

[vrna\\_ptable\\_from\\_string\(\)](#), [vrna\\_db\\_flatten\(\)](#), [vrna\\_db\\_flatten\\_to\(\)](#)

### 15.63.2.5 VRNA\_BRACKETS\_SQR

```
#define VRNA_BRACKETS_SQR 64U
```

```
#include <ViennaRNA/utils/structures.h>
```

Bitflag to indicate secondary structure notations using square brackets, [ ]

See also

[vrna\\_ptable\\_from\\_string\(\)](#), [vrna\\_db\\_flatten\(\)](#), [vrna\\_db\\_flatten\\_to\(\)](#)

### 15.63.2.6 VRNA\_BRACKETS\_DEFAULT

```
#define VRNA_BRACKETS_DEFAULT
```

```
#include <ViennaRNA/utils/structures.h>
```

Value:

```
(VRNA_BRACKETS_RND | \
 VRNA_BRACKETS_CLY | \
 VRNA_BRACKETS_ANG | \
 VRNA_BRACKETS_SQR)
```

Default bitmask to indicate secondary structure notation using any pair of brackets.

See also

[vrna\\_ptable\\_from\\_string\(\)](#), [vrna\\_db\\_flatten\(\)](#), [vrna\\_db\\_flatten\\_to\(\)](#)

## 15.63.3 Function Documentation

### 15.63.3.1 vrna\_db\_pack()

```
char* vrna_db_pack (
    const char * struc )
```

```
#include <ViennaRNA/utils/structures.h>
```

Pack secondary secondary structure, 5:1 compression using base 3 encoding.

Returns a binary string encoding of the secondary structure using a 5:1 compression scheme. The string is NULL terminated and can therefore be used with standard string functions such as strcmp(). Useful for programs that need to keep many structures in memory.

See also

[vrna\\_db\\_unpack\(\)](#)

## Parameters

<i>struc</i>	The secondary structure in dot-bracket notation
--------------	---

## Returns

The binary encoded structure

15.63.3.2 `vrna_db_unpack()`

```
char* vrna_db_unpack (
    const char * packed )
```

```
#include <ViennaRNA/utils/structures.h>
```

Unpack secondary structure previously packed with [vrna\\_db\\_pack\(\)](#)

Translate a compressed binary string produced by [vrna\\_db\\_pack\(\)](#) back into the familiar dot-bracket notation.

## See also

[vrna\\_db\\_pack\(\)](#)

## Parameters

<i>packed</i>	The binary encoded packed secondary structure
---------------	---

## Returns

The unpacked secondary structure in dot-bracket notation

15.63.3.3 `vrna_db_flatten()`

```
vrna_db_flatten (
    char * structure,
    unsigned int options )
```

```
#include <ViennaRNA/utils/structures.h>
```

Substitute pairs of brackets in a string with parenthesis.

This function can be used to replace brackets of unusual types, such as angular brackets `<>`, to dot-bracket format. The `options` parameter is used to specify which types of brackets will be replaced by round parenthesis `()`.

## See also

[vrna\\_db\\_flatten\\_to\(\)](#), [VRNA\\_BRACKETS\\_RND](#), [VRNA\\_BRACKETS\\_ANG](#), [VRNA\\_BRACKETS\\_CLY](#), [VRNA\\_BRACKETS\\_SQR](#), [VRNA\\_BRACKETS\\_DEFAULT](#)

## Parameters

<i>structure</i>	The structure string where brackets are flattened in-place
<i>options</i>	A bitmask to specify which types of brackets should be flattened out

**SWIG Wrapper Notes** This function flattens an input structure string in-place! The second parameter is optional and defaults to [VRNA\\_BRACKETS\\_DEFAULT](#).

An overloaded version of this function exists, where an additional second parameter can be passed to specify the target brackets, i.e. the type of matching pair characters all brackets will be flattened to. Therefore, in the scripting language interface this function is a replacement for [vrna\\_db\\_flatten\\_to\(\)](#).

15.63.3.4 `vrna_db_flatten_to()`

```
void vrna_db_flatten_to (
    char * string,
    const char target[3],
    unsigned int options )
```

```
#include <ViennaRNA/utils/structures.h>
```

Substitute pairs of brackets in a string with another type of pair characters.

This function can be used to replace brackets in a structure annotation string, such as square brackets `[]`, to another type of pair characters, e.g. angular brackets `<>`.

The `target` array must contain a character for the 'pair open' annotation at position 0, and one for 'pair close' at position 1. `Options` parameter is used to specify which types of brackets will be replaced by the new pairs.

## See also

[vrna\\_db\\_flatten\(\)](#), [VRNA\\_BRACKETS\\_RND](#), [VRNA\\_BRACKETS\\_ANG](#), [VRNA\\_BRACKETS\\_CLY](#), [VRNA\\_BRACKETS\\_SQR](#), [VRNA\\_BRACKETS\\_DEFAULT](#)

## Parameters

<i>string</i>	The structure string where brackets are flattened in-place
<i>target</i>	The new pair characters the string will be flattened to
<i>options</i>	A bitmask to specify which types of brackets should be flattened out

**SWIG Wrapper Notes** This function is available as an overloaded version of [vrna\\_db\\_flatten\(\)](#)

15.63.3.5 `vrna_db_from_ptable()`

```
char* vrna_db_from_ptable (
    short * pt )
```

```
#include <ViennaRNA/utils/structures.h>
```

Convert a pair table into dot-parenthesis notation.

**Parameters**

<i>pt</i>	The pair table to be copied
-----------	-----------------------------

**Returns**

A char pointer to the dot-bracket string

15.63.3.6 `vrna_db_from_WUSS()`

```
char* vrna_db_from_WUSS (
    const char * wuss )
```

```
#include <ViennaRNA/utils/structures.h>
```

Convert a WUSS annotation string to dot-bracket format.

**Note**

This function flattens all brackets, and treats pseudo-knots annotated by matching pairs of upper/lowercase letters as unpaired nucleotides

**See also**

[Washington University Secondary Structure \(WUSS\) notation](#)

**Parameters**

<i>wuss</i>	The input string in WUSS notation
-------------	-----------------------------------

**Returns**

A dot-bracket notation of the input secondary structure

### 15.63.3.7 `vrna_db_from_plist()`

```
char* vrna_db_from_plist (
    vrna_ep_t * pairs,
    unsigned int n )
```

```
#include <ViennaRNA/utils/structures.h>
```

Convert a list of base pairs into dot-bracket notation.

See also

[vrna\\_plist\(\)](#)

#### Parameters

<i>pairs</i>	A <a href="#">vrna_ep_t</a> containing the pairs to be included in the dot-bracket string
<i>n</i>	The length of the structure (number of nucleotides)

#### Returns

The dot-bracket string containing the provided base pairs

### 15.63.3.8 `vrna_db_to_element_string()`

```
char* vrna_db_to_element_string (
    const char * structure )
```

```
#include <ViennaRNA/utils/structures.h>
```

Convert a secondary structure in dot-bracket notation to a nucleotide annotation of loop contexts.

#### Parameters

<i>structure</i>	The secondary structure in dot-bracket notation
------------------	---

#### Returns

A string annotating each nucleotide according to it's structural context

## 15.64 Pair Table Representation of Secondary Structures

### 15.64.1 Detailed Description

Collaboration diagram for Pair Table Representation of Secondary Structures:

#### Functions

- short \* [vrna\\_ptable](#) (const char \*structure)  
*Create a pair table from a dot-bracket notation of a secondary structure.*
- short \* [vrna\\_ptable\\_from\\_string](#) (const char \*string, unsigned int options)  
*Create a pair table for a secondary structure string.*
- short \* [vrna\\_pt\\_pk\\_get](#) (const char \*structure)  
*Create a pair table of a secondary structure (pseudo-knot version)*
- short \* [vrna\\_ptable\\_copy](#) (const short \*pt)  
*Get an exact copy of a pair table.*
- short \* [vrna\\_pt\\_aln\\_get](#) (const char \*structure)  
*Create a pair table of a secondary structure (snoop align version)*
- short \* [vrna\\_pt\\_snoop\\_get](#) (const char \*structure)  
*Create a pair table of a secondary structure (snoop version)*

### 15.64.2 Function Documentation

#### 15.64.2.1 vrna\_ptable()

```
short* vrna_ptable (
    const char * structure )
```

```
#include <ViennaRNA/utils/structures.h>
```

Create a pair table from a dot-bracket notation of a secondary structure.

Returns a newly allocated table, such that table[i]=j if (i,j) pair or 0 if i is unpaired, table[0] contains the length of the structure.

See also

[vrna\\_ptable\\_from\\_string\(\)](#), [vrna\\_db\\_from\\_ptable\(\)](#)

#### Parameters

<i>structure</i>	The secondary structure in dot-bracket notation
------------------	---

**Returns**

A pointer to the created pair\_table

**15.64.2.2 vrna\_ptable\_from\_string()**

```
short* vrna_ptable_from_string (
    const char * string,
    unsigned int options )
```

```
#include <ViennaRNA/utils/structures.h>
```

Create a pair table for a secondary structure string.

This function takes an input string of a secondary structure annotation in [Dot-Bracket Notation](#) (a.k.a. [Dot-Parenthesis Notation](#)) or [Extended Dot-Bracket Notation](#), and converts it into a pair table representation.

**See also**

[vrna\\_ptable\(\)](#), [vrna\\_db\\_from\\_ptable\(\)](#), [vrna\\_db\\_flatten\\_to\(\)](#), [VRNA\\_BRACKETS\\_RND](#), [VRNA\\_BRACKETS\\_ANG](#), [VRNA\\_BRACKETS\\_CLY](#), [VRNA\\_BRACKETS\\_SQR](#), [VRNA\\_BRACKETS\\_DEFAULT](#)

**Parameters**

<i>string</i>	Secondary structure in <a href="#">Extended Dot-Bracket Notation</a>
<i>options</i>	A bitmask to specify which brackets are recognized during conversion to pair table

**Returns**

A pointer to a new pair table of the provided secondary structure

**15.64.2.3 vrna\_pt\_pk\_get()**

```
short* vrna_pt_pk_get (
    const char * structure )
```

```
#include <ViennaRNA/utils/structures.h>
```

Create a pair table of a secondary structure (pseudo-knot version)

Returns a newly allocated table, such that table[i]=j if (i,j) pair or 0 if i is unpaired, table[0] contains the length of the structure.

In contrast to [vrna\\_ptable\(\)](#) this function also recognizes the base pairs denoted by '[' and ']' brackets.



**Parameters**

<i>structure</i>	The secondary structure in (extended) dot-bracket notation
------------------	--

**Returns**

A pointer to the created pair\_table

**15.64.2.4 vrna\_ptable\_copy()**

```
short* vrna_ptable_copy (
    const short * pt )
```

```
#include <ViennaRNA/utils/structures.h>
```

Get an exact copy of a pair table.

**Parameters**

<i>pt</i>	The pair table to be copied
-----------	-----------------------------

**Returns**

A pointer to the copy of 'pt'

**15.64.2.5 vrna\_pt\_snoop\_get()**

```
short* vrna_pt_snoop_get (
    const char * structure )
```

```
#include <ViennaRNA/utils/structures.h>
```

Create a pair table of a secondary structure (snoop version)

returns a newly allocated table, such that: table[i]=j if (i,j) pair or 0 if i is unpaired, table[0] contains the length of the structure. The special pseudoknotted H/ACA-mRNA structure is taken into account.

## 15.65 Pair List Representation of Secondary Structures

### 15.65.1 Detailed Description

Collaboration diagram for Pair List Representation of Secondary Structures:

#### Data Structures

- struct [vrna\\_elem\\_prob\\_s](#)  
Data structure representing a single entry of an element probability list (e.g. list of pair probabilities) [More...](#)

#### Macros

- #define [VRNA\\_PLIST\\_TYPE\\_BASEPAIR](#) 0  
A Base Pair element.
- #define [VRNA\\_PLIST\\_TYPE\\_GQUAD](#) 1  
A G-Quadruplex element.
- #define [VRNA\\_PLIST\\_TYPE\\_H\\_MOTIF](#) 2  
A Hairpin loop motif element.
- #define [VRNA\\_PLIST\\_TYPE\\_I\\_MOTIF](#) 3  
An Internal loop motif element.
- #define [VRNA\\_PLIST\\_TYPE\\_UD\\_MOTIF](#) 4  
An Unstructured Domain motif element.
- #define [VRNA\\_PLIST\\_TYPE\\_STACK](#) 5  
A Base Pair stack element.

#### Typedefs

- typedef struct [vrna\\_elem\\_prob\\_s](#) [vrna\\_ep\\_t](#)  
Convenience typedef for data structure [vrna\\_elem\\_prob\\_s](#).

#### Functions

- [vrna\\_ep\\_t](#) \* [vrna\\_plist](#) (const char \*struc, float pr)  
Create a [vrna\\_ep\\_t](#) from a dot-bracket string.

### 15.65.2 Data Structure Documentation

#### 15.65.2.1 struct vrna\_elem\_prob\_s

Data structure representing a single entry of an element probability list (e.g. list of pair probabilities)

#### See also

[vrna\\_plist\(\)](#), [vrna\\_plist\\_from\\_probs\(\)](#), [vrna\\_db\\_from\\_plist\(\)](#), [VRNA\\_PLIST\\_TYPE\\_BASEPAIR](#), [VRNA\\_PLIST\\_TYPE\\_GQUAD](#), [VRNA\\_PLIST\\_TYPE\\_H\\_MOTIF](#), [VRNA\\_PLIST\\_TYPE\\_I\\_MOTIF](#), [VRNA\\_PLIST\\_TYPE\\_UD\\_MOTIF](#), [VRNA\\_PLIST\\_TYPE\\_STACK](#)

**Data Fields**

- int `i`  
*Start position (usually 5' nucleotide that starts the element, e.g. base pair)*
- int `j`  
*End position (usually 3' nucleotide that ends the element, e.g. base pair)*
- float `p`  
*Probability of the element.*
- int `type`  
*Type of the element.*

**15.65.3 Function Documentation****15.65.3.1 vrna\_plist()**

```
vrna_ep_t* vrna_plist (
    const char * struc,
    float pr )
```

```
#include <ViennaRNA/utils/structures.h>
```

Create a `vrna_ep_t` from a dot-bracket string.

The dot-bracket string is parsed and for each base pair an entry in the plist is created. The probability of each pair in the list is set by a function parameter.

The end of the plist is marked by sequence positions `i` as well as `j` equal to 0. This condition should be used to stop looping over its entries

**Parameters**

<code>struc</code>	The secondary structure in dot-bracket notation
<code>pr</code>	The probability for each base pair used in the plist

**Returns**

The plist array

## 15.66 Helix List Representation of Secondary Structures

### 15.66.1 Detailed Description

Collaboration diagram for Helix List Representation of Secondary Structures:

#### Data Structures

- struct [vrna\\_hx\\_s](#)  
*Data structure representing an entry of a helix list. [More...](#)*

#### Typedefs

- typedef struct [vrna\\_hx\\_s](#) [vrna\\_hx\\_t](#)  
*Convenience typedef for data structure [vrna\\_hx\\_s](#).*

#### Functions

- [vrna\\_hx\\_t \\* vrna\\_hx\\_from\\_ptable](#) (short \*pt)  
*Convert a pair table representation of a secondary structure into a helix list.*
- [vrna\\_hx\\_t \\* vrna\\_hx\\_merge](#) (const [vrna\\_hx\\_t](#) \*list, int maxdist)  
*Create a merged helix list from another helix list.*

### 15.66.2 Data Structure Documentation

#### 15.66.2.1 struct vrna\_hx\_s

Data structure representing an entry of a helix list.

### 15.66.3 Function Documentation

#### 15.66.3.1 vrna\_hx\_from\_ptable()

```
vrna_hx_t* vrna_hx_from_ptable (
    short * pt )
```

```
#include <ViennaRNA/utils/structures.h>
```

Convert a pair table representation of a secondary structure into a helix list.

**Parameters**

<i>pt</i>	The secondary structure in pair table representation
-----------	--

**Returns**

The secondary structure represented as a helix list

## 15.67 Tree Representation of Secondary Structures

### 15.67.1 Detailed Description

Secondary structures can be readily represented as trees, where internal nodes represent base pairs, and leaves represent unpaired nucleotides. The dot-bracket structure string already is a tree represented by a string of parenthesis (base pairs) and dots for the leaf nodes (unpaired nucleotides).

See [Tree Representations of Secondary Structures](#) for a detailed description on tree representation of secondary structures. Collaboration diagram for Tree Representation of Secondary Structures:

### Macros

- `#define VRNA_STRUCTURE_TREE_HIT 1U`  
*Homeomorphically Irreducible [Tree](#) (HIT) representation of a secondary structure.*
- `#define VRNA_STRUCTURE_TREE_SHAPIRO_SHORT 2U`  
*(short) Coarse Grained representation of a secondary structure*
- `#define VRNA_STRUCTURE_TREE_SHAPIRO 3U`  
*(full) Coarse Grained representation of a secondary structure*
- `#define VRNA_STRUCTURE_TREE_SHAPIRO_EXT 4U`  
*(extended) Coarse Grained representation of a secondary structure*
- `#define VRNA_STRUCTURE_TREE_SHAPIRO_WEIGHT 5U`  
*(weighted) Coarse Grained representation of a secondary structure*
- `#define VRNA_STRUCTURE_TREE_EXPANDED 6U`  
*Expanded [Tree](#) representation of a secondary structure.*

### Functions

- `char * vrna_db_to_tree_string (const char *structure, unsigned int type)`  
*Convert a Dot-Bracket structure string into tree string representation.*
- `char * vrna_tree_string_unweight (const char *structure)`  
*Remove weights from a linear string tree representation of a secondary structure.*
- `char * vrna_tree_string_to_db (const char *tree)`  
*Convert a linear tree string representation of a secondary structure back to Dot-Bracket notation.*

### 15.67.2 Macro Definition Documentation

#### 15.67.2.1 VRNA\_STRUCTURE\_TREE\_HIT

```
#define VRNA_STRUCTURE_TREE_HIT 1U
#include <ViennaRNA/utils/structures.h>
```

Homeomorphically Irreducible [Tree](#) (HIT) representation of a secondary structure.

See also

[vrna\\_db\\_to\\_tree\\_string\(\)](#)

**15.67.2.2 VRNA\_STRUCTURE\_TREE\_SHAPIRO\_SHORT**

```
#define VRNA_STRUCTURE_TREE_SHAPIRO_SHORT 2U  
  
#include <ViennaRNA/utils/structures.h>
```

(short) Coarse Grained representation of a secondary structure

See also

[vrna\\_db\\_to\\_tree\\_string\(\)](#)

**15.67.2.3 VRNA\_STRUCTURE\_TREE\_SHAPIRO**

```
#define VRNA_STRUCTURE_TREE_SHAPIRO 3U  
  
#include <ViennaRNA/utils/structures.h>
```

(full) Coarse Grained representation of a secondary structure

See also

[vrna\\_db\\_to\\_tree\\_string\(\)](#)

**15.67.2.4 VRNA\_STRUCTURE\_TREE\_SHAPIRO\_EXT**

```
#define VRNA_STRUCTURE_TREE_SHAPIRO_EXT 4U  
  
#include <ViennaRNA/utils/structures.h>
```

(extended) Coarse Grained representation of a secondary structure

See also

[vrna\\_db\\_to\\_tree\\_string\(\)](#)

**15.67.2.5 VRNA\_STRUCTURE\_TREE\_SHAPIRO\_WEIGHT**

```
#define VRNA_STRUCTURE_TREE_SHAPIRO_WEIGHT 5U  
  
#include <ViennaRNA/utils/structures.h>
```

(weighted) Coarse Grained representation of a secondary structure

See also

[vrna\\_db\\_to\\_tree\\_string\(\)](#)

### 15.67.2.6 VRNA\_STRUCTURE\_TREE\_EXPANDED

```
#define VRNA_STRUCTURE_TREE_EXPANDED 6U

#include <ViennaRNA/utils/structures.h>
```

Expanded [Tree](#) representation of a secondary structure.

See also

[vrna\\_db\\_to\\_tree\\_string\(\)](#)

## 15.67.3 Function Documentation

### 15.67.3.1 vrna\_db\_to\_tree\_string()

```
char* vrna_db_to_tree_string (
    const char * structure,
    unsigned int type )

#include <ViennaRNA/utils/structures.h>
```

Convert a Dot-Bracket structure string into tree string representation.

This function allows one to convert a secondary structure in dot-bracket notation into one of the various tree representations for secondary structures. The resulting tree is then represented as a string of parenthesis and node symbols, similar to to the Newick format.

Currently we support conversion into the following formats, denoted by the value of parameter `type`:

- [VRNA\\_STRUCTURE\\_TREE\\_HIT](#) - Homeomorphically Irreducible [Tree](#) (HIT) representation of a secondary structure. (See also Fontana et al. 1993 [8])
- [VRNA\\_STRUCTURE\\_TREE\\_SHAPIRO\\_SHORT](#) - (short) Coarse Grained representation of a secondary structure (same as Shapiro 1988 [20], but with root node `R` and without `S` nodes for the stems)
- [VRNA\\_STRUCTURE\\_TREE\\_SHAPIRO](#) - (full) Coarse Grained representation of a secondary structure (See also Shapiro 1988 [20])
- [VRNA\\_STRUCTURE\\_TREE\\_SHAPIRO\\_EXT](#) - (extended) Coarse Grained representation of a secondary structure (same as Shapiro 1988 [20], but external nodes denoted as `E` )
- [VRNA\\_STRUCTURE\\_TREE\\_SHAPIRO\\_WEIGHT](#) - (weighted) Coarse Grained representation of a secondary structure (same as [VRNA\\_STRUCTURE\\_TREE\\_SHAPIRO\\_EXT](#) but with additional weights for number of unpaired nucleotides in loop, and number of pairs in stems)
- [VRNA\\_STRUCTURE\\_TREE\\_EXPANDED](#) - Expanded [Tree](#) representation of a secondary structure.

See also

[Tree Representations of Secondary Structures](#)



## Parameters

<i>structure</i>	The null-terminated dot-bracket structure string
<i>type</i>	A switch to determine the type of tree string representation

## Returns

A tree representation of the input `structure`

15.67.3.2 `vrna_tree_string_unweight()`

```
char* vrna_tree_string_unweight (
    const char * structure )
```

```
#include <ViennaRNA/utils/structures.h>
```

Remove weights from a linear string tree representation of a secondary structure.

This function strips the weights of a linear string tree representation such as `HIT`, or Coarse Grained `Tree` sensu Shapiro [20]

## See also

[vrna\\_db\\_to\\_tree\\_string\(\)](#)

## Parameters

<i>structure</i>	A linear string tree representation of a secondary structure with weights
------------------	---

## Returns

A linear string tree representation of a secondary structure without weights

15.67.3.3 `vrna_tree_string_to_db()`

```
char* vrna_tree_string_to_db (
    const char * tree )
```

```
#include <ViennaRNA/utils/structures.h>
```

Convert a linear tree string representation of a secondary structure back to Dot-Bracket notation.

## Warning

This function only accepts *Expanded* and *HIT* tree representations!

See also

[vrna\\_db\\_to\\_tree\\_string\(\)](#), [VRNA\\_STRUCTURE\\_TREE\\_EXPANDED](#), [VRNA\\_STRUCTURE\\_TREE\\_HIT](#),  
[Tree Representations of Secondary Structures](#)

**Parameters**

<i>tree</i>	A linear tree string representation of a secondary structure
-------------	--

**Returns**

A dot-bracket notation of the secondary structure provided in `tree`

## 15.68 Deprecated Interface for Secondary Structure Utilities

### 15.68.1 Detailed Description

Collaboration diagram for Deprecated Interface for Secondary Structure Utilities:

#### Files

- file [RNAstruct.h](#)  
*Parsing and Coarse Graining of Structures.*

#### Functions

- char \* [b2HIT](#) (const char \*structure)  
*Converts the full structure from bracket notation to the HIT notation including root.*
- char \* [b2C](#) (const char \*structure)  
*Converts the full structure from bracket notation to the a coarse grained notation using the 'H' 'B' 'I' 'M' and 'R' identifiers.*
- char \* [b2Shapiro](#) (const char \*structure)  
*Converts the full structure from bracket notation to the weighted coarse grained notation using the 'H' 'B' 'I' 'M' 'S' 'E' and 'R' identifiers.*
- char \* [add\\_root](#) (const char \*structure)  
*Adds a root to an un-rooted tree in any except bracket notation.*
- char \* [expand\\_Shapiro](#) (const char \*coarse)  
*Inserts missing 'S' identifiers in unweighted coarse grained structures as obtained from [b2C\(\)](#).*
- char \* [expand\\_Full](#) (const char \*structure)  
*Convert the full structure from bracket notation to the expanded notation including root.*
- char \* [unexpand\\_Full](#) (const char \*ffull)  
*Restores the bracket notation from an expanded full or HIT tree, that is any tree using only identifiers 'U' 'P' and 'R'.*
- char \* [unweight](#) (const char \*wcoarse)  
*Strip weights from any weighted tree.*
- void [unexpand\\_aligned\\_F](#) (char \*align[2])  
*Converts two aligned structures in expanded notation.*
- void [parse\\_structure](#) (const char \*structure)  
*Collects a statistic of structure elements of the full structure in bracket notation.*
- char \* [pack\\_structure](#) (const char \*struc)  
*Pack secondary secondary structure, 5:1 compression using base 3 encoding.*
- char \* [unpack\\_structure](#) (const char \*packed)  
*Unpack secondary structure previously packed with [pack\\_structure\(\)](#)*
- short \* [make\\_pair\\_table](#) (const char \*structure)  
*Create a pair table of a secondary structure.*
- short \* [copy\\_pair\\_table](#) (const short \*pt)  
*Get an exact copy of a pair table.*
- short \* [alimake\\_pair\\_table](#) (const char \*structure)
- short \* [make\\_pair\\_table\\_snoop](#) (const char \*structure)
- int [bp\\_distance](#) (const char \*str1, const char \*str2)  
*Compute the "base pair" distance between two secondary structures s1 and s2.*

- unsigned int \* [make\\_referenceBP\\_array](#) (short \*reference\_pt, unsigned int turn)  
*Make a reference base pair count matrix.*
- unsigned int \* [compute\\_BPdifferences](#) (short \*pt1, short \*pt2, unsigned int turn)  
*Make a reference base pair distance matrix.*
- void [parenthesis\\_structure](#) (char \*structure, [vrna\\_bp\\_stack\\_t](#) \*bp, int length)  
*Create a dot-bracket/parenthesis structure from backtracking stack.*
- void [parenthesis\\_zuker](#) (char \*structure, [vrna\\_bp\\_stack\\_t](#) \*bp, int length)  
*Create a dot-bracket/parenthesis structure from backtracking stack obtained by Zuker suboptimal calculation in `cofold.c`.*
- void [bppm\\_to\\_structure](#) (char \*structure, [FLT\\_OR\\_DBL](#) \*pr, unsigned int length)  
*Create a dot-bracket like structure string from base pair probability matrix.*
- char [bppm\\_symbol](#) (const float \*x)  
*Get a pseudo dot bracket notation for a given probability information.*

## Variables

- int [loop\\_size](#) [2000]  
*contains a list of all loop sizes. `loop_size[0]` contains the number of external bases.*
- int [helix\\_size](#) [2000]  
*contains a list of all stack sizes.*
- int [loop\\_degree](#) [2000]  
*contains the corresponding list of loop degrees.*
- int [loops](#)  
*contains the number of loops ( and therefore of stacks ).*
- int [unpaired](#)  
*contains the number of unpaired bases.*
- int [pairs](#)  
*contains the number of base pairs in the last parsed structure.*

## 15.68.2 Function Documentation

### 15.68.2.1 b2HIT()

```
char* b2HIT (
    const char * structure )
```

```
#include <ViennaRNA/RNAstruct.h>
```

Converts the full structure from bracket notation to the HIT notation including root.

**Deprecated** See [vrna\\_db\\_to\\_tree\\_string\(\)](#) and [VRNA\\_STRUCTURE\\_TREE\\_HIT](#) for a replacement

#### Parameters

<code>structure</code>	
------------------------	--

## Returns

### 15.68.2.2 b2C()

```
char* b2C (
    const char * structure )
```

```
#include <ViennaRNA/RNAstruct.h>
```

Converts the full structure from bracket notation to the a coarse grained notation using the 'H' 'B' 'I' 'M' and 'R' identifiers.

**Deprecated** See [vrna\\_db\\_to\\_tree\\_string\(\)](#) and [VRNA\\_STRUCTURE\\_TREE\\_SHAPIRO\\_SHORT](#) for a replacement

## Parameters

<i>structure</i>	
------------------	--

## Returns

### 15.68.2.3 b2Shapiro()

```
char* b2Shapiro (
    const char * structure )
```

```
#include <ViennaRNA/RNAstruct.h>
```

Converts the full structure from bracket notation to the *weighted* coarse grained notation using the 'H' 'B' 'I' 'M' 'S' 'E' and 'R' identifiers.

**Deprecated** See [vrna\\_db\\_to\\_tree\\_string\(\)](#) and [VRNA\\_STRUCTURE\\_TREE\\_SHAPIRO\\_WEIGHT](#) for a replacement

## Parameters

<i>structure</i>	
------------------	--

## Returns

### 15.68.2.4 add\_root()

```
char* add_root (
    const char * structure )
```

```
#include <ViennaRNA/RNAstruct.h>
```

Adds a root to an un-rooted tree in any except bracket notation.

## Parameters

<i>structure</i>	
------------------	--

## Returns

### 15.68.2.5 expand\_Shapiro()

```
char* expand_Shapiro (
    const char * coarse )
```

```
#include <ViennaRNA/RNAstruct.h>
```

Inserts missing 'S' identifiers in unweighted coarse grained structures as obtained from [b2C\(\)](#).

## Parameters

<i>coarse</i>	
---------------	--

## Returns

### 15.68.2.6 expand\_Full()

```
char* expand_Full (
    const char * structure )
```

```
#include <ViennaRNA/RNAstruct.h>
```

Convert the full structure from bracket notation to the expanded notation including root.

**Parameters**

<i>structure</i>	
------------------	--

**Returns****15.68.2.7 unexpand\_Full()**

```
char* unexpand_Full (
    const char * ffull )
```

```
#include <ViennaRNA/RNAstruct.h>
```

Restores the bracket notation from an expanded full or HIT tree, that is any tree using only identifiers 'U' 'P' and 'R'.

**Parameters**

<i>ffull</i>	
--------------	--

**Returns****15.68.2.8 unweight()**

```
char* unweight (
    const char * wcoarse )
```

```
#include <ViennaRNA/RNAstruct.h>
```

Strip weights from any weighted tree.

**Parameters**

<i>wcoarse</i>	
----------------	--

**Returns**



### 15.68.2.9 unexpand\_aligned\_F()

```
void unexpand_aligned_F (
    char * align[2] )
```

```
#include <ViennaRNA/RNAstruct.h>
```

Converts two aligned structures in expanded notation.

Takes two aligned structures as produced by [tree\\_edit\\_distance\(\)](#) function back to bracket notation with '\_' as the gap character. The result overwrites the input.

#### Parameters

<i>align</i>	
--------------	--

### 15.68.2.10 parse\_structure()

```
void parse_structure (
    const char * structure )
```

```
#include <ViennaRNA/RNAstruct.h>
```

Collects a statistic of structure elements of the full structure in bracket notation.

The function writes to the following global variables: [loop\\_size](#), [loop\\_degree](#), [helix\\_size](#), [loops](#), [pairs](#), [unpaired](#)

#### Parameters

<i>structure</i>	
------------------	--

#### Returns

### 15.68.2.11 pack\_structure()

```
char* pack_structure (
    const char * struc )
```

```
#include <ViennaRNA/utils/structures.h>
```

Pack secondary secondary structure, 5:1 compression using base 3 encoding.

Returns a binary string encoding of the secondary structure using a 5:1 compression scheme. The string is NULL terminated and can therefore be used with standard string functions such as `strcmp()`. Useful for programs that need to keep many structures in memory.

**Deprecated** Use [vrna\\_db\\_pack\(\)](#) as a replacement

**Parameters**

<i>struct</i>	The secondary structure in dot-bracket notation
---------------	---

**Returns**

The binary encoded structure

**15.68.2.12 unpack\_structure()**

```
char* unpack_structure (
    const char * packed )
```

```
#include <ViennaRNA/utils/structures.h>
```

Unpack secondary structure previously packed with [pack\\_structure\(\)](#)

Translate a compressed binary string produced by [pack\\_structure\(\)](#) back into the familiar dot-bracket notation.

**Deprecated** Use [vrna\\_db\\_unpack\(\)](#) as a replacement

**Parameters**

<i>packed</i>	The binary encoded packed secondary structure
---------------	---

**Returns**

The unpacked secondary structure in dot-bracket notation

**15.68.2.13 make\_pair\_table()**

```
short* make_pair_table (
    const char * structure )
```

```
#include <ViennaRNA/utils/structures.h>
```

Create a pair table of a secondary structure.

Returns a newly allocated table, such that table[i]=j if (i,j) pair or 0 if i is unpaired, table[0] contains the length of the structure.

**Deprecated** Use [vrna\\_ptable\(\)](#) instead

## Parameters

<i>structure</i>	The secondary structure in dot-bracket notation
------------------	---

## Returns

A pointer to the created pair\_table

## 15.68.2.14 copy\_pair\_table()

```
short* copy_pair_table (
    const short * pt )
```

```
#include <ViennaRNA/utils/structures.h>
```

Get an exact copy of a pair table.

**Deprecated** Use `vrna_ptable_copy()` instead

## Parameters

<i>pt</i>	The pair table to be copied
-----------	-----------------------------

## Returns

A pointer to the copy of 'pt'

## 15.68.2.15 alimake\_pair\_table()

```
short* alimake_pair_table (
    const char * structure )
```

```
#include <ViennaRNA/utils/structures.h>
```

Pair table for snoop align

**Deprecated** Use `vrna_pt_alig_get()` instead!

**15.68.2.16 make\_pair\_table\_snoop()**

```
short* make_pair_table_snoop (
    const char * structure )
```

```
#include <ViennaRNA/utils/structures.h>
```

returns a newly allocated table, such that: table[i]=j if (i,j) pair or 0 if i is unpaired, table[0] contains the length of the structure. The special pseudoknotted H/ACA-mRNA structure is taken into account.

**Deprecated** Use [vrna\\_pt\\_snoop\\_get\(\)](#) instead!

**15.68.2.17 bp\_distance()**

```
int bp_distance (
    const char * str1,
    const char * str2 )
```

```
#include <ViennaRNA/utils/structures.h>
```

Compute the "base pair" distance between two secondary structures s1 and s2.

The sequences should have the same length. dist = number of base pairs in one structure but not in the other same as edit distance with open-pair close-pair as move-set

**Deprecated** Use [vrna\\_bp\\_distance](#) instead

**Parameters**

<i>str1</i>	First structure in dot-bracket notation
<i>str2</i>	Second structure in dot-bracket notation

**Returns**

The base pair distance between str1 and str2

**15.68.2.18 make\_referenceBP\_array()**

```
unsigned int* make_referenceBP_array (
    short * reference_pt,
    unsigned int turn )
```

```
#include <ViennaRNA/utils/structures.h>
```

Make a reference base pair count matrix.

Get an upper triangular matrix containing the number of basepairs of a reference structure for each interval [i,j] with i<j. Access it via iidx!!!

**Deprecated** Use `vrna_refBPcnt_matrix()` instead

#### 15.68.2.19 `compute_BPdifferences()`

```
unsigned int* compute_BPdifferences (
    short * pt1,
    short * pt2,
    unsigned int turn )
```

```
#include <ViennaRNA/utils/structures.h>
```

Make a reference base pair distance matrix.

Get an upper triangular matrix containing the base pair distance of two reference structures for each interval  $[i,j]$  with  $i < j$ . Access it via `iindx!!!`

**Deprecated** Use `vrna_refBPdist_matrix()` instead

#### 15.68.2.20 `parenthesis_structure()`

```
void parenthesis_structure (
    char * structure,
    vrna_bp_stack_t * bp,
    int length )
```

```
#include <ViennaRNA/utils/structures.h>
```

Create a dot-bracket/parenthesis structure from backtracking stack.

**Deprecated** use `vrna_parenthesis_structure()` instead

#### Note

This function is threadsafe

### 15.68.2.21 parenthesis\_zuker()

```
void parenthesis_zuker (
    char * structure,
    vrna_bp_stack_t * bp,
    int length )
```

```
#include <ViennaRNA/utils/structures.h>
```

Create a dot-bracket/parenthesis structure from backtracking stack obtained by zucker suboptimal calculation in cofold.c.

**Deprecated** use `vrna_parenthesis_zuker` instead

#### Note

This function is threadsafe

### 15.68.2.22 bppm\_to\_structure()

```
void bppm_to_structure (
    char * structure,
    FLT_OR_DBL * pr,
    unsigned int length )
```

```
#include <ViennaRNA/utils/structures.h>
```

Create a dot-bracket like structure string from base pair probability matrix.

**Deprecated** Use `vrna_db_from_probs()` instead!

### 15.68.2.23 bppm\_symbol()

```
char bppm_symbol (
    const float * x )
```

```
#include <ViennaRNA/utils/structures.h>
```

Get a pseudo dot bracket notation for a given probability information.

**Deprecated** Use `vrna_bpp_symbol()` instead!

## 15.69 Multiple Sequence Alignment Utilities

Functions to extract features from and to manipulate multiple sequence alignments.

### 15.69.1 Detailed Description

Functions to extract features from and to manipulate multiple sequence alignments.

Collaboration diagram for Multiple Sequence Alignment Utilities:

#### Modules

- [Deprecated Interface for Multiple Sequence Alignment Utilities](#)

#### Files

- file [alignments.h](#)  
*Various utility- and helper-functions for sequence alignments and comparative structure prediction.*

#### Data Structures

- struct [vrna\\_pinfo\\_s](#)  
*A base pair info structure. [More...](#)*

#### Macros

- `#define VRNA\_ALN\_DEFAULT 0U`  
*Use default alignment settings.*
- `#define VRNA\_ALN\_RNA 1U`  
*Convert to RNA alphabet.*
- `#define VRNA\_ALN\_DNA 2U`  
*Convert to DNA alphabet.*
- `#define VRNA\_ALN\_UPPERCASE 4U`  
*Convert to uppercase nucleotide letters.*
- `#define VRNA\_ALN\_LOWERCASE 8U`  
*Convert to lowercase nucleotide letters.*
- `#define VRNA\_MEASURE\_SHANNON\_ENTROPY 1U`  
*Flag indicating Shannon Entropy measure.*

#### Typedefs

- typedef struct [vrna\\_pinfo\\_s](#) [vrna\\_pinfo\\_t](#)  
*Typename for the base pair info representing data structure [vrna\\_pinfo\\_s](#).*

## Functions

- int `vrna_aln_mpi` (const char \*\*alignment)  
*Get the mean pairwise identity in steps from ?to?(ident)*
- `vrna_pinfo_t * vrna_aln_pinfo` (`vrna_fold_compound_t *vc`, const char \*structure, double threshold)  
*Retrieve an array of `vrna_pinfo_t` structures from precomputed pair probabilities.*
- char \*\* `vrna_aln_slice` (const char \*\*alignment, unsigned int i, unsigned int j)  
*Slice out a subalignment from a larger alignment.*
- void `vrna_aln_free` (char \*\*alignment)  
*Free memory occupied by a set of aligned sequences.*
- char \*\* `vrna_aln_uppercase` (const char \*\*alignment)  
*Create a copy of an alignment with only uppercase letters in the sequences.*
- char \*\* `vrna_aln_toRNA` (const char \*\*alignment)  
*Create a copy of an alignment where DNA alphabet is replaced by RNA alphabet.*
- char \*\* `vrna_aln_copy` (const char \*\*alignment, unsigned int options)  
*Make a copy of a multiple sequence alignment.*
- float \* `vrna_aln_conservation_struct` (const char \*\*alignment, const char \*structure, const `vrna_md_t *md`)  
*Compute base pair conservation of a consensus structure.*
- float \* `vrna_aln_conservation_col` (const char \*\*alignment, const `vrna_md_t *md_p`, unsigned int options)  
*Compute nucleotide conservation in an alignment.*
- char \* `vrna_aln_consensus_sequence` (const char \*\*alignment, const `vrna_md_t *md_p`)  
*Compute the consensus sequence for a given multiple sequence alignment.*
- char \* `vrna_aln_consensus_mis` (const char \*\*alignment, const `vrna_md_t *md_p`)  
*Compute the Most Informative Sequence (MIS) for a given multiple sequence alignment.*

## 15.69.2 Data Structure Documentation

### 15.69.2.1 struct vrna\_pinfo\_s

A base pair info structure.

For each base pair (i,j) with i,j in [0, n-1] the structure lists:

- its probability 'p'
- an entropy-like measure for its well-definedness 'ent'
- the frequency of each type of pair in 'bp[]'
  - 'bp[0]' contains the number of non-compatible sequences
  - 'bp[1]' the number of CG pairs, etc.

### Data Fields

- unsigned `i`  
*nucleotide position i*
- unsigned `j`  
*nucleotide position j*
- float `p`  
*Probability.*
- float `ent`  
*Pseudo entropy for  $p(i, j) = S_i + S_j - p_{ij} * \ln(p_{ij})$ .*
- short `bp` [8]  
*Frequencies of pair\_types.*
- char `comp`  
*1 iff pair is in mfe structure*



### 15.69.3 Macro Definition Documentation

#### 15.69.3.1 VRNA\_MEASURE\_SHANNON\_ENTROPY

```
#define VRNA_MEASURE_SHANNON_ENTROPY 1U

#include <ViennaRNA/utils/alignments.h>
```

Flag indicating Shannon Entropy measure.

Shannon Entropy is defined as  $H = -\sum_c p_c \cdot \log_2 p_c$

### 15.69.4 Function Documentation

#### 15.69.4.1 vrna\_aln\_mpi()

```
int vrna_aln_mpi (
    const char ** alignment )

#include <ViennaRNA/utils/alignments.h>
```

Get the mean pairwise identity in steps from ?to?(ident)

##### Parameters

<i>alignment</i>	Aligned sequences
------------------	-------------------

##### Returns

The mean pairwise identity

#### 15.69.4.2 vrna\_aln\_pinfo()

```
vrna_pinfo_t* vrna_aln_pinfo (
    vrna_fold_compound_t * vc,
    const char * structure,
    double threshold )

#include <ViennaRNA/utils/alignments.h>
```

Retrieve an array of `vrna_pinfo_t` structures from precomputed pair probabilities.

This array of structures contains information about positionwise pair probabilities, base pair entropy and more

See also

[vrna\\_pinfo\\_t](#), and [vrna\\_pf\(\)](#)

## Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> of type <a href="#">VRNA_FC_TYPE_COMPARATIVE</a> with precomputed partition function matrices
<i>structure</i>	An optional structure in dot-bracket notation (Maybe NULL)
<i>threshold</i>	Do not include results with pair probabilities below threshold

## Returns

The [vrna\\_pinfo\\_t](#) array

15.69.4.3 [vrna\\_aln\\_slice\(\)](#)

```
char** vrna_aln_slice (
    const char ** alignment,
    unsigned int i,
    unsigned int j )

#include <ViennaRNA/utils/alignments.h>
```

Slice out a subalignment from a larger alignment.

## Note

The user is responsible to free the memory occupied by the returned subalignment

## See also

[vrna\\_aln\\_free\(\)](#)

## Parameters

<i>alignment</i>	The input alignment
<i>i</i>	The first column of the subalignment (1-based)
<i>j</i>	The last column of the subalignment (1-based)

## Returns

The subalignment between column *i* and *j*

15.69.4.4 [vrna\\_aln\\_free\(\)](#)

```
void vrna_aln_free (
    char ** alignment )
```

```
#include <ViennaRNA/Utils/alignments.h>
```

Free memory occupied by a set of aligned sequences.

## Parameters

<i>alignment</i>	The input alignment
------------------	---------------------

15.69.4.5 `vrna_aln_uppercase()`

```
char** vrna_aln_uppercase (  
    const char ** alignment )
```

```
#include <ViennaRNA/utils/alignments.h>
```

Create a copy of an alignment with only uppercase letters in the sequences.

## See also

[vrna\\_aln\\_copy](#)

## Parameters

<i>alignment</i>	The input sequence alignment (last entry must be <i>NULL</i> terminated)
------------------	--

## Returns

A copy of the input alignment where lowercase sequence letters are replaced by uppercase letters

15.69.4.6 `vrna_aln_toRNA()`

```
char** vrna_aln_toRNA (  
    const char ** alignment )
```

```
#include <ViennaRNA/utils/alignments.h>
```

Create a copy of an alignment where DNA alphabet is replaced by RNA alphabet.

## See also

[vrna\\_aln\\_copy](#)

## Parameters

<i>alignment</i>	The input sequence alignment (last entry must be <i>NULL</i> terminated)
------------------	--

**Returns**

A copy of the input alignment where DNA alphabet is replaced by RNA alphabet (T -> U)

**15.69.4.7 vrna\_aln\_copy()**

```
char** vrna_aln_copy (
    const char ** alignment,
    unsigned int options )

#include <ViennaRNA/utils/alignments.h>
```

Make a copy of a multiple sequence alignment.

This function allows one to create a copy of a multiple sequence alignment. The `options` parameter additionally allows for sequence manipulation, such as converting DNA to RNA alphabet, and conversion to uppercase letters.

**See also**

[vrna\\_aln\\_copy\(\)](#), [VRNA\\_ALN\\_RNA](#), [VRNA\\_ALN\\_UPPERCASE](#), [VRNA\\_ALN\\_DEFAULT](#)

**Parameters**

<i>alignment</i>	The input sequence alignment (last entry must be <i>NULL</i> terminated)
<i>options</i>	Option flags indicating whether the aligned sequences should be converted

**Returns**

A (manipulated) copy of the input alignment

**15.69.4.8 vrna\_aln\_conservation\_struct()**

```
float * vrna_aln_conservation_struct (
    const char ** alignment,
    const char * structure,
    const vrna_md_t * md )

#include <ViennaRNA/utils/alignments.h>
```

Compute base pair conservation of a consensus structure.

This function computes the base pair conservation (fraction of canonical base pairs) of a consensus structure given a multiple sequence alignment. The base pair types that are considered canonical may be specified using the [vrna\\_md\\_t.pair](#) array. Passing *NULL* as parameter `md` results in default pairing rules, i.e. canonical Watson-Crick and GU Wobble pairs.

## Parameters

<i>alignment</i>	The input sequence alignment (last entry must be <i>NULL</i> terminated)
<i>structure</i>	The consensus structure in dot-bracket notation
<i>md</i>	Model details that specify compatible base pairs (Maybe <i>NULL</i> )

## Returns

A 1-based vector of base pair conservations

**SWIG Wrapper Notes** This function is available in an overloaded form where the last parameter may be omitted, indicating `md = NULL`

15.69.4.9 `vrna_aln_conservation_col()`

```
float * vrna_aln_conservation_col (
    const char ** alignment,
    const vrna_md_t * md,
    unsigned int options )
```

```
#include <ViennaRNA/utils/alignments.h>
```

Compute nucleotide conservation in an alignment.

This function computes the conservation of nucleotides in alignment columns. The simplest measure is Shannon Entropy and can be selected by passing the `VRNA_MEASURE_SHANNON_ENTROPY` flag in the `options` parameter.

## Note

Currently, only `VRNA_MEASURE_SHANNON_ENTROPY` is supported as conservation measure.

## See also

`VRNA_MEASURE_SHANNON_ENTROPY`

## Parameters

<i>alignment</i>	The input sequence alignment (last entry must be <i>NULL</i> terminated)
<i>md</i>	Model details that specify known nucleotides (Maybe <i>NULL</i> )
<i>options</i>	A flag indicating which measure of conservation should be applied

## Returns

A 1-based vector of column conservations

**SWIG Wrapper Notes** This function is available in an overloaded form where the last two parameters may be omitted, indicating `md = NULL`, and `options = VRNA_MEASURE_SHANNON_ENTROPY`,

respectively.

#### 15.69.4.10 `vrna_aln_consensus_sequence()`

```
char* vrna_aln_consensus_sequence (
    const char ** alignment,
    const vrna_md_t * md_p )

#include <ViennaRNA/utils/alignments.h>
```

Compute the consensus sequence for a given multiple sequence alignment.

##### Parameters

<i>alignment</i>	The input sequence alignment (last entry must be <i>NULL</i> terminated)
<i>md_p</i>	Model details that specify known nucleotides (Maybe <i>NULL</i> )

##### Returns

The consensus sequence of the alignment, i.e. the most frequent nucleotide for each alignment column

#### 15.69.4.11 `vrna_aln_consensus_mis()`

```
char* vrna_aln_consensus_mis (
    const char ** alignment,
    const vrna_md_t * md_p )

#include <ViennaRNA/utils/alignments.h>
```

Compute the Most Informative Sequence (MIS) for a given multiple sequence alignment.

The most informative sequence (MIS) [9] displays for each alignment column the nucleotides with frequency greater than the background frequency, projected into IUPAC notation. Columns where gaps are over-represented are in lower case.

##### Parameters

<i>alignment</i>	The input sequence alignment (last entry must be <i>NULL</i> terminated)
<i>md_p</i>	Model details that specify known nucleotides (Maybe <i>NULL</i> )

##### Returns

The most informative sequence for the alignment



## 15.70 Deprecated Interface for Multiple Sequence Alignment Utilities

### 15.70.1 Detailed Description

Collaboration diagram for Deprecated Interface for Multiple Sequence Alignment Utilities:

#### Typedefs

- typedef struct [vrna\\_pinfo\\_s](#) [pair\\_info](#)  
*Old typename of [vrna\\_pinfo\\_s](#).*

#### Functions

- int [get\\_mpi](#) (char \*Aseq[ ], int n\_seq, int length, int \*mini)  
*Get the mean pairwise identity in steps from ?to?(ident)*
- void [encode\\_al\\_i\\_sequence](#) (const char \*sequence, short \*S, short \*s5, short \*s3, char \*ss, unsigned short \*as, int [circ](#))  
*Get arrays with encoded sequence of the alignment.*
- void [alloc\\_sequence\\_arrays](#) (const char \*\*sequences, short \*\*\*S, short \*\*\*S5, short \*\*\*S3, unsigned short \*\*\*a2s, char \*\*\*Ss, int [circ](#))  
*Allocate memory for sequence array used to deal with aligned sequences.*
- void [free\\_sequence\\_arrays](#) (unsigned int n\_seq, short \*\*\*S, short \*\*\*S5, short \*\*\*S3, unsigned short \*\*\*a2s, char \*\*\*Ss)  
*Free the memory of the sequence arrays used to deal with aligned sequences.*

### 15.70.2 Typedef Documentation

#### 15.70.2.1 pair\_info

```
typedef struct vrna\_pinfo\_s pair\_info

#include <ViennaRNA/utils/alignments.h>
```

Old typename of [vrna\\_pinfo\\_s](#).

**Deprecated** Use [vrna\\_pinfo\\_t](#) instead!

### 15.70.3 Function Documentation

15.70.3.1 `get_mpi()`

```
int get_mpi (
    char * Alseq[],
    int n_seq,
    int length,
    int * mini )
```

```
#include <ViennaRNA/utils/alignments.h>
```

Get the mean pairwise identity in steps from ?to?(ident)

**Deprecated** Use `vrna_aln_mpi()` as a replacement

## Parameters

<i>Alseq</i>	
<i>n_seq</i>	The number of sequences in the alignment
<i>length</i>	The length of the alignment
<i>mini</i>	

## Returns

The mean pairwise identity

15.70.3.2 `encode_aln_sequence()`

```
void encode_aln_sequence (
    const char * sequence,
    short * S,
    short * s5,
    short * s3,
    char * ss,
    unsigned short * as,
    int circ )
```

```
#include <ViennaRNA/utils/alignments.h>
```

Get arrays with encoded sequence of the alignment.

this function assumes that in *S*, *S5*, *s3*, *ss* and *as* enough space is already allocated (size must be at least sequence length+2)

## Parameters

<i>sequence</i>	The gapped sequence from the alignment
<i>S</i>	pointer to an array that holds encoded sequence
<i>s5</i>	pointer to an array that holds the next base 5' of alignment position i
<i>s3</i>	pointer to an array that holds the next base 3' of alignment position i
<i>ss</i>	
<i>as</i>	
<i>circ</i>	assume the molecules to be circular instead of linear (circ=0)

**15.70.3.3 alloc\_sequence\_arrays()**

```
void alloc_sequence_arrays (
    const char ** sequences,
    short *** S,
    short *** S5,
    short *** S3,
    unsigned short *** a2s,
    char *** Ss,
    int circ )
```

```
#include <ViennaRNA/utils/alignments.h>
```

Allocate memory for sequence array used to deal with aligned sequences.

Note that these arrays will also be initialized according to the sequence alignment given

See also

[free\\_sequence\\_arrays\(\)](#)

**Parameters**

<i>sequences</i>	The aligned sequences
<i>S</i>	A pointer to the array of encoded sequences
<i>S5</i>	A pointer to the array that contains the next 5' nucleotide of a sequence position
<i>S3</i>	A pointer to the array that contains the next 3' nucleotide of a sequence position
<i>a2s</i>	A pointer to the array that contains the alignment to sequence position mapping
<i>Ss</i>	A pointer to the array that contains the ungapped sequence
<i>circ</i>	assume the molecules to be circular instead of linear (circ=0)

**15.70.3.4 free\_sequence\_arrays()**

```
void free_sequence_arrays (
    unsigned int n_seq,
    short *** S,
    short *** S5,
    short *** S3,
    unsigned short *** a2s,
    char *** Ss )
```

```
#include <ViennaRNA/utils/alignments.h>
```

Free the memory of the sequence arrays used to deal with aligned sequences.

This function frees the memory previously allocated with [alloc\\_sequence\\_arrays\(\)](#)

See also

[alloc\\_sequence\\_arrays\(\)](#)

## Parameters

<i>n_seq</i>	The number of aligned sequences
<i>S</i>	A pointer to the array of encoded sequences
<i>S5</i>	A pointer to the array that contains the next 5' nucleotide of a sequence position
<i>S3</i>	A pointer to the array that contains the next 3' nucleotide of a sequence position
<i>a2s</i>	A pointer to the array that contains the alignment to sequence position mapping
<i>Ss</i>	A pointer to the array that contains the ungapped sequence

## 15.71 Files and I/O

Functions to parse, write, and convert various file formats and to deal with file system related issues.

### 15.71.1 Detailed Description

Functions to parse, write, and convert various file formats and to deal with file system related issues.

Collaboration diagram for Files and I/O:

#### Modules

- [Nucleic Acid Sequences and Structures](#)  
*Functions to read/write different file formats for nucleic acid sequences and secondary structures.*
- [Multiple Sequence Alignments](#)  
*Functions to read/write multiple sequence alignments (MSA) in various file formats.*
- [Command Files](#)  
*Functions to parse and interpret the content of [Command Files](#).*

#### Files

- file [commands.h](#)  
*Parse and apply different commands that alter the behavior of secondary structure prediction and evaluation.*
- file [ribo.h](#)  
*Parse RiboSum Scoring Matrices for Covariance Scoring of Alignments.*
- file [file\\_formats.h](#)  
*Read and write different file formats for RNA sequences, structures.*
- file [file\\_formats\\_msa.h](#)  
*Functions dealing with file formats for Multiple Sequence Alignments (MSA)*
- file [utils.h](#)  
*Several utilities for file handling.*

#### Functions

- float \*\* [get\\_ribosum](#) (const char \*\*Aseq, int n\_seq, int length)  
*Retrieve a RiboSum Scoring Matrix for a given Alignment.*
- float \*\* [readribosum](#) (char \*name)  
*Read a RiboSum or other user-defined Scoring Matrix and Store into global Memory.*
- void [vrna\\_file\\_copy](#) (FILE \*from, FILE \*to)  
*Inefficient 'cp'.*
- char \* [vrna\\_read\\_line](#) (FILE \*fp)  
*Read a line of arbitrary length from a stream.*
- int [vrna\\_mkdir\\_p](#) (const char \*path)  
*Recursively create a directory tree.*
- char \* [vrna\\_basename](#) (const char \*path)  
*Extract the filename from a file path.*
- char \* [vrna\\_dirname](#) (const char \*path)  
*Extract the directory part of a file path.*
- char \* [vrna\\_filename\\_sanitize](#) (const char \*name, const char \*replacement)  
*Sanitize a file name.*
- int [vrna\\_file\\_exists](#) (const char \*filename)  
*Check if a file already exists in the file system.*

## 15.71.2 Function Documentation

### 15.71.2.1 `vrna_read_line()`

```
char* vrna_read_line (
    FILE * fp )
```

```
#include <ViennaRNA/io/utils.h>
```

Read a line of arbitrary length from a stream.

Returns a pointer to the resulting string. The necessary memory is allocated and should be released using `free()` when the string is no longer needed.

#### Parameters

<i>fp</i>	A file pointer to the stream where the function should read from
-----------	--

#### Returns

A pointer to the resulting string

### 15.71.2.2 `vrna_filename_sanitize()`

```
char* vrna_filename_sanitize (
    const char * name,
    const char * replacement )
```

```
#include <ViennaRNA/io/utils.h>
```

Sanitize a file name.

Returns a new file name where all invalid characters are substituted by a replacement character. If no replacement character is supplied, invalid characters are simply removed from the filename. File names may also never exceed a length of 255 characters. Longer file names will undergo a 'smart' truncation process, where the filenames' suffix, i.e. everything after the last dot '.', is attempted to be kept intact. Hence, only the filename part before the suffix is reduced in such a way that the total filename complies to the length restriction of 255 characters. If no suffix is present or the suffix itself already exceeds the maximum length, the filename is simply truncated from the back of the string.

For now we consider the following characters invalid:

- backslash '\'
- slash '/'
- question mark '?'

- percent sign `%`
- asterisk `*`
- colon `:`
- pipe symbol `|`
- double quote `"`
- triangular brackets `<` and `>`

Furthermore, the (resulting) file name must not be a reserved file name, such as:

- `.`
- `..`

#### Note

This function allocates a new block of memory for the sanitized string. It also may return (a) NULL if the input is pointing to NULL, or (b) an empty string if the input only consists of invalid characters which are simply removed!

#### Parameters

<i>name</i>	The input file name
<i>replacement</i>	The replacement character, or NULL

#### Returns

The sanitized file name, or NULL

#### 15.71.2.3 `vrna_file_exists()`

```
int vrna_file_exists (
    const char * filename )
```

```
#include <ViennaRNA/io/utils.h>
```

Check if a file already exists in the file system.

#### Parameters

<i>filename</i>	The name of (path to) the file to check for existence
-----------------	---

#### Returns

0 if it doesn't exist, 1 otherwise

## 15.72 Nucleic Acid Sequences and Structures

Functions to read/write different file formats for nucleic acid sequences and secondary structures.

### 15.72.1 Detailed Description

Functions to read/write different file formats for nucleic acid sequences and secondary structures.

Collaboration diagram for Nucleic Acid Sequences and Structures:

#### Files

- file [file\\_formats.h](#)  
*Read and write different file formats for RNA sequences, structures.*

#### Macros

- `#define VRNA_OPTION_MULTILINE 32U`  
*Tell a function that an input is assumed to span several lines.*
- `#define VRNA_CONSTRAINT_MULTILINE 32U`  
*parse multiline constraint*

#### Functions

- void [vrna\\_file\\_helixlist](#) (const char \*seq, const char \*db, float energy, FILE \*file)  
*Print a secondary structure as helix list.*
- void [vrna\\_file\\_connect](#) (const char \*seq, const char \*db, float energy, const char \*identifier, FILE \*file)  
*Print a secondary structure as connect table.*
- void [vrna\\_file\\_bpseq](#) (const char \*seq, const char \*db, FILE \*file)  
*Print a secondary structure in bpseq format.*
- void [vrna\\_file\\_json](#) (const char \*seq, const char \*db, double energy, const char \*identifier, FILE \*file)  
*Print a secondary structure in jsonformat.*
- unsigned int [vrna\\_file\\_fasta\\_read\\_record](#) (char \*\*header, char \*\*sequence, char \*\*\*rest, FILE \*file, unsigned int options)  
*Get a (fasta) data set from a file or stdin.*
- char \* [vrna\\_extract\\_record\\_rest\\_structure](#) (const char \*\*lines, unsigned int length, unsigned int option)  
*Extract a dot-bracket structure string from (multiline)character array.*
- int [vrna\\_file\\_SHAPE\\_read](#) (const char \*file\_name, int length, double default\_value, char \*sequence, double \*values)  
*Read data from a given SHAPE reactivity input file.*
- void [vrna\\_extract\\_record\\_rest\\_constraint](#) (char \*\*cstruc, const char \*\*lines, unsigned int option)  
*Extract a hard constraint encoded as pseudo dot-bracket string.*
- unsigned int [read\\_record](#) (char \*\*header, char \*\*sequence, char \*\*\*rest, unsigned int options)  
*Get a data record from stdin.*



## 15.72.2 Macro Definition Documentation

### 15.72.2.1 VRNA\_OPTION\_MULTILINE

```
#define VRNA_OPTION_MULTILINE 32U
```

```
#include <ViennaRNA/io/file_formats.h>
```

Tell a function that an input is assumed to span several lines.

If used as input-option a function might also be returning this state telling that it has read data from multiple lines.

See also

[vrna\\_extract\\_record\\_rest\\_structure\(\)](#), [vrna\\_file\\_fasta\\_read\\_record\(\)](#)

### 15.72.2.2 VRNA\_CONSTRAINT\_MULTILINE

```
#define VRNA_CONSTRAINT_MULTILINE 32U
```

```
#include <ViennaRNA/io/file_formats.h>
```

parse multiline constraint

**Deprecated** see [vrna\\_extract\\_record\\_rest\\_structure\(\)](#)

## 15.72.3 Function Documentation

### 15.72.3.1 vrna\_file\_helixlist()

```
void vrna_file_helixlist (
    const char * seq,
    const char * db,
    float energy,
    FILE * file )
```

```
#include <ViennaRNA/io/file_formats.h>
```

Print a secondary structure as helix list.

## Parameters

<i>seq</i>	The RNA sequence
<i>db</i>	The structure in dot-bracket format
<i>energy</i>	Free energy of the structure in kcal/mol
<i>file</i>	The file handle used to print to (print defaults to 'stdout' if(file == NULL) )

15.72.3.2 `vrna_file_connect()`

```
void vrna_file_connect (
    const char * seq,
    const char * db,
    float energy,
    const char * identifier,
    FILE * file )

#include <ViennaRNA/io/file_formats.h>
```

Print a secondary structure as connect table.

Connect table file format looks like this:

```
300 ENERGY = 7.0 example
 1 G      0   2   22   1
 2 G      1   3   21   2
```

where the headerline is followed by 6 columns with:

1. Base number: index n
2. Base (A, C, G, T, U, X)
3. Index n-1 (0 if first nucleotide)
4. Index n+1 (0 if last nucleotide)
5. Number of the base to which n is paired. No pairing is indicated by 0 (zero).
6. Natural numbering.

## Parameters

<i>seq</i>	The RNA sequence
<i>db</i>	The structure in dot-bracket format
<i>energy</i>	The free energy of the structure
<i>identifier</i>	An optional identifier for the sequence
<i>file</i>	The file handle used to print to (print defaults to 'stdout' if(file == NULL) )

### 15.72.3.3 vrna\_file\_bpseq()

```
void vrna_file_bpseq (
    const char * seq,
    const char * db,
    FILE * file )

#include <ViennaRNA/io/file_formats.h>
```

Print a secondary structure in bpseq format.

#### Parameters

<i>seq</i>	The RNA sequence
<i>db</i>	The structure in dot-bracket format
<i>file</i>	The file handle used to print to (print defaults to 'stdout' if(file == NULL) )

### 15.72.3.4 vrna\_file\_json()

```
void vrna_file_json (
    const char * seq,
    const char * db,
    double energy,
    const char * identifier,
    FILE * file )

#include <ViennaRNA/io/file_formats.h>
```

Print a secondary structure in jsonformat.

#### Parameters

<i>seq</i>	The RNA sequence
<i>db</i>	The structure in dot-bracket format
<i>energy</i>	The free energy
<i>identifier</i>	An identifier for the sequence
<i>file</i>	The file handle used to print to (print defaults to 'stdout' if(file == NULL) )

### 15.72.3.5 vrna\_file\_fasta\_read\_record()

```
unsigned int vrna_file_fasta_read_record (
    char ** header,
    char ** sequence,
    char *** rest,
    FILE * file,
    unsigned int options )
```

```
#include <ViennaRNA/io/file_formats.h>
```

Get a (fasta) data set from a file or stdin.

This function may be used to obtain complete datasets from a filehandle or stdin. A dataset is always defined to contain at least a sequence. If data starts with a fasta header, i.e. a line like

```
>some header info
```

then `vrna_file_fasta_read_record()` will assume that the sequence that follows the header may span over several lines. To disable this behavior and to assign a single line to the argument 'sequence' one can pass `VRNA_INPUT_NO_SPAN` in the 'options' argument. If no fasta header is read in the beginning of a data block, a sequence must not span over multiple lines!

Unless the options `VRNA_INPUT_NOSKIP_COMMENTS` or `VRNA_INPUT_NOSKIP_BLANK_LINES` are passed, a sequence may be interrupted by lines starting with a comment character or empty lines.

A sequence is regarded as completely read if it was either assumed to not span over multiple lines, a secondary structure or structure constraint follows the sequence on the next line, or a new header marks the beginning of a new sequence...

All lines following the sequence (this includes comments) that do not initiate a new dataset according to the above definition are available through the line-array 'rest'. Here one can usually find the structure constraint or other information belonging to the current dataset. Filling of 'rest' may be prevented by passing `VRNA_INPUT_NO_REST` to the options argument.

#### Note

This function will exit any program with an error message if no sequence could be read!

This function is NOT threadsafe! It uses a global variable to store information about the next data block.

The main purpose of this function is to be able to easily parse blocks of data in the header of a loop where all calculations for the appropriate data is done inside the loop. The loop may be then left on certain return values, e.g.:

```
char *id, *seq, **rest;
int i;
id = seq = NULL;
rest = NULL;
while(!(vrna_file_fasta_read_record(&id, &seq, &rest, NULL, 0) & (
    VRNA_INPUT_ERROR | VRNA_INPUT_QUIT))){
    if(id) printf("%s\n", id);
    printf("%s\n", seq);
    if(rest)
        for(i=0;rest[i];i++){
            printf("%s\n", rest[i]);
            free(rest[i]);
        }
    free(rest);
    free(seq);
    free(id);
}
```

In the example above, the while loop will be terminated when `vrna_file_fasta_read_record()` returns either an error, EOF, or a user initiated quit request.

As long as data is read from stdin (we are passing NULL as the file pointer), the id is printed if it is available for the current block of data. The sequence will be printed in any case and if some more lines belong to the current block of data each line will be printed as well.

#### Note

Do not forget to free the memory occupied by header, sequence and rest!

## Parameters

<i>header</i>	A pointer which will be set such that it points to the header of the record
<i>sequence</i>	A pointer which will be set such that it points to the sequence of the record
<i>rest</i>	A pointer which will be set such that it points to an array of lines which also belong to the record
<i>file</i>	A file handle to read from (if NULL, this function reads from stdin)
<i>options</i>	Some options which may be passed to alter the behavior of the function, use 0 for no options

## Returns

A flag with information about what the function actually did read

15.72.3.6 `vrna_extract_record_rest_structure()`

```
char* vrna_extract_record_rest_structure (
    const char ** lines,
    unsigned int length,
    unsigned int option )

#include <ViennaRNA/io/file_formats.h>
```

Extract a dot-bracket structure string from (multiline)character array.

This function extracts a dot-bracket structure string from the 'rest' array as returned by [vrna\\_file\\_fasta\\_read\\_record\(\)](#) and returns it. All occurrences of comments within the 'lines' array will be skipped as long as they do not break the structure string. If no structure could be read, this function returns NULL.

## Precondition

The argument 'lines' has to be a 2-dimensional character array as obtained by [vrna\\_file\\_fasta\\_read\\_record\(\)](#)

## See also

[vrna\\_file\\_fasta\\_read\\_record\(\)](#)

## Parameters

<i>lines</i>	The (multiline) character array to be parsed
<i>length</i>	The assumed length of the dot-bracket string (passing a value < 1 results in no length limit)
<i>option</i>	Some options which may be passed to alter the behavior of the function, use 0 for no options

## Returns

The dot-bracket string read from lines or NULL

15.72.3.7 `vrna_file_SHAPE_read()`

```
int vrna_file_SHAPE_read (
    const char * file_name,
    int length,
    double default_value,
    char * sequence,
    double * values )
```

```
#include <ViennaRNA/io/file_formats.h>
```

Read data from a given SHAPE reactivity input file.

This function parses the informations from a given file and stores the result in the preallocated string sequence and the double array values.

## Parameters

<i>file_name</i>	Path to the constraints file
<i>length</i>	Length of the sequence (file entries exceeding this limit will cause an error)
<i>default_value</i>	Value for missing indices
<i>sequence</i>	Pointer to an array used for storing the sequence obtained from the SHAPE reactivity file
<i>values</i>	Pointer to an array used for storing the values obtained from the SHAPE reactivity file

15.72.3.8 `vrna_extract_record_rest_constraint()`

```
void vrna_extract_record_rest_constraint (
    char ** cstruc,
    const char ** lines,
    unsigned int option )
```

```
#include <ViennaRNA/io/file_formats.h>
```

Extract a hard constraint encoded as pseudo dot-bracket string.

**Deprecated** Use `vrna_extract_record_rest_structure()` instead!

## Precondition

The argument 'lines' has to be a 2-dimensional character array as obtained by `vrna_file_fasta_read_record()`

## See also

`vrna_file_fasta_read_record()`, `VRNA_CONSTRAINT_DB_PIPE`, `VRNA_CONSTRAINT_DB_DOT`, `VRNA_CONSTRAINT_DB_`  
`VRNA_CONSTRAINT_DB_ANG_BRACK`, `VRNA_CONSTRAINT_DB_RND_BRACK`

## Parameters

<i>cstruc</i>	A pointer to a character array that is used as pseudo dot-bracket output
<i>lines</i>	A 2-dimensional character array with the extension lines from the FASTA input
<i>option</i>	The option flags that define the behavior and recognition pattern of this function

## 15.72.3.9 read\_record()

```
unsigned int read_record (
    char ** header,
    char ** sequence,
    char *** rest,
    unsigned int options )
```

```
#include <ViennaRNA/io/file_formats.h>
```

Get a data record from stdin.

**Deprecated** This function is deprecated! Use `vrna_file_fasta_read_record()` as a replacment.

## 15.73 Multiple Sequence Alignments

Functions to read/write multiple sequence alignments (MSA) in various file formats.

### 15.73.1 Detailed Description

Functions to read/write multiple sequence alignments (MSA) in various file formats.

Collaboration diagram for Multiple Sequence Alignments:

#### Files

- file [file\\_formats\\_msa.h](#)  
*Functions dealing with file formats for Multiple Sequence Alignments (MSA)*

#### Macros

- `#define VRNA_FILE_FORMAT_MSA_CLUSTAL 1U`  
*Option flag indicating ClustalW formatted files.*
- `#define VRNA_FILE_FORMAT_MSA_STOCKHOLM 2U`  
*Option flag indicating Stockholm 1.0 formatted files.*
- `#define VRNA_FILE_FORMAT_MSA_FASTA 4U`  
*Option flag indicating FASTA (Pearson) formatted files.*
- `#define VRNA_FILE_FORMAT_MSA_MAF 8U`  
*Option flag indicating MAF formatted files.*
- `#define VRNA_FILE_FORMAT_MSA_MIS 16U`  
*Option flag indicating most informative sequence (MIS) output.*
- `#define VRNA_FILE_FORMAT_MSA_DEFAULT`  
*Option flag indicating the set of default file formats.*
- `#define VRNA_FILE_FORMAT_MSA_NOCHECK 4096U`  
*Option flag to disable validation of the alignment.*
- `#define VRNA_FILE_FORMAT_MSA_UNKNOWN 8192U`  
*Return flag of `vrna_file_msa_detect_format()` to indicate unknown or malformed alignment.*
- `#define VRNA_FILE_FORMAT_MSA_APPEND 16384U`  
*Option flag indicating to append data to a multiple sequence alignment file rather than overwriting it.*
- `#define VRNA_FILE_FORMAT_MSA_QUIET 32768U`  
*Option flag to suppress unnecessary spam messages on `stderr`*
- `#define VRNA_FILE_FORMAT_MSA_SILENT 65536U`  
*Option flag to completely silence any warnings on `stderr`*



## Functions

- int [vrna\\_file\\_msa\\_read](#) (const char \*filename, char \*\*\*names, char \*\*\*aln, char \*\*id, char \*\*structure, unsigned int options)  
*Read a multiple sequence alignment from file.*
- int [vrna\\_file\\_msa\\_read\\_record](#) (FILE \*fp, char \*\*\*names, char \*\*\*aln, char \*\*id, char \*\*structure, unsigned int options)  
*Read a multiple sequence alignment from file handle.*
- unsigned int [vrna\\_file\\_msa\\_detect\\_format](#) (const char \*filename, unsigned int options)  
*Detect the format of a multiple sequence alignment file.*
- int [vrna\\_file\\_msa\\_write](#) (const char \*filename, const char \*\*names, const char \*\*aln, const char \*id, const char \*structure, const char \*source, unsigned int options)  
*Write multiple sequence alignment file.*

### 15.73.2 Macro Definition Documentation

#### 15.73.2.1 VRNA\_FILE\_FORMAT\_MSA\_CLUSTAL

```
#define VRNA_FILE_FORMAT_MSA_CLUSTAL 1U  
  
#include <ViennaRNA/io/file_formats_msa.h>
```

Option flag indicating ClustalW formatted files.

See also

[vrna\\_file\\_msa\\_read\(\)](#), [vrna\\_file\\_msa\\_read\\_record\(\)](#), [vrna\\_file\\_msa\\_detect\\_format\(\)](#)

#### 15.73.2.2 VRNA\_FILE\_FORMAT\_MSA\_STOCKHOLM

```
#define VRNA_FILE_FORMAT_MSA_STOCKHOLM 2U  
  
#include <ViennaRNA/io/file_formats_msa.h>
```

Option flag indicating Stockholm 1.0 formatted files.

See also

[vrna\\_file\\_msa\\_read\(\)](#), [vrna\\_file\\_msa\\_read\\_record\(\)](#), [vrna\\_file\\_msa\\_detect\\_format\(\)](#)

### 15.73.2.3 VRNA\_FILE\_FORMAT\_MSA\_FASTA

```
#define VRNA_FILE_FORMAT_MSA_FASTA 4U

#include <ViennaRNA/io/file_formats_msa.h>
```

Option flag indicating FASTA (Pearson) formatted files.

See also

[vrna\\_file\\_msa\\_read\(\)](#), [vrna\\_file\\_msa\\_read\\_record\(\)](#), [vrna\\_file\\_msa\\_detect\\_format\(\)](#)

### 15.73.2.4 VRNA\_FILE\_FORMAT\_MSA\_MAF

```
#define VRNA_FILE_FORMAT_MSA_MAF 8U

#include <ViennaRNA/io/file_formats_msa.h>
```

Option flag indicating MAF formatted files.

See also

[vrna\\_file\\_msa\\_read\(\)](#), [vrna\\_file\\_msa\\_read\\_record\(\)](#), [vrna\\_file\\_msa\\_detect\\_format\(\)](#)

### 15.73.2.5 VRNA\_FILE\_FORMAT\_MSA\_MIS

```
#define VRNA_FILE_FORMAT_MSA_MIS 16U

#include <ViennaRNA/io/file_formats_msa.h>
```

Option flag indicating most informative sequence (MIS) output.

The default reference sequence output for an alignment is simply a consensus sequence. This flag allows to write the most informative sequence (MIS) instead.

See also

[vrna\\_file\\_msa\\_write\(\)](#)

**15.73.2.6 VRNA\_FILE\_FORMAT\_MSA\_DEFAULT**

```
#define VRNA_FILE_FORMAT_MSA_DEFAULT

#include <ViennaRNA/io/file_formats_msa.h>
```

**Value:**

```
( \
  VRNA_FILE_FORMAT_MSA_CLUSTAL \
  | VRNA_FILE_FORMAT_MSA_STOCKHOLM \
  | VRNA_FILE_FORMAT_MSA_FASTA \
  | VRNA_FILE_FORMAT_MSA_MAF \
)
```

Option flag indicating the set of default file formats.

**See also**

[vrna\\_file\\_msa\\_read\(\)](#), [vrna\\_file\\_msa\\_read\\_record\(\)](#), [vrna\\_file\\_msa\\_detect\\_format\(\)](#)

**15.73.2.7 VRNA\_FILE\_FORMAT\_MSA\_NOCHECK**

```
#define VRNA_FILE_FORMAT_MSA_NOCHECK 4096U

#include <ViennaRNA/io/file_formats_msa.h>
```

Option flag to disable validation of the alignment.

**See also**

[vrna\\_file\\_msa\\_read\(\)](#), [vrna\\_file\\_msa\\_read\\_record\(\)](#)

**15.73.2.8 VRNA\_FILE\_FORMAT\_MSA\_UNKNOWN**

```
#define VRNA_FILE_FORMAT_MSA_UNKNOWN 8192U

#include <ViennaRNA/io/file_formats_msa.h>
```

Return flag of [vrna\\_file\\_msa\\_detect\\_format\(\)](#) to indicate unknown or malformed alignment.

**See also**

[vrna\\_file\\_msa\\_detect\\_format\(\)](#)

#### 15.73.2.9 VRNA\_FILE\_FORMAT\_MSA\_APPEND

```
#define VRNA_FILE_FORMAT_MSA_APPEND 16384U

#include <ViennaRNA/io/file_formats_msa.h>
```

Option flag indicating to append data to a multiple sequence alignment file rather than overwriting it.

See also

[vrna\\_file\\_msa\\_write\(\)](#)

#### 15.73.2.10 VRNA\_FILE\_FORMAT\_MSA\_QUIET

```
#define VRNA_FILE_FORMAT_MSA_QUIET 32768U

#include <ViennaRNA/io/file_formats_msa.h>
```

Option flag to suppress unnecessary spam messages on `stderr`

See also

[vrna\\_file\\_msa\\_read\(\)](#), [vrna\\_file\\_msa\\_read\\_record\(\)](#)

#### 15.73.2.11 VRNA\_FILE\_FORMAT\_MSA\_SILENT

```
#define VRNA_FILE_FORMAT_MSA_SILENT 65536U

#include <ViennaRNA/io/file_formats_msa.h>
```

Option flag to completely silence any warnings on `stderr`

See also

[vrna\\_file\\_msa\\_read\(\)](#), [vrna\\_file\\_msa\\_read\\_record\(\)](#)

### 15.73.3 Function Documentation

15.73.3.1 `vrna_file_msa_read()`

```
vrna_file_msa_read (
    const char * filename,
    char *** names,
    char *** aln,
    char ** id,
    char ** structure,
    unsigned int options )

#include <ViennaRNA/io/file_formats_msa.h>
```

Read a multiple sequence alignment from file.

This function reads the (first) multiple sequence alignment from an input file. The read alignment is split into the sequence id/name part and the actual sequence information and stored in memory as arrays of ids/names and sequences. If the alignment file format allows for additional information, such as an ID of the entire alignment or consensus structure information, this data is retrieved as well and made available. The `options` parameter allows to specify the set of alignment file formats that should be used to retrieve the data. If 0 is passed as option, the list of alignment file formats defaults to `VRNA_FILE_FORMAT_MSA_DEFAULT`.

Currently, the list of parsable multiple sequence alignment file formats consists of:

- [ClustalW format](#)
- [Stockholm 1.0 format](#)
- [FASTA \(Pearson\) format](#)
- [MAF format](#)

**Note**

After successfully reading an alignment, this function performs a validation of the data that includes uniqueness of the sequence identifiers, and equal sequence lengths. This check can be deactivated by passing `VRNA_FILE_FORMAT_MSA_NOCHECK` in the `options` parameter.

It is the users responsibility to free any memory occupied by the output arguments `names`, `aln`, `id`, and `structure` after calling this function. The function automatically sets the latter two arguments to `NULL` in case no corresponding data could be retrieved from the input alignment.

**See also**

[vrna\\_file\\_msa\\_read\\_record\(\)](#), [VRNA\\_FILE\\_FORMAT\\_MSA\\_CLUSTAL](#), [VRNA\\_FILE\\_FORMAT\\_MSA\\_STOCKHOLM](#), [VRNA\\_FILE\\_FORMAT\\_MSA\\_FASTA](#), [VRNA\\_FILE\\_FORMAT\\_MSA\\_MAF](#), [VRNA\\_FILE\\_FORMAT\\_MSA\\_DEFAULT](#), [VRNA\\_FILE\\_FORMAT\\_MSA\\_NOCHECK](#)

**Parameters**

<i>filename</i>	The name of input file that contains the alignment
<i>names</i>	An address to the pointer where sequence identifiers should be written to
<i>aln</i>	An address to the pointer where aligned sequences should be written to
<i>id</i>	An address to the pointer where the alignment ID should be written to (Maybe NULL)
<i>structure</i>	An address to the pointer where consensus structure information should be written to (Maybe NULL)
<i>options</i>	Options to manipulate the behavior of this function

**Returns**

The number of sequences in the alignment, or -1 if no alignment record could be found

**SWIG Wrapper Notes** In the target scripting language, only the first and last argument, `filename` and `options`, are passed to the corresponding function. The other arguments, which serve as output in the C-library, are available as additional return values. Hence, a function call in python may look like this:

```
num_seq, names, aln, id, structure = RNA.file_msa_read("msa.stk", RNA.FILE_FORMAT_MSA_STOCKHOLM)
```

After successfully reading the first record, the variable `num_seq` contains the number of sequences in the alignment (the actual return value of the C-function), while the variables `names`, `aln`, `id`, and `structure` are lists of the sequence names and aligned sequences, as well as strings holding the alignment ID and the structure as stated in the `SS_cons` line, respectively. Note, the last two return values may be empty strings in case the alignment does not provide the required data.

This function exists as an overloaded version where the `options` parameter may be omitted! In that case, the `options` parameter defaults to [VRNA\\_FILE\\_FORMAT\\_MSA\\_STOCKHOLM](#).

**15.73.3.2 vrna\_file\_msa\_read\_record()**

```
vrna_file_msa_read_record (
    FILE * fp,
    char *** names,
    char *** aln,
    char ** id,
    char ** structure,
    unsigned int options )

#include <ViennaRNA/io/file_formats_msa.h>
```

Read a multiple sequence alignment from file handle.

Similar to [vrna\\_file\\_msa\\_read\(\)](#), this function reads a multiple sequence alignment from an input file handle. Since using a file handle, this function is not limited to the first alignment record, but allows for looping over all alignments within the input.

The read alignment is split into the sequence id/name part and the actual sequence information and stored in memory as arrays of ids/names and sequences. If the alignment file format allows for additional information, such as an ID of the entire alignment or consensus structure information, this data is retrieved as well and made available. The `options` parameter allows to specify the alignment file format used to retrieve the data. A single format must be specified here, see [vrna\\_file\\_msa\\_detect\\_format\(\)](#) for helping to determine the correct MSA file format.

Currently, the list of parsable multiple sequence alignment file formats consists of:

- [ClustalW format](#)
- [Stockholm 1.0 format](#)
- [FASTA \(Pearson\) format](#)
- [MAF format](#)

**Note**

After successfully reading an alignment, this function performs a validation of the data that includes uniqueness of the sequence identifiers, and equal sequence lengths. This check can be deactivated by passing [VRNA\\_FILE\\_FORMAT\\_MSA\\_NOCHECK](#) in the `options` parameter.

It is the users responsibility to free any memory occupied by the output arguments `names`, `aln`, `id`, and `structure` after calling this function. The function automatically sets the latter two arguments to `NULL` in case no corresponding data could be retrieved from the input alignment.

**See also**

[vrna\\_file\\_msa\\_read\(\)](#), [vrna\\_file\\_msa\\_detect\\_format\(\)](#), [VRNA\\_FILE\\_FORMAT\\_MSA\\_CLUSTAL](#), [VRNA\\_FILE\\_FORMAT\\_MSA\\_FASTA](#), [VRNA\\_FILE\\_FORMAT\\_MSA\\_MAF](#), [VRNA\\_FILE\\_FORMAT\\_MSA\\_DEFAULT](#), [VRNA\\_FILE\\_FORMAT\\_MSA\\_NOCHECK](#)

**Parameters**

<i>fp</i>	The file pointer the data will be retrieved from
<i>names</i>	An address to the pointer where sequence identifiers should be written to
<i>aln</i>	An address to the pointer where aligned sequences should be written to
<i>id</i>	An address to the pointer where the alignment ID should be written to (Maybe NULL)
<i>structure</i>	An address to the pointer where consensus structure information should be written to (Maybe NULL)
<i>options</i>	Options to manipulate the behavior of this function

**Returns**

The number of sequences in the alignment, or -1 if no alignment record could be found

**SWIG Wrapper Notes** In the target scripting language, only the first and last argument, `fp` and `options`, are passed to the corresponding function. The other arguments, which serve as output in the C-library, are available as additional return values. Hence, a function call in python may look like this:

```
f = open('msa.stk', 'r')
num_seq, names, aln, id, structure = RNA.file_msa_read_record(f, RNA.FILE_FORMAT_MSA_STOCKHOLM)
f.close()
```

After successfully reading the first record, the variable `num_seq` contains the number of sequences in the alignment (the actual return value of the C-function), while the variables `names`, `aln`, `id`, and `structure` are lists of the sequence names and aligned sequences, as well as strings holding the alignment ID and the structure as stated in the `SS_cons` line, respectively. Note, the last two return values may be empty strings in case the alignment does not provide the required data.

This function exists as an overloaded version where the `options` parameter may be omitted! In that case, the `options` parameter defaults to [VRNA\\_FILE\\_FORMAT\\_MSA\\_STOCKHOLM](#).

**15.73.3.3 vrna\_file\_msa\_detect\_format()**

```
vrna_file_msa_detect_format (
    const char * filename,
    unsigned int options )
```

```
#include <ViennaRNA/io/file_formats_msa.h>
```

Detect the format of a multiple sequence alignment file.

This function attempts to determine the format of a file that supposedly contains a multiple sequence alignment (MSA). This is useful in cases where a MSA file contains more than a single record and therefore [vrna\\_file\\_msa\\_read\(\)](#) can not be applied, since it only retrieves the first. Here, one can try to guess the correct file format using this function and then loop over the file, record by record using one of the low-level record retrieval functions for the corresponding MSA file format.

#### Note

This function parses the entire first record within the specified file. As a result, it returns [VRNA\\_FILE\\_FORMAT\\_MSA\\_UNKNOWN](#) not only if it can't detect the file's format, but also in cases where the file doesn't contain sequences!

#### See also

[vrna\\_file\\_msa\\_read\(\)](#), [vrna\\_file\\_stockholm\\_read\\_record\(\)](#), [vrna\\_file\\_clustal\\_read\\_record\(\)](#), [vrna\\_file\\_fasta\\_read\\_record\(\)](#)

#### Parameters

<i>filename</i>	The name of input file that contains the alignment
<i>options</i>	Options to manipulate the behavior of this function

#### Returns

The MSA file format, or [VRNA\\_FILE\\_FORMAT\\_MSA\\_UNKNOWN](#)

**SWIG Wrapper Notes** This function exists as an overloaded version where the `options` parameter may be omitted! In that case, the `options` parameter defaults to [VRNA\\_FILE\\_FORMAT\\_MSA\\_DEFAULT](#).

#### 15.73.3.4 vrna\_file\_msa\_write()

```
vrna_file_msa_write (
    const char * filename,
    const char ** names,
    const char ** aln,
    const char * id,
    const char * structure,
    const char * source,
    unsigned int options )

#include <ViennaRNA/io/file_formats_msa.h>
```

Write multiple sequence alignment file.

#### Note

Currently, we only support [Stockholm 1.0 format](#) output

#### See also

[VRNA\\_FILE\\_FORMAT\\_MSA\\_STOCKHOLM](#), [VRNA\\_FILE\\_FORMAT\\_MSA\\_APPEND](#), [VRNA\\_FILE\\_FORMAT\\_MSA\\_MIS](#)



## Parameters

<i>filename</i>	The output filename
<i>names</i>	The array of sequence names / identifies
<i>aln</i>	The array of aligned sequences
<i>id</i>	An optional ID for the alignment
<i>structure</i>	An optional consensus structure
<i>source</i>	A string describing the source of the alignment
<i>options</i>	Options to manipulate the behavior of this function

## Returns

Non-null upon successfully writing the alignment to file

**SWIG Wrapper Notes** In the target scripting language, this function exists as a set of overloaded versions, where the last four parameters may be omitted. If the `options` parameter is missing the options default to ([VRNA\\_FILE\\_FORMAT\\_MSA\\_STOCKHOLM](#) | [VRNA\\_FILE\\_FORMAT\\_MSA\\_APPEND](#)).

## 15.74 Command Files

Functions to parse and interpret the content of [Command Files](#).

### 15.74.1 Detailed Description

Functions to parse and interpret the content of [Command Files](#).

Collaboration diagram for Command Files:

#### Files

- file [commands.h](#)

*Parse and apply different commands that alter the behavior of secondary structure prediction and evaluation.*

#### Macros

- `#define VRNA_CMD_PARSE_HC 1U`  
*Command parse/apply flag indicating hard constraints.*
- `#define VRNA_CMD_PARSE_SC 2U`  
*Command parse/apply flag indicating soft constraints.*
- `#define VRNA_CMD_PARSE_UD 4U`  
*Command parse/apply flag indicating unstructured domains.*
- `#define VRNA_CMD_PARSE_SD 8U`  
*Command parse/apply flag indicating structured domains.*
- `#define VRNA_CMD_PARSE_DEFAULTS`  
*Command parse/apply flag indicating default set of commands.*

#### Typedefs

- `typedef struct vrna_command_s * vrna_cmd_t`  
*A data structure that contains commands.*

#### Functions

- `vrna_cmd_t vrna_file_commands_read` (const char \*filename, unsigned int options)  
*Extract a list of commands from a command file.*
- `int vrna_file_commands_apply` (vrna\_fold\_compound\_t \*vc, const char \*filename, unsigned int options)  
*Apply a list of commands from a command file.*
- `int vrna_commands_apply` (vrna\_fold\_compound\_t \*vc, vrna\_cmd\_t commands, unsigned int options)  
*Apply a list of commands to a [vrna\\_fold\\_compound\\_t](#).*
- `void vrna_commands_free` (vrna\_cmd\_t commands)  
*Free memory occupied by a list of commands.*

## 15.74.2 Macro Definition Documentation

### 15.74.2.1 VRNA\_CMD\_PARSE\_HC

```
#define VRNA_CMD_PARSE_HC 1U
```

```
#include <ViennaRNA/commands.h>
```

Command parse/apply flag indicating hard constraints.

See also

[vrna\\_cmd\\_t](#), [vrna\\_file\\_commands\\_read\(\)](#), [vrna\\_file\\_commands\\_apply\(\)](#), [vrna\\_commands\\_apply\(\)](#)

### 15.74.2.2 VRNA\_CMD\_PARSE\_SC

```
#define VRNA_CMD_PARSE_SC 2U
```

```
#include <ViennaRNA/commands.h>
```

Command parse/apply flag indicating soft constraints.

See also

[vrna\\_cmd\\_t](#), [vrna\\_file\\_commands\\_read\(\)](#), [vrna\\_file\\_commands\\_apply\(\)](#), [vrna\\_commands\\_apply\(\)](#)

### 15.74.2.3 VRNA\_CMD\_PARSE\_UD

```
#define VRNA_CMD_PARSE_UD 4U
```

```
#include <ViennaRNA/commands.h>
```

Command parse/apply flag indicating unstructured domains.

See also

[vrna\\_cmd\\_t](#), [vrna\\_file\\_commands\\_read\(\)](#), [vrna\\_file\\_commands\\_apply\(\)](#), [vrna\\_commands\\_apply\(\)](#)

#### 15.74.2.4 VRNA\_CMD\_PARSE\_SD

```
#define VRNA_CMD_PARSE_SD 8U
```

```
#include <ViennaRNA/commands.h>
```

Command parse/apply flag indicating structured domains.

See also

[vrna\\_cmd\\_t](#), [vrna\\_file\\_commands\\_read\(\)](#), [vrna\\_file\\_commands\\_apply\(\)](#), [vrna\\_commands\\_apply\(\)](#)

#### 15.74.2.5 VRNA\_CMD\_PARSE\_DEFAULTS

```
#define VRNA_CMD_PARSE_DEFAULTS
```

```
#include <ViennaRNA/commands.h>
```

**Value:**

```
(VRNA_CMD_PARSE_HC \
    | VRNA_CMD_PARSE_SC \
    | VRNA_CMD_PARSE_UD \
    | VRNA_CMD_PARSE_SD \
)
```

Command parse/apply flag indicating default set of commands.

See also

[vrna\\_cmd\\_t](#), [vrna\\_file\\_commands\\_read\(\)](#), [vrna\\_file\\_commands\\_apply\(\)](#), [vrna\\_commands\\_apply\(\)](#)

### 15.74.3 Function Documentation

#### 15.74.3.1 vrna\_file\_commands\_read()

```
vrna_cmd_t vrna_file_commands_read (
    const char * filename,
    unsigned int options )
```

```
#include <ViennaRNA/commands.h>
```

Extract a list of commands from a command file.

Read a list of commands specified in the input file and return them as list of abstract commands

See also

[vrna\\_commands\\_apply\(\)](#), [vrna\\_file\\_commands\\_apply\(\)](#), [vrna\\_commands\\_free\(\)](#)

## Parameters

<i>filename</i>	The filename
<i>options</i>	Options to limit the type of commands read from the file

## Returns

A list of abstract commands

15.74.3.2 `vrna_file_commands_apply()`

```
int vrna_file_commands_apply (
    vrna_fold_compound_t * vc,
    const char * filename,
    unsigned int options )
```

```
#include <ViennaRNA/commands.h>
```

Apply a list of commands from a command file.

This function is a shortcut to directly parse a commands file and apply all successfully parsed commands to a `vrna_fold_compound_t` data structure. It is the same as:

```
int r;
struct vrna_command_s *cmds;

cmds = vrna_file_commands_read(filename, options);
r = vrna_commands_apply(vc, cmds, options);

vrna_commands_free(cmds);

return r;
```

## Parameters

<i>vc</i>	The <code>vrna_fold_compound_t</code> the command list will be applied to
<i>filename</i>	The filename
<i>options</i>	Options to limit the type of commands read from the file

## Returns

The number of commands successfully applied

**SWIG Wrapper Notes** This function is attached as method `file_commands_apply()` to objects of type `fold_↔compound`

### 15.74.3.3 vrna\_commands\_apply()

```
int vrna_commands_apply (
    vrna_fold_compound_t * vc,
    vrna_cmd_t commands,
    unsigned int options )
```

```
#include <ViennaRNA/commands.h>
```

Apply a list of commands to a [vrna\\_fold\\_compound\\_t](#).

#### Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> the command list will be applied to
<i>commands</i>	The commands to apply
<i>options</i>	Options to limit the type of commands read from the file

#### Returns

The number of commands successfully applied

### 15.74.3.4 vrna\_commands\_free()

```
void vrna_commands_free (
    vrna_cmd_t commands )
```

```
#include <ViennaRNA/commands.h>
```

Free memory occupied by a list of commands.

Release memory occupied by a list of commands

#### Parameters

<i>commands</i>	A pointer to a list of commands
-----------------	---------------------------------

## 15.75 Plotting

Functions for Creating Secondary Structure Plots, Dot-Plots, and More.

### 15.75.1 Detailed Description

Functions for Creating Secondary Structure Plots, Dot-Plots, and More.

Collaboration diagram for Plotting:

#### Modules

- [Annotation](#)

*Functions to generate annotations for Secondary Structure Plots, Dot-Plots, and Others.*

#### Files

- file [alignments.h](#)

*Various functions for plotting Sequence / Structure Alignments.*

- file [layouts.h](#)

*Secondary structure plot layout algorithms.*

- file [naview.h](#)

- file [probabilities.h](#)

*Various functions for plotting RNA secondary structures, dot-plots and other visualizations.*

- file [structures.h](#)

*Various functions for plotting RNA secondary structures.*

- file [utils.h](#)

*Various utilities to assist in plotting secondary structures and consensus structures.*

#### Data Structures

- struct [COORDINATE](#)

*this is a workaround for the SWIG Perl Wrapper RNA plot function that returns an array of type [COORDINATE](#) More...*

- struct [vrna\\_dotplot\\_auxdata\\_t](#)

#### Macros

- `#define VRNA\_PLOT\_TYPE\_SIMPLE 0`

*Definition of Plot type simple*

- `#define VRNA\_PLOT\_TYPE\_NAVIEW 1`

*Definition of Plot type Naview*

- `#define VRNA\_PLOT\_TYPE\_CIRCULAR 2`

*Definition of Plot type Circular*

## Functions

- `int PS_color_aln` (const char \*structure, const char \*filename, const char \*seqs[], const char \*names[])  
*Produce PostScript sequence alignment color-annotated by consensus structure.*
- `int vrna_file_PS_aln` (const char \*filename, const char \*\*seqs, const char \*\*names, const char \*structure, int columns)
- `int vrna_file_PS_aln_sub` (const char \*filename, const char \*\*seqs, const char \*\*names, const char \*structure, int start, int end, int columns)
- `int aliPS_color_aln` (const char \*structure, const char \*filename, const char \*seqs[], const char \*names[])
- `int simple_xy_coordinates` (short \*pair\_table, float \*X, float \*Y)  
*Calculate nucleotide coordinates for secondary structure plot the Simple way*
- `int simple_circplot_coordinates` (short \*pair\_table, float \*x, float \*y)  
*Calculate nucleotide coordinates for Circular Plot*
- `int PS_dot_plot_list` (char \*seq, char \*filename, `plist` \*pl, `plist` \*mf, char \*comment)  
*Produce a postscript dot-plot from two pair lists.*
- `int PS_dot_plot` (char \*string, char \*file)  
*Produce postscript dot-plot.*
- `int vrna_file_PS_rnaplot` (const char \*seq, const char \*structure, const char \*file, `vrna_md_t` \*md\_p)  
*Produce a secondary structure graph in PostScript and write it to 'filename'.*
- `int vrna_file_PS_rnaplot_a` (const char \*seq, const char \*structure, const char \*file, const char \*pre, const char \*post, `vrna_md_t` \*md\_p)  
*Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename'.*
- `int gmlRNA` (char \*string, char \*structure, char \*ssfile, char option)  
*Produce a secondary structure graph in Graph Meta Language (gml) and write it to a file.*
- `int ssv_rna_plot` (char \*string, char \*structure, char \*ssfile)  
*Produce a secondary structure graph in SStructView format.*
- `int svg_rna_plot` (char \*string, char \*structure, char \*ssfile)  
*Produce a secondary structure plot in SVG format and write it to a file.*
- `int xrna_plot` (char \*string, char \*structure, char \*ssfile)  
*Produce a secondary structure plot for further editing in XRNA.*
- `int PS_rna_plot` (char \*string, char \*structure, char \*file)  
*Produce a secondary structure graph in PostScript and write it to 'filename'.*
- `int PS_rna_plot_a` (char \*string, char \*structure, char \*file, char \*pre, char \*post)  
*Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename'.*
- `int PS_rna_plot_a_gquad` (char \*string, char \*structure, char \*ssfile, char \*pre, char \*post)  
*Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename' (detect and draw g-quadruplexes)*

## Variables

- `int rna_plot_type`  
*Switch for changing the secondary structure layout algorithm.*

## 15.75.2 Data Structure Documentation

### 15.75.2.1 struct COORDINATE

this is a workaround for the SWIG Perl Wrapper RNA plot function that returns an array of type `COORDINATE`



## 15.75.2.2 struct vrna\_dotplot\_auxdata\_t

Collaboration diagram for vrna\_dotplot\_auxdata\_t:

## 15.75.3 Macro Definition Documentation

## 15.75.3.1 VRNA\_PLOT\_TYPE\_SIMPLE

```
#define VRNA_PLOT_TYPE_SIMPLE 0

#include <ViennaRNA/plotting/layouts.h>
```

Definition of Plot type *simple*

This is the plot type definition for several RNA structure plotting functions telling them to use **Simple** plotting algorithm

See also

[rna\\_plot\\_type](#), [vrna\\_file\\_PS\\_rnaplot\\_a\(\)](#), [vrna\\_file\\_PS\\_rnaplot\(\)](#), [svg\\_rna\\_plot\(\)](#), [gmlRNA\(\)](#), [ssv\\_rna\\_plot\(\)](#), [xrna\\_plot\(\)](#)

## 15.75.3.2 VRNA\_PLOT\_TYPE\_NAVIEW

```
#define VRNA_PLOT_TYPE_NAVIEW 1

#include <ViennaRNA/plotting/layouts.h>
```

Definition of Plot type *Naview*

This is the plot type definition for several RNA structure plotting functions telling them to use **Naview** plotting algorithm

See also

[rna\\_plot\\_type](#), [vrna\\_file\\_PS\\_rnaplot\\_a\(\)](#), [vrna\\_file\\_PS\\_rnaplot\(\)](#), [svg\\_rna\\_plot\(\)](#), [gmlRNA\(\)](#), [ssv\\_rna\\_plot\(\)](#), [xrna\\_plot\(\)](#)

### 15.75.3.3 VRNA\_PLOT\_TYPE\_CIRCULAR

```
#define VRNA_PLOT_TYPE_CIRCULAR 2

#include <ViennaRNA/plotting/layouts.h>
```

Definition of Plot type *Circular*

This is the plot type definition for several RNA structure plotting functions telling them to produce a **Circular plot**

See also

[rna\\_plot\\_type](#), [vrna\\_file\\_PS\\_rnaplot\\_a\(\)](#), [vrna\\_file\\_PS\\_rnaplot\(\)](#), [svg\\_rna\\_plot\(\)](#), [gmlRNA\(\)](#), [ssv\\_rna\\_plot\(\)](#), [xrna\\_plot\(\)](#)

## 15.75.4 Function Documentation

### 15.75.4.1 vrna\_file\_PS\_aln()

```
int vrna_file_PS_aln (
    const char * filename,
    const char ** seqs,
    const char ** names,
    const char * structure,
    int columns )

#include <ViennaRNA/plotting/alignments.h>
```

Parameters

<i>columns</i>	The number of columns before the alignment is wrapped as a new block (values less than 1 indicate no wrapping)
----------------	--

### 15.75.4.2 vrna\_file\_PS\_aln\_sub()

```
int vrna_file_PS_aln_sub (
    const char * filename,
    const char ** seqs,
    const char ** names,
    const char * structure,
    int start,
    int end,
    int columns )

#include <ViennaRNA/plotting/alignments.h>
```

## Parameters

<i>columns</i>	The number of columns before the alignment is wrapped as a new block (values less than 1 indicate no wrapping)
----------------	--

## 15.75.4.3 aliPS\_color\_aln()

```
int aliPS_color_aln (
    const char * structure,
    const char * filename,
    const char * seqs[],
    const char * names[] )

#include <ViennaRNA/plotting/alignments.h>
```

PS\_color\_aln for duplexes

## 15.75.4.4 simple\_xy\_coordinates()

```
int simple_xy_coordinates (
    short * pair_table,
    float * X,
    float * Y )

#include <ViennaRNA/plotting/layouts.h>
```

Calculate nucleotide coordinates for secondary structure plot the *Simple way*

## See also

[make\\_pair\\_table\(\)](#), [rna\\_plot\\_type](#), [simple\\_circplot\\_coordinates\(\)](#), [naview\\_xy\\_coordinates\(\)](#), [vrna\\_file\\_PS\\_rnaplot\\_a\(\)](#), [vrna\\_file\\_PS\\_rnaplot](#), [svg\\_rna\\_plot\(\)](#)

## Parameters

<i>pair_table</i>	The pair table of the secondary structure
<i>X</i>	a pointer to an array with enough allocated space to hold the x coordinates
<i>Y</i>	a pointer to an array with enough allocated space to hold the y coordinates

## Returns

length of sequence on success, 0 otherwise

15.75.4.5 `simple_circplot_coordinates()`

```
int simple_circplot_coordinates (
    short * pair_table,
    float * x,
    float * y )

#include <ViennaRNA/plotting/layouts.h>
```

Calculate nucleotide coordinates for *Circular Plot*

This function calculates the coordinates of nucleotides mapped in equal distances onto a unit circle.

**Note**

In order to draw nice arcs using quadratic bezier curves that connect base pairs one may calculate a second tangential point  $P^t$  in addition to the actual  $R^2$  coordinates. the simplest way to do so may be to compute a radius scaling factor  $rs$  in the interval  $[0, 1]$  that weights the proportion of base pair span to the actual length of the sequence. This scaling factor can then be used to calculate the coordinates for  $P^t$ , i.e.  $P_x^t[i] = X[i] * rs$  and  $P_y^t[i] = Y[i] * rs$ .

**See also**

[make\\_pair\\_table\(\)](#), [rna\\_plot\\_type](#), [simple\\_xy\\_coordinates\(\)](#), [naview\\_xy\\_coordinates\(\)](#), [vrna\\_file\\_PS\\_rnaplot\\_a\(\)](#), [vrna\\_file\\_PS\\_rnaplot](#), [svg\\_rna\\_plot\(\)](#)

**Parameters**

<i>pair_table</i>	The pair table of the secondary structure
<i>x</i>	a pointer to an array with enough allocated space to hold the x coordinates
<i>y</i>	a pointer to an array with enough allocated space to hold the y coordinates

**Returns**

length of sequence on success, 0 otherwise

15.75.4.6 `PS_dot_plot_list()`

```
int PS_dot_plot_list (
    char * seq,
    char * filename,
    plist * pl,
    plist * mf,
    char * comment )

#include <ViennaRNA/plotting/probabilities.h>
```

Produce a postscript dot-plot from two pair lists.

This function reads two plist structures (e.g. base pair probabilities and a secondary structure) as produced by [assign\\_plist\\_from\\_pr\(\)](#) and [assign\\_plist\\_from\\_db\(\)](#) and produces a postscript "dot plot" that is written to 'filename'. Using base pair probabilities in the first and mfe structure in the second plist, the resulting "dot plot" represents each base pairing probability by a square of corresponding area in a upper triangle matrix. The lower part of the matrix contains the minimum free energy structure.

See also

[assign\\_plist\\_from\\_pr\(\)](#), [assign\\_plist\\_from\\_db\(\)](#)

#### Parameters

<i>seq</i>	The RNA sequence
<i>filename</i>	A filename for the postscript output
<i>pl</i>	The base pair probability pairlist
<i>mf</i>	The mfe secondary structure pairlist
<i>comment</i>	A comment

#### Returns

1 if postscript was successfully written, 0 otherwise

#### 15.75.4.7 PS\_dot\_plot()

```
int PS_dot_plot (
    char * string,
    char * file )

#include <ViennaRNA/plotting/probabilities.h>
```

Produce postscript dot-plot.

Wrapper to PS\_dot\_plot\_list

Reads base pair probabilities produced by [pf\\_fold\(\)](#) from the global array [pr](#) and the pair list [base\\_pair](#) produced by [fold\(\)](#) and produces a postscript "dot plot" that is written to 'filename'. The "dot plot" represents each base pairing probability by a square of corresponding area in a upper triangle matrix. The lower part of the matrix contains the minimum free energy

#### Note

DO NOT USE THIS FUNCTION ANYMORE SINCE IT IS NOT THREADSAFE

**Deprecated** This function is deprecated and will be removed soon! Use [PS\\_dot\\_plot\\_list\(\)](#) instead!

#### 15.75.4.8 vrna\_file\_PS\_rnaplot()

```
int vrna_file_PS_rnaplot (
    const char * seq,
    const char * structure,
    const char * file,
    vrna_md_t * md_p )

#include <ViennaRNA/plotting/structures.h>
```

Produce a secondary structure graph in PostScript and write it to 'filename'.

Note that this function has changed from previous versions and now expects the structure to be plotted in dot-bracket notation as an argument. It does not make use of the global [base\\_pair](#) array anymore.

## Parameters

<i>seq</i>	The RNA sequence
<i>structure</i>	The secondary structure in dot-bracket notation
<i>file</i>	The filename of the postscript output
<i>md_p</i>	Model parameters used to generate a commandline option string in the output (Maybe NULL)

## Returns

1 on success, 0 otherwise

15.75.4.9 `vrna_file_PS_rnaplot_a()`

```
int vrna_file_PS_rnaplot_a (
    const char * seq,
    const char * structure,
    const char * file,
    const char * pre,
    const char * post,
    vrna_md_t * md_p )
```

```
#include <ViennaRNA/plotting/structures.h>
```

Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename'.

Same as `vrna_file_PS_rnaplot()` but adds extra PostScript macros for various annotations (see generated PS code). The 'pre' and 'post' variables contain PostScript code that is verbatim copied in the resulting PS file just before and after the structure plot. If both arguments ('pre' and 'post') are NULL, no additional macros will be printed into the PostScript.

## Parameters

<i>seq</i>	The RNA sequence
<i>structure</i>	The secondary structure in dot-bracket notation
<i>file</i>	The filename of the postscript output
<i>pre</i>	PostScript code to appear before the secondary structure plot
<i>post</i>	PostScript code to appear after the secondary structure plot
<i>md_p</i>	Model parameters used to generate a commandline option string in the output (Maybe NULL)

## Returns

1 on success, 0 otherwise

15.75.4.10 `gmlRNA()`

```
int gmlRNA (
    char * string,
```

```
char * structure,  
char * ssfile,  
char option )
```

```
#include <ViennaRNA/plotting/structures.h>
```

Produce a secondary structure graph in Graph Meta Language (gml) and write it to a file.

If 'option' is an uppercase letter the RNA sequence is used to label nodes, if 'option' equals 'X' or 'x' the resulting file will contain coordinates for an initial layout of the graph.

#### Parameters

<i>string</i>	The RNA sequence
<i>structure</i>	The secondary structure in dot-bracket notation
<i>ssfile</i>	The filename of the gml output
<i>option</i>	The option flag

#### Returns

1 on success, 0 otherwise

#### 15.75.4.11 ssv\_rna\_plot()

```
int ssv_rna_plot (  
    char * string,  
    char * structure,  
    char * ssfile )
```

```
#include <ViennaRNA/plotting/structures.h>
```

Produce a secondary structure graph in SStructView format.

Write coord file for SStructView

#### Parameters

<i>string</i>	The RNA sequence
<i>structure</i>	The secondary structure in dot-bracket notation
<i>ssfile</i>	The filename of the ssv output

#### Returns

1 on success, 0 otherwise

#### 15.75.4.12 `svg_rna_plot()`

```
int svg_rna_plot (
    char * string,
    char * structure,
    char * ssfile )

#include <ViennaRNA/plotting/structures.h>
```

Produce a secondary structure plot in SVG format and write it to a file.

##### Parameters

<i>string</i>	The RNA sequence
<i>structure</i>	The secondary structure in dot-bracket notation
<i>ssfile</i>	The filename of the svg output

##### Returns

1 on success, 0 otherwise

#### 15.75.4.13 `xrna_plot()`

```
int xrna_plot (
    char * string,
    char * structure,
    char * ssfile )

#include <ViennaRNA/plotting/structures.h>
```

Produce a secondary structure plot for further editing in XRNA.

##### Parameters

<i>string</i>	The RNA sequence
<i>structure</i>	The secondary structure in dot-bracket notation
<i>ssfile</i>	The filename of the xrna output

##### Returns

1 on success, 0 otherwise

#### 15.75.4.14 `PS_rna_plot()`

```
int PS_rna_plot (
    char * string,
```



```
char * structure,  
char * file )
```

```
#include <ViennaRNA/plotting/structures.h>
```

Produce a secondary structure graph in PostScript and write it to 'filename'.

**Deprecated** Use `vrna_file_PS_rnaplot()` instead!

#### 15.75.4.15 PS\_rna\_plot\_a()

```
int PS_rna_plot_a (  
    char * string,  
    char * structure,  
    char * file,  
    char * pre,  
    char * post )
```

```
#include <ViennaRNA/plotting/structures.h>
```

Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename'.

**Deprecated** Use `vrna_file_PS_rnaplot_a()` instead!

#### 15.75.4.16 PS\_rna\_plot\_a\_gquad()

```
int PS_rna_plot_a_gquad (  
    char * string,  
    char * structure,  
    char * ssfile,  
    char * pre,  
    char * post )
```

```
#include <ViennaRNA/plotting/structures.h>
```

Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename' (detect and draw g-quadruplexes)

**Deprecated** Use `vrna_file_PS_rnaplot_a()` instead!

### 15.75.5 Variable Documentation

### 15.75.5.1 rna\_plot\_type

```
int rna_plot_type
```

```
#include <ViennaRNA/plotting/layouts.h>
```

Switch for changing the secondary structure layout algorithm.

Current possibilities are 0 for a simple radial drawing or 1 for the modified radial drawing taken from the *naview* program of [5].

#### Note

To provide thread safety please do not rely on this global variable in future implementations but pass a plot type flag directly to the function that decides which layout algorithm it may use!

#### See also

[VRNA\\_PLOT\\_TYPE\\_SIMPLE](#), [VRNA\\_PLOT\\_TYPE\\_NAVIEW](#), [VRNA\\_PLOT\\_TYPE\\_CIRCULAR](#)

## 15.76 Annotation

Functions to generate annotations for Secondary Structure Plots, Dot-Plots, and Others.

### 15.76.1 Detailed Description

Functions to generate annotations for Secondary Structure Plots, Dot-Plots, and Others.

Collaboration diagram for Annotation:

#### Functions

- `char ** vrna_annotate_covar_struct` (const char \*\*alignment, const char \*structure, `vrna_md_t` \*md)  
*Produce covariance annotation for an alignment given a secondary structure.*
- `vrna_cpair_t * vrna_annotate_covar_pairs` (const char \*\*alignment, `vrna_ep_t` \*pl, `vrna_ep_t` \*mfel, double threshold, `vrna_md_t` \*md)  
*Produce covariance annotation for an alignment given a set of base pairs.*

## 15.77 Search Algorithms

Implementations of various search algorithms to detect strings of objects within other strings of objects.

### 15.77.1 Detailed Description

Implementations of various search algorithms to detect strings of objects within other strings of objects.

Collaboration diagram for Search Algorithms:

#### Files

- file [BoyerMoore.h](#)  
*Variants of the Boyer-Moore string search algorithm.*

#### Functions

- `const unsigned int * vrna_search_BMH_num` (`const unsigned int *needle`, `size_t needle_size`, `const unsigned int *haystack`, `size_t haystack_size`, `size_t start`, `size_t *badchars`, `unsigned char cyclic`)  
*Search for a string of elements in a larger string of elements using the Boyer-Moore-Horspool algorithm.*
- `const char * vrna_search_BMH` (`const char *needle`, `size_t needle_size`, `const char *haystack`, `size_t haystack_size`, `size_t start`, `size_t *badchars`, `unsigned char cyclic`)  
*Search for an ASCII pattern within a larger ASCII string using the Boyer-Moore-Horspool algorithm.*
- `size_t * vrna_search_BM_BCT_num` (`const unsigned int *pattern`, `size_t pattern_size`, `unsigned int num_max`)  
*Retrieve a Boyer-Moore Bad Character Table for a pattern of elements represented by natural numbers.*
- `size_t * vrna_search_BM_BCT` (`const char *pattern`)  
*Retrieve a Boyer-Moore Bad Character Table for a NULL-terminated pattern of ASCII characters.*

### 15.77.2 Function Documentation

#### 15.77.2.1 vrna\_search\_BMH\_num()

```
const unsigned int* vrna_search_BMH_num (
    const unsigned int * needle,
    size_t needle_size,
    const unsigned int * haystack,
    size_t haystack_size,
    size_t start,
    size_t * badchars,
    unsigned char cyclic )

#include <ViennaRNA/search/BoyerMoore.h>
```

Search for a string of elements in a larger string of elements using the Boyer-Moore-Horspool algorithm.

To speed-up subsequent searches with this function, the Bad Character Table should be precomputed and passed as argument `badchars`.

#### See also

[vrna\\_search\\_BM\\_BCT\\_num\(\)](#), [vrna\\_search\\_BMH\(\)](#)

## Parameters

<i>needle</i>	The pattern of object representations to search for
<i>needle_size</i>	The size (length) of the pattern provided in <i>needle</i>
<i>haystack</i>	The string of objects the search will be performed on
<i>haystack_size</i>	The size (length) of the <i>haystack</i> string
<i>start</i>	The position within <i>haystack</i> where to start the search
<i>badchars</i>	A pre-computed Bad Character Table obtained from <a href="#">vrna_search_BM_BCT_num()</a> (If NULL, a Bad Character Table will be generated automatically)
<i>cyclic</i>	Allow for cyclic matches if non-zero, stop search at end of haystack otherwise

## Returns

A pointer to the first occurrence of *needle* within *haystack* after position *start*

## 15.77.2.2 vrna\_search\_BMH()

```
const char* vrna_search_BMH (
    const char * needle,
    size_t needle_size,
    const char * haystack,
    size_t haystack_size,
    size_t start,
    size_t * badchars,
    unsigned char cyclic )
```

```
#include <ViennaRNA/search/BoyerMoore.h>
```

Search for an ASCII pattern within a larger ASCII string using the Boyer-Moore-Horspool algorithm.

To speed-up subsequent searches with this function, the Bad Character Table should be precomputed and passed as argument *badchars*. Furthermore, both, the lengths of *needle* and the length of *haystack* should be pre-computed and must be passed along with each call.

## See also

[vrna\\_search\\_BM\\_BCT\(\)](#), [vrna\\_search\\_BMH\\_num\(\)](#)

## Parameters

<i>needle</i>	The NULL-terminated ASCII pattern to search for
<i>needle_size</i>	The size (length) of the pattern provided in <i>needle</i>
<i>haystack</i>	The NULL-terminated ASCII string of the search will be performed on
<i>haystack_size</i>	The size (length) of the <i>haystack</i> string
<i>start</i>	The position within <i>haystack</i> where to start the search
<i>badchars</i>	A pre-computed Bad Character Table obtained from <a href="#">vrna_search_BM_BCT()</a> (If NULL, a Bad Character Table will be generated automatically)
<i>cyclic</i>	Allow for cyclic matches if non-zero, stop search at end of haystack otherwise

**Returns**

A pointer to the first occurrence of `needle` within `haystack` after position `start`

**15.77.2.3 `vrna_search_BM_BCT_num()`**

```
size_t* vrna_search_BM_BCT_num (
    const unsigned int * pattern,
    size_t pattern_size,
    unsigned int num_max )

#include <ViennaRNA/search/BoyerMoore.h>
```

Retrieve a Boyer-Moore Bad Character Table for a pattern of elements represented by natural numbers.

**Note**

We store the maximum number representation of an element `num_max` at position 0. So the actual bad character table `T` starts at `T[1]` for an element represented by number 0.

**See also**

[vrna\\_search\\_BMH\\_num\(\)](#), [vrna\\_search\\_BM\\_BCT\(\)](#)

**Parameters**

<i>pattern</i>	The pattern of element representations used in the subsequent search
<i>pattern_size</i>	The size (length) of the pattern provided in <code>pattern</code>
<i>num_max</i>	The maximum number representation of an element, i.e. the size of the alphabet

**Returns**

A Bad Character Table for use in our Boyer-Moore search algorithm implementation(s)

**15.77.2.4 `vrna_search_BM_BCT()`**

```
size_t* vrna_search_BM_BCT (
    const char * pattern )

#include <ViennaRNA/search/BoyerMoore.h>
```

Retrieve a Boyer-Moore Bad Character Table for a NULL-terminated pattern of ASCII characters.

**Note**

We store the maximum number representation of an element, i.e. 127 at position 0. So the actual bad character table `T` starts at `T[1]` for an element represented by ASCII code 0.

**See also**

[vrna\\_search\\_BMH\(\)](#), [vrna\\_search\\_BM\\_BCT\\_num\(\)](#)

**Parameters**

<i>pattern</i>	The NULL-terminated pattern of ASCII characters used in the subsequent search
----------------	---

**Returns**

A Bad Character Table for use in our Boyer-Moore search algorithm implementation(s)

## 15.78 Combinatorics Algorithms

Implementations to solve various combinatorial aspects for strings of objects.

### 15.78.1 Detailed Description

Implementations to solve various combinatorial aspects for strings of objects.

Collaboration diagram for Combinatorics Algorithms:

#### Files

- file [combinatorics.h](#)  
*Various implementations that deal with combinatorial aspects of objects.*

#### Functions

- unsigned int \*\* [vrna\\_enumerate\\_necklaces](#) (const unsigned int \*type\_counts)  
*Enumerate all necklaces with fixed content.*
- unsigned int [vrna\\_rotational\\_symmetry\\_num](#) (const unsigned int \*string, size\_t string\_length)  
*Determine the order of rotational symmetry for a string of objects represented by natural numbers.*
- unsigned int [vrna\\_rotational\\_symmetry\\_pos\\_num](#) (const unsigned int \*string, size\_t string\_length, unsigned int \*\*positions)  
*Determine the order of rotational symmetry for a string of objects represented by natural numbers.*
- unsigned int [vrna\\_rotational\\_symmetry](#) (const char \*string)  
*Determine the order of rotational symmetry for a NULL-terminated string of ASCII characters.*
- unsigned int [vrna\\_rotational\\_symmetry\\_pos](#) (const char \*string, unsigned int \*\*positions)  
*Determine the order of rotational symmetry for a NULL-terminated string of ASCII characters.*
- unsigned int [vrna\\_rotational\\_symmetry\\_db](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, const char \*structure)  
*Determine the order of rotational symmetry for a dot-bracket structure.*
- unsigned int [vrna\\_rotational\\_symmetry\\_db\\_pos](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, const char \*structure, unsigned int \*\*positions)  
*Determine the order of rotational symmetry for a dot-bracket structure.*

### 15.78.2 Function Documentation

#### 15.78.2.1 vrna\_enumerate\_necklaces()

```
unsigned int ** vrna_enumerate_necklaces (
    const unsigned int * type_counts )

#include <ViennaRNA/combinatorics.h>
```

Enumerate all necklaces with fixed content.

This function implements *A fast algorithm to generate necklaces with fixed content* as published by Joe Sawada in 2003 [19].

The function receives a list of counts (the elements on the necklace) for each type of object within a necklace. The list starts at index 0 and ends with an entry that has a count of 0. The algorithm then enumerates all non-cyclic permutations of the content, returned as a list of necklaces. This list, again, is zero-terminated, i.e. the last entry of the list is a NULL pointer.



## Parameters

<i>type_counts</i>	A 0-terminated list of entity counts
--------------------	--------------------------------------

## Returns

A list of all non-cyclic permutations of the entities

**SWIG Wrapper Notes** This function is available as global function **enumerate\_necklaces()** which accepts lists input, and produces list of lists output.

## 15.78.2.2 vrna\_rotational\_symmetry\_num()

```
unsigned int vrna_rotational_symmetry_num (
    const unsigned int * string,
    size_t string_length )
```

```
#include <ViennaRNA/combinatorics.h>
```

Determine the order of rotational symmetry for a string of objects represented by natural numbers.

The algorithm applies a fast search of the provided string within itself, assuming the end of the string wraps around to connect with its start. For example, a string of the form 011011 has rotational symmetry of order 2

This is a simplified version of [vrna\\_rotational\\_symmetry\\_pos\\_num\(\)](#) that may be useful if one is only interested in the degree of rotational symmetry but not the actual set of rotational symmetric strings.

## See also

[vrna\\_rotational\\_symmetry\\_pos\\_num\(\)](#), [vrna\\_rotational\\_symmetry\(\)](#)

## Parameters

<i>string</i>	The string of elements encoded as natural numbers
<i>string_length</i>	The length of the string

## Returns

The order of rotational symmetry

**SWIG Wrapper Notes** This function is available as global function **rotational\_symmetry()**. See [vrna\\_rotational\\_symmetry\\_pos\(\)](#) for details. Note, that in the target language the length of the list *string* is always known a-priori, so the parameter *string\_length* must be omitted.

### 15.78.2.3 vrna\_rotational\_symmetry\_pos\_num()

```
unsigned int vrna_rotational_symmetry_pos_num (
    const unsigned int * string,
    size_t string_length,
    unsigned int ** positions )
```

```
#include <ViennaRNA/combinatorics.h>
```

Determine the order of rotational symmetry for a string of objects represented by natural numbers.

The algorithm applies a fast search of the provided string within itself, assuming the end of the string wraps around to connect with it's start. For example, a string of the form 011011 has rotational symmetry of order 2

If the argument `positions` is not `NULL`, the function stores an array of string start positions for rotational shifts that map the string back onto itself. This array has length of order of rotational symmetry, i.e. the number returned by this function. The first element `positions[0]` always contains a shift value of 0 representing the trivial rotation.

#### Note

Do not forget to release the memory occupied by `positions` after a successful execution of this function.

#### See also

[vrna\\_rotational\\_symmetry\\_num\(\)](#), [vrna\\_rotational\\_symmetry\(\)](#), [vrna\\_rotational\\_symmetry\\_pos\(\)](#)

#### Parameters

<i>string</i>	The string of elements encoded as natural numbers
<i>string_length</i>	The length of the string
<i>positions</i>	A pointer to an (undefined) list of alternative string start positions that lead to an identity mapping (may be <code>NULL</code> )

#### Returns

The order of rotational symmetry

**SWIG Wrapper Notes** This function is available as global function **rotational\_symmetry()**. See [vrna\\_rotational\\_symmetry\\_pos\(\)](#) for details. Note, that in the target language the length of the list `string` is always known a-priori, so the parameter `string_length` must be omitted.

### 15.78.2.4 vrna\_rotational\_symmetry()

```
unsigned int vrna_rotational_symmetry (
    const char * string )
```

```
#include <ViennaRNA/combinatorics.h>
```

Determine the order of rotational symmetry for a `NULL`-terminated string of ASCII characters.

The algorithm applies a fast search of the provided string within itself, assuming the end of the string wraps around to connect with it's start. For example, a string of the form AABAAB has rotational symmetry of order 2

This is a simplified version of [vrna\\_rotational\\_symmetry\\_pos\(\)](#) that may be useful if one is only interested in the degree of rotational symmetry but not the actual set of rotational symmetric strings.

See also

[vrna\\_rotational\\_symmetry\\_pos\(\)](#), [vrna\\_rotational\\_symmetry\\_num\(\)](#)

Parameters

<i>string</i>	A NULL-terminated string of characters
---------------	--

Returns

The order of rotational symmetry

**SWIG Wrapper Notes** This function is available as global function **rotational\_symmetry()**. See [vrna\\_rotational\\_symmetry\\_pos\(\)](#) for details.

#### 15.78.2.5 vrna\_rotational\_symmetry\_pos()

```
unsigned int vrna_rotational_symmetry_pos (
    const char * string,
    unsigned int ** positions )
```

```
#include <ViennaRNA/combinatorics.h>
```

Determine the order of rotational symmetry for a NULL-terminated string of ASCII characters.

The algorithm applies a fast search of the provided string within itself, assuming the end of the string wraps around to connect with its start. For example, a string of the form `AABAAB` has rotational symmetry of order 2.

If the argument `positions` is not `NULL`, the function stores an array of string start positions for rotational shifts that map the string back onto itself. This array has length of order of rotational symmetry, i.e. the number returned by this function. The first element `positions[0]` always contains a shift value of 0 representing the trivial rotation.

Note

Do not forget to release the memory occupied by `positions` after a successful execution of this function.

See also

[vrna\\_rotational\\_symmetry\(\)](#), [vrna\\_rotational\\_symmetry\\_num\(\)](#), [vrna\\_rotational\\_symmetry\\_num\\_pos\(\)](#)

Parameters

<i>string</i>	A NULL-terminated string of characters
<i>positions</i>	A pointer to an (undefined) list of alternative string start positions that lead to an identity mapping (may be <code>NULL</code> )

**Returns**

The order of rotational symmetry

**15.78.2.6 vrna\_rotational\_symmetry\_db()**

```
unsigned int vrna_rotational_symmetry_db (
    vrna_fold_compound_t * fc,
    const char * structure )
```

```
#include <ViennaRNA/combinatorics.h>
```

Determine the order of rotational symmetry for a dot-bracket structure.

Given a (permutation of multiple) RNA strand(s) and a particular secondary structure in dot-bracket notation, compute the degree of rotational symmetry. In case there is only a single linear RNA strand, the structure always has degree 1, as there are no rotational symmetries due to the direction of the nucleic acid sequence and the fixed positions of 5' and 3' ends. However, for circular RNAs, rotational symmetries might arise if the sequence consists of a concatenation of  $k$  identical subsequences.

This is a simplified version of [vrna\\_rotational\\_symmetry\\_db\\_pos\(\)](#) that may be useful if one is only interested in the degree of rotational symmetry but not the actual set of rotational symmetric strings.

**See also**

[vrna\\_rotational\\_symmetry\\_db\\_pos\(\)](#), [vrna\\_rotational\\_symmetry\(\)](#), [vrna\\_rotational\\_symmetry\\_num\(\)](#)

**Parameters**

<i>fc</i>	A fold_compound data structure containing the nucleic acid sequence(s), their order, and model settings
<i>structure</i>	The dot-bracket structure the degree of rotational symmetry is checked for

**Returns**

The degree of rotational symmetry of the `structure` (0 in case of any errors)

**SWIG Wrapper Notes** This function is attached as method **rotational\_symmetry\_db()** to objects of type `fold_compound` (i.e. [vrna\\_fold\\_compound\\_t](#)). See [vrna\\_rotational\\_symmetry\\_db\\_pos\(\)](#) for details.

**15.78.2.7 vrna\_rotational\_symmetry\_db\_pos()**

```
unsigned int vrna_rotational_symmetry_db_pos (
    vrna_fold_compound_t * fc,
```

```

    const char * structure,
    unsigned int ** positions )

#include <ViennaRNA/combinatorics.h>

```

Determine the order of rotational symmetry for a dot-bracket structure.

Given a (permutation of multiple) RNA strand(s) and a particular secondary structure in dot-bracket notation, compute the degree of rotational symmetry. In case there is only a single linear RNA strand, the structure always has degree 1, as there are no rotational symmetries due to the direction of the nucleic acid sequence and the fixed positions of 5' and 3' ends. However, for circular RNAs, rotational symmetries might arise if the sequence consists of a concatenation of  $k$  identical subsequences.

If the argument `positions` is not `NULL`, the function stores an array of string start positions for rotational shifts that map the string back onto itself. This array has length of order of rotational symmetry, i.e. the number returned by this function. The first element `positions[0]` always contains a shift value of 0 representing the trivial rotation.

#### Note

Do not forget to release the memory occupied by `positions` after a successful execution of this function.

#### See also

[vrna\\_rotational\\_symmetry\\_db\(\)](#), [vrna\\_rotational\\_symmetry\\_pos\(\)](#), [vrna\\_rotational\\_symmetry\\_pos\\_num\(\)](#)

#### Parameters

<i>fc</i>	A fold_compound data structure containing the nucleic acid sequence(s), their order, and model settings
<i>structure</i>	The dot-bracket structure the degree of rotational symmetry is checked for
<i>positions</i>	A pointer to an (undefined) list of alternative string start positions that lead to an identity mapping (may be <code>NULL</code> )

#### Returns

The degree of rotational symmetry of the `structure` (0 in case of any errors)

**SWIG Wrapper Notes** This function is attached as method **rotational\_symmetry\_db()** to objects of type `fold_compound` (i.e. [vrna\\_fold\\_compound\\_t](#)). Thus, the first argument must be omitted. In contrast to our C-implementation, this function doesn't simply return the order of rotational symmetry of the secondary structure, but returns the list `position` of cyclic permutation shifts that result in a rotationally symmetric structure. The length of the list then determines the order of rotational symmetry.

## 15.79 (Abstract) Data Structures

All datastructures and typedefs shared among the ViennaRNA Package can be found here.

### 15.79.1 Detailed Description

All datastructures and typedefs shared among the ViennaRNA Package can be found here.

Collaboration diagram for (Abstract) Data Structures:

### Modules

- [The Fold Compound](#)  
*This module provides interfaces that deal with the most basic data structure used in structure predicting and energy evaluating function of the RNAlib.*
- [The Dynamic Programming Matrices](#)  
*This module provides interfaces that deal with creation and destruction of dynamic programming matrices used within the RNAlib.*
- [Hash Tables](#)  
*Various implementations of hash table functions.*
- [Buffers](#)  
*Functions that provide dynamically buffered stream-like data structures.*

### Files

- file [dp\\_matrices.h](#)  
*Functions to deal with standard dynamic programming (DP) matrices.*
- file [basic.h](#)  
*Various data structures and pre-processor macros.*

### Data Structures

- struct [vrna\\_basepair\\_s](#)  
*Base pair data structure used in subopt.c. [More...](#)*
- struct [vrna\\_cpair\\_s](#)  
*this datastructure is used as input parameter in functions of PS\_dot.c [More...](#)*
- struct [vrna\\_color\\_s](#)
- struct [vrna\\_data\\_linear\\_s](#)
- struct [vrna\\_sect\\_s](#)  
*Stack of partial structures for backtracking. [More...](#)*
- struct [vrna\\_bp\\_stack\\_s](#)  
*Base pair stack element. [More...](#)*
- struct [pu\\_contrib](#)  
*contributions to p\_u [More...](#)*
- struct [interact](#)  
*interaction data structure for RNAup [More...](#)*

- struct [pu\\_out](#)  
*Collection of all free\_energy of beeing unpaired values for output. [More...](#)*
- struct [constrain](#)  
*constraints for cofolding [More...](#)*
- struct [duplexT](#)  
*Data structure for RNAduplex. [More...](#)*
- struct [node](#)  
*Data structure for RNAsnoop (fold energy list) [More...](#)*
- struct [snoopT](#)  
*Data structure for RNAsnoop. [More...](#)*
- struct [dupVar](#)  
*Data structure used in RNAPkplex. [More...](#)*

## Typedefs

- typedef struct [vrna\\_basepair\\_s](#) [vrna\\_basepair\\_t](#)  
*Typename for the base pair representing data structure [vrna\\_basepair\\_s](#).*
- typedef struct [vrna\\_elem\\_prob\\_s](#) [vrna\\_plist\\_t](#)  
*Typename for the base pair list representing data structure [vrna\\_elem\\_prob\\_s](#).*
- typedef struct [vrna\\_bp\\_stack\\_s](#) [vrna\\_bp\\_stack\\_t](#)  
*Typename for the base pair stack representing data structure [vrna\\_bp\\_stack\\_s](#).*
- typedef struct [vrna\\_cpair\\_s](#) [vrna\\_cpair\\_t](#)  
*Typename for data structure [vrna\\_cpair\\_s](#).*
- typedef struct [vrna\\_sect\\_s](#) [vrna\\_sect\\_t](#)  
*Typename for stack of partial structures [vrna\\_sect\\_s](#).*
- typedef double [FLT\\_OR\\_DBL](#)  
*Typename for floating point number in partition function computations.*
- typedef struct [vrna\\_basepair\\_s](#) PAIR  
*Old typename of [vrna\\_basepair\\_s](#).*
- typedef struct [vrna\\_elem\\_prob\\_s](#) plist  
*Old typename of [vrna\\_elem\\_prob\\_s](#).*
- typedef struct [vrna\\_cpair\\_s](#) cpair  
*Old typename of [vrna\\_cpair\\_s](#).*
- typedef struct [vrna\\_sect\\_s](#) sect  
*Old typename of [vrna\\_sect\\_s](#).*
- typedef struct [vrna\\_bp\\_stack\\_s](#) bondT  
*Old typename of [vrna\\_bp\\_stack\\_s](#).*
- typedef struct [pu\\_contrib](#) [pu\\_contrib](#)  
*contributions to  $p_u$*
- typedef struct [interact](#) [interact](#)  
*interaction data structure for RNAup*
- typedef struct [pu\\_out](#) [pu\\_out](#)  
*Collection of all free\_energy of beeing unpaired values for output.*
- typedef struct [constrain](#) [constrain](#)  
*constraints for cofolding*
- typedef struct [node](#) [folden](#)  
*Data structure for RNAsnoop (fold energy list)*
- typedef struct [dupVar](#) [dupVar](#)  
*Data structure used in RNAPkplex.*

## Functions

- void `vrna_C11_features` (void)

*Dummy symbol to check whether the library was build using C11/C++11 features.*

## 15.79.2 Data Structure Documentation

### 15.79.2.1 struct `vrna_basepair_s`

Base pair data structure used in `subopt.c`.

### 15.79.2.2 struct `vrna_cpair_s`

this datastructure is used as input parameter in functions of `PS_dot.c`

### 15.79.2.3 struct `vrna_color_s`

### 15.79.2.4 struct `vrna_data_linear_s`

Collaboration diagram for `vrna_data_linear_s`:

### 15.79.2.5 struct `vrna_sect_s`

Stack of partial structures for backtracking.

### 15.79.2.6 struct `vrna_bp_stack_s`

Base pair stack element.

### 15.79.2.7 struct `pu_contrib`

contributions to `p_u`

## Data Fields

- double \*\* `H`  
*hairpin loops*
- double \*\* `I`  
*interior loops*
- double \*\* `M`  
*multi loops*
- double \*\* `E`  
*exterior loop*
- int `length`  
*length of the input sequence*
- int `w`  
*longest unpaired region*



## 15.79.2.8 struct interact

interaction data structure for RNAup

## Data Fields

- double \* [Pi](#)  
*probabilities of interaction*
- double \* [Gi](#)  
*free energies of interaction*
- double [Gikjl](#)  
*full free energy for interaction between  $[k,i]$   $k < i$  in longer seq and  $[j,l]$   $j < l$  in shorter seq*
- double [Gikjl\\_wo](#)  
*Gikjl without contributions for prob\_unpaired.*
- int [i](#)  
 *$k < i$  in longer seq*
- int [k](#)  
 *$k < i$  in longer seq*
- int [j](#)  
 *$j < l$  in shorter seq*
- int [l](#)  
 *$j < l$  in shorter seq*
- int [length](#)  
*length of longer sequence*

## 15.79.2.9 struct pu\_out

Collection of all free\_energy of beeing unpaired values for output.

## Data Fields

- int [len](#)  
*sequence length*
- int [u\\_vals](#)  
*number of different -u values*
- int [contribs](#)  
*[-c "SHIME"]*
- char \*\* [header](#)  
*header line*
- double \*\* [u\\_values](#)  
*(the -u values \* [-c "SHIME"]) \* seq len*

## 15.79.2.10 struct constrain

constraints for cofolding

#### 15.79.2.11 struct duplexT

Data structure for RNAduplex.

#### 15.79.2.12 struct node

Data structure for RNAsnoop (fold energy list)

Collaboration diagram for node:

#### 15.79.2.13 struct snoopT

Data structure for RNAsnoop.

#### 15.79.2.14 struct dupVar

Data structure used in RNApkplex.

### 15.79.3 Typedef Documentation

#### 15.79.3.1 PAIR

```
typedef struct vrna_basepair_s PAIR  
  
#include <ViennaRNA/datastructures/basic.h>  
  
Old typename of vrna_basepair_s.
```

**Deprecated** Use vrna\_basepair\_t instead!

#### 15.79.3.2 plist

```
typedef struct vrna_elem_prob_s plist  
  
#include <ViennaRNA/datastructures/basic.h>  
  
Old typename of vrna_elem_prob_s.
```

**Deprecated** Use vrna\_ep\_t or vrna\_elem\_prob\_s instead!

### 15.79.3.3 cpair

```
typedef struct vrna_cpair_s cpair
```

```
#include <ViennaRNA/datastructures/basic.h>
```

Old typename of `vrna_cpair_s`.

**Deprecated** Use `vrna_cpair_t` instead!

### 15.79.3.4 sect

```
typedef struct vrna_sect_s sect
```

```
#include <ViennaRNA/datastructures/basic.h>
```

Old typename of `vrna_sect_s`.

**Deprecated** Use `vrna_sect_t` instead!

### 15.79.3.5 bondT

```
typedef struct vrna_bp_stack_s bondT
```

```
#include <ViennaRNA/datastructures/basic.h>
```

Old typename of `vrna_bp_stack_s`.

**Deprecated** Use `vrna_bp_stack_t` instead!

## 15.79.4 Function Documentation

#### 15.79.4.1 vrna\_C11\_features()

```
void vrna_C11_features (
    void )
```

```
#include <ViennaRNA/datastructures/basic.h>
```

Dummy symbol to check whether the library was build using C11/C++11 features.

By default, several data structures of our new v3.0 API use C11/C++11 features, such as unnamed unions, unnamed structs. However, these features can be deactivated at compile time to allow building the library and executables with compilers that do not support these features.

Now, the problem arises that once our static library is compiled and a third-party application is supposed to link against it, it needs to know, at compile time, how to correctly address particular data structures. This is usually implicitly taken care of through the API exposed in our header files. Unfortunately, we had some preprocessor directives in our header files that changed the API depending on the capabilities of the compiler the third-party application is build with. This in turn prohibited the use of an RNALib compiled without C11/C++11 support in a program that compiles/links with enabled C11/C++11 support and vice-versa.

Therefore, we introduce this dummy symbol which can be used to check, whether the static library was build with C11/C++11 features.

#### Note

If the symbol is present, the library was build with enabled C11/C++11 features support and no action is required. However, if the symbol is missing in RNALib  $\geq 2.2.9$ , programs that link to RNALib must define a pre-processor identifier `VRNA_DISABLE_C11_FEATURES` before including any ViennaRNA Package header file, for instance by adding a `CPPFLAG`

```
CPPFLAGS+=-DVRNA_DISABLE_C11_FEATURES
```

#### Since

v2.2.9

## 15.80 Messages

Functions to print various kind of messages.

### 15.80.1 Detailed Description

Functions to print various kind of messages.

Collaboration diagram for Messages:

#### Functions

- void [vrna\\_message\\_error](#) (const char \*format,...)  
*Print an error message and die.*
- void [vrna\\_message\\_verror](#) (const char \*format, va\_list args)  
*Print an error message and die.*
- void [vrna\\_message\\_warning](#) (const char \*format,...)  
*Print a warning message.*
- void [vrna\\_message\\_vwarning](#) (const char \*format, va\_list args)  
*Print a warning message.*
- void [vrna\\_message\\_info](#) (FILE \*fp, const char \*format,...)  
*Print an info message.*
- void [vrna\\_message\\_vinfo](#) (FILE \*fp, const char \*format, va\_list args)  
*Print an info message.*
- void [vrna\\_message\\_input\\_seq\\_simple](#) (void)  
*Print a line to stdout that asks for an input sequence.*
- void [vrna\\_message\\_input\\_seq](#) (const char \*s)  
*Print a line with a user defined string and a ruler to stdout.*

### 15.80.2 Function Documentation

#### 15.80.2.1 [vrna\\_message\\_error\(\)](#)

```
void vrna_message_error (  
    const char * format,  
    ... )
```

```
#include <ViennaRNA/utils/basic.h>
```

Print an error message and die.

This function is a wrapper to `fprintf(stderr, ...)` that puts a capital **ERROR:** in front of the message and then exits the calling program.

See also

[vrna\\_message\\_verror\(\)](#), [vrna\\_message\\_warning\(\)](#), [vrna\\_message\\_info\(\)](#)

## Parameters

<i>format</i>	The error message to be printed
...	Optional arguments for the formatted message string

15.80.2.2 `vrna_message_verror()`

```
void vrna_message_verror (
    const char * format,
    va_list args )
```

```
#include <ViennaRNA/utils/basic.h>
```

Print an error message and die.

This function is a wrapper to `vfprintf(stderr, ...)` that puts a capital **ERROR:** in front of the message and then exits the calling program.

## See also

[vrna\\_message\\_error\(\)](#), [vrna\\_message\\_warning\(\)](#), [vrna\\_message\\_info\(\)](#)

## Parameters

<i>format</i>	The error message to be printed
<i>args</i>	The argument list for the formatted message string

15.80.2.3 `vrna_message_warning()`

```
void vrna_message_warning (
    const char * format,
    ... )
```

```
#include <ViennaRNA/utils/basic.h>
```

Print a warning message.

This function is a wrapper to `fprintf(stderr, ...)` that puts a capital **WARNING:** in front of the message.

## See also

[vrna\\_message\\_vwarning\(\)](#), [vrna\\_message\\_error\(\)](#), [vrna\\_message\\_info\(\)](#)

## Parameters

<i>format</i>	The warning message to be printed
...	Optional arguments for the formatted message string

15.80.2.4 `vrna_message_vwarning()`

```
void vrna_message_vwarning (
    const char * format,
    va_list args )
```

```
#include <ViennaRNA/utils/basic.h>
```

Print a warning message.

This function is a wrapper to `fprintf(stderr, ...)` that puts a capital **WARNING:** in front of the message.

## See also

[vrna\\_message\\_vwarning\(\)](#), [vrna\\_message\\_error\(\)](#), [vrna\\_message\\_info\(\)](#)

## Parameters

<i>format</i>	The warning message to be printed
<i>args</i>	The argument list for the formatted message string

15.80.2.5 `vrna_message_info()`

```
void vrna_message_info (
    FILE * fp,
    const char * format,
    ... )
```

```
#include <ViennaRNA/utils/basic.h>
```

Print an info message.

This function is a wrapper to `fprintf(...)`.

## See also

[vrna\\_message\\_vinfo\(\)](#), [vrna\\_message\\_error\(\)](#), [vrna\\_message\\_warning\(\)](#)

## Parameters

<i>fp</i>	The file pointer where the message is printed to
<i>format</i>	The warning message to be printed
...	Optional arguments for the formatted message string

15.80.2.6 `vrna_message_vinfo()`

```
void vrna_message_vinfo (
    FILE * fp,
    const char * format,
    va_list args )
```

```
#include <ViennaRNA/utils/basic.h>
```

Print an info message.

This function is a wrapper to `fprintf(...)`.

## See also

[vrna\\_message\\_vinfo\(\)](#), [vrna\\_message\\_error\(\)](#), [vrna\\_message\\_warning\(\)](#)

## Parameters

<i>fp</i>	The file pointer where the message is printed to
<i>format</i>	The info message to be printed
<i>args</i>	The argument list for the formatted message string

15.80.2.7 `vrna_message_input_seq_simple()`

```
void vrna_message_input_seq_simple (
    void )
```

```
#include <ViennaRNA/utils/basic.h>
```

Print a line to `stdout` that asks for an input sequence.

There will also be a ruler (scale line) printed that helps orientation of the sequence positions



**15.80.2.8 vrna\_message\_input\_seq()**

```
void vrna_message_input_seq (  
    const char * s )
```

```
#include <ViennaRNA/utils/basic.h>
```

Print a line with a user defined string and a ruler to stdout.

(usually this is used to ask for user input) There will also be a ruler (scale line) printed that helps orientation of the sequence positions

**Parameters**

s	A user defined string that will be printed to stdout
---	--

## 15.81 Unit Conversion

Functions to convert between various physical units.

### 15.81.1 Detailed Description

Functions to convert between various physical units.

Collaboration diagram for Unit Conversion:

#### Files

- file [units.h](#)  
*Physical Units and Functions to convert them into each other.*

#### Enumerations

- enum [vrna\\_unit\\_energy\\_e](#) {  
[VRNA\\_UNIT\\_J](#), [VRNA\\_UNIT\\_KJ](#), [VRNA\\_UNIT\\_CAL\\_IT](#), [VRNA\\_UNIT\\_DACAL\\_IT](#),  
[VRNA\\_UNIT\\_KCAL\\_IT](#), [VRNA\\_UNIT\\_CAL](#), [VRNA\\_UNIT\\_DACAL](#), [VRNA\\_UNIT\\_KCAL](#),  
[VRNA\\_UNIT\\_G\\_TNT](#), [VRNA\\_UNIT\\_KG\\_TNT](#), [VRNA\\_UNIT\\_T\\_TNT](#), [VRNA\\_UNIT\\_EV](#),  
[VRNA\\_UNIT\\_WH](#), [VRNA\\_UNIT\\_KWH](#) }  
*Energy / Work Units.*
- enum [vrna\\_unit\\_temperature\\_e](#) {  
[VRNA\\_UNIT\\_K](#), [VRNA\\_UNIT\\_DEG\\_C](#), [VRNA\\_UNIT\\_DEG\\_F](#), [VRNA\\_UNIT\\_DEG\\_R](#),  
[VRNA\\_UNIT\\_DEG\\_N](#), [VRNA\\_UNIT\\_DEG\\_DE](#), [VRNA\\_UNIT\\_DEG\\_RE](#), [VRNA\\_UNIT\\_DEG\\_RO](#) }  
*Temperature Units.*

#### Functions

- double [vrna\\_convert\\_energy](#) (double energy, [vrna\\_unit\\_energy\\_e](#) from, [vrna\\_unit\\_energy\\_e](#) to)  
*Convert between energy / work units.*
- double [vrna\\_convert\\_temperature](#) (double temp, [vrna\\_unit\\_temperature\\_e](#) from, [vrna\\_unit\\_temperature\\_e](#) to)  
*Convert between temperature units.*

### 15.81.2 Enumeration Type Documentation

#### 15.81.2.1 vrna\_unit\_energy\_e

```
enum vrna_unit_energy_e
#include <ViennaRNA/units.h>
```

Energy / Work Units.

See also

[vrna\\_convert\\_energy\(\)](#)

## Enumerator

VRNA_UNIT_J	Joule ( $1 J = 1 kg \cdot m^2 s^{-2}$ )
VRNA_UNIT_KJ	Kilojoule ( $1 kJ = 1,000 J$ )
VRNA_UNIT_CAL_IT	Calorie (International (Steam) Table, $1 cal_{IT} = 4.1868 J$ )
VRNA_UNIT_DACAL_IT	Decacalorie (International (Steam) Table, $1 dacal_{IT} = 10 cal_{IT} = 41.868 J$ )
VRNA_UNIT_KCAL_IT	Kilocalorie (International (Steam) Table, $1 kcal_{IT} = 4.1868 kJ$ )
VRNA_UNIT_CAL	Calorie (Thermochemical, $1 cal_{th} = 4.184 J$ )
VRNA_UNIT_DACAL	Decacalorie (Thermochemical, $1 dacal_{th} = 10 cal_{th} = 41.84 J$ )
VRNA_UNIT_KCAL	Kilocalorie (Thermochemical, $1 kcal_{th} = 4.184 kJ$ )
VRNA_UNIT_G_TNT	g TNT ( $1 g \text{ TNT} = 1,000 cal_{th} = 4,184 J$ )
VRNA_UNIT_KG_TNT	kg TNT ( $1 kg \text{ TNT} = 1,000 kcal_{th} = 4,184 kJ$ )
VRNA_UNIT_T_TNT	ton TNT ( $1 t \text{ TNT} = 1,000,000 kcal_{th} = 4,184 MJ$ )
VRNA_UNIT_EV	Electronvolt ( $1 eV = 1.602176565 \times 10^{-19} J$ )
VRNA_UNIT_WH	Watt hour ( $1 W \cdot h = 1 W \cdot 3,600 s = 3,600 J = 3.6 kJ$ )
VRNA_UNIT_KWH	Kilowatt hour ( $1 kW \cdot h = 1 kW \cdot 3,600 s = 3,600 kJ = 3.6 MJ$ )

## 15.81.2.2 vrna\_unit\_temperature\_e

```
enum vrna_unit_temperature_e
```

```
#include <ViennaRNA/units.h>
```

Temperature Units.

See also

```
vrna_convert_temperature()
```

## Enumerator

VRNA_UNIT_K	Kelvin (K)
VRNA_UNIT_DEG_C	Degree Celcius (°C) ( $[^{\circ}C] = [K] - 273.15$ )
VRNA_UNIT_DEG_F	Degree Fahrenheit (°F) ( $[^{\circ}F] = [K] \times \frac{9}{5} - 459.67$ )
VRNA_UNIT_DEG_R	Degree Rankine (°R) ( $[^{\circ}R] = [K] \times \frac{9}{5}$ )
VRNA_UNIT_DEG_N	Degree Newton (°N) ( $[^{\circ}N] = ([K] - 273.15) \times \frac{33}{100}$ )
VRNA_UNIT_DEG_DE	Degree Delisle (°De) ( $[^{\circ}De] = (373.15 - [K]) \times \frac{3}{2}$ )
VRNA_UNIT_DEG_RE	Degree Réaumur (°Ré) ( $[^{\circ}Re] = ([K] - 273.15) \times \frac{4}{5}$ )
VRNA_UNIT_DEG_RO	Degree Rømer (°Rø) ( $[^{\circ}R] = ([K] - 273.15) \times \frac{21}{40} + 7.5$ )

### 15.81.3 Function Documentation

#### 15.81.3.1 `vrna_convert_energy()`

```
double vrna_convert_energy (
    double energy,
    vrna_unit_energy_e from,
    vrna_unit_energy_e to )
```

```
#include <ViennaRNA/units.h>
```

Convert between energy / work units.

See also

[vrna\\_unit\\_energy\\_e](#)

##### Parameters

<i>energy</i>	Input energy value
<i>from</i>	Input unit
<i>to</i>	Output unit

##### Returns

Energy value in Output unit

#### 15.81.3.2 `vrna_convert_temperature()`

```
double vrna_convert_temperature (
    double temp,
    vrna_unit_temperature_e from,
    vrna_unit_temperature_e to )
```

```
#include <ViennaRNA/units.h>
```

Convert between temperature units.

See also

[vrna\\_unit\\_temperature\\_e](#)

##### Parameters

<i>temp</i>	Input temperature value
<i>from</i>	Input unit
<i>to</i>	Output unit

**Returns**

Temperature value in Output unit

## 15.82 The Fold Compound

This module provides interfaces that deal with the most basic data structure used in structure predicting and energy evaluating function of the RNAlib.

### 15.82.1 Detailed Description

This module provides interfaces that deal with the most basic data structure used in structure predicting and energy evaluating function of the RNAlib.

Throughout the entire RNAlib, the `vrna_fold_compound_t`, is used to group information and data that is required for structure prediction and energy evaluation. Here, you'll find interface functions to create, modify, and delete `vrna_fold_compound_t` data structures. Collaboration diagram for The Fold Compound:

#### Files

- file `fold_compound.h`  
*The Basic Fold Compound API.*

#### Data Structures

- struct `vrna_fc_s`  
*The most basic data structure required by many functions throughout the RNAlib. [More...](#)*

#### Macros

- `#define VRNA_STATUS_MFE_PRE` (unsigned char)1  
*Status message indicating that MFE computations are about to begin.*
- `#define VRNA_STATUS_MFE_POST` (unsigned char)2  
*Status message indicating that MFE computations are finished.*
- `#define VRNA_STATUS_PF_PRE` (unsigned char)3  
*Status message indicating that Partition function computations are about to begin.*
- `#define VRNA_STATUS_PF_POST` (unsigned char)4  
*Status message indicating that Partition function computations are finished.*
- `#define VRNA_OPTION_DEFAULT` 0U  
*Option flag to specify default settings/requirements.*
- `#define VRNA_OPTION_MFE` 1U  
*Option flag to specify requirement of Minimum Free Energy (MFE) DP matrices and corresponding set of energy parameters.*
- `#define VRNA_OPTION_PF` 2U  
*Option flag to specify requirement of Partition Function (PF) DP matrices and corresponding set of Boltzmann factors.*
- `#define VRNA_OPTION_HYBRID` 4U  
*Option flag to specify requirement of dimer DP matrices.*
- `#define VRNA_OPTION_EVAL_ONLY` 8U  
*Option flag to specify that neither MFE, nor PF DP matrices are required.*
- `#define VRNA_OPTION_WINDOW` 16U  
*Option flag to specify requirement of DP matrices for local folding approaches.*

## Typedefs

- typedef struct [vrna\\_fc\\_s](#) [vrna\\_fold\\_compound\\_t](#)  
*Typename for the fold\_compound data structure [vrna\\_fc\\_s](#).*
- typedef void() [vrna\\_callback\\_free\\_auxdata](#)(void \*data)  
*Callback to free memory allocated for auxiliary user-provided data.*
- typedef void() [vrna\\_callback\\_recursion\\_status](#)(unsigned char status, void \*data)  
*Callback to perform specific user-defined actions before, or after recursive computations.*

## Enumerations

- enum [vrna\\_fc\\_type\\_e](#) { [VRNA\\_FC\\_TYPE\\_SINGLE](#), [VRNA\\_FC\\_TYPE\\_COMPARATIVE](#) }  
*An enumerator that is used to specify the type of a [vrna\\_fold\\_compound\\_t](#).*

## Functions

- [vrna\\_fold\\_compound\\_t](#) \* [vrna\\_fold\\_compound](#) (const char \*sequence, [vrna\\_md\\_t](#) \*md\_p, unsigned int options)  
*Retrieve a [vrna\\_fold\\_compound\\_t](#) data structure for single sequences and hybridizing sequences.*
- [vrna\\_fold\\_compound\\_t](#) \* [vrna\\_fold\\_compound\\_comparative](#) (const char \*\*sequences, [vrna\\_md\\_t](#) \*md\_p, unsigned int options)  
*Retrieve a [vrna\\_fold\\_compound\\_t](#) data structure for sequence alignments.*
- void [vrna\\_fold\\_compound\\_free](#) ([vrna\\_fold\\_compound\\_t](#) \*fc)  
*Free memory occupied by a [vrna\\_fold\\_compound\\_t](#).*
- void [vrna\\_fold\\_compound\\_add\\_auxdata](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, void \*data, [vrna\\_callback\\_free\\_auxdata](#) \*f)  
*Add auxiliary data to the [vrna\\_fold\\_compound\\_t](#).*
- void [vrna\\_fold\\_compound\\_add\\_callback](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, [vrna\\_callback\\_recursion\\_status](#) \*f)  
*Add a recursion status callback to the [vrna\\_fold\\_compound\\_t](#).*

## 15.82.2 Data Structure Documentation

### 15.82.2.1 struct [vrna\\_fc\\_s](#)

The most basic data structure required by many functions throughout the RNAlib.

#### Note

Please read the documentation of this data structure carefully! Some attributes are only available for specific types this data structure can adopt.

#### Warning

Reading/Writing from/to attributes that are not within the scope of the current type usually result in undefined behavior!



See also

[vrna\\_fold\\_compound\\_t.type](#), [vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_comparative\(\)](#), [vrna\\_fold\\_compound\\_free\(\)](#), [VRNA\\_FC\\_TYPE\\_SINGLE](#), [VRNA\\_FC\\_TYPE\\_COMPARATIVE](#)

**SWIG Wrapper Notes** This data structure is wrapped as an object **fold\_compound** with several related functions attached as methods.

A new **fold\_compound** can be obtained by calling one of its constructors:

- *fold\_compound(seq)* – Initialize with a single sequence, or two concatenated sequences separated by an ampersand character '&' (for cofolding)
- *fold\_compound(aln)* – Initialize with a sequence alignment *aln* stored as a list of sequences (with gap characters)

The resulting object has a list of attached methods which in most cases directly correspond to functions that mainly operate on the corresponding C data structure:

- *type()* – Get the type of the *fold\_compound* (See [vrna\\_fc\\_type\\_e](#))
- *length()* – Get the length of the sequence(s) or alignment stored within the *fold\_compound*

Collaboration diagram for `vrna_fc_s`:

## Data Fields

### Common data fields

- const [vrna\\_fc\\_type\\_e](#) *type*  
*The type of the [vrna\\_fold\\_compound\\_t](#).*
- unsigned int [length](#)  
*The length of the sequence (or sequence alignment)*
- int [cutpoint](#)  
*The position of the (cofold) cutpoint within the provided sequence. If there is no cutpoint, this field will be set to -1.*
- unsigned int \* [strand\\_number](#)  
*The strand number a particular nucleotide is associated with.*
- unsigned int \* **strand\_order**
- unsigned int \* **strand\_start**
- unsigned int \* **strand\_end**
- unsigned int **strands**
- [vrna\\_seq\\_t](#) \* **nucleotides**
- [vrna\\_hc\\_t](#) \* **hc**  
*The hard constraints data structure used for structure prediction.*
- [vrna\\_mx\\_mfe\\_t](#) \* **matrices**  
*The MFE DP matrices.*
- [vrna\\_mx\\_pf\\_t](#) \* **exp\_matrices**  
*The PF DP matrices.*
- [vrna\\_param\\_t](#) \* **params**  
*The precomputed free energy contributions for each type of loop.*
- [vrna\\_exp\\_param\\_t](#) \* **exp\_params**  
*The precomputed free energy contributions as Boltzmann factors.*
- int \* **iindx**  
*DP matrix accessor.*
- int \* **jindx**

*DP matrix accessor.*

### User-defined data fields

- `vrna_callback_recursion_status * stat_cb`  
*Recursion status callback (usually called just before, and after recursive computations in the library).*
- `void * auxdata`  
*A pointer to auxiliary, user-defined data.*
- `vrna_callback_free_auxdata * free_auxdata`  
*A callback to free auxiliary user data whenever the fold\_compound itself is free'd.*

### Secondary Structure Decomposition (grammar) related data fields

- `vrna_sd_t * domains_struct`  
*Additional structured domains.*
- `vrna_ud_t * domains_up`  
*Additional unstructured domains.*
- `vrna_gr_aux_t * aux_grammar`

### Data fields available for single/hybrid structure prediction

### Data fields for consensus structure prediction

### Additional data fields for Distance Class Partitioning

*These data fields are typically populated with meaningful data only if used in the context of Distance Class Partitioning*

- `unsigned int maxD1`  
*Maximum allowed base pair distance to first reference.*
- `unsigned int maxD2`  
*Maximum allowed base pair distance to second reference.*
- `short * reference_pt1`  
*A pairtable of the first reference structure.*
- `short * reference_pt2`  
*A pairtable of the second reference structure.*
- `unsigned int * referenceBPs1`  
*Matrix containing number of basepairs of reference structure1 in interval [i,j].*
- `unsigned int * referenceBPs2`  
*Matrix containing number of basepairs of reference structure2 in interval [i,j].*
- `unsigned int * bpdist`  
*Matrix containing base pair distance of reference structure 1 and 2 on interval [i,j].*
- `unsigned int * mm1`  
*Maximum matching matrix, reference struct 1 disallowed.*
- `unsigned int * mm2`  
*Maximum matching matrix, reference struct 2 disallowed.*

### Additional data fields for local folding

*These data fields are typically populated with meaningful data only if used in the context of local folding*

- `int window_size`  
*window size for local folding sliding window approach*
- `char ** ptype_local`  
*Pair type array (for local folding)*

### 15.82.2.1.1 Field Documentation

#### 15.82.2.1.1.1 type

```
const vrna\_fc\_type\_e vrna_fc_s::type
```

The type of the [vrna\\_fold\\_compound\\_t](#).

Currently possible values are [VRNA\\_FC\\_TYPE\\_SINGLE](#), and [VRNA\\_FC\\_TYPE\\_COMPARATIVE](#)

#### Warning

Do not edit this attribute, it will be automagically set by the corresponding `get()` methods for the [vrna\\_fold\\_compound\\_t](#). The value specified in this attribute dictates the set of other attributes to use within this data structure.

#### 15.82.2.1.1.2 stat\_cb

```
vrna\_callback\_recursion\_status\* vrna_fc_s::stat_cb
```

Recursion status callback (usually called just before, and after recursive computations in the library.

#### See also

[vrna\\_callback\\_recursion\\_status\(\)](#), [vrna\\_fold\\_compound\\_add\\_callback\(\)](#)

#### 15.82.2.1.1.3 auxdata

```
void* vrna_fc_s::auxdata
```

A pointer to auxiliary, user-defined data.

#### See also

[vrna\\_fold\\_compound\\_add\\_auxdata\(\)](#), [vrna\\_fold\\_compound\\_t.free\\_auxdata](#)

#### 15.82.2.1.1.4 free\_auxdata

```
vrna\_callback\_free\_auxdata\* vrna_fc_s::free_auxdata
```

A callback to free auxiliary user data whenever the `fold_compound` itself is free'd.

#### See also

[vrna\\_fold\\_compound\\_t.auxdata](#), [vrna\\_callback\\_free\\_auxdata\(\)](#)

#### 15.82.2.1.1.5 sequence

```
char* vrna_fc_s::sequence
```

The input sequence string.

##### Warning

Only available if

```
type==VRNA_FC_TYPE_SINGLE
```

#### 15.82.2.1.1.6 sequence\_encoding

```
short* vrna_fc_s::sequence_encoding
```

Numerical encoding of the input sequence.

##### See also

```
vrna_sequence_encode()
```

##### Warning

Only available if

```
type==VRNA_FC_TYPE_SINGLE
```

#### 15.82.2.1.1.7 ptype

```
char* vrna_fc_s::ptype
```

Pair type array.

Contains the numerical encoding of the pair type for each pair (i,j) used in MFE, Partition function and Evaluation computations.

##### Note

This array is always indexed via jindx, in contrast to previously different indexing between mfe and pf variants!

##### Warning

Only available if

```
type==VRNA_FC_TYPE_SINGLE
```

##### See also

```
vrna\_idx\_col\_wise\(\), vrna\_ptypes\(\)
```

#### 15.82.2.1.1.8 ptype\_pf\_compat

```
char* vrna_fc_s::ptype_pf_compat
```

ptype array indexed via iindx

**Deprecated** This attribute will vanish in the future! It's meant for backward compatibility only!

#### Warning

Only available if

```
type==VRNA_FC_TYPE_SINGLE
```

#### 15.82.2.1.1.9 sc

```
vrna_sc_t* vrna_fc_s::sc
```

The soft constraints for usage in structure prediction and evaluation.

#### Warning

Only available if

```
type==VRNA_FC_TYPE_SINGLE
```

#### 15.82.2.1.1.10 sequences

```
char** vrna_fc_s::sequences
```

The aligned sequences.

#### Note

The end of the alignment is indicated by a NULL pointer in the second dimension

#### Warning

Only available if

```
type==VRNA_FC_TYPE_COMPARATIVE
```

#### 15.82.2.1.1.11 n\_seq

```
unsigned int vrna_fc_s::n_seq
```

The number of sequences in the alignment.

##### Warning

Only available if

```
type==VRNA_FC_TYPE_COMPARATIVE
```

#### 15.82.2.1.1.12 cons\_seq

```
char* vrna_fc_s::cons_seq
```

The consensus sequence of the aligned sequences.

##### Warning

Only available if

```
type==VRNA_FC_TYPE_COMPARATIVE
```

#### 15.82.2.1.1.13 S\_cons

```
short* vrna_fc_s::S_cons
```

Numerical encoding of the consensus sequence.

##### Warning

Only available if

```
type==VRNA_FC_TYPE_COMPARATIVE
```

#### 15.82.2.1.1.14 S

```
short** vrna_fc_s::S
```

Numerical encoding of the sequences in the alignment.

##### Warning

Only available if

```
type==VRNA_FC_TYPE_COMPARATIVE
```

**15.82.2.1.1.15 S5**

```
short** vrna_fc_s::S5
```

S5[s][i] holds next base 5' of i in sequence s.

**Warning**

Only available if

```
type==VRNA_FC_TYPE_COMPARATIVE
```

**15.82.2.1.1.16 S3**

```
short** vrna_fc_s::S3
```

S3[s][i] holds next base 3' of i in sequence s.

**Warning**

Only available if

```
type==VRNA_FC_TYPE_COMPARATIVE
```

**15.82.2.1.1.17 pscore**

```
int* vrna_fc_s::pscore
```

Precomputed array of pair types expressed as pairing scores.

**Warning**

Only available if

```
type==VRNA_FC_TYPE_COMPARATIVE
```

**15.82.2.1.1.18 pscore\_local**

```
int** vrna_fc_s::pscore_local
```

Precomputed array of pair types expressed as pairing scores.

**Warning**

Only available if

```
type==VRNA_FC_TYPE_COMPARATIVE
```

#### 15.82.2.1.1.19 pscore\_pf\_compat

```
short* vrna_fc_s::pscore_pf_compat
```

Precomputed array of pair types expressed as pairing scores indexed via iindx.

**Deprecated** This attribute will vanish in the future!

#### Warning

Only available if

```
type==VRNA_FC_TYPE_COMPARATIVE
```

#### 15.82.2.1.1.20 scs

```
vrna_sc_t** vrna_fc_s::scs
```

A set of soft constraints (for each sequence in the alignment)

#### Warning

Only available if

```
type==VRNA_FC_TYPE_COMPARATIVE
```

### 15.82.3 Macro Definition Documentation

#### 15.82.3.1 VRNA\_STATUS\_MFE\_PRE

```
#define VRNA_STATUS_MFE_PRE (unsigned char)1
```

```
#include <ViennaRNA/fold_compound.h>
```

Status message indicating that MFE computations are about to begin.

#### See also

[vrna\\_fold\\_compound\\_t.stat\\_cb](#), [vrna\\_callback\\_recursion\\_status\(\)](#), [vrna\\_mfe\(\)](#), [vrna\\_fold\(\)](#), [vrna\\_circfold\(\)](#), [vrna\\_alifold\(\)](#), [vrna\\_circalifold\(\)](#), [vrna\\_cofold\(\)](#)



### 15.82.3.2 VRNA\_STATUS\_MFE\_POST

```
#define VRNA_STATUS_MFE_POST (unsigned char)2  
  
#include <ViennaRNA/fold_compound.h>
```

Status message indicating that MFE computations are finished.

See also

[vrna\\_fold\\_compound\\_t.stat\\_cb](#), [vrna\\_callback\\_recursion\\_status\(\)](#), [vrna\\_mfe\(\)](#), [vrna\\_fold\(\)](#), [vrna\\_circfold\(\)](#),  
[vrna\\_alifold\(\)](#), [vrna\\_circalifold\(\)](#), [vrna\\_cofold\(\)](#)

### 15.82.3.3 VRNA\_STATUS\_PF\_PRE

```
#define VRNA_STATUS_PF_PRE (unsigned char)3  
  
#include <ViennaRNA/fold_compound.h>
```

Status message indicating that Partition function computations are about to begin.

See also

[vrna\\_fold\\_compound\\_t.stat\\_cb](#), [vrna\\_callback\\_recursion\\_status\(\)](#), [vrna\\_pf\(\)](#)

### 15.82.3.4 VRNA\_STATUS\_PF\_POST

```
#define VRNA_STATUS_PF_POST (unsigned char)4  
  
#include <ViennaRNA/fold_compound.h>
```

Status message indicating that Partition function computations are finished.

See also

[vrna\\_fold\\_compound\\_t.stat\\_cb](#), [vrna\\_callback\\_recursion\\_status\(\)](#), [vrna\\_pf\(\)](#)

### 15.82.3.5 VRNA\_OPTION\_MFE

```
#define VRNA_OPTION_MFE 1U  
  
#include <ViennaRNA/fold_compound.h>
```

Option flag to specify requirement of Minimum Free Energy (MFE) DP matrices and corresponding set of energy parameters.

See also

[vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_comparative\(\)](#), [VRNA\\_OPTION\\_EVAL\\_ONLY](#)

### 15.82.3.6 VRNA\_OPTION\_PF

```
#define VRNA_OPTION_PF 2U  
  
#include <ViennaRNA/fold_compound.h>
```

Option flag to specify requirement of Partition Function (PF) DP matrices and corresponding set of Boltzmann factors.

See also

[vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_comparative\(\)](#), [VRNA\\_OPTION\\_EVAL\\_ONLY](#)

### 15.82.3.7 VRNA\_OPTION\_EVAL\_ONLY

```
#define VRNA_OPTION_EVAL_ONLY 8U  
  
#include <ViennaRNA/fold_compound.h>
```

Option flag to specify that neither MFE, nor PF DP matrices are required.

Use this flag in conjunction with [VRNA\\_OPTION\\_MFE](#), and [VRNA\\_OPTION\\_PF](#) to save memory for a [vrna\\_fold\\_compound\\_t](#) obtained from [vrna\\_fold\\_compound\(\)](#), or [vrna\\_fold\\_compound\\_comparative\(\)](#) in cases where only energy evaluation but no structure prediction is required.

See also

[vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_comparative\(\)](#), [vrna\\_eval\\_structure\(\)](#)

## 15.82.4 Typedef Documentation

### 15.82.4.1 vrna\_callback\_free\_auxdata

```
typedef void() vrna_callback_free_auxdata(void *data)  
  
#include <ViennaRNA/fold_compound.h>
```

Callback to free memory allocated for auxiliary user-provided data.

This type of user-implemented function usually deletes auxiliary data structures. The user must take care to free all the memory occupied by the data structure passed.

**Notes on Callback Functions** This callback is supposed to free memory occupied by an auxiliary data structure. It will be called when the [vrna\\_fold\\_compound\\_t](#) is erased from memory through a call to [vrna\\_fold\\_compound\\_free\(\)](#) and will be passed the address of memory previously bound to the [vrna\\_fold\\_compound\\_t](#) via [vrna\\_fold\\_compound\\_add\\_auxdata\(\)](#).

See also

[vrna\\_fold\\_compound\\_add\\_auxdata\(\)](#), [vrna\\_fold\\_compound\\_free\(\)](#), [vrna\\_fold\\_compound\\_add\\_callback\(\)](#)

## Parameters

<i>data</i>	The data that needs to be free'd
-------------	----------------------------------

15.82.4.2 `vrna_callback_recursion_status`

```
typedef void() vrna_callback_recursion_status(unsigned char status, void *data)
```

```
#include <ViennaRNA/fold_compound.h>
```

Callback to perform specific user-defined actions before, or after recursive computations.

**Notes on Callback Functions** This function will be called to notify a third-party implementation about the status of a currently ongoing recursion. The purpose of this callback mechanism is to provide users with a simple way to ensure pre- and post conditions for auxiliary mechanisms attached to our implementations.

## See also

[vrna\\_fold\\_compound\\_add\\_auxdata\(\)](#), [vrna\\_fold\\_compound\\_add\\_callback\(\)](#), [vrna\\_mfe\(\)](#), [vrna\\_pf\(\)](#), [VRNA\\_STATUS\\_MFE\\_PRE](#), [VRNA\\_STATUS\\_MFE\\_POST](#), [VRNA\\_STATUS\\_PF\\_PRE](#), [VRNA\\_STATUS\\_PF\\_POST](#)

## Parameters

<i>status</i>	The status indicator
<i>data</i>	The data structure that was assigned with <a href="#">vrna_fold_compound_add_auxdata()</a>
<i>status</i>	The status indicator

## 15.82.5 Enumeration Type Documentation

15.82.5.1 `vrna_fc_type_e`

```
enum vrna_fc_type_e
```

```
#include <ViennaRNA/fold_compound.h>
```

An enumerator that is used to specify the type of a [vrna\\_fold\\_compound\\_t](#).

## Enumerator

VRNA_FC_TYPE_SINGLE	Type is suitable for single, and hybridizing sequences
VRNA_FC_TYPE_COMPARATIVE	Type is suitable for sequence alignments (consensus structure prediction)

## 15.82.6 Function Documentation

### 15.82.6.1 `vrna_fold_compound()`

```
vrna_fold_compound_t* vrna_fold_compound (
    const char * sequence,
    vrna_md_t * md_p,
    unsigned int options )

#include <ViennaRNA/fold_compound.h>
```

Retrieve a `vrna_fold_compound_t` data structure for single sequences and hybridizing sequences.

This function provides an easy interface to obtain a prefilled `vrna_fold_compound_t` by passing a single sequence, or two concatenated sequences as input. For the latter, sequences need to be separated by an '&' character like this:

```
char *sequence = "GGGG&CCCC";
```

The optional parameter `md_p` can be used to specify the model details for successive computations based on the content of the generated `vrna_fold_compound_t`. Passing NULL will instruct the function to use default model details. The third parameter `options` may be used to specify dynamic programming (DP) matrix requirements.

#### Options

- `VRNA_OPTION_DEFAULT` - Option flag to specify default settings/requirements.
- `VRNA_OPTION_MFE` - Option flag to specify requirement of Minimum Free Energy (MFE) DP matrices and corresponding set of energy parameters.
- `VRNA_OPTION_PF` - Option flag to specify requirement of Partition Function (PF) DP matrices and corresponding set of Boltzmann factors.
- `VRNA_OPTION_WINDOW` - Option flag to specify requirement of DP matrices for local folding approaches.

The above options may be OR-ed together.

If you just need the folding compound serving as a container for your data, you can simply pass `VRNA_OPTION_DEFAULT` to the `option` parameter. This creates a `vrna_fold_compound_t` without DP matrices, thus saving memory. Subsequent calls of any structure prediction function will then take care of allocating the memory required for the DP matrices. If you only intend to evaluate structures instead of actually predicting them, you may use the `VRNA_OPTION_EVAL_ONLY` macro. This will seriously speedup the creation of the `vrna_fold_compound_t`.

#### Note

The sequence string must be uppercase, and should contain only RNA (resp. DNA) alphabet depending on what energy parameter set is used

#### See also

[vrna\\_fold\\_compound\\_free\(\)](#), [vrna\\_fold\\_compound\\_comparative\(\)](#), [vrna\\_md\\_t](#)

## Parameters

<i>sequence</i>	A single sequence, or two concatenated sequences seperated by an '&' character
<i>md_p</i>	An optional set of model details
<i>options</i>	The options for DP matrices memory allocation

## Returns

A prefilled `vrna_fold_compound_t` ready to be used for computations (may be `NULL` on error)

15.82.6.2 `vrna_fold_compound_comparative()`

```
vrna_fold_compound_t* vrna_fold_compound_comparative (
    const char ** sequences,
    vrna_md_t * md_p,
    unsigned int options )
```

```
#include <ViennaRNA/fold_compound.h>
```

Retrieve a `vrna_fold_compound_t` data structure for sequence alignments.

This function provides an easy interface to obtain a prefilled `vrna_fold_compound_t` by passing an alignment of sequences.

The optional parameter `md_p` can be used to specify the model details for successive computations based on the content of the generated `vrna_fold_compound_t`. Passing `NULL` will instruct the function to use default model details. The third parameter `options` may be used to specify dynamic programming (DP) matrix requirements.

## Options

- `VRNA_OPTION_DEFAULT` - Option flag to specify default settings/requirements.
- `VRNA_OPTION_MFE` - Option flag to specify requirement of Minimum Free Energy (MFE) DP matrices and corresponding set of energy parameters.
- `VRNA_OPTION_PF` - Option flag to specify requirement of Partition Function (PF) DP matrices and corresponding set of Boltzmann factors.
- `VRNA_OPTION_WINDOW` - Option flag to specify requirement of DP matrices for local folding approaches.

The above options may be OR-ed together.

If you just need the folding compound serving as a container for your data, you can simply pass `VRNA_OPTION_DEFAULT` to the `option` parameter. This creates a `vrna_fold_compound_t` without DP matrices, thus saving memory. Subsequent calls of any structure prediction function will then take care of allocating the memory required for the DP matrices. If you only intend to evaluate structures instead of actually predicting them, you may use the `VRNA_OPTION_EVAL_ONLY` macro. This will seriously speedup the creation of the `vrna_fold_compound_t`.

## Note

The sequence strings must be uppercase, and should contain only RNA (resp. DNA) alphabet including gap characters depending on what energy parameter set is used.

## See also

`vrna_fold_compound_free()`, `vrna_fold_compound()`, `vrna_md_t`, `VRNA_OPTION_MFE`, `VRNA_OPTION_PF`, `VRNA_OPTION_EVAL_ONLY`, `read_clustal()`

## Parameters

<i>sequences</i>	A sequence alignment including 'gap' characters
<i>md_p</i>	An optional set of model details
<i>options</i>	The options for DP matrices memory allocation

## Returns

A prefilled `vrna_fold_compound_t` ready to be used for computations (may be `NULL` on error)

15.82.6.3 `vrna_fold_compound_free()`

```
void vrna_fold_compound_free (
    vrna_fold_compound_t * fc )
```

```
#include <ViennaRNA/fold_compound.h>
```

Free memory occupied by a `vrna_fold_compound_t`.

## See also

[vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_comparative\(\)](#), [vrna\\_mx\\_mfe\\_free\(\)](#), [vrna\\_mx\\_pf\\_free\(\)](#)

## Parameters

<i>fc</i>	The <code>vrna_fold_compound_t</code> that is to be erased from memory
-----------	--

15.82.6.4 `vrna_fold_compound_add_auxdata()`

```
void vrna_fold_compound_add_auxdata (
    vrna_fold_compound_t * fc,
    void * data,
    vrna_callback_free_auxdata * f )
```

```
#include <ViennaRNA/fold_compound.h>
```

Add auxiliary data to the `vrna_fold_compound_t`.

This function allows one to bind arbitrary data to a `vrna_fold_compound_t` which may later on be used by one of the callback functions, e.g. [vrna\\_callback\\_recursion\\_status\(\)](#). To allow for proper cleanup of the memory occupied by this auxiliary data, the user may also provide a pointer to a cleanup function that free's the corresponding memory. This function will be called automatically when the `vrna_fold_compound_t` is free'd with [vrna\\_fold\\_compound\\_free\(\)](#).

**Note**

Before attaching the arbitrary data pointer, this function will call the [vrna\\_callback\\_free\\_auxdata\(\)](#) on any pre-existing data that is already attached.

**See also**

[vrna\\_callback\\_free\\_auxdata\(\)](#)

**Parameters**

<i>fc</i>	The fold_compound the arbitrary data pointer should be associated with
<i>data</i>	A pointer to an arbitrary data structure
<i>f</i>	A pointer to function that free's memory occupied by the arbitrary data (May be NULL)

**15.82.6.5 vrna\_fold\_compound\_add\_callback()**

```
void vrna_fold_compound_add_callback (
    vrna_fold_compound_t * fc,
    vrna_callback_recursion_status * f )
```

```
#include <ViennaRNA/fold_compound.h>
```

Add a recursion status callback to the [vrna\\_fold\\_compound\\_t](#).

Binding a recursion status callback function to a [vrna\\_fold\\_compound\\_t](#) allows one to perform arbitrary operations just before, or after an actual recursive computations, e.g. MFE prediction, is performed by the RNAlib. The callback function will be provided with a pointer to its [vrna\\_fold\\_compound\\_t](#), and a status message. Hence, it has complete access to all variables that influence the recursive computations.

**See also**

[vrna\\_callback\\_recursion\\_status\(\)](#), [vrna\\_fold\\_compound\\_t](#), [VRNA\\_STATUS\\_MFE\\_PRE](#), [VRNA\\_STATUS\\_MFE\\_POST](#), [VRNA\\_STATUS\\_PF\\_PRE](#), [VRNA\\_STATUS\\_PF\\_POST](#)

**Parameters**

<i>fc</i>	The fold_compound the callback function should be attached to
<i>f</i>	The pointer to the recursion status callback function

## 15.83 The Dynamic Programming Matrices

This module provides interfaces that deal with creation and destruction of dynamic programming matrices used within the RNAlib.

### 15.83.1 Detailed Description

This module provides interfaces that deal with creation and destruction of dynamic programming matrices used within the RNAlib.

Collaboration diagram for The Dynamic Programming Matrices:

### Data Structures

- struct [vrna\\_mx\\_mfe\\_s](#)  
*Minimum Free Energy (MFE) Dynamic Programming (DP) matrices data structure required within the [vrna\\_fold\\_compound\\_t](#). [More...](#)*
- struct [vrna\\_mx\\_pf\\_s](#)  
*Partition function (PF) Dynamic Programming (DP) matrices data structure required within the [vrna\\_fold\\_compound\\_t](#). [More...](#)*

### Typedefs

- typedef struct [vrna\\_mx\\_mfe\\_s](#) [vrna\\_mx\\_mfe\\_t](#)  
*Typename for the Minimum Free Energy (MFE) DP matrices data structure [vrna\\_mx\\_mfe\\_s](#).*
- typedef struct [vrna\\_mx\\_pf\\_s](#) [vrna\\_mx\\_pf\\_t](#)  
*Typename for the Partition Function (PF) DP matrices data structure [vrna\\_mx\\_pf\\_s](#).*

### Enumerations

- enum [vrna\\_mx\\_type\\_e](#) { [VRNA\\_MX\\_DEFAULT](#), [VRNA\\_MX\\_WINDOW](#), [VRNA\\_MX\\_2DFOLD](#) }  
*An enumerator that is used to specify the type of a polymorphic Dynamic Programming (DP) matrix data structure.*

### Functions

- int [vrna\\_mx\\_add](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_mx\\_type\\_e](#) type, unsigned int options)  
*Add Dynamic Programming (DP) matrices (allocate memory)*
- void [vrna\\_mx\\_mfe\\_free](#) ([vrna\\_fold\\_compound\\_t](#) \*vc)  
*Free memory occupied by the Minimum Free Energy (MFE) Dynamic Programming (DP) matrices.*
- void [vrna\\_mx\\_pf\\_free](#) ([vrna\\_fold\\_compound\\_t](#) \*vc)  
*Free memory occupied by the Partition Function (PF) Dynamic Programming (DP) matrices.*



## 15.83.2 Data Structure Documentation

### 15.83.2.1 struct vrna\_mx\_mfe\_s

Minimum Free Energy (MFE) Dynamic Programming (DP) matrices data structure required within the [vrna\\_fold\\_compound\\_t](#).

#### Data Fields

##### Common fields for MFE matrices

- [vrna\\_mx\\_type\\_e](#) type
- unsigned int [length](#)

*Length of the sequence, therefore an indicator of the size of the DP matrices.*

##### Default DP matrices

###### Note

*These data fields are available if*

```
vrna_mx_mfe_t.type == VRNA_MX_DEFAULT
```

##### Local Folding DP matrices using window approach

###### Note

*These data fields are available if*

```
vrna_mx_mfe_t.type == VRNA_MX_WINDOW
```

##### Distance Class DP matrices

###### Note

*These data fields are available if*

```
vrna_mx_mfe_t.type == VRNA_MX_2DFOLD
```

### 15.83.2.2 struct vrna\_mx\_pf\_s

Partition function (PF) Dynamic Programming (DP) matrices data structure required within the [vrna\\_fold\\_compound\\_t](#).

## Data Fields

## Common fields for DP matrices

- [vrna\\_mx\\_type\\_e](#) type
- unsigned int **length**
- [FLT\\_OR\\_DBL](#) \* **scale**
- [FLT\\_OR\\_DBL](#) \* **expMLbase**

## Default PF matrices

## Note

*These data fields are available if*

```
vrna_mx_pf_t.type == VRNA_MX_DEFAULT
```

## Local Folding DP matrices using window approach

## Note

*These data fields are available if*

```
vrna_mx_mfe_t.type == VRNA_MX_WINDOW
```

## Distance Class DP matrices

## Note

*These data fields are available if*

```
vrna_mx_pf_t.type == VRNA_MX_2DFOLD
```

## 15.83.3 Enumeration Type Documentation

## 15.83.3.1 vrna\_mx\_type\_e

```
enum vrna_mx_type_e
```

```
#include <ViennaRNA/dp_matrices.h>
```

An enumerator that is used to specify the type of a polymorphic Dynamic Programming (DP) matrix data structure.

## See also

[vrna\\_mx\\_mfe\\_t](#), [vrna\\_mx\\_pf\\_t](#)

## Enumerator

VRNA_MX_DEFAULT	Default DP matrices.
VRNA_MX_WINDOW	DP matrices suitable for local structure prediction using window approach.  See also  <a href="#">vrna_mfe_window()</a> , <a href="#">vrna_mfe_window_zscore()</a> , <a href="#">pfl_fold()</a>
VRNA_MX_2DFOLD	DP matrices suitable for distance class partitioned structure prediction.  See also  <a href="#">vrna_mfe_TwoD()</a> , <a href="#">vrna_pf_TwoD()</a>

### 15.83.4 Function Documentation

#### 15.83.4.1 `vrna_mx_add()`

```
int vrna_mx_add (
    vrna_fold_compound_t * vc,
    vrna_mx_type_e type,
    unsigned int options )

#include <ViennaRNA/dp_matrices.h>
```

Add Dynamic Programming (DP) matrices (allocate memory)

This function adds DP matrices of a specific type to the provided `vrna_fold_compound_t`, such that successive DP recursion can be applied. The function caller has to specify which type of DP matrix is requested, see `vrna_mx_type_e`, and what kind of recursive algorithm will be applied later on, using the parameters `type`, and `options`, respectively. For the latter, Minimum free energy (MFE), and Partition function (PF) computations are distinguished. A third option that may be passed is `VRNA_OPTION_HYBRID`, indicating that auxiliary DP arrays are required for RNA-RNA interaction prediction.

#### Note

Usually, there is no need to call this function, since the constructors of `vrna_fold_compound_t` are handling all the DP matrix memory allocation.

#### See also

`vrna_mx_mfe_add()`, `vrna_mx_pf_add()`, `vrna_fold_compound()`, `vrna_fold_compound_comparative()`, `vrna_fold_compound_free()`, `vrna_mx_pf_free()`, `vrna_mx_mfe_free()`, `vrna_mx_type_e`, `VRNA_OPTION_MFE`, `VRNA_OPTION_PF`, `VRNA_OPTION_HYBRID`, `VRNA_OPTION_EVAL_ONLY`

#### Parameters

<code>vc</code>	The <code>vrna_fold_compound_t</code> that holds pointers to the DP matrices
<code>type</code>	The type of DP matrices requested
<code>options</code>	Option flags that specify the kind of DP matrices, such as MFE or PF arrays, and auxiliary requirements

#### Returns

1 if DP matrices were properly allocated and attached, 0 otherwise

#### 15.83.4.2 `vrna_mx_mfe_free()`

```
void vrna_mx_mfe_free (
    vrna_fold_compound_t * vc )

#include <ViennaRNA/dp_matrices.h>
```

Free memory occupied by the Minimum Free Energy (MFE) Dynamic Programming (DP) matrices.

See also

[vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_comparative\(\)](#), [vrna\\_fold\\_compound\\_free\(\)](#), [vrna\\_mx\\_pf\\_free\(\)](#)

Parameters

vc	The <a href="#">vrna_fold_compound_t</a> storing the MFE DP matrices that are to be erased from memory
----	--

#### 15.83.4.3 vrna\_mx\_pf\_free()

```
void vrna_mx_pf_free (
    vrna_fold_compound_t * vc )
```

```
#include <ViennaRNA/dp_matrices.h>
```

Free memory occupied by the Partition Function (PF) Dynamic Programming (DP) matrices.

See also

[vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_comparative\(\)](#), [vrna\\_fold\\_compound\\_free\(\)](#), [vrna\\_mx\\_mfe\\_free\(\)](#)

Parameters

vc	The <a href="#">vrna_fold_compound_t</a> storing the PF DP matrices that are to be erased from memory
----	---

## 15.84 Hash Tables

Various implementations of hash table functions.

### 15.84.1 Detailed Description

Various implementations of hash table functions.

Hash tables are common data structures that allow for fast random access to the data that is stored within.

Here, we provide an abstract implementation of a hash table interface and a concrete implementation for pairs of secondary structure and corresponding free energy value. Collaboration diagram for Hash Tables:

#### Files

- file [hash\\_tables.h](#)  
*Implementations of hash table functions.*

#### Data Structures

- struct [vrna\\_ht\\_entry\\_db\\_t](#)  
*Default hash table entry. [More...](#)*

#### Abstract interface

- typedef struct vrna\_hash\_table\_s \* [vrna\\_hash\\_table\\_t](#)  
*A hash table object.*
- typedef int() [vrna\\_callback\\_ht\\_compare\\_entries](#)(void \*x, void \*y)  
*Callback function to compare two hash table entries.*
- typedef unsigned int() [vrna\\_callback\\_ht\\_hash\\_function](#)(void \*x, unsigned long hashtable\_size)  
*Callback function to generate a hash key, i.e. hash function.*
- typedef int() [vrna\\_callback\\_ht\\_free\\_entry](#)(void \*x)  
*Callback function to free a hash table entry.*
- [vrna\\_hash\\_table\\_t](#) [vrna\\_ht\\_init](#) (unsigned int b, [vrna\\_callback\\_ht\\_compare\\_entries](#) \*compare\_function, [vrna\\_callback\\_ht\\_hash\\_function](#) \*hash\_function, [vrna\\_callback\\_ht\\_free\\_entry](#) \*free\_hash\_entry)  
*Get an initialized hash table.*
- unsigned long [vrna\\_ht\\_size](#) ([vrna\\_hash\\_table\\_t](#) ht)  
*Get the size of the hash table.*
- unsigned long [vrna\\_ht\\_collisions](#) (struct vrna\_hash\_table\_s \*ht)  
*Get the number of collisions in the hash table.*
- void \* [vrna\\_ht\\_get](#) ([vrna\\_hash\\_table\\_t](#) ht, void \*x)  
*Get an element from the hash table.*
- int [vrna\\_ht\\_insert](#) ([vrna\\_hash\\_table\\_t](#) ht, void \*x)  
*Insert an object into a hash table.*
- void [vrna\\_ht\\_remove](#) ([vrna\\_hash\\_table\\_t](#) ht, void \*x)  
*Remove an object from the hash table.*
- void [vrna\\_ht\\_clear](#) ([vrna\\_hash\\_table\\_t](#) ht)  
*Clear the hash table.*
- void [vrna\\_ht\\_free](#) ([vrna\\_hash\\_table\\_t](#) ht)  
*Free all memory occupied by the hash table.*

## Dot-Bracket / Free Energy entries

- int [vrna\\_ht\\_db\\_comp](#) (void \*x, void \*y)  
*Default hash table entry comparison.*
- unsigned int [vrna\\_ht\\_db\\_hash\\_func](#) (void \*x, unsigned long hashtable\_size)  
*Default hash function.*
- int [vrna\\_ht\\_db\\_free\\_entry](#) (void \*hash\_entry)  
*Default function to free memory occupied by a hash entry.*

## 15.84.2 Data Structure Documentation

### 15.84.2.1 struct vrna\_ht\_entry\_db\_t

Default hash table entry.

See also

[vrna\\_ht\\_init\(\)](#), [vrna\\_ht\\_db\\_comp\(\)](#), [vrna\\_ht\\_db\\_hash\\_func\(\)](#), [vrna\\_ht\\_db\\_free\\_entry\(\)](#)

#### Data Fields

- char \* [structure](#)
- float [energy](#)

### 15.84.2.1.1 Field Documentation

#### 15.84.2.1.1.1 structure

```
char* vrna_ht_entry_db_t::structure
```

A secondary structure in dot-bracket notation

#### 15.84.2.1.1.2 energy

```
float vrna_ht_entry_db_t::energy
```

The free energy of `structure`

## 15.84.3 Typedef Documentation

### 15.84.3.1 vrna\_hash\_table\_t

```
typedef struct vrna_hash_table_s* vrna_hash_table_t  
  
#include <ViennaRNA/datastructures/hash_tables.h>
```

A hash table object.

See also

[vrna\\_ht\\_init\(\)](#), [vrna\\_ht\\_free\(\)](#)

### 15.84.3.2 vrna\_callback\_ht\_compare\_entries

```
typedef int() vrna_callback_ht_compare_entries(void *x, void *y)  
  
#include <ViennaRNA/datastructures/hash_tables.h>
```

Callback function to compare two hash table entries.

See also

[vrna\\_ht\\_init\(\)](#), [vrna\\_ht\\_db\\_comp\(\)](#)

Parameters

<i>x</i>	A hash table entry
<i>y</i>	A hash table entry

Returns

-1 if *x* is smaller, +1 if *x* is larger than *y*. 0 if  $x == y$

### 15.84.3.3 vrna\_callback\_ht\_hash\_function

```
typedef unsigned int() vrna_callback_ht_hash_function(void *x, unsigned long hashtable_size)  
  
#include <ViennaRNA/datastructures/hash_tables.h>
```

Callback function to generate a hash key, i.e. hash function.

See also

[vrna\\_ht\\_init\(\)](#), [vrna\\_ht\\_db\\_hash\\_func\(\)](#)

**Parameters**

<i>x</i>	A hash table entry
<i>hashtable_size</i>	The size of the hash table

**Returns**

The hash table key for entry *x*

**15.84.3.4 vrna\_callback\_ht\_free\_entry**

```
typedef int() vrna_callback_ht_free_entry(void *x)
```

```
#include <ViennaRNA/datastructures/hash_tables.h>
```

Callback function to free a hash table entry.

**See also**

[vrna\\_ht\\_init\(\)](#), [vrna\\_ht\\_db\\_free\\_entry\(\)](#)

**Parameters**

<i>x</i>	A hash table entry
----------	--------------------

**Returns**

0 on success

**15.84.4 Function Documentation****15.84.4.1 vrna\_ht\_init()**

```
vrna_hash_table_t vrna_ht_init (
    unsigned int b,
    vrna_callback_ht_compare_entries * compare_function,
    vrna_callback_ht_hash_function * hash_function,
    vrna_callback_ht_free_entry * free_hash_entry )
```

```
#include <ViennaRNA/datastructures/hash_tables.h>
```

Get an initialized hash table.

This function returns a ready-to-use hash table with pre-allocated memory for a particular number of entries.



**Note**

If all function pointers are `NULL`, this function initializes the hash table with *default functions*, i.e.

- `vrna_ht_db_comp()` for the `compare_function`,
- `vrna_ht_db_hash_func()` for the `hash_function`, and
- `vrna_ht_db_free_entry()` for the `free_hash_entry`

arguments.

**Warning**

If `hash_bits` is larger than 27 you have to compile it with the flag `gcc -mmodel=large`.

**Parameters**

<i>b</i>	Number of bits for the hash table. This determines the size ( $2^b - 1$ ).
<i>compare_function</i>	A function pointer to compare any two entries in the hash table (may be <code>NULL</code> )
<i>hash_function</i>	A function pointer to retrieve the hash value of any entry (may be <code>NULL</code> )
<i>free_hash_entry</i>	A function pointer to free the memory occupied by any entry (may be <code>NULL</code> )

**Returns**

An initialized, empty hash table, or `NULL` on any error

**15.84.4.2 vrna\_ht\_size()**

```
unsigned long vrna_ht_size (
    vrna_hash_table_t ht )
```

```
#include <ViennaRNA/datastructures/hash_tables.h>
```

Get the size of the hash table.

**Parameters**

<i>ht</i>	The hash table
-----------	----------------

**Returns**

The size of the hash table, i.e. the maximum number of entries

**15.84.4.3 vrna\_ht\_collisions()**

```
unsigned long vrna_ht_collisions (
    struct vrna_hash_table_s * ht )
```

```
#include <ViennaRNA/datastructures/hash_tables.h>
```

Get the number of collisions in the hash table.

## Parameters

<i>ht</i>	The hash table
-----------	----------------

## Returns

The number of collisions in the hash table

15.84.4.4 `vrna_ht_get()`

```
void* vrna_ht_get (
    vrna_hash_table_t ht,
    void * x )
```

```
#include <ViennaRNA/datastructures/hash_tables.h>
```

Get an element from the hash table.

This function takes an object `x` and performs a look-up whether the object is stored within the hash table `ht`. If the object is already stored in `ht`, the function simply returns the entry, otherwise it returns `NULL`.

## See also

[vrna\\_ht\\_insert\(\)](#), [vrna\\_hash\\_delete\(\)](#), [vrna\\_ht\\_init\(\)](#)

## Parameters

<i>ht</i>	The hash table
<i>x</i>	The hash entry to look-up

## Returns

The entry `x` if it is stored in `ht`, `NULL` otherwise

15.84.4.5 `vrna_ht_insert()`

```
int vrna_ht_insert (
    vrna_hash_table_t ht,
    void * x )
```

```
#include <ViennaRNA/datastructures/hash_tables.h>
```

Insert an object into a hash table.

Writes the pointer to your hash entry into the table.

**Warning**

In case of collisions, this function simply increments the hash key until a free entry in the hash table is found.

**See also**

[vrna\\_ht\\_init\(\)](#), [vrna\\_hash\\_delete\(\)](#), [vrna\\_ht\\_clear\(\)](#)

**Parameters**

<i>ht</i>	The hash table
<i>x</i>	The hash entry

**Returns**

0 on success, 1 if the value is already in the hash table, -1 on error.

**15.84.4.6 vrna\_ht\_remove()**

```
void vrna_ht_remove (
    vrna_hash_table_t ht,
    void * x )
```

```
#include <ViennaRNA/datastructures/hash_tables.h>
```

Remove an object from the hash table.

Deletes the pointer to your hash entry from the table.

**Note**

This function doesn't free any memory occupied by the hash entry.

**Parameters**

<i>ht</i>	The hash table
<i>x</i>	The hash entry

**15.84.4.7 vrna\_ht\_clear()**

```
void vrna_ht_clear (
    vrna_hash_table_t ht )
```

```
#include <ViennaRNA/datastructures/hash_tables.h>
```

Clear the hash table.

This function removes all entries from the hash table and automatically free's the memory occupied by each entry using the `bound ()` function.

See also

[vrna\\_ht\\_free\(\)](#), [vrna\\_ht\\_init\(\)](#)

Parameters

<i>ht</i>	The hash table
-----------	----------------

#### 15.84.4.8 `vrna_ht_free()`

```
void vrna_ht_free (
    vrna_hash_table_t ht )
```

```
#include <ViennaRNA/datastructures/hash_tables.h>
```

Free all memory occupied by the hash table.

This function removes all entries from the hash table by calling the [vrna\\_callback\\_ht\\_free\\_entry\(\)](#) function for each entry. Finally, the memory occupied by the hash table itself is free'd as well.

Parameters

<i>ht</i>	The hash table
-----------	----------------

#### 15.84.4.9 `vrna_ht_db_comp()`

```
int vrna_ht_db_comp (
    void * x,
    void * y )
```

```
#include <ViennaRNA/datastructures/hash_tables.h>
```

Default hash table entry comparison.

This is the default comparison function for hash table entries. It assumes the both entries `x` and `y` are of type [vrna\\_ht\\_entry\\_db\\_t](#) and compares the `structure` attribute of both entries

See also

[vrna\\_ht\\_entry\\_db\\_t](#), [vrna\\_ht\\_init\(\)](#), [vrna\\_ht\\_db\\_hash\\_func\(\)](#), [vrna\\_ht\\_db\\_free\\_entry\(\)](#)

## Parameters

<i>x</i>	A hash table entry of type <a href="#">vrna_ht_entry_db_t</a>
<i>y</i>	A hash table entry of type <a href="#">vrna_ht_entry_db_t</a>

## Returns

-1 if *x* is smaller, +1 if *x* is larger than *y*. 0 if both are equal.

15.84.4.10 `vrna_ht_db_hash_func()`

```
unsigned int vrna_ht_db_hash_func (
    void * x,
    unsigned long hashtable_size )
```

```
#include <ViennaRNA/datastructures/hash_tables.h>
```

Default hash function.

This is the default hash function for hash table insertion/lookup. It assumes that entries are of type [vrna\\_ht\\_entry\\_db\\_t](#) and uses the Bob Jenkins 1996 mix function to create a hash key from the `structure` attribute of the hash entry.

## See also

[vrna\\_ht\\_entry\\_db\\_t](#), [vrna\\_ht\\_init\(\)](#), [vrna\\_ht\\_db\\_comp\(\)](#), [vrna\\_ht\\_db\\_free\\_entry\(\)](#)

## Parameters

<i>x</i>	A hash table entry to compute the key for
<i>hashtable_size</i>	The size of the hash table

## Returns

The hash key for entry *x*

15.84.4.11 `vrna_ht_db_free_entry()`

```
int vrna_ht_db_free_entry (
    void * hash_entry )
```

```
#include <ViennaRNA/datastructures/hash_tables.h>
```

Default function to free memory occupied by a hash entry.

This function assumes that hash entries are of type [vrna\\_ht\\_entry\\_db\\_t](#) and free's the memory occupied by that entry.

See also

[vrna\\_ht\\_entry\\_db\\_t](#), [vrna\\_ht\\_init\(\)](#), [vrna\\_ht\\_db\\_comp\(\)](#), [vrna\\_ht\\_db\\_hash\\_func\(\)](#)

Parameters

<i>hash_entry</i>	The hash entry to remove from memory
-------------------	--------------------------------------

Returns

0 on success

## 15.85 Buffers

Functions that provide dynamically buffered stream-like data structures.

### 15.85.1 Detailed Description

Functions that provide dynamically buffered stream-like data structures.

Collaboration diagram for Buffers:

#### Files

- file [char\\_stream.h](#)  
*Implementation of a dynamic, buffered character stream.*
- file [stream\\_output.h](#)  
*An implementation of a buffered, ordered stream output data structure.*

#### Typedefs

- typedef struct vrna\_ordered\_stream\_s \* [vrna\\_ostream\\_t](#)  
*An ordered output stream structure with unordered insert capabilities.*
- typedef void() [vrna\\_callback\\_stream\\_output](#)(void \*auxdata, unsigned int i, void \*data)  
*Ordered stream processing callback.*

#### Functions

- [vrna\\_cstr\\_t vrna\\_cstr](#) (size\_t size, FILE \*output)  
*Create a dynamic char \* stream data structure.*
- void [vrna\\_cstr\\_free](#) (vrna\_cstr\_t buf)  
*Free the memory occupied by a dynamic char \* stream data structure.*
- void [vrna\\_cstr\\_close](#) (vrna\_cstr\_t buf)  
*Free the memory occupied by a dynamic char \* stream and close the output stream.*
- void [vrna\\_cstr\\_fflush](#) (struct vrna\_cstr\_s \*buf)  
*Flush the dynamic char \* output stream.*
- [vrna\\_ostream\\_t vrna\\_ostream\\_init](#) ([vrna\\_callback\\_stream\\_output](#) \*output, void \*auxdata)  
*Get an initialized ordered output stream.*
- void [vrna\\_ostream\\_free](#) ([vrna\\_ostream\\_t](#) dat)  
*Free an initialized ordered output stream.*
- void [vrna\\_ostream\\_request](#) ([vrna\\_ostream\\_t](#) dat, unsigned int num)  
*Request index in ordered output stream.*
- void [vrna\\_ostream\\_provide](#) ([vrna\\_ostream\\_t](#) dat, unsigned int i, void \*data)  
*Provide output stream data for a particular index.*



## 15.85.2 Typedef Documentation

### 15.85.2.1 `vrna_callback_stream_output`

```
typedef void() vrna_callback_stream_output(void *auxdata, unsigned int i, void *data)
```

```
#include <ViennaRNA/datastructures/stream_output.h>
```

Ordered stream processing callback.

This callback will be processed in sequential order as soon as sequential data in the output stream becomes available.

#### Note

The callback must also release the memory occupied by the data passed since the stream will lose any reference to it after the callback has been executed.

#### Parameters

<i>auxdata</i>	A shared pointer for all calls, as provided by the second argument to <a href="#">vrna_ostream_init()</a>
<i>i</i>	The index number of the data passed to <i>data</i>
<i>data</i>	A block of data ready for processing

## 15.85.3 Function Documentation

### 15.85.3.1 `vrna_cstr()`

```
vrna_cstr_t vrna_cstr (
    size_t size,
    FILE * output )
```

```
#include <ViennaRNA/datastructures/char_stream.h>
```

Create a dynamic `char *` stream data structure.

#### See also

[vrna\\_cstr\\_free\(\)](#), [vrna\\_cstr\\_close\(\)](#), [vrna\\_cstr\\_fflush\(\)](#), [vrna\\_cstr\\_printf\(\)](#)

#### Parameters

<i>size</i>	The initial size of the buffer in characters
<i>output</i>	An optional output file stream handle that is used to write the collected data to (defaults to <i>stdout</i> if <i>NULL</i> )

### 15.85.3.2 vrna\_cstr\_free()

```
void vrna_cstr_free (
    vrna_cstr_t buf )
```

```
#include <ViennaRNA/datastructures/char_stream.h>
```

Free the memory occupied by a dynamic char \* stream data structure.

This function first flushes any remaining character data within the stream and then free's the memory occupied by the data structure.

See also

[vrna\\_cstr\\_close\(\)](#), [vrna\\_cstr\\_fflush\(\)](#), [vrna\\_cstr\(\)](#)

#### Parameters

<i>buf</i>	The dynamic char * stream data structure to free
------------	--

### 15.85.3.3 vrna\_cstr\_close()

```
void vrna_cstr_close (
    vrna_cstr_t buf )
```

```
#include <ViennaRNA/datastructures/char_stream.h>
```

Free the memory occupied by a dynamic char \* stream and close the output stream.

This function first flushes any remaining character data within the stream then closes the attached output file stream (if any), and finally free's the memory occupied by the data structure.

See also

[vrna\\_cstr\\_free\(\)](#), [vrna\\_cstr\\_fflush\(\)](#), [vrna\\_cstr\(\)](#)

#### Parameters

<i>buf</i>	The dynamic char * stream data structure to free
------------	--

### 15.85.3.4 vrna\_cstr\_fflush()

```
void vrna_cstr_fflush (
```

```
struct vrna_cstr_s * buf )
```

```
#include <ViennaRNA/datastructures/char_stream.h>
```

Flush the dynamic char \* output stream.

This function flushes the collected char \* stream, either by writing to the attached file handle, or simply by writing to *stdout* if no file handle has been attached upon construction using [vrna\\_cstr\(\)](#).

#### Postcondition

The stream buffer is empty after execution of this function

#### See also

[vrna\\_cstr\(\)](#), [vrna\\_cstr\\_close\(\)](#), [vrna\\_cstr\\_free\(\)](#)

#### Parameters

<i>buf</i>	The dynamic char * stream data structure to flush
------------	---

#### 15.85.3.5 vrna\_ostream\_init()

```
vrna_ostream_t vrna_ostream_init (
    vrna_callback_stream_output * output,
    void * auxdata )
```

```
#include <ViennaRNA/datastructures/stream_output.h>
```

Get an initialized ordered output stream.

#### See also

[vrna\\_ostream\\_free\(\)](#), [vrna\\_ostream\\_request\(\)](#), [vrna\\_ostream\\_provide\(\)](#)

#### Parameters

<i>output</i>	A callback function that processes and releases data in the stream
<i>auxdata</i>	A pointer to auxiliary data passed as first argument to the <i>output</i> callback

#### Returns

An initialized ordered output stream

### 15.85.3.6 `vrna_ostream_free()`

```
void vrna_ostream_free (
    vrna_ostream_t dat )
```

```
#include <ViennaRNA/datastructures/stream_output.h>
```

Free an initialized ordered output stream.

See also

[vrna\\_ostream\\_init\(\)](#)

#### Parameters

<i>dat</i>	The output stream for which occupied memory should be free'd
------------	--

### 15.85.3.7 `vrna_ostream_request()`

```
void vrna_ostream_request (
    vrna_ostream_t dat,
    unsigned int num )
```

```
#include <ViennaRNA/datastructures/stream_output.h>
```

Request index in ordered output stream.

This function must be called prior to [vrna\\_ostream\\_provide\(\)](#) to indicate that data associated with a certain index number is expected to be inserted into the stream in the future.

See also

[vrna\\_ostream\\_init\(\)](#), [vrna\\_ostream\\_provide\(\)](#), [vrna\\_ostream\\_free\(\)](#)

#### Parameters

<i>dat</i>	The output stream for which the index is requested
<i>num</i>	The index to request data for

### 15.85.3.8 `vrna_ostream_provide()`

```
void vrna_ostream_provide (
    vrna_ostream_t dat,
    unsigned int i,
    void * data )
```

```
#include <ViennaRNA/datastructures/stream_output.h>
```

Provide output stream data for a particular index.

#### Precondition

The index data is provided for must have been requested using [vrna\\_ostream\\_request\(\)](#) beforehand.

#### See also

[vrna\\_ostream\\_request\(\)](#)

#### Parameters

<i>dat</i>	The output stream for which data is provided
<i>i</i>	The index of the provided data
<i>data</i>	The data provided



## Chapter 16

# Data Structure Documentation

### 16.1 `_struct_en` Struct Reference

Data structure for [energy\\_of\\_move\(\)](#)

#### 16.1.1 Detailed Description

Data structure for [energy\\_of\\_move\(\)](#)

The documentation for this struct was generated from the following file:

- ViennaRNA/move\_set.h

### 16.2 `LIST` Struct Reference

Collaboration diagram for `LIST`:

The documentation for this struct was generated from the following file:

- ViennaRNA/datastructures/lists.h

### 16.3 `LST_BUCKET` Struct Reference

Collaboration diagram for `LST_BUCKET`:

The documentation for this struct was generated from the following file:

- ViennaRNA/datastructures/lists.h

## 16.4 Postorder\_list Struct Reference

Postorder data structure.

### 16.4.1 Detailed Description

Postorder data structure.

The documentation for this struct was generated from the following file:

- [ViennaRNA/dist\\_vars.h](#)

## 16.5 swString Struct Reference

Some other data structure.

### 16.5.1 Detailed Description

Some other data structure.

The documentation for this struct was generated from the following file:

- [ViennaRNA/dist\\_vars.h](#)

## 16.6 Tree Struct Reference

[Tree](#) data structure.

Collaboration diagram for Tree:

### 16.6.1 Detailed Description

[Tree](#) data structure.

The documentation for this struct was generated from the following file:

- [ViennaRNA/dist\\_vars.h](#)



## 16.7 TwoDpfold\_vars Struct Reference

Variables compound for 2Dfold partition function folding.

Collaboration diagram for TwoDpfold\_vars:

### Data Fields

- char \* [ptype](#)  
*Precomputed array of pair types.*
- char \* [sequence](#)  
*The input sequence.*
- short \* [S1](#)  
*The input sequences in numeric form.*
- unsigned int [maxD1](#)  
*Maximum allowed base pair distance to first reference.*
- unsigned int [maxD2](#)  
*Maximum allowed base pair distance to second reference.*
- int \* [my\\_iindx](#)  
*Index for moving in quadratic distance dimensions.*
- int \* [jindx](#)  
*Index for moving in the triangular matrix qm1.*
- unsigned int \* [referenceBPs1](#)  
*Matrix containing number of basepairs of reference structure1 in interval [i,j].*
- unsigned int \* [referenceBPs2](#)  
*Matrix containing number of basepairs of reference structure2 in interval [i,j].*
- unsigned int \* [bpdist](#)  
*Matrix containing base pair distance of reference structure 1 and 2 on interval [i,j].*
- unsigned int \* [mm1](#)  
*Maximum matching matrix, reference struct 1 disallowed.*
- unsigned int \* [mm2](#)  
*Maximum matching matrix, reference struct 2 disallowed.*

### 16.7.1 Detailed Description

Variables compound for 2Dfold partition function folding.

**Deprecated** This data structure will be removed from the library soon! Use [vrna\\_fold\\_compound\\_t](#) and the corresponding functions [vrna\\_fold\\_compound\\_TwoD\(\)](#), [vrna\\_pf\\_TwoD\(\)](#), and [vrna\\_fold\\_compound\\_free\(\)](#) instead!

The documentation for this struct was generated from the following file:

- [ViennaRNA/2Dpfold.h](#)

## 16.8 vrna\_dimer\_conc\_s Struct Reference

Data structure for concentration dependency computations.

### Data Fields

- double [Ac\\_start](#)  
*start concentration A*
- double [Bc\\_start](#)  
*start concentration B*
- double [ABc](#)  
*End concentration AB.*

### 16.8.1 Detailed Description

Data structure for concentration dependency computations.

The documentation for this struct was generated from the following file:

- ViennaRNA/[concentrations.h](#)

## 16.9 vrna\_hc\_bp\_storage\_t Struct Reference

A base pair hard constraint.

### 16.9.1 Detailed Description

A base pair hard constraint.

The documentation for this struct was generated from the following file:

- ViennaRNA/constraints/[hard.h](#)

## 16.10 vrna\_sc\_bp\_storage\_t Struct Reference

A base pair constraint.

### 16.10.1 Detailed Description

A base pair constraint.

The documentation for this struct was generated from the following file:

- ViennaRNA/constraints/[soft.h](#)

## 16.11 vrna\_sc\_motif\_s Struct Reference

The documentation for this struct was generated from the following file:

- ViennaRNA/constraints/[ligand.h](#)

## 16.12 vrna\_structured\_domains\_s Struct Reference

The documentation for this struct was generated from the following file:

- ViennaRNA/[structured\\_domains.h](#)

## 16.13 vrna\_subopt\_sol\_s Struct Reference

Solution element from subopt.c.

### Data Fields

- float [energy](#)  
*Free Energy of structure in kcal/mol.*
- char \* [structure](#)  
*Structure in dot-bracket notation.*

### 16.13.1 Detailed Description

Solution element from subopt.c.

The documentation for this struct was generated from the following file:

- ViennaRNA/[subopt.h](#)

## 16.14 vrna\_unstructured\_domain\_motif\_s Struct Reference

The documentation for this struct was generated from the following file:

- ViennaRNA/[unstructured\\_domains.h](#)



## Chapter 17

# File Documentation

### 17.1 ViennaRNA/2Dfold.h File Reference

MFE structures for base pair distance classes.

Include dependency graph for 2Dfold.h:

#### Data Structures

- struct [vrna\\_sol\\_TwoD\\_t](#)  
*Solution element returned from [vrna\\_mfe\\_TwoD\(\)](#) [More...](#)*
- struct [TwoDfold\\_vars](#)  
*Variables compound for 2Dfold MFE folding. [More...](#)*

#### Typedefs

- typedef struct [vrna\\_sol\\_TwoD\\_t](#) [vrna\\_sol\\_TwoD\\_t](#)  
*Solution element returned from [vrna\\_mfe\\_TwoD\(\)](#)*
- typedef struct [TwoDfold\\_vars](#) [TwoDfold\\_vars](#)  
*Variables compound for 2Dfold MFE folding.*

#### Functions

- [vrna\\_sol\\_TwoD\\_t](#) \* [vrna\\_mfe\\_TwoD](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int distance1, int distance2)  
*Compute MFE's and representative for distance partitioning.*
- char \* [vrna\\_backtrack5\\_TwoD](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int k, int l, unsigned int j)  
*Backtrack a minimum free energy structure from a 5' section of specified length.*
- [TwoDfold\\_vars](#) \* [get\\_TwoDfold\\_variables](#) (const char \*seq, const char \*structure1, const char \*structure2, int circ)  
*Get a structure of type [TwoDfold\\_vars](#) prefilled with current global settings.*
- void [destroy\\_TwoDfold\\_variables](#) ([TwoDfold\\_vars](#) \*our\_variables)  
*Destroy a [TwoDfold\\_vars](#) datastructure without memory loss.*
- [vrna\\_sol\\_TwoD\\_t](#) \* [TwoDfoldList](#) ([TwoDfold\\_vars](#) \*vars, int distance1, int distance2)  
*Compute MFE's and representative for distance partitioning.*
- char \* [TwoDfold\\_backtrack\\_f5](#) (unsigned int j, int k, int l, [TwoDfold\\_vars](#) \*vars)  
*Backtrack a minimum free energy structure from a 5' section of specified length.*

### 17.1.1 Detailed Description

MFE structures for base pair distance classes.

## 17.2 ViennaRNA/2Dpfold.h File Reference

Partition function implementations for base pair distance classes.

Include dependency graph for 2Dpfold.h:

### Data Structures

- struct [vrna\\_sol\\_TwoD\\_pf\\_t](#)  
*Solution element returned from [vrna\\_pf\\_TwoD\(\)](#) [More...](#)*
- struct [TwoDpfold\\_vars](#)  
*Variables compound for 2Dfold partition function folding.*

### Typedefs

- typedef struct [vrna\\_sol\\_TwoD\\_pf\\_t](#) [vrna\\_sol\\_TwoD\\_pf\\_t](#)  
*Solution element returned from [vrna\\_pf\\_TwoD\(\)](#)*

### Functions

- [vrna\\_sol\\_TwoD\\_pf\\_t](#) \* [vrna\\_pf\\_TwoD](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int maxDistance1, int maxDistance2)  
*Compute the partition function for all distance classes.*
- char \* [vrna\\_pbacktrack\\_TwoD](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int d1, int d2)  
*Sample secondary structure representatives from a set of distance classes according to their Boltzmann probability.*
- char \* [vrna\\_pbacktrack5\\_TwoD](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int d1, int d2, unsigned int length)  
*Sample secondary structure representatives with a specified length from a set of distance classes according to their Boltzmann probability.*
- [TwoDpfold\\_vars](#) \* [get\\_TwoDpfold\\_variables](#) (const char \*seq, const char \*structure1, char \*structure2, int circ)  
*Get a datastructure containing all necessary attributes and global folding switches.*
- void [destroy\\_TwoDpfold\\_variables](#) ([TwoDpfold\\_vars](#) \*vars)  
*Free all memory occupied by a [TwoDpfold\\_vars](#) datastructure.*
- [vrna\\_sol\\_TwoD\\_pf\\_t](#) \* [TwoDpfoldList](#) ([TwoDpfold\\_vars](#) \*vars, int maxDistance1, int maxDistance2)  
*Compute the partition function for all distance classes.*
- char \* [TwoDpfold\\_pbacktrack](#) ([TwoDpfold\\_vars](#) \*vars, int d1, int d2)  
*Sample secondary structure representatives from a set of distance classes according to their Boltzmann probability.*
- char \* [TwoDpfold\\_pbacktrack5](#) ([TwoDpfold\\_vars](#) \*vars, int d1, int d2, unsigned int length)  
*Sample secondary structure representatives with a specified length from a set of distance classes according to their Boltzmann probability.*

### 17.2.1 Detailed Description

Partition function implementations for base pair distance classes.

### 17.2.2 Function Documentation

#### 17.2.2.1 `get_TwoDpfold_variables()`

```
TwoDpfold_vars* get_TwoDpfold_variables (
    const char * seq,
    const char * structure1,
    char * structure2,
    int circ )
```

Get a datastructure containing all necessary attributes and global folding switches.

This function prepares all necessary attributes and matrices etc which are needed for a call of `TwoDpfold()`. A snapshot of all current global model switches (dangles, temperature and so on) is done and stored in the returned datastructure. Additionally, all matrices that will hold the partition function values are prepared.

**Deprecated** Use the new API that relies on `vrna_fold_compound_t` and the corresponding functions `vrna_fold_compound_TwoD()`, `vrna_pf_TwoD()`, and `vrna_fold_compound_free()` instead!

#### Parameters

<i>seq</i>	the RNA sequence in uppercase format with letters from the alphabet {AUCG}
<i>structure1</i>	the first reference structure in dot-bracket notation
<i>structure2</i>	the second reference structure in dot-bracket notation
<i>circ</i>	a switch indicating if the sequence is linear (0) or circular (1)

#### Returns

the datastructure containing all necessary partition function attributes

#### 17.2.2.2 `destroy_TwoDpfold_variables()`

```
void destroy_TwoDpfold_variables (
    TwoDpfold_vars * vars )
```

Free all memory occupied by a `TwoDpfold_vars` datastructure.

This function free's all memory occupied by a datastructure obtained from `get_TwoDpfold_variables()` or `get_TwoDpfold_variables_from_MFE()`

**Deprecated** Use the new API that relies on `vrna_fold_compound_t` and the corresponding functions `vrna_fold_compound_TwoD()`, `vrna_pf_TwoD()`, and `vrna_fold_compound_free()` instead!

See also

[get\\_TwoDpfold\\_variables\(\)](#), [get\\_TwoDpfold\\_variables\\_from\\_MFE\(\)](#)

Parameters

<i>vars</i>	the datastructure to be free'd
-------------	--------------------------------

### 17.2.2.3 TwoDpfoldList()

```
vrna_sol_TwoD_pf_t* TwoDpfoldList (
    TwoDpfold_vars * vars,
    int maxDistance1,
    int maxDistance2 )
```

Compute the partition function for all distance classes.

This function computes the partition functions for all distance classes according the two reference structures specified in the datastructure 'vars'. Similar to TwoDfold() the arguments maxDistance1 and maxDistance2 specify the maximum distance to both reference structures. A value of '-1' in either of them makes the appropriate distance restrictionless, i.e. all basepair distances to the reference are taken into account during computation. In case there is a restriction, the returned solution contains an entry where the attribute k=-1 contains the partition function for all structures exceeding the restriction. A values of **INF** in the attribute 'k' of the returned list denotes the end of the list

**Deprecated** Use the new API that relies on [vrna\\_fold\\_compound\\_t](#) and the corresponding functions [vrna\\_fold\\_compound\\_TwoD\(\)](#), [vrna\\_pf\\_TwoD\(\)](#), and [vrna\\_fold\\_compound\\_free\(\)](#) instead!

See also

[get\\_TwoDpfold\\_variables\(\)](#), [destroy\\_TwoDpfold\\_variables\(\)](#), [vrna\\_sol\\_TwoD\\_pf\\_t](#)

Parameters

<i>vars</i>	the datastructure containing all necessary folding attributes and matrices
<i>maxDistance1</i>	the maximum basepair distance to reference1 (may be -1)
<i>maxDistance2</i>	the maximum basepair distance to reference2 (may be -1)

Returns

a list of partition funtions for the appropriate distance classes

### 17.2.2.4 TwoDpfold\_pbacktrack()

```
char* TwoDpfold_pbacktrack (
    TwoDpfold_vars * vars,
```



```
int d1,
int d2 )
```

Sample secondary structure representatives from a set of distance classes according to their Boltzmann probability.

If the argument 'd1' is set to '-1', the structure will be backtracked in the distance class where all structures exceeding the maximum basepair distance to either of the references reside.

#### Precondition

The argument 'vars' must contain precalculated partition function matrices, i.e. a call to `TwoDpfold()` preceding this function is mandatory!

**Deprecated** Use the new API that relies on [vrna\\_fold\\_compound\\_t](#) and the corresponding functions `vrna_fold_compound_TwoD()`, `vrna_pf_TwoD()`, `vrna_pbacktrack_TwoD()`, and `vrna_fold_compound_free()` instead!

#### See also

`TwoDpfold()`

#### Parameters

in	<i>vars</i>	the datastructure containing all necessary folding attributes and matrices
in	<i>d1</i>	the distance to reference1 (may be -1)
in	<i>d2</i>	the distance to reference2

#### Returns

A sampled secondary structure in dot-bracket notation

#### 17.2.2.5 TwoDpfold\_pbacktrack5()

```
char* TwoDpfold_pbacktrack5 (
    TwoDpfold_vars * vars,
    int d1,
    int d2,
    unsigned int length )
```

Sample secondary structure representatives with a specified length from a set of distance classes according to their Boltzmann probability.

This function does essentially the same as [TwoDpfold\\_pbacktrack\(\)](#) with the only difference that partial structures, i.e. structures beginning from the 5' end with a specified length of the sequence, are backtracked

#### Note

This function does not work (since it makes no sense) for circular RNA sequences!

**Precondition**

The argument 'vars' must contain precalculated partition function matrices, i.e. a call to `TwoDpfold()` preceding this function is mandatory!

**Deprecated** Use the new API that relies on [vrna\\_fold\\_compound\\_t](#) and the corresponding functions `vrna_fold_compound_TwoD()`, `vrna_pf_TwoD()`, `vrna_pbacktrack5_TwoD()`, and `vrna_fold_compound_free()` instead!

**See also**

[TwoDpfold\\_pbacktrack\(\)](#), `TwoDpfold()`

**Parameters**

in	<i>vars</i>	the datastructure containing all necessary folding attributes and matrices
in	<i>d1</i>	the distance to reference1 (may be -1)
in	<i>d2</i>	the distance to reference2
in	<i>length</i>	the length of the structure beginning from the 5' end

**Returns**

A sampled secondary structure in dot-bracket notation

## 17.3 ViennaRNA/alifold.h File Reference

Functions for comparative structure prediction using RNA sequence alignments.

Include dependency graph for `alifold.h`:

**Functions**

- float [energy\\_of\\_alistruct](#) (const char \*\*sequences, const char \*structure, int n\_seq, float \*energy)  
*Calculate the free energy of a consensus structure given a set of aligned sequences.*
- void [update\\_alifold\\_params](#) (void)  
*Update the energy parameters for alifold function.*
- float [alifold](#) (const char \*\*strings, char \*structure)  
*Compute MFE and according consensus structure of an alignment of sequences.*
- float [circaifold](#) (const char \*\*strings, char \*structure)  
*Compute MFE and according structure of an alignment of sequences assuming the sequences are circular instead of linear.*
- void [free\\_alifold\\_arrays](#) (void)  
*Free the memory occupied by MFE alifold functions.*

- float [alipf\\_fold\\_par](#) (const char \*\*sequences, char \*structure, [vrna\\_ep\\_t](#) \*\*pl, [vrna\\_exp\\_param\\_t](#) \*parameters, int calculate\_bppm, int is\_constrained, int is\_circular)
- float [alipf\\_fold](#) (const char \*\*sequences, char \*structure, [vrna\\_ep\\_t](#) \*\*pl)  
*The partition function version of [alifold\(\)](#) works in analogy to [pf\\_fold\(\)](#). Pair probabilities and information about sequence covariations are returned via the 'pi' variable as a list of [vrna\\_pinfo\\_t](#) structs. The list is terminated by the first entry with  $pi.i = 0$ .*
- float [alipf\\_circ\\_fold](#) (const char \*\*sequences, char \*structure, [vrna\\_ep\\_t](#) \*\*pl)
- [FLT\\_OR\\_DBL](#) \* [export\\_ali\\_bppm](#) (void)  
*Get a pointer to the base pair probability array.*
- void [free\\_alipf\\_arrays](#) (void)  
*Free the memory occupied by folding matrices allocated by [alipf\\_fold](#), [alipf\\_circ\\_fold](#), etc.*
- char \* [alipbacktrack](#) (double \*prob)  
*Sample a consensus secondary structure from the Boltzmann ensemble according its probability.*
- int [get\\_alipf\\_arrays](#) (short \*\*\*S\_p, short \*\*\*S5\_p, short \*\*\*S3\_p, unsigned short \*\*\*a2s\_p, char \*\*\*Ss↔\_p, [FLT\\_OR\\_DBL](#) \*\*qb\_p, [FLT\\_OR\\_DBL](#) \*\*qm\_p, [FLT\\_OR\\_DBL](#) \*\*q1k\_p, [FLT\\_OR\\_DBL](#) \*\*qln\_p, short \*\*pscore)  
*Get pointers to (almost) all relevant arrays used in alifold's partition function computation.*

## Variables

- double [cv\\_fact](#)  
*This variable controls the weight of the covariance term in the energy function of alignment folding algorithms.*
- double [nc\\_fact](#)  
*This variable controls the magnitude of the penalty for non-compatible sequences in the covariance term of alignment folding algorithms.*

## 17.3.1 Detailed Description

Functions for comparative structure prediction using RNA sequence alignments.

## 17.3.2 Function Documentation

### 17.3.2.1 [energy\\_of\\_alistruct\(\)](#)

```
float energy_of_alistruct (
    const char ** sequences,
    const char * structure,
    int n_seq,
    float * energy )
```

Calculate the free energy of a consensus structure given a set of aligned sequences.

**Deprecated** Usage of this function is discouraged! Use [vrna\\_eval\\_structure\(\)](#), and [vrna\\_eval\\_covar\\_structure\(\)](#) instead!

## Parameters

<i>sequences</i>	The NULL terminated array of sequences
<i>structure</i>	The consensus structure
<i>n_seq</i>	The number of sequences in the alignment
<i>energy</i>	A pointer to an array of at least two floats that will hold the free energies (energy[0] will contain the free energy, energy[1] will be filled with the covariance energy term)

## Returns

free energy in kcal/mol

17.3.2.2 `update_alifold_params()`

```
void update_alifold_params (
    void )
```

Update the energy parameters for alifold function.

Call this to recalculate the pair matrix and energy parameters after a change in folding parameters like [temperature](#)

**Deprecated** Usage of this function is discouraged! The new API uses [vrna\\_fold\\_compound\\_t](#) to lump all folding related necessities together, including the energy parameters. Use `vrna_update_fold_params()` to update the energy parameters within a [vrna\\_fold\\_compound\\_t](#).

## 17.3.3 Variable Documentation

17.3.3.1 `cv_fact`

```
double cv_fact
```

This variable controls the weight of the covariance term in the energy function of alignment folding algorithms.

**Deprecated** See [vrna\\_md\\_t.cv\\_fact](#), and [vrna\\_mfe\(\)](#) to avoid using global variables

Default is 1.

17.3.3.2 `nc_fact`

```
double nc_fact
```

This variable controls the magnitude of the penalty for non-compatible sequences in the covariance term of alignment folding algorithms.

**Deprecated** See [vrna\\_md\\_t.nc\\_fact](#), and [vrna\\_mfe\(\)](#) to avoid using global variables

Default is 1.

## 17.4 ViennaRNA/aln\_util.h File Reference

Use [ViennaRNA/utis/alignments.h](#) instead.

Include dependency graph for aln\_util.h:

### 17.4.1 Detailed Description

Use [ViennaRNA/utis/alignments.h](#) instead.

**Deprecated** Use [ViennaRNA/utis/alignments.h](#) instead

## 17.5 ViennaRNA/alphabet.h File Reference

Functions to process, convert, and generally handle different nucleotide and/or base pair alphabets.

Include dependency graph for alphabet.h:

This graph shows which files directly or indirectly include this file:

### Functions

- char \* [vrna\\_ptypes](#) (const short \*S, [vrna\\_md\\_t](#) \*md)  
*Get an array of the numerical encoding for each possible base pair (i,j)*
- short \* [vrna\\_seq\\_encode](#) (const char \*sequence, [vrna\\_md\\_t](#) \*md)  
*Get a numerical representation of the nucleotide sequence.*
- short \* [vrna\\_seq\\_encode\\_simple](#) (const char \*sequence, [vrna\\_md\\_t](#) \*md)  
*Get a numerical representation of the nucleotide sequence (simple version)*
- int [vrna\\_nucleotide\\_encode](#) (char c, [vrna\\_md\\_t](#) \*md)  
*Encode a nucleotide character to numerical value.*
- char [vrna\\_nucleotide\\_decode](#) (int enc, [vrna\\_md\\_t](#) \*md)  
*Decode a numerical representation of a nucleotide back into nucleotide alphabet.*

### 17.5.1 Detailed Description

Functions to process, convert, and generally handle different nucleotide and/or base pair alphabets.

,

## 17.6 ViennaRNA/boltzmann\_sampling.h File Reference

Boltzmann Sampling of secondary structures from the ensemble.

Include dependency graph for boltzmann\_sampling.h:

This graph shows which files directly or indirectly include this file:

### Functions

- char \* [vrna\\_pbacktrack5](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int length)  
*Sample a secondary structure of a subsequence from the Boltzmann ensemble according its probability.*
- char \*\* [vrna\\_pbacktrack\\_nr](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int num\_samples)  
*Samples multiple secondary structures non-redundantly from the Boltzmann ensemble according its probability.*
- char \* [vrna\\_pbacktrack](#) ([vrna\\_fold\\_compound\\_t](#) \*vc)  
*Sample a secondary structure (consensus structure) from the Boltzmann ensemble according its probability.*

### 17.6.1 Detailed Description

Boltzmann Sampling of secondary structures from the ensemble.

A.k.a. Stochastic backtracking

## 17.7 ViennaRNA/centroid.h File Reference

Centroid structure computation.

Include dependency graph for centroid.h:

This graph shows which files directly or indirectly include this file:

### Functions

- char \* [vrna\\_centroid](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, double \*dist)  
*Get the centroid structure of the ensemble.*
- char \* [vrna\\_centroid\\_from\\_plist](#) (int length, double \*dist, [vrna\\_ep\\_t](#) \*pl)  
*Get the centroid structure of the ensemble.*
- char \* [vrna\\_centroid\\_from\\_probs](#) (int length, double \*dist, [FLT\\_OR\\_DBL](#) \*probs)  
*Get the centroid structure of the ensemble.*
- char \* [get\\_centroid\\_struct\\_pl](#) (int length, double \*dist, [vrna\\_ep\\_t](#) \*pl)  
*Get the centroid structure of the ensemble.*
- char \* [get\\_centroid\\_struct\\_pr](#) (int length, double \*dist, [FLT\\_OR\\_DBL](#) \*pr)  
*Get the centroid structure of the ensemble.*

### 17.7.1 Detailed Description

Centroid structure computation.

### 17.7.2 Function Documentation

#### 17.7.2.1 `get_centroid_struct_pl()`

```
char* get_centroid_struct_pl (
    int length,
    double * dist,
    vrna_ep_t * pl )
```

Get the centroid structure of the ensemble.

**Deprecated** This function was renamed to [vrna\\_centroid\\_from\\_plist\(\)](#)

#### 17.7.2.2 `get_centroid_struct_pr()`

```
char* get_centroid_struct_pr (
    int length,
    double * dist,
    FLT_OR_DBL * pr )
```

Get the centroid structure of the ensemble.

**Deprecated** This function was renamed to [vrna\\_centroid\\_from\\_probs\(\)](#)

## 17.8 ViennaRNA/char\_stream.h File Reference

Use [ViennaRNA/datastructures/char\\_stream.h](#) instead.

Include dependency graph for char\_stream.h:

### 17.8.1 Detailed Description

Use [ViennaRNA/datastructures/char\\_stream.h](#) instead.

**Deprecated** Use [ViennaRNA/datastructures/char\\_stream.h](#) instead

## 17.9 ViennaRNA/datastructures/char\_stream.h File Reference

Implementation of a dynamic, buffered character stream.

Include dependency graph for char\_stream.h:

This graph shows which files directly or indirectly include this file:

### Functions

- `vrna_cstr_t vrna_cstr` (`size_t size`, `FILE *output`)  
*Create a dynamic char \* stream data structure.*
- `void vrna_cstr_free` (`vrna_cstr_t buf`)  
*Free the memory occupied by a dynamic char \* stream data structure.*
- `void vrna_cstr_close` (`vrna_cstr_t buf`)  
*Free the memory occupied by a dynamic char \* stream and close the output stream.*
- `void vrna_cstr_fflush` (`struct vrna_cstr_s *buf`)  
*Flush the dynamic char \* output stream.*

### 17.9.1 Detailed Description

Implementation of a dynamic, buffered character stream.

,

## 17.10 ViennaRNA/cofold.h File Reference

MFE implementations for RNA-RNA interaction.

Include dependency graph for cofold.h:



## Functions

- float [cofold](#) (const char \*sequence, char \*structure)  
*Compute the minimum free energy of two interacting RNA molecules.*
- float [cofold\\_par](#) (const char \*string, char \*structure, [vrna\\_param\\_t](#) \*parameters, int is\_constrained)  
*Compute the minimum free energy of two interacting RNA molecules.*
- void [free\\_co\\_arrays](#) (void)  
*Free memory occupied by [cofold\(\)](#)*
- void [update\\_cofold\\_params](#) (void)  
*Recalculate parameters.*
- void [update\\_cofold\\_params\\_par](#) ([vrna\\_param\\_t](#) \*parameters)  
*Recalculate parameters.*
- void [export\\_cofold\\_arrays\\_gg](#) (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*fc\_p, int \*\*ggg\_p, int \*\*indx\_p, char \*\*ptype\_p)  
*Export the arrays of partition function cofold (with gquadruplex support)*
- void [export\\_cofold\\_arrays](#) (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*fc\_p, int \*\*indx\_p, char \*\*ptype\_p)  
*Export the arrays of partition function cofold.*
- void [get\\_monomere\\_mfes](#) (float \*e1, float \*e2)  
*get\_monomer\_free\_energies*
- void [initialize\\_cofold](#) (int length)

### 17.10.1 Detailed Description

MFE implementations for RNA-RNA interaction.

## 17.11 ViennaRNA/combinatorics.h File Reference

Various implementations that deal with combinatorial aspects of objects.

## Functions

- unsigned int \*\* [vrna\\_enumerate\\_necklaces](#) (const unsigned int \*type\_counts)  
*Enumerate all necklaces with fixed content.*
- unsigned int [vrna\\_rotational\\_symmetry\\_num](#) (const unsigned int \*string, size\_t string\_length)  
*Determine the order of rotational symmetry for a string of objects represented by natural numbers.*
- unsigned int [vrna\\_rotational\\_symmetry\\_pos\\_num](#) (const unsigned int \*string, size\_t string\_length, unsigned int \*\*positions)  
*Determine the order of rotational symmetry for a string of objects represented by natural numbers.*
- unsigned int [vrna\\_rotational\\_symmetry](#) (const char \*string)  
*Determine the order of rotational symmetry for a NULL-terminated string of ASCII characters.*
- unsigned int [vrna\\_rotational\\_symmetry\\_pos](#) (const char \*string, unsigned int \*\*positions)  
*Determine the order of rotational symmetry for a NULL-terminated string of ASCII characters.*
- unsigned int [vrna\\_rotational\\_symmetry\\_db](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, const char \*structure)  
*Determine the order of rotational symmetry for a dot-bracket structure.*
- unsigned int [vrna\\_rotational\\_symmetry\\_db\\_pos](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, const char \*structure, unsigned int \*\*positions)  
*Determine the order of rotational symmetry for a dot-bracket structure.*

### 17.11.1 Detailed Description

Various implementations that deal with combinatorial aspects of objects.

,

## 17.12 ViennaRNA/commands.h File Reference

Parse and apply different commands that alter the behavior of secondary structure prediction and evaluation.

Include dependency graph for commands.h:

### Macros

- `#define VRNA_CMD_PARSE_HC 1U`  
*Command parse/apply flag indicating hard constraints.*
- `#define VRNA_CMD_PARSE_SC 2U`  
*Command parse/apply flag indicating soft constraints.*
- `#define VRNA_CMD_PARSE_UD 4U`  
*Command parse/apply flag indicating unstructured domains.*
- `#define VRNA_CMD_PARSE_SD 8U`  
*Command parse/apply flag indicating structured domains.*
- `#define VRNA_CMD_PARSE_DEFAULTS`  
*Command parse/apply flag indicating default set of commands.*

### Typedefs

- `typedef struct vrna_command_s * vrna_cmd_t`  
*A data structure that contains commands.*

### Functions

- `vrna_cmd_t vrna_file_commands_read` (const char \*filename, unsigned int options)  
*Extract a list of commands from a command file.*
- `int vrna_file_commands_apply` (vrna\_fold\_compound\_t \*vc, const char \*filename, unsigned int options)  
*Apply a list of commands from a command file.*
- `int vrna_commands_apply` (vrna\_fold\_compound\_t \*vc, vrna\_cmd\_t commands, unsigned int options)  
*Apply a list of commands to a vrna\_fold\_compound\_t.*
- `void vrna_commands_free` (vrna\_cmd\_t commands)  
*Free memory occupied by a list of commands.*

### 17.12.1 Detailed Description

Parse and apply different commands that alter the behavior of secondary structure prediction and evaluation.

, ,

## 17.13 ViennaRNA/concentrations.h File Reference

Concentration computations for RNA-RNA interactions.

Include dependency graph for concentrations.h:

This graph shows which files directly or indirectly include this file:

### Data Structures

- struct [vrna\\_dimer\\_conc\\_s](#)  
*Data structure for concentration dependency computations.*

### Functions

- [vrna\\_dimer\\_conc\\_t](#) \* [get\\_concentrations](#) (double FEAB, double FEAA, double FEBB, double FEA, double FEB, double \*startconc)  
*Given two start monomer concentrations a and b, compute the concentrations in thermodynamic equilibrium of all dimers and the monomers.*
- typedef struct [vrna\\_dimer\\_conc\\_s](#) [vrna\\_dimer\\_conc\\_t](#)  
*Typename for the data structure that stores the dimer concentrations, [vrna\\_dimer\\_conc\\_s](#), as required by [vrna\\_pf\\_dimer\\_concentration\(\)](#)*
- typedef struct [vrna\\_dimer\\_conc\\_s](#) ConcEnt  
*Backward compatibility typedef for [vrna\\_dimer\\_conc\\_s](#).*
- [vrna\\_dimer\\_conc\\_t](#) \* [vrna\\_pf\\_dimer\\_concentrations](#) (double FcAB, double FcAA, double FcBB, double FEA, double FEB, const double \*startconc, const [vrna\\_exp\\_param\\_t](#) \*exp\_params)  
*Given two start monomer concentrations a and b, compute the concentrations in thermodynamic equilibrium of all dimers and the monomers.*

#### 17.13.1 Detailed Description

Concentration computations for RNA-RNA interactions.

#### 17.13.2 Function Documentation

### 17.13.2.1 `get_concentrations()`

```
vrna_dimer_conc_t* get_concentrations (
    double FEAB,
    double FEAA,
    double FEBB,
    double FEA,
    double FEB,
    double * startconc )
```

Given two start monomer concentrations a and b, compute the concentrations in thermodynamic equilibrium of all dimers and the monomers.

This function takes an array 'startconc' of input concentrations with alternating entries for the initial concentrations of molecules A and B (terminated by two zeroes), then computes the resulting equilibrium concentrations from the free energies for the dimers. Dimer free energies should be the dimer-only free energies, i.e. the FcAB entries from the `vrna_dimer_pf_t` struct.

**Deprecated** { Use `vrna_pf_dimer_concentrations()` instead!}

#### Parameters

<i>FEAB</i>	Free energy of AB dimer (FcAB entry)
<i>FEAA</i>	Free energy of AA dimer (FcAB entry)
<i>FEBB</i>	Free energy of BB dimer (FcAB entry)
<i>FEA</i>	Free energy of monomer A
<i>FEB</i>	Free energy of monomer B
<i>startconc</i>	List of start concentrations [a0],[b0],[a1],[b1],...,[an],[bn],[0],[0]

#### Returns

`vrna_dimer_conc_t` array containing the equilibrium energies and start concentrations

## 17.14 ViennaRNA/constraints.h File Reference

Use [ViennaRNA/constraints/basic.h](#) instead.

Include dependency graph for constraints.h:

### 17.14.1 Detailed Description

Use [ViennaRNA/constraints/basic.h](#) instead.

**Deprecated** Use [ViennaRNA/constraints/basic.h](#) instead

## 17.15 ViennaRNA/constraints/hard.h File Reference

Functions and data structures for handling of secondary structure hard constraints.

Include dependency graph for hard.h:

This graph shows which files directly or indirectly include this file:

### Data Structures

- struct [vrna\\_hc\\_bp\\_storage\\_t](#)  
*A base pair hard constraint.*
- struct [vrna\\_hc\\_s](#)  
*The hard constraints data structure. [More...](#)*
- struct [vrna\\_hc\\_up\\_s](#)  
*A single hard constraint for a single nucleotide. [More...](#)*

### Macros

- `#define VRNA_CONSTRAINT_NO_HEADER 0`  
*do not print the header information line*
- `#define VRNA_CONSTRAINT_DB 16384U`  
*Flag for [vrna\\_constraints\\_add\(\)](#) to indicate that constraint is passed in pseudo dot-bracket notation.*
- `#define VRNA_CONSTRAINT_DB_ENFORCE_BP 32768U`  
*Switch for dot-bracket structure constraint to enforce base pairs.*
- `#define VRNA_CONSTRAINT_DB_PIPE 65536U`  
*Flag that is used to indicate the pipe '|' sign in pseudo dot-bracket notation of hard constraints.*
- `#define VRNA_CONSTRAINT_DB_DOT 131072U`  
*dot '.' switch for structure constraints (no constraint at all)*
- `#define VRNA_CONSTRAINT_DB_X 262144U`  
*'x' switch for structure constraint (base must not pair)*
- `#define VRNA_CONSTRAINT_DB_ANG_BRACK 524288U`  
*angle brackets '<', '>' switch for structure constraint (paired downstream/upstream)*
- `#define VRNA_CONSTRAINT_DB_RND_BRACK 1048576U`  
*round brackets '(', ')' switch for structure constraint (base i pairs base j)*
- `#define VRNA_CONSTRAINT_DB_INTRAMOL 2097152U`  
*Flag that is used to indicate the character 'I' in pseudo dot-bracket notation of hard constraints.*
- `#define VRNA_CONSTRAINT_DB_INTERMOL 4194304U`  
*Flag that is used to indicate the character 'e' in pseudo dot-bracket notation of hard constraints.*
- `#define VRNA_CONSTRAINT_DB_GQUAD 8388608U`  
*'+' switch for structure constraint (base is involved in a gquad)*
- `#define VRNA_CONSTRAINT_DB_WUSS 33554432U`  
*Flag to indicate Washington University Secondary Structure (WUSS) notation of the hard constraint string.*
- `#define VRNA_CONSTRAINT_DB_DEFAULT`  
*Switch for dot-bracket structure constraint with default symbols.*

- `#define VRNA_CONSTRAINT_CONTEXT_EXT_LOOP` (unsigned char)0x01  
*Hard constraints flag, base pair in the exterior loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_HP_LOOP` (unsigned char)0x02  
*Hard constraints flag, base pair encloses hairpin loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_INT_LOOP` (unsigned char)0x04  
*Hard constraints flag, base pair encloses an interior loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_INT_LOOP_ENC` (unsigned char)0x08  
*Hard constraints flag, base pair encloses a multi branch loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_MB_LOOP` (unsigned char)0x10  
*Hard constraints flag, base pair is enclosed in an interior loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_MB_LOOP_ENC` (unsigned char)0x20  
*Hard constraints flag, base pair is enclosed in a multi branch loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_ENFORCE` (unsigned char)0x40  
*Hard constraint flag to indicate enforcement of constraints.*
- `#define VRNA_CONSTRAINT_CONTEXT_NO_REMOVE` (unsigned char)0x80  
*Hard constraint flag to indicate not to remove base pairs that conflict with a given constraint.*
- `#define VRNA_CONSTRAINT_CONTEXT_NONE` (unsigned char)0  
*Constraint context flag that forbids any loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_CLOSING_LOOPS`  
*Constraint context flag indicating base pairs that close any loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_ENCLOSED_LOOPS`  
*Constraint context flag indicating base pairs enclosed by any loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_ALL_LOOPS`  
*Constraint context flag indicating any loop context.*

## Typedefs

- `typedef struct vrna_hc_s vrna_hc_t`  
*Typename for the hard constraints data structure `vrna_hc_s`.*
- `typedef struct vrna_hc_up_s vrna_hc_up_t`  
*Typename for the single nucleotide hard constraint data structure `vrna_hc_up_s`.*
- `typedef unsigned char() vrna_callback_hc_evaluate(int i, int j, int k, int l, unsigned char d, void *data)`  
*Callback to evaluate whether or not a particular decomposition step is contributing to the solution space.*

## Enumerations

- `enum vrna_hc_type_e { VRNA_HC_DEFAULT, VRNA_HC_WINDOW }`  
*The hard constraints type.*

## Functions

- `void vrna_message_constraint_options` (unsigned int option)  
*Print a help message for pseudo dot-bracket structure constraint characters to stdout. (constraint support is specified by option parameter)*
- `void vrna_message_constraint_options_all` (void)  
*Print structure constraint characters to stdout (full constraint support)*
- `void vrna_hc_init` (`vrna_fold_compound_t` \*vc)  
*Initialize/Reset hard constraints to default values.*
- `void vrna_hc_add_up` (`vrna_fold_compound_t` \*vc, int i, unsigned char option)

- *Make a certain nucleotide unpaired.*  
 • int [vrna\\_hc\\_add\\_up\\_batch](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_hc\\_up\\_t](#) \*constraints)
- *Apply a list of hard constraints for single nucleotides.*  
 • void [vrna\\_hc\\_add\\_bp](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int i, int j, unsigned char option)
- *Favorize/Enforce a certain base pair (i,j)*  
 • void [vrna\\_hc\\_add\\_bp\\_nonspecific](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int i, int d, unsigned char option)
- *Enforce a nucleotide to be paired (upstream/downstream)*  
 • void [vrna\\_hc\\_free](#) ([vrna\\_hc\\_t](#) \*hc)
- *Free the memory allocated by a [vrna\\_hc\\_t](#) data structure.*  
 • void [vrna\\_hc\\_add\\_f](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_callback\\_hc\\_evaluate](#) \*f)
- *Add a function pointer pointer for the generic hard constraint feature.*  
 • void [vrna\\_hc\\_add\\_data](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, void \*data, [vrna\\_callback\\_free\\_auxdata](#) \*f)
- *Add an auxiliary data structure for the generic hard constraints callback function.*  
 • int [vrna\\_hc\\_add\\_from\\_db](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*constraint, unsigned int options)
- *Add hard constraints from pseudo dot-bracket notation.*  
 • void [print\\_tty\\_constraint](#) (unsigned int option)
- *Print structure constraint characters to stdout. (constraint support is specified by option parameter)*  
 • void [print\\_tty\\_constraint\\_full](#) (void)
- *Print structure constraint characters to stdout (full constraint support)*  
 • void [constrain\\_ptypes](#) (const char \*constraint, unsigned int length, char \*ptype, int \*BP, int min\_loop\_size, unsigned int idx\_type)
- *Insert constraining pair types according to constraint structure string.*

### 17.15.1 Detailed Description

Functions and data structures for handling of secondary structure hard constraints.

### 17.15.2 Macro Definition Documentation

#### 17.15.2.1 VRNA\_CONSTRAINT\_NO\_HEADER

```
#define VRNA_CONSTRAINT_NO_HEADER 0
```

do not print the header information line

**Deprecated** This mode is not supported anymore!

#### 17.15.2.2 VRNA\_CONSTRAINT\_DB\_ANG\_BRACK

```
#define VRNA_CONSTRAINT_DB_ANG_BRACK 524288U
```

angle brackets '<', '>' switch for structure constraint (paired downstream/upstream)

See also

[vrna\\_hc\\_add\\_from\\_db\(\)](#), [vrna\\_constraints\\_add\(\)](#), [vrna\\_message\\_constraint\\_options\(\)](#), [vrna\\_message\\_constraint\\_options\\_all\(\)](#)

### 17.15.3 Enumeration Type Documentation

#### 17.15.3.1 vrna\_hc\_type\_e

enum [vrna\\_hc\\_type\\_e](#)

The hard constraints type.

Global and local structure prediction methods use a slightly different way to handle hard constraints internally. This enum is used to distinguish both types.

##### Enumerator

VRNA_HC_DEFAULT	Default Hard Constraints.
VRNA_HC_WINDOW	Hard Constraints suitable for local structure prediction using window approach.  See also <a href="#">vrna_mfe_window()</a> , <a href="#">vrna_mfe_window_zscore()</a> , <a href="#">pfl_fold()</a>

### 17.15.4 Function Documentation

#### 17.15.4.1 vrna\_hc\_add\_data()

```
void vrna_hc_add_data (
    vrna\_fold\_compound\_t * vc,
    void * data,
    vrna\_callback\_free\_auxdata * f )
```

Add an auxiliary data structure for the generic hard constraints callback function.

##### See also

[vrna\\_hc\\_add\\_f\(\)](#)

##### Parameters

<i>vc</i>	The fold compound the generic hard constraint function should be bound to
<i>data</i>	A pointer to the data structure that holds required data for function 'f'
<i>f</i>	A pointer to a function that free's the memory occupied by <i>data</i> (Maybe NULL)



17.15.4.2 `print_tty_constraint()`

```
void print_tty_constraint (
    unsigned int option )
```

Print structure constraint characters to stdout. (constraint support is specified by option parameter)

**Deprecated** Use `vrna_message_constraints()` instead!

## Parameters

<i>option</i>	Option switch that tells which constraint help will be printed
---------------	--

17.15.4.3 `print_tty_constraint_full()`

```
void print_tty_constraint_full (
    void )
```

Print structure constraint characters to stdout (full constraint support)

**Deprecated** Use `vrna_message_constraint_options_all()` instead!

17.15.4.4 `constrain_ptypes()`

```
void constrain_ptypes (
    const char * constraint,
    unsigned int length,
    char * ptype,
    int * BP,
    int min_loop_size,
    unsigned int idx_type )
```

Insert constraining pair types according to constraint structure string.

**Deprecated** Do not use this function anymore! Structure constraints are now handled through `vrna_hc_t` and related functions.

## Parameters

<i>constraint</i>	The structure constraint string
<i>length</i>	The actual length of the sequence (constraint may be shorter)
<i>ptype</i>	A pointer to the basepair type array
<i>BP</i>	(not used anymore)
<i>min_loop_size</i>	The minimal loop size (usually <a href="#">TURN</a> )
<i>idx_type</i>	Define the access type for base pair type array (0 = indx, 1 = iindx)

## 17.16 ViennaRNA/constraints/ligand.h File Reference

Functions for incorporation of ligands binding to hairpin and interior loop motifs using the soft constraints framework.

Include dependency graph for ligand.h:

This graph shows which files directly or indirectly include this file:

### Data Structures

- struct [vrna\\_sc\\_motif\\_s](#)

### Functions

- int [vrna\\_sc\\_add\\_hi\\_motif](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*seq, const char \*structure, [FLT\\_OR\\_DBL](#) energy, unsigned int options)  
*Add soft constraints for hairpin or interior loop binding motif.*

### 17.16.1 Detailed Description

Functions for incorporation of ligands binding to hairpin and interior loop motifs using the soft constraints framework.

## 17.17 ViennaRNA/constraints/SHAPE.h File Reference

This module provides function to incorporate SHAPE reactivity data into the folding recursions by means of soft constraints.

Include dependency graph for SHAPE.h:

This graph shows which files directly or indirectly include this file:

## Functions

- int [vrna\\_sc\\_add\\_SHAPE\\_deigan](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const double \*reactivities, double m, double b, unsigned int options)  
*Add SHAPE reactivity data as soft constraints (Deigan et al. method)*
- int [vrna\\_sc\\_add\\_SHAPE\\_deigan.ali](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*\*shape\_files, const int \*shape\_file\_association, double m, double b, unsigned int options)  
*Add SHAPE reactivity data from files as soft constraints for consensus structure prediction (Deigan et al. method)*
- int [vrna\\_sc\\_add\\_SHAPE\\_zarringhalam](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const double \*reactivities, double b, double default\_value, const char \*shape\_conversion, unsigned int options)  
*Add SHAPE reactivity data as soft constraints (Zarringhalam et al. method)*
- int [vrna\\_sc\\_SHAPE\\_parse\\_method](#) (const char \*method\_string, char \*method, float \*param\_1, float \*param\_2)  
*Parse a character string and extract the encoded SHAPE reactivity conversion method and possibly the parameters for conversion into pseudo free energies.*
- int [vrna\\_sc\\_SHAPE\\_to\\_pr](#) (const char \*shape\_conversion, double \*values, int length, double default\_value)  
*Convert SHAPE reactivity values to probabilities for being unpaired.*

### 17.17.1 Detailed Description

This module provides function to incorporate SHAPE reactivity data into the folding recursions by means of soft constraints.

### 17.17.2 Function Documentation

#### 17.17.2.1 [vrna\\_sc\\_SHAPE\\_parse\\_method\(\)](#)

```
int vrna_sc_SHAPE_parse_method (
    const char * method_string,
    char * method,
    float * param_1,
    float * param_2 )
```

Parse a character string and extract the encoded SHAPE reactivity conversion method and possibly the parameters for conversion into pseudo free energies.

#### Parameters

<i>method_string</i>	The string that contains the encoded SHAPE reactivity conversion method
<i>method</i>	A pointer to the memory location where the method character will be stored
<i>param_1</i>	A pointer to the memory location where the first parameter of the corresponding method will be stored
<i>param_2</i>	A pointer to the memory location where the second parameter of the corresponding method will be stored

**Returns**

1 on successful extraction of the method, 0 on errors

**17.18 ViennaRNA/constraints/soft.h File Reference**

Functions and data structures for secondary structure soft constraints.

Include dependency graph for soft.h:

This graph shows which files directly or indirectly include this file:

**Data Structures**

- struct [vrna\\_sc\\_bp\\_storage\\_t](#)  
*A base pair constraint.*
- struct [vrna\\_sc\\_s](#)  
*The soft constraints data structure. [More...](#)*

**Typedefs**

- typedef struct [vrna\\_sc\\_s](#) [vrna\\_sc\\_t](#)  
*Typename for the soft constraints data structure [vrna\\_sc\\_s](#).*
- typedef int() [vrna\\_callback\\_sc\\_energy](#)(int i, int j, int k, int l, unsigned char d, void \*data)  
*Callback to retrieve pseudo energy contribution for soft constraint feature.*
- typedef [FLT\\_OR\\_DBL](#)() [vrna\\_callback\\_sc\\_exp\\_energy](#)(int i, int j, int k, int l, unsigned char d, void \*data)  
*Callback to retrieve pseudo energy contribution as Boltzmann Factors for soft constraint feature.*
- typedef [vrna\\_basepair\\_t](#) \*() [vrna\\_callback\\_sc\\_backtrack](#)(int i, int j, int k, int l, unsigned char d, void \*data)  
*Callback to retrieve auxiliary base pairs for soft constraint feature.*

**Enumerations**

- enum [vrna\\_sc\\_type\\_e](#) { [VRNA\\_SC\\_DEFAULT](#), [VRNA\\_SC\\_WINDOW](#) }  
*The type of a soft constraint.*

## Functions

- void [vrna\\_sc\\_init](#) ([vrna\\_fold\\_compound\\_t](#) \*vc)  
*Initialize an empty soft constraints data structure within a [vrna\\_fold\\_compound\\_t](#).*
- void [vrna\\_sc\\_set\\_bp](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const [FLT\\_OR\\_DBL](#) \*\*constraints, unsigned int options)  
*Set soft constraints for paired nucleotides.*
- void [vrna\\_sc\\_add\\_bp](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int i, int j, [FLT\\_OR\\_DBL](#) energy, unsigned int options)  
*Add soft constraints for paired nucleotides.*
- void [vrna\\_sc\\_set\\_up](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const [FLT\\_OR\\_DBL](#) \*constraints, unsigned int options)  
*Set soft constraints for unpaired nucleotides.*
- void [vrna\\_sc\\_add\\_up](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int i, [FLT\\_OR\\_DBL](#) energy, unsigned int options)  
*Add soft constraints for unpaired nucleotides.*
- void [vrna\\_sc\\_remove](#) ([vrna\\_fold\\_compound\\_t](#) \*vc)  
*Remove soft constraints from [vrna\\_fold\\_compound\\_t](#).*
- void [vrna\\_sc\\_free](#) ([vrna\\_sc\\_t](#) \*sc)  
*Free memory occupied by a [vrna\\_sc\\_t](#) data structure.*
- void [vrna\\_sc\\_add\\_data](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, void \*data, [vrna\\_callback\\_free\\_auxdata](#) \*free\_data)  
*Add an auxiliary data structure for the generic soft constraints callback function.*
- void [vrna\\_sc\\_add\\_f](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_callback\\_sc\\_energy](#) \*f)  
*Bind a function pointer for generic soft constraint feature (MFE version)*
- void [vrna\\_sc\\_add\\_bt](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_callback\\_sc\\_backtrack](#) \*f)  
*Bind a backtracking function pointer for generic soft constraint feature.*
- void [vrna\\_sc\\_add\\_exp\\_f](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_callback\\_sc\\_exp\\_energy](#) \*exp\_f)  
*Bind a function pointer for generic soft constraint feature (PF version)*

### 17.18.1 Detailed Description

Functions and data structures for secondary structure soft constraints.

### 17.18.2 Enumeration Type Documentation

#### 17.18.2.1 [vrna\\_sc\\_type\\_e](#)

enum [vrna\\_sc\\_type\\_e](#)

The type of a soft constraint.

#### Enumerator

<a href="#">VRNA_SC_DEFAULT</a>	Default Soft Constraints.
<a href="#">VRNA_SC_WINDOW</a>	Soft Constraints suitable for local structure prediction using window approach.  See also  <a href="#">vrna_mfe_window()</a> , <a href="#">vrna_mfe_window_zscore()</a> , <a href="#">pfl_fold()</a>

## 17.19 ViennaRNA/constraints\_hard.h File Reference

Use [ViennaRNA/constraints/hard.h](#) instead.

Include dependency graph for constraints\_hard.h:

### 17.19.1 Detailed Description

Use [ViennaRNA/constraints/hard.h](#) instead.

**Deprecated** Use [ViennaRNA/constraints/hard.h](#) instead

## 17.20 ViennaRNA/constraints\_ligand.h File Reference

Use [ViennaRNA/constraints/ligand.h](#) instead.

Include dependency graph for constraints\_ligand.h:

### 17.20.1 Detailed Description

Use [ViennaRNA/constraints/ligand.h](#) instead.

**Deprecated** Use [ViennaRNA/constraints/ligand.h](#) instead

## 17.21 ViennaRNA/constraints\_SHAPE.h File Reference

Use [ViennaRNA/constraints/SHAPE.h](#) instead.

Include dependency graph for constraints\_SHAPE.h:

### 17.21.1 Detailed Description

Use [ViennaRNA/constraints/SHAPE.h](#) instead.

**Deprecated** Use [ViennaRNA/constraints/SHAPE.h](#) instead

## 17.22 ViennaRNA/constraints\_soft.h File Reference

Use [ViennaRNA/constraints/soft.h](#) instead.

Include dependency graph for constraints\_soft.h:

### 17.22.1 Detailed Description

Use [ViennaRNA/constraints/soft.h](#) instead.

**Deprecated** Use [ViennaRNA/constraints/soft.h](#) instead

## 17.23 ViennaRNA/convert\_epars.h File Reference

Use [ViennaRNA/params/convert.h](#) instead.

Include dependency graph for convert\_epars.h:

### 17.23.1 Detailed Description

Use [ViennaRNA/params/convert.h](#) instead.

**Deprecated** Use [ViennaRNA/params/convert.h](#) instead

## 17.24 ViennaRNA/data\_structures.h File Reference

Use [ViennaRNA/datastructures/basic.h](#) instead.

Include dependency graph for data\_structures.h:

This graph shows which files directly or indirectly include this file:

### 17.24.1 Detailed Description

Use [ViennaRNA/datastructures/basic.h](#) instead.

**Deprecated** Use [ViennaRNA/datastructures/basic.h](#) instead

## 17.25 ViennaRNA/datastructures/hash\_tables.h File Reference

Implementations of hash table functions.

### Data Structures

- struct [vrna\\_ht\\_entry\\_db\\_t](#)  
*Default hash table entry. [More...](#)*

### Functions

#### Dot-Bracket / Free Energy entries

- int [vrna\\_ht\\_db\\_comp](#) (void \*x, void \*y)  
*Default hash table entry comparison.*
- unsigned int [vrna\\_ht\\_db\\_hash\\_func](#) (void \*x, unsigned long hashtable\_size)  
*Default hash function.*
- int [vrna\\_ht\\_db\\_free\\_entry](#) (void \*hash\_entry)  
*Default function to free memory occupied by a hash entry.*

### Abstract interface

- typedef struct vrna\_hash\_table\_s \* [vrna\\_hash\\_table\\_t](#)  
*A hash table object.*
- typedef int() [vrna\\_callback\\_ht\\_compare\\_entries](#)(void \*x, void \*y)  
*Callback function to compare two hash table entries.*
- typedef unsigned int() [vrna\\_callback\\_ht\\_hash\\_function](#)(void \*x, unsigned long hashtable\_size)  
*Callback function to generate a hash key, i.e. hash function.*
- typedef int() [vrna\\_callback\\_ht\\_free\\_entry](#)(void \*x)  
*Callback function to free a hash table entry.*
- [vrna\\_hash\\_table\\_t](#) [vrna\\_ht\\_init](#) (unsigned int b, [vrna\\_callback\\_ht\\_compare\\_entries](#) \*compare\_function, [vrna\\_callback\\_ht\\_hash\\_function](#) \*hash\_function, [vrna\\_callback\\_ht\\_free\\_entry](#) \*free\_hash\_entry)  
*Get an initialized hash table.*
- unsigned long [vrna\\_ht\\_size](#) ([vrna\\_hash\\_table\\_t](#) ht)  
*Get the size of the hash table.*
- unsigned long [vrna\\_ht\\_collisions](#) (struct vrna\_hash\_table\_s \*ht)  
*Get the number of collisions in the hash table.*
- void \* [vrna\\_ht\\_get](#) ([vrna\\_hash\\_table\\_t](#) ht, void \*x)  
*Get an element from the hash table.*
- int [vrna\\_ht\\_insert](#) ([vrna\\_hash\\_table\\_t](#) ht, void \*x)  
*Insert an object into a hash table.*
- void [vrna\\_ht\\_remove](#) ([vrna\\_hash\\_table\\_t](#) ht, void \*x)  
*Remove an object from the hash table.*
- void [vrna\\_ht\\_clear](#) ([vrna\\_hash\\_table\\_t](#) ht)  
*Clear the hash table.*
- void [vrna\\_ht\\_free](#) ([vrna\\_hash\\_table\\_t](#) ht)  
*Free all memory occupied by the hash table.*



### 17.25.1 Detailed Description

Implementations of hash table functions.

## 17.26 ViennaRNA/dist\_vars.h File Reference

Global variables for Distance-Package.

This graph shows which files directly or indirectly include this file:

### Data Structures

- struct [Postorder\\_list](#)  
*Postorder data structure.*
- struct [Tree](#)  
*Tree data structure.*
- struct [swString](#)  
*Some other data structure.*

### Variables

- int [edit\\_backtrack](#)  
*Produce an alignment of the two structures being compared by tracing the editing path giving the minimum distance.*
- char \* [aligned\\_line](#) [4]  
*Contains the two aligned structures after a call to one of the distance functions with [edit\\_backtrack](#) set to 1.*
- int [cost\\_matrix](#)  
*Specify the cost matrix to be used for distance calculations.*

### 17.26.1 Detailed Description

Global variables for Distance-Package.

### 17.26.2 Variable Documentation

#### 17.26.2.1 [edit\\_backtrack](#)

```
int edit_backtrack
```

Produce an alignment of the two structures being compared by tracing the editing path giving the minimum distance.

set to 1 if you want backtracking

### 17.26.2.2 cost\_matrix

```
int cost_matrix
```

Specify the cost matrix to be used for distance calculations.

if 0, use the default cost matrix (upper matrix in example), otherwise use Shapiro's costs (lower matrix).

## 17.27 ViennaRNA/dp\_matrices.h File Reference

Functions to deal with standard dynamic programming (DP) matrices.

Include dependency graph for dp\_matrices.h:

This graph shows which files directly or indirectly include this file:

### Data Structures

- struct [vrna\\_mx\\_mfe\\_s](#)  
*Minimum Free Energy (MFE) Dynamic Programming (DP) matrices data structure required within the [vrna\\_fold\\_compound\\_t](#). [More...](#)*
- struct [vrna\\_mx\\_pf\\_s](#)  
*Partition function (PF) Dynamic Programming (DP) matrices data structure required within the [vrna\\_fold\\_compound\\_t](#). [More...](#)*

### Typedefs

- typedef struct [vrna\\_mx\\_mfe\\_s](#) [vrna\\_mx\\_mfe\\_t](#)  
*Typename for the Minimum Free Energy (MFE) DP matrices data structure [vrna\\_mx\\_mfe\\_s](#).*
- typedef struct [vrna\\_mx\\_pf\\_s](#) [vrna\\_mx\\_pf\\_t](#)  
*Typename for the Partition Function (PF) DP matrices data structure [vrna\\_mx\\_pf\\_s](#).*

### Enumerations

- enum [vrna\\_mx\\_type\\_e](#) { [VRNA\\_MX\\_DEFAULT](#), [VRNA\\_MX\\_WINDOW](#), [VRNA\\_MX\\_2DFOLD](#) }  
*An enumerator that is used to specify the type of a polymorphic Dynamic Programming (DP) matrix data structure.*

### Functions

- int [vrna\\_mx\\_add](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_mx\\_type\\_e](#) type, unsigned int options)  
*Add Dynamic Programming (DP) matrices (allocate memory)*
- void [vrna\\_mx\\_mfe\\_free](#) ([vrna\\_fold\\_compound\\_t](#) \*vc)  
*Free memory occupied by the Minimum Free Energy (MFE) Dynamic Programming (DP) matrices.*
- void [vrna\\_mx\\_pf\\_free](#) ([vrna\\_fold\\_compound\\_t](#) \*vc)  
*Free memory occupied by the Partition Function (PF) Dynamic Programming (DP) matrices.*

### 17.27.1 Detailed Description

Functions to deal with standard dynamic programming (DP) matrices.

## 17.28 ViennaRNA/duplex.h File Reference

Functions for simple RNA-RNA duplex interactions.

Include dependency graph for duplex.h:

### 17.28.1 Detailed Description

Functions for simple RNA-RNA duplex interactions.

## 17.29 ViennaRNA/edit\_cost.h File Reference

global variables for Edit Costs included by treedist.c and stringdist.c

### 17.29.1 Detailed Description

global variables for Edit Costs included by treedist.c and stringdist.c

## 17.30 ViennaRNA/energy\_const.h File Reference

Use [ViennaRNA/params/constants.h](#) instead.

Include dependency graph for energy\_const.h:

### 17.30.1 Detailed Description

Use [ViennaRNA/params/constants.h](#) instead.

**Deprecated** Use [ViennaRNA/params/constants.h](#) instead

## 17.31 ViennaRNA/energy\_par.h File Reference

Use [ViennaRNA/params/default.h](#) instead.

Include dependency graph for energy\_par.h:

### 17.31.1 Detailed Description

Use [ViennaRNA/params/default.h](#) instead.

**Deprecated** Use [ViennaRNA/params/default.h](#) instead

## 17.32 ViennaRNA/equilibrium\_probs.h File Reference

Equilibrium Probability implementations.

Include dependency graph for equilibrium\_probs.h:

This graph shows which files directly or indirectly include this file:

### Functions

- void [vrna\\_pf\\_dimer\\_probs](#) (double FAB, double FA, double FB, [vrna\\_ep\\_t](#) \*prAB, const [vrna\\_ep\\_t](#) \*prA, const [vrna\\_ep\\_t](#) \*prB, int Alength, const [vrna\\_exp\\_param\\_t](#) \*exp\_params)  
*Compute Boltzmann probabilities of dimerization without homodimers.*
- double [vrna\\_pr\\_structure](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, const char \*structure)  
*Compute the equilibrium probability of a particular secondary structure.*

### Base pair related probability computations

- double [vrna\\_mean\\_bp\\_distance\\_pr](#) (int length, [FLT\\_OR\\_DBL](#) \*pr)  
*Get the mean base pair distance in the thermodynamic ensemble from a probability matrix.*
- double [vrna\\_mean\\_bp\\_distance](#) ([vrna\\_fold\\_compound\\_t](#) \*vc)  
*Get the mean base pair distance in the thermodynamic ensemble.*
- double [vrna\\_ensemble\\_defect](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, const char \*structure)  
*Compute the Ensemble Defect for a given target structure.*
- [vrna\\_ep\\_t](#) \* [vrna\\_stack\\_prob](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, double cutoff)  
*Compute stacking probabilities.*

### 17.32.1 Detailed Description

Equilibrium Probability implementations.

This file includes various implementations for equilibrium probability computations based on the partition function of an RNA sequence, two concatenated sequences, or a sequence alignment.

## 17.33 ViennaRNA/eval.h File Reference

Functions and variables related to energy evaluation of sequence/structure pairs.

Include dependency graph for eval.h:

This graph shows which files directly or indirectly include this file:

### Macros

- `#define VRNA_VERBOSITY_QUIET -1`  
*Quiet level verbosity setting.*
- `#define VRNA_VERBOSITY_DEFAULT 1`  
*Default level verbosity setting.*

### Functions

- `int vrna_eval_loop_pt (vrna_fold_compound_t *vc, int i, const short *pt)`  
*Calculate energy of a loop.*
- `int vrna_eval_loop_pt_v (vrna_fold_compound_t *vc, int i, const short *pt, int verbosity_level)`  
*Calculate energy of a loop.*
- `float vrna_eval_move (vrna_fold_compound_t *vc, const char *structure, int m1, int m2)`  
*Calculate energy of a move (closing or opening of a base pair)*
- `int vrna_eval_move_pt (vrna_fold_compound_t *vc, short *pt, int m1, int m2)`  
*Calculate energy of a move (closing or opening of a base pair)*
- `float energy_of_structure (const char *string, const char *structure, int verbosity_level)`  
*Calculate the free energy of an already folded RNA using global model detail settings.*
- `float energy_of_struct_par (const char *string, const char *structure, vrna_param_t *parameters, int verbosity_level)`  
*Calculate the free energy of an already folded RNA.*
- `float energy_of_circ_structure (const char *string, const char *structure, int verbosity_level)`  
*Calculate the free energy of an already folded circular RNA.*
- `float energy_of_circ_struct_par (const char *string, const char *structure, vrna_param_t *parameters, int verbosity_level)`  
*Calculate the free energy of an already folded circular RNA.*
- `int energy_of_structure_pt (const char *string, short *ptable, short *s, short *s1, int verbosity_level)`  
*Calculate the free energy of an already folded RNA.*

- int `energy_of_struct_pt_par` (const char \*string, short \*ptable, short \*s, short \*s1, `vrna_param_t` \*parameters, int verbosity\_level)  
*Calculate the free energy of an already folded RNA.*
- float `energy_of_move` (const char \*string, const char \*structure, int m1, int m2)  
*Calculate energy of a move (closing or opening of a base pair)*
- int `energy_of_move_pt` (short \*pt, short \*s, short \*s1, int m1, int m2)  
*Calculate energy of a move (closing or opening of a base pair)*
- int `loop_energy` (short \*ptable, short \*s, short \*s1, int i)  
*Calculate energy of a loop.*
- float `energy_of_struct` (const char \*string, const char \*structure)
- int `energy_of_struct_pt` (const char \*string, short \*ptable, short \*s, short \*s1)
- float `energy_of_circ_struct` (const char \*string, const char \*structure)

### Basic Energy Evaluation Interface with Dot-Bracket Structure String

- float `vrna_eval_structure` (`vrna_fold_compound_t` \*vc, const char \*structure)  
*Calculate the free energy of an already folded RNA.*
- float `vrna_eval_covar_structure` (`vrna_fold_compound_t` \*vc, const char \*structure)  
*Calculate the pseudo energy derived by the covariance scores of a set of aligned sequences.*
- float `vrna_eval_structure_verbose` (`vrna_fold_compound_t` \*vc, const char \*structure, FILE \*file)  
*Calculate the free energy of an already folded RNA and print contributions on a per-loop base.*
- float `vrna_eval_structure_v` (`vrna_fold_compound_t` \*vc, const char \*structure, int verbosity\_level, FILE \*file)  
*Calculate the free energy of an already folded RNA and print contributions on a per-loop base.*
- float `vrna_eval_structure_cstr` (`vrna_fold_compound_t` \*vc, const char \*structure, int verbosity\_level, `vrna_cstr_t` output\_stream)

### Basic Energy Evaluation Interface with Structure Pair Table

- int `vrna_eval_structure_pt` (`vrna_fold_compound_t` \*vc, const short \*pt)  
*Calculate the free energy of an already folded RNA.*
- int `vrna_eval_structure_pt_verbose` (`vrna_fold_compound_t` \*vc, const short \*pt, FILE \*file)  
*Calculate the free energy of an already folded RNA.*
- int `vrna_eval_structure_pt_v` (`vrna_fold_compound_t` \*vc, const short \*pt, int verbosity\_level, FILE \*file)  
*Calculate the free energy of an already folded RNA.*

### Simplified Energy Evaluation with Sequence and Dot-Bracket Strings

- float `vrna_eval_structure_simple` (const char \*string, const char \*structure)  
*Calculate the free energy of an already folded RNA.*
- float `vrna_eval_circ_structure` (const char \*string, const char \*structure)  
*Evaluate the free energy of a sequence/structure pair where the sequence is circular.*
- float `vrna_eval_gquad_structure` (const char \*string, const char \*structure)  
*Evaluate the free energy of a sequence/structure pair where the structure may contain G-Quadruplexes.*
- float `vrna_eval_circ_gquad_structure` (const char \*string, const char \*structure)  
*Evaluate the free energy of a sequence/structure pair where the sequence is circular and the structure may contain G-Quadruplexes.*
- float `vrna_eval_structure_simple_verbose` (const char \*string, const char \*structure, FILE \*file)  
*Calculate the free energy of an already folded RNA and print contributions per loop.*
- float `vrna_eval_structure_simple_v` (const char \*string, const char \*structure, int verbosity\_level, FILE \*file)  
*Calculate the free energy of an already folded RNA and print contributions per loop.*
- float `vrna_eval_circ_structure_v` (const char \*string, const char \*structure, int verbosity\_level, FILE \*file)  
*Evaluate free energy of a sequence/structure pair, assume sequence to be circular and print contributions per loop.*

- float [vrna\\_eval\\_gquad\\_structure\\_v](#) (const char \*string, const char \*structure, int verbosity\_level, FILE \*file)  
*Evaluate free energy of a sequence/structure pair, allow for G-Quadruplexes in the structure and print contributions per loop.*
- float [vrna\\_eval\\_circ\\_gquad\\_structure\\_v](#) (const char \*string, const char \*structure, int verbosity\_level, FILE \*file)  
*Evaluate free energy of a sequence/structure pair, assume sequence to be circular, allow for G-Quadruplexes in the structure, and print contributions per loop.*

### Simplified Energy Evaluation with Sequence Alignments and Consensus Structure Dot-Bracket String

- float [vrna\\_eval\\_consensus\\_structure\\_simple](#) (const char \*\*alignment, const char \*structure)  
*Calculate the free energy of an already folded RNA sequence alignment.*
- float [vrna\\_eval\\_circ\\_consensus\\_structure](#) (const char \*\*alignment, const char \*structure)  
*Evaluate the free energy of a multiple sequence alignment/consensus structure pair where the sequences are circular.*
- float [vrna\\_eval\\_gquad\\_consensus\\_structure](#) (const char \*\*alignment, const char \*structure)  
*Evaluate the free energy of a multiple sequence alignment/consensus structure pair where the structure may contain G-Quadruplexes.*
- float [vrna\\_eval\\_circ\\_gquad\\_consensus\\_structure](#) (const char \*\*alignment, const char \*structure)  
*Evaluate the free energy of a multiple sequence alignment/consensus structure pair where the sequence is circular and the structure may contain G-Quadruplexes.*
- float [vrna\\_eval\\_consensus\\_structure\\_simple\\_verbose](#) (const char \*\*alignment, const char \*structure, FILE \*file)  
*Evaluate the free energy of a consensus structure for an RNA sequence alignment and print contributions per loop.*
- float [vrna\\_eval\\_consensus\\_structure\\_simple\\_v](#) (const char \*\*alignment, const char \*structure, int verbosity\_level, FILE \*file)  
*Evaluate the free energy of a consensus structure for an RNA sequence alignment and print contributions per loop.*
- float [vrna\\_eval\\_circ\\_consensus\\_structure\\_v](#) (const char \*\*alignment, const char \*structure, int verbosity\_level, FILE \*file)  
*Evaluate the free energy of a consensus structure for an alignment of circular RNA sequences and print contributions per loop.*
- float [vrna\\_eval\\_gquad\\_consensus\\_structure\\_v](#) (const char \*\*alignment, const char \*structure, int verbosity\_level, FILE \*file)  
*Evaluate the free energy of a consensus structure for an RNA sequence alignment, allow for annotated G-Quadruplexes in the structure and print contributions per loop.*
- float [vrna\\_eval\\_circ\\_gquad\\_consensus\\_structure\\_v](#) (const char \*\*alignment, const char \*structure, int verbosity\_level, FILE \*file)  
*Evaluate the free energy of a consensus structure for an alignment of circular RNA sequences, allow for annotated G-Quadruplexes in the structure and print contributions per loop.*

### Simplified Energy Evaluation with Sequence String and Structure Pair Table

- int [vrna\\_eval\\_structure\\_pt\\_simple](#) (const char \*string, const short \*pt)  
*Calculate the free energy of an already folded RNA.*
- int [vrna\\_eval\\_structure\\_pt\\_simple\\_verbose](#) (const char \*string, const short \*pt, FILE \*file)  
*Calculate the free energy of an already folded RNA.*
- int [vrna\\_eval\\_structure\\_pt\\_simple\\_v](#) (const char \*string, const short \*pt, int verbosity\_level, FILE \*file)  
*Calculate the free energy of an already folded RNA.*

### Simplified Energy Evaluation with Sequence Alignment and Consensus Structure Pair Table

- int [vrna\\_eval\\_consensus\\_structure\\_pt\\_simple](#) (const char \*\*alignment, const short \*pt)  
*Evaluate the Free Energy of a Consensus Secondary Structure given a Sequence Alignment.*
- int [vrna\\_eval\\_consensus\\_structure\\_pt\\_simple\\_verbose](#) (const char \*\*alignment, const short \*pt, FILE \*file)
- int [vrna\\_eval\\_consensus\\_structure\\_pt\\_simple\\_v](#) (const char \*\*alignment, const short \*pt, int verbosity\_level, FILE \*file)

## Variables

- int [cut\\_point](#)  
*first pos of second seq for cofolding*
- int [eos\\_debug](#)  
*verbose info from energy\_of\_struct*

### 17.33.1 Detailed Description

Functions and variables related to energy evaluation of sequence/structure pairs.

## 17.34 ViennaRNA/exterior\_loops.h File Reference

Use [ViennaRNA/loops/external.h](#) instead.

Include dependency graph for exterior\_loops.h:

### 17.34.1 Detailed Description

Use [ViennaRNA/loops/external.h](#) instead.

**Deprecated** Use [ViennaRNA/loops/external.h](#) instead

## 17.35 ViennaRNA/file\_formats.h File Reference

Use [ViennaRNA/io/file\\_formats.h](#) instead.

Include dependency graph for file\_formats.h:

### 17.35.1 Detailed Description

Use [ViennaRNA/io/file\\_formats.h](#) instead.

**Deprecated** Use [ViennaRNA/io/file\\_formats.h](#) instead



## 17.36 ViennaRNA/io/file\_formats.h File Reference

Read and write different file formats for RNA sequences, structures.

Include dependency graph for file\_formats.h:

This graph shows which files directly or indirectly include this file:

### Macros

- `#define VRNA_OPTION_MULTILINE 32U`  
*Tell a function that an input is assumed to span several lines.*
- `#define VRNA_CONSTRAINT_MULTILINE 32U`  
*parse multiline constraint*

### Functions

- void `vrna_file_helixlist` (const char \*seq, const char \*db, float energy, FILE \*file)  
*Print a secondary structure as helix list.*
- void `vrna_file_connect` (const char \*seq, const char \*db, float energy, const char \*identifier, FILE \*file)  
*Print a secondary structure as connect table.*
- void `vrna_file_bpseq` (const char \*seq, const char \*db, FILE \*file)  
*Print a secondary structure in bpseq format.*
- void `vrna_file_json` (const char \*seq, const char \*db, double energy, const char \*identifier, FILE \*file)  
*Print a secondary structure in jsonformat.*
- unsigned int `vrna_file_fasta_read_record` (char \*\*header, char \*\*sequence, char \*\*\*rest, FILE \*file, unsigned int options)  
*Get a (fasta) data set from a file or stdin.*
- char \* `vrna_extract_record_rest_structure` (const char \*\*lines, unsigned int length, unsigned int option)  
*Extract a dot-bracket structure string from (multiline)character array.*
- int `vrna_file_SHAPE_read` (const char \*file\_name, int length, double default\_value, char \*sequence, double \*values)  
*Read data from a given SHAPE reactivity input file.*
- void `vrna_extract_record_rest_constraint` (char \*\*cstruc, const char \*\*lines, unsigned int option)  
*Extract a hard constraint encoded as pseudo dot-bracket string.*
- unsigned int `read_record` (char \*\*header, char \*\*sequence, char \*\*\*rest, unsigned int options)  
*Get a data record from stdin.*

#### 17.36.1 Detailed Description

Read and write different file formats for RNA sequences, structures.

,

## 17.37 ViennaRNA/file\_formats\_msa.h File Reference

Use [ViennaRNA/io/file\\_formats\\_msa.h](#) instead.

Include dependency graph for file\_formats\_msa.h:

### 17.37.1 Detailed Description

Use [ViennaRNA/io/file\\_formats\\_msa.h](#) instead.

**Deprecated** Use [ViennaRNA/io/file\\_formats\\_msa.h](#) instead

## 17.38 ViennaRNA/io/file\_formats\_msa.h File Reference

Functions dealing with file formats for Multiple Sequence Alignments (MSA)

Include dependency graph for file\_formats\_msa.h:

This graph shows which files directly or indirectly include this file:

### Macros

- `#define VRNA_FILE_FORMAT_MSA_CLUSTAL 1U`  
*Option flag indicating ClustalW formatted files.*
- `#define VRNA_FILE_FORMAT_MSA_STOCKHOLM 2U`  
*Option flag indicating Stockholm 1.0 formatted files.*
- `#define VRNA_FILE_FORMAT_MSA_FASTA 4U`  
*Option flag indicating FASTA (Pearson) formatted files.*
- `#define VRNA_FILE_FORMAT_MSA_MAF 8U`  
*Option flag indicating MAF formatted files.*
- `#define VRNA_FILE_FORMAT_MSA_MIS 16U`  
*Option flag indicating most informative sequence (MIS) output.*
- `#define VRNA_FILE_FORMAT_MSA_DEFAULT`  
*Option flag indicating the set of default file formats.*
- `#define VRNA_FILE_FORMAT_MSA_NOCHECK 4096U`  
*Option flag to disable validation of the alignment.*
- `#define VRNA_FILE_FORMAT_MSA_UNKNOWN 8192U`  
*Return flag of `vrna_file_msa_detect_format()` to indicate unknown or malformed alignment.*
- `#define VRNA_FILE_FORMAT_MSA_APPEND 16384U`  
*Option flag indicating to append data to a multiple sequence alignment file rather than overwriting it.*
- `#define VRNA_FILE_FORMAT_MSA_QUIET 32768U`  
*Option flag to suppress unnecessary spam messages on `stderr`*
- `#define VRNA_FILE_FORMAT_MSA_SILENT 65536U`  
*Option flag to completely silence any warnings on `stderr`*

## Functions

- int [vrna\\_file\\_msa\\_read](#) (const char \*filename, char \*\*\*names, char \*\*\*aln, char \*\*id, char \*\*structure, unsigned int options)  
*Read a multiple sequence alignment from file.*
- int [vrna\\_file\\_msa\\_read\\_record](#) (FILE \*fp, char \*\*\*names, char \*\*\*aln, char \*\*id, char \*\*structure, unsigned int options)  
*Read a multiple sequence alignment from file handle.*
- unsigned int [vrna\\_file\\_msa\\_detect\\_format](#) (const char \*filename, unsigned int options)  
*Detect the format of a multiple sequence alignment file.*
- int [vrna\\_file\\_msa\\_write](#) (const char \*filename, const char \*\*names, const char \*\*aln, const char \*id, const char \*structure, const char \*source, unsigned int options)  
*Write multiple sequence alignment file.*

### 17.38.1 Detailed Description

Functions dealing with file formats for Multiple Sequence Alignments (MSA)

, ,

## 17.39 ViennaRNA/file\_utils.h File Reference

Use [ViennaRNA/io/utils.h](#) instead.

Include dependency graph for file\_utils.h:

### 17.39.1 Detailed Description

Use [ViennaRNA/io/utils.h](#) instead.

**Deprecated** Use [ViennaRNA/io/utils.h](#) instead

## 17.40 ViennaRNA/findpath.h File Reference

A breadth-first search heuristic for optimal direct folding paths.

Include dependency graph for findpath.h:

## Data Structures

- struct [vrna\\_path\\_s](#)  
*An element of a refolding path list. [More...](#)*

## Typedefs

- typedef struct [vrna\\_path\\_s](#) [vrna\\_path\\_t](#)  
*Typename for the refolding path data structure [vrna\\_path\\_s](#).*
- typedef struct [vrna\\_path\\_s](#) [path\\_t](#)  
*Old typename of [vrna\\_path\\_s](#).*

## Functions

- int [vrna\\_path\\_findpath\\_saddle](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*s1, const char \*s2, int width)  
*Find energy of a saddle point between 2 structures (search only direct path)*
- int [vrna\\_path\\_findpath\\_saddle\\_ub](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*s1, const char \*s2, int width, int maxE)  
*Find energy of a saddle point between 2 structures (search only direct path)*
- [vrna\\_path\\_t](#) \* [vrna\\_path\\_findpath](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*s1, const char \*s2, int width)  
*Find refolding path between 2 structures (search only direct path)*
- [vrna\\_path\\_t](#) \* [vrna\\_path\\_findpath\\_ub](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*s1, const char \*s2, int width, int maxE)  
*Find refolding path between 2 structures (search only direct path)*
- int [find\\_saddle](#) (const char \*seq, const char \*s1, const char \*s2, int width)  
*Find energy of a saddle point between 2 structures (search only direct path)*
- void [free\\_path](#) ([vrna\\_path\\_t](#) \*path)  
*Free memory allocated by [get\\_path\(\)](#) function.*
- [vrna\\_path\\_t](#) \* [get\\_path](#) (const char \*seq, const char \*s1, const char \*s2, int width)  
*Find refolding path between 2 structures (search only direct path)*

### 17.40.1 Detailed Description

A breadth-first search heuristic for optimal direct folding paths.

## 17.41 ViennaRNA/fold.h File Reference

MFE calculations for single RNA sequences.

Include dependency graph for fold.h:

## Functions

- float [fold\\_par](#) (const char \*sequence, char \*structure, [vrna\\_param\\_t](#) \*parameters, int is\_constrained, int is\_circular)  
*Compute minimum free energy and an appropriate secondary structure of an RNA sequence.*
- float [fold](#) (const char \*sequence, char \*structure)  
*Compute minimum free energy and an appropriate secondary structure of an RNA sequence.*
- float [circfold](#) (const char \*sequence, char \*structure)  
*Compute minimum free energy and an appropriate secondary structure of a circular RNA sequence.*
- void [free\\_arrays](#) (void)  
*Free arrays for mfe folding.*
- void [update\\_fold\\_params](#) (void)  
*Recalculate energy parameters.*
- void [update\\_fold\\_params\\_par](#) ([vrna\\_param\\_t](#) \*parameters)  
*Recalculate energy parameters.*
- void [export\\_fold\\_arrays](#) (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*indx\_p, char \*\*ptype\_p)
- void [export\\_fold\\_arrays\\_par](#) (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*indx\_p, char \*\*ptype\_p, [vrna\\_param\\_t](#) \*\*P\_p)
- void [export\\_circfold\\_arrays](#) (int \*Fc\_p, int \*FcH\_p, int \*FcI\_p, int \*FcM\_p, int \*\*fM2\_p, int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*indx\_p, char \*\*ptype\_p)
- void [export\\_circfold\\_arrays\\_par](#) (int \*Fc\_p, int \*FcH\_p, int \*FcI\_p, int \*FcM\_p, int \*\*fM2\_p, int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*indx\_p, char \*\*ptype\_p, [vrna\\_param\\_t](#) \*\*P\_p)
- int [LoopEnergy](#) (int n1, int n2, int type, int type\_2, int si1, int sj1, int sp1, int sq1)
- int [HairpinE](#) (int size, int type, int si1, int sj1, const char \*string)
- void [initialize\\_fold](#) (int length)

### 17.41.1 Detailed Description

MFE calculations for single RNA sequences.

## 17.42 ViennaRNA/fold\_compound.h File Reference

The Basic Fold Compound API.

Include dependency graph for fold\_compound.h:

This graph shows which files directly or indirectly include this file:

## Data Structures

- struct [vrna\\_fc\\_s](#)  
*The most basic data structure required by many functions throughout the RNAlib. [More...](#)*

## Macros

- `#define VRNA_STATUS_MFE_PRE` (unsigned char)1  
*Status message indicating that MFE computations are about to begin.*
- `#define VRNA_STATUS_MFE_POST` (unsigned char)2  
*Status message indicating that MFE computations are finished.*
- `#define VRNA_STATUS_PF_PRE` (unsigned char)3  
*Status message indicating that Partition function computations are about to begin.*
- `#define VRNA_STATUS_PF_POST` (unsigned char)4  
*Status message indicating that Partition function computations are finished.*
- `#define VRNA_OPTION_DEFAULT` 0U  
*Option flag to specify default settings/requirements.*
- `#define VRNA_OPTION_MFE` 1U  
*Option flag to specify requirement of Minimum Free Energy (MFE) DP matrices and corresponding set of energy parameters.*
- `#define VRNA_OPTION_PF` 2U  
*Option flag to specify requirement of Partition Function (PF) DP matrices and corresponding set of Boltzmann factors.*
- `#define VRNA_OPTION_HYBRID` 4U  
*Option flag to specify requirement of dimer DP matrices.*
- `#define VRNA_OPTION_EVAL_ONLY` 8U  
*Option flag to specify that neither MFE, nor PF DP matrices are required.*
- `#define VRNA_OPTION_WINDOW` 16U  
*Option flag to specify requirement of DP matrices for local folding approaches.*

## Typedefs

- `typedef struct vrna_fc_s vrna_fold_compound_t`  
*Typename for the fold\_compound data structure `vrna_fc_s`.*
- `typedef void() vrna_callback_free_auxdata(void *data)`  
*Callback to free memory allocated for auxiliary user-provided data.*
- `typedef void() vrna_callback_recursion_status(unsigned char status, void *data)`  
*Callback to perform specific user-defined actions before, or after recursive computations.*

## Enumerations

- `enum vrna_fc_type_e { VRNA_FC_TYPE_SINGLE, VRNA_FC_TYPE_COMPARATIVE }`  
*An enumerator that is used to specify the type of a `vrna_fold_compound_t`.*

## Functions

- `vrna_fold_compound_t * vrna_fold_compound` (const char \*sequence, `vrna_md_t` \*md\_p, unsigned int options)  
*Retrieve a `vrna_fold_compound_t` data structure for single sequences and hybridizing sequences.*
- `vrna_fold_compound_t * vrna_fold_compound_comparative` (const char \*\*sequences, `vrna_md_t` \*md\_p, unsigned int options)  
*Retrieve a `vrna_fold_compound_t` data structure for sequence alignments.*
- `void vrna_fold_compound_free` (`vrna_fold_compound_t` \*fc)  
*Free memory occupied by a `vrna_fold_compound_t`.*
- `void vrna_fold_compound_add_auxdata` (`vrna_fold_compound_t` \*fc, void \*data, `vrna_callback_free_auxdata` \*f)  
*Add auxiliary data to the `vrna_fold_compound_t`.*
- `void vrna_fold_compound_add_callback` (`vrna_fold_compound_t` \*fc, `vrna_callback_recursion_status` \*f)  
*Add a recursion status callback to the `vrna_fold_compound_t`.*

### 17.42.1 Detailed Description

The Basic Fold Compound API.

## 17.43 ViennaRNA/fold\_vars.h File Reference

Here all all declarations of the global variables used throughout RNAlib.

Include dependency graph for fold\_vars.h:

This graph shows which files directly or indirectly include this file:

### Variables

- int `fold_constrained`  
*Global switch to activate/deactivate folding with structure constraints.*
- int `csv`  
*generate comma seperated output*
- char \* `RibosumFile`
- int `james_rule`
- int `logML`
- int `cut_point`  
*Marks the position (starting from 1) of the first nucleotide of the second molecule within the concatenated sequence.*
- `bondT * base_pair`  
*Contains a list of base pairs after a call to `fold()`.*
- `FLT_OR_DBL * pr`  
*A pointer to the base pair probability matrix.*
- int \* `iindx`  
*index array to move through pr.*

### 17.43.1 Detailed Description

Here all all declarations of the global variables used throughout RNAlib.

### 17.43.2 Variable Documentation

#### 17.43.2.1 RibosumFile

```
char* RibosumFile
```

warning this variable will vanish in the future ribosums will be compiled in instead

#### 17.43.2.2 james\_rule

```
int james_rule
```

interior loops of size 2 get energy 0.8Kcal and no mismatches, default 1

#### 17.43.2.3 logML

```
int logML
```

use logarithmic multiloop energy function

#### 17.43.2.4 cut\_point

```
int cut_point
```

Marks the position (starting from 1) of the first nucleotide of the second molecule within the concatenated sequence.

To evaluate the energy of a duplex structure (a structure formed by two strands), concatenate the two sequences and set it to the first base of the second strand in the concatenated sequence. The default value of -1 stands for single molecule folding. The cut\_point variable is also used by [vrna\\_file\\_PS\\_rnaplot\(\)](#) and [PS\\_dot\\_plot\(\)](#) to mark the chain break in postscript plots.

#### 17.43.2.5 base\_pair

```
bondT* base_pair
```

Contains a list of base pairs after a call to [fold\(\)](#).

base\_pair[0].i contains the total number of pairs.

**Deprecated** Do not use this variable anymore!

#### 17.43.2.6 pr

```
FLT_OR_DBL* pr
```

A pointer to the base pair probability matrix.

**Deprecated** Do not use this variable anymore!



### 17.43.2.7 iindx

```
int* iindx
```

index array to move through pr.

The probability for base i and j to form a pair is in pr[iindx[i]-j].

**Deprecated** Do not use this variable anymore!

## 17.44 ViennaRNA/gquad.h File Reference

G-quadruplexes.

Include dependency graph for gquad.h:

### Functions

- int [get\\_gquad\\_matrix](#) (short \*S, [vrna\\_param\\_t](#) \*P)  
*Get a triangular matrix prefilled with minimum free energy contributions of G-quadruplexes.*
- int [parse\\_gquad](#) (const char \*struc, int \*L, int l[3])
- PRIVATE int [backtrack\\_GQuad\\_IntLoop](#) (int c, int i, int j, int type, short \*S, int \*ggg, int \*index, int \*p, int \*q, [vrna\\_param\\_t](#) \*P)
- PRIVATE int [backtrack\\_GQuad\\_IntLoop\\_L](#) (int c, int i, int j, int type, short \*S, int \*\*ggg, int maxdist, int \*p, int \*q, [vrna\\_param\\_t](#) \*P)

### 17.44.1 Detailed Description

G-quadruplexes.

## 17.45 ViennaRNA/grammar.h File Reference

Implementations for the RNA folding grammar.

Include dependency graph for grammar.h:

This graph shows which files directly or indirectly include this file:

### Data Structures

- struct [vrna\\_gr\\_aux\\_s](#)

### 17.45.1 Detailed Description

Implementations for the RNA folding grammar.

## 17.46 ViennaRNA/hairpin\_loops.h File Reference

Use [ViennaRNA/loops/hairpin.h](#) instead.

Include dependency graph for hairpin\_loops.h:

### 17.46.1 Detailed Description

Use [ViennaRNA/loops/hairpin.h](#) instead.

**Deprecated** Use [ViennaRNA/loops/hairpin.h](#) instead

## 17.47 ViennaRNA/interior\_loops.h File Reference

Use [ViennaRNA/loops/internal.h](#) instead.

Include dependency graph for interior\_loops.h:

### 17.47.1 Detailed Description

Use [ViennaRNA/loops/internal.h](#) instead.

**Deprecated** Use [ViennaRNA/loops/internal.h](#) instead

## 17.48 ViennaRNA/inverse.h File Reference

Inverse folding routines.

### Functions

- float [inverse\\_fold](#) (char \*start, const char \*target)  
*Find sequences with predefined structure.*
- float [inverse\\_pf\\_fold](#) (char \*start, const char \*target)  
*Find sequence that maximizes probability of a predefined structure.*

## Variables

- char \* [symbolset](#)  
*This global variable points to the allowed bases, initially "AUGC". It can be used to design sequences from reduced alphabets.*
- float [final\\_cost](#)
- int [give\\_up](#)
- int [inv\\_verbose](#)

### 17.48.1 Detailed Description

Inverse folding routines.

## 17.49 ViennaRNA/Lfold.h File Reference

Functions for locally optimal MFE structure prediction.

Include dependency graph for Lfold.h:

## Functions

- float [Lfold](#) (const char \*string, const char \*structure, int maxdist)  
*The local analog to [fold\(\)](#).*
- float [Lfoldz](#) (const char \*string, const char \*structure, int maxdist, int zsc, double min\_z)

### 17.49.1 Detailed Description

Functions for locally optimal MFE structure prediction.

## 17.50 ViennaRNA/loop\_energies.h File Reference

Use [ViennaRNA/loops/all.h](#) instead.

Include dependency graph for loop\_energies.h:

### 17.50.1 Detailed Description

Use [ViennaRNA/loops/all.h](#) instead.

**Deprecated** Use [ViennaRNA/loops/all.h](#) instead

## 17.51 ViennaRNA/loops/all.h File Reference

Energy evaluation for MFE and partition function calculations.

Include dependency graph for all.h:

This graph shows which files directly or indirectly include this file:

### 17.51.1 Detailed Description

Energy evaluation for MFE and partition function calculations.

,

This file contains functions for the calculation of the free energy  $\Delta G$  of a hairpin- [ [E\\_Hairpin\(\)](#) ] or interior-loop [ [E\\_IntLoop\(\)](#) ].

The unit of the free energy returned is  $10^{-2} * \text{kcal/mol}$

In case of computing the partition function, this file also supplies functions which return the Boltzmann weights  $e^{-\Delta G/kT}$  for a hairpin- [ [exp\\_E\\_Hairpin\(\)](#) ] or interior-loop [ [exp\\_E\\_IntLoop\(\)](#) ].

## 17.52 ViennaRNA/loops/external.h File Reference

Energy evaluation of exterior loops for MFE and partition function calculations.

Include dependency graph for external.h:

This graph shows which files directly or indirectly include this file:

### Functions

- int [E\\_Stem](#) (int type, int si1, int sj1, int extLoop, [vrna\\_param\\_t](#) \*P)  
*Compute the energy contribution of a stem branching off a loop-region.*
- [FLT\\_OR\\_DBL exp\\_E\\_ExtLoop](#) (int type, int si1, int sj1, [vrna\\_exp\\_param\\_t](#) \*P)
- [FLT\\_OR\\_DBL exp\\_E\\_Stem](#) (int type, int si1, int sj1, int extLoop, [vrna\\_exp\\_param\\_t](#) \*P)

### Basic free energy interface

- int [vrna\\_E\\_ext\\_stem](#) (unsigned int type, int n5d, int n3d, [vrna\\_param\\_t](#) \*p)  
*Evaluate a stem branching off the exterior loop.*
- int [vrna\\_E\\_ext\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j)  
*Evaluate the free energy of a base pair in the exterior loop.*
- int [vrna\\_E\\_ext\\_loop\\_5](#) ([vrna\\_fold\\_compound\\_t](#) \*fc)
- int [vrna\\_E\\_ext\\_loop\\_3](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i)

**Boltzmann weight (partition function) interface**

- typedef struct vrna\_mx\_pf\_aux\_el\_s \* [vrna\\_mx\\_pf\\_aux\\_el\\_t](#)  
*Auxiliary helper arrays for fast exterior loop computations.*
- [FLT\\_OR\\_DBL vrna\\_exp\\_E\\_ext\\_stem](#) (unsigned int type, int n5d, int n3d, [vrna\\_exp\\_param\\_t](#) \*p)  
*Evaluate a stem branching off the exterior loop (Boltzmann factor version)*
- struct vrna\_mx\_pf\_aux\_el\_s \* [vrna\\_exp\\_E\\_ext\\_fast\\_init](#) ([vrna\\_fold\\_compound\\_t](#) \*fc)
- void [vrna\\_exp\\_E\\_ext\\_fast\\_rotate](#) (struct vrna\_mx\_pf\_aux\_el\_s \*aux\_mx)
- void [vrna\\_exp\\_E\\_ext\\_fast\\_free](#) (struct vrna\_mx\_pf\_aux\_el\_s \*aux\_mx)
- [FLT\\_OR\\_DBL vrna\\_exp\\_E\\_ext\\_fast](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j, struct vrna\_mx\_pf\_aux\_el\_s \*aux\_mx)
- void [vrna\\_exp\\_E\\_ext\\_fast\\_update](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int j, struct vrna\_mx\_pf\_aux\_el\_s \*aux\_mx)

**17.52.1 Detailed Description**

Energy evaluation of exterior loops for MFE and partition function calculations.

, ,

**17.53 ViennaRNA/loops/hairpin.h File Reference**

Energy evaluation of hairpin loops for MFE and partition function calculations.

Include dependency graph for hairpin.h:

This graph shows which files directly or indirectly include this file:

**Functions**

- int [vrna\\_BT\\_hp\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j, int en, [vrna\\_bp\\_stack\\_t](#) \*bp\_stack, int \*stack\_count)  
*Backtrack a hairpin loop closed by (i, j).*

**Basic free energy interface**

- int [vrna\\_E\\_hp\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j)  
*Evaluate the free energy of a hairpin loop and consider hard constraints if they apply.*
- int [vrna\\_E\\_ext\\_hp\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j)  
*Evaluate the free energy of an exterior hairpin loop and consider possible hard constraints.*
- int [vrna\\_eval\\_ext\\_hp\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j)  
*Evaluate free energy of an exterior hairpin loop.*
- int [vrna\\_eval\\_hp\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j)  
*Evaluate free energy of a hairpin loop.*
- PRIVATE int [E\\_Hairpin](#) (int size, int type, int si1, int sj1, const char \*string, [vrna\\_param\\_t](#) \*P)  
*Compute the Energy of a hairpin-loop.*

**Boltzmann weight (partition function) interface**

- PRIVATE [FLT\\_OR\\_DBL exp\\_E\\_Hairpin](#) (int u, int type, short si1, short sj1, const char \*string, [vrna\\_exp\\_param\\_t](#) \*P)  
*Compute Boltzmann weight  $e^{-\Delta G/kT}$  of a hairpin loop.*
- [FLT\\_OR\\_DBL vrna\\_exp\\_E\\_hp\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j)  
*High-Level function for hairpin loop energy evaluation (partition function variant)*

### 17.53.1 Detailed Description

Energy evaluation of hairpin loops for MFE and partition function calculations.

, ,

## 17.54 ViennaRNA/loops/internal.h File Reference

Energy evaluation of interior loops for MFE and partition function calculations.

Include dependency graph for internal.h:

This graph shows which files directly or indirectly include this file:

### Functions

- int [vrna\\_BT\\_stack](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int \*i, int \*j, int \*en, [vrna\\_bp\\_stack\\_t](#) \*bp\_stack, int \*stack↵\_count)  
*Backtrack a stacked pair closed by  $(i, j)$ .*
- int [vrna\\_BT\\_int\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int \*i, int \*j, int en, [vrna\\_bp\\_stack\\_t](#) \*bp\_stack, int \*stack↵\_count)  
*Backtrack an interior loop closed by  $(i, j)$ .*
- PRIVATE int [E\\_IntLoop](#) (int n1, int n2, int type, int type\_2, int si1, int sj1, int sp1, int sq1, [vrna\\_param\\_t](#) \*P)
- PRIVATE [FLT\\_OR\\_DBL exp\\_E\\_IntLoop](#) (int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short sq1, [vrna\\_exp\\_param\\_t](#) \*P)

### Basic free energy interface

- int [vrna\\_E\\_int\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j)
- int [vrna\\_eval\\_int\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j, int k, int l)  
*Evaluate the free energy contribution of an interior loop with delimiting base pairs  $(i, j)$  and  $(k, l)$ .*
- int [vrna\\_E\\_ext\\_int\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j, int \*ip, int \*iq)
- int [vrna\\_E\\_stack](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j)

### Boltzmann weight (partition function) interface

- [FLT\\_OR\\_DBL vrna\\_exp\\_E\\_int\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j)
- [FLT\\_OR\\_DBL vrna\\_exp\\_E\\_interior\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j, int k, int l)

### 17.54.1 Detailed Description

Energy evaluation of interior loops for MFE and partition function calculations.

, ,

## 17.55 ViennaRNA/loops/multibranch.h File Reference

Energy evaluation of multibranch loops for MFE and partition function calculations.

Include dependency graph for multibranch.h:

This graph shows which files directly or indirectly include this file:

### Functions

- int [vrna\\_BT\\_mb\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int \*i, int \*j, int \*k, int en, int \*component1, int \*component2)  
*Backtrack the decomposition of a multi branch loop closed by (i, j).*

### Basic free energy interface

- int [vrna\\_E\\_mb\\_loop\\_stack](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j)  
*Evaluate energy of a multi branch helices stacking onto closing pair (i,j)*
- int [vrna\\_E\\_mb\\_loop\\_fast](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j, int \*dmli1, int \*dmli2)
- int [E\\_ml\\_rightmost\\_stem](#) (int i, int j, [vrna\\_fold\\_compound\\_t](#) \*fc)
- int [vrna\\_E\\_ml\\_stems\\_fast](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j, int \*fmi, int \*dmli)

### Boltzmann weight (partition function) interface

- typedef struct [vrna\\_mx\\_pf\\_aux\\_ml\\_s](#) \* [vrna\\_mx\\_pf\\_aux\\_ml\\_t](#)  
*Auxiliary helper arrays for fast exterior loop computations.*
- [FLT\\_OR\\_DBL](#) [vrna\\_exp\\_E\\_mb\\_loop\\_fast](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j, [vrna\\_mx\\_pf\\_aux\\_ml\\_t](#) aux\_mx)
- [vrna\\_mx\\_pf\\_aux\\_ml\\_t](#) [vrna\\_exp\\_E\\_ml\\_fast\\_init](#) ([vrna\\_fold\\_compound\\_t](#) \*fc)
- void [vrna\\_exp\\_E\\_ml\\_fast\\_rotate](#) ([vrna\\_mx\\_pf\\_aux\\_ml\\_t](#) aux\_mx)
- void [vrna\\_exp\\_E\\_ml\\_fast\\_free](#) ([vrna\\_mx\\_pf\\_aux\\_ml\\_t](#) aux\_mx)
- const [FLT\\_OR\\_DBL](#) \* [vrna\\_exp\\_E\\_ml\\_fast\\_qqm](#) (struct [vrna\\_mx\\_pf\\_aux\\_ml\\_s](#) \*aux\_mx)
- const [FLT\\_OR\\_DBL](#) \* [vrna\\_exp\\_E\\_ml\\_fast\\_qqm1](#) (struct [vrna\\_mx\\_pf\\_aux\\_ml\\_s](#) \*aux\_mx)
- [FLT\\_OR\\_DBL](#) [vrna\\_exp\\_E\\_ml\\_fast](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, int i, int j, [vrna\\_mx\\_pf\\_aux\\_ml\\_t](#) aux\_mx)

### 17.55.1 Detailed Description

Energy evaluation of multibranch loops for MFE and partition function calculations.

’ ’

## 17.56 ViennaRNA/LPfold.h File Reference

Partition function and equilibrium probability implementation for the sliding window algorithm.

Include dependency graph for LPfold.h:

### Functions

- void [update\\_pf\\_paramsLP](#) (int length)
- [vrna\\_ep\\_t](#) \* [pfl\\_fold](#) (char \*sequence, int winSize, int pairSize, float cutoffb, double \*\*pU, [vrna\\_ep\\_t](#) \*\*dpp2, FILE \*pUfp, FILE \*spup)  
*Compute partition functions for locally stable secondary structures.*
- [vrna\\_ep\\_t](#) \* [pfl\\_fold\\_par](#) (char \*sequence, int winSize, int pairSize, float cutoffb, double \*\*pU, [vrna\\_ep\\_t](#) \*\*dpp2, FILE \*pUfp, FILE \*spup, [vrna\\_exp\\_param\\_t](#) \*parameters)  
*Compute partition functions for locally stable secondary structures.*
- void [putoutpU\\_prob](#) (double \*\*pU, int length, int ulength, FILE \*fp, int energies)  
*Writes the unpaired probabilities (pU) or opening energies into a file.*
- void [putoutpU\\_prob\\_bin](#) (double \*\*pU, int length, int ulength, FILE \*fp, int energies)  
*Writes the unpaired probabilities (pU) or opening energies into a binary file.*
- void [init\\_pf\\_foldLP](#) (int length)

### 17.56.1 Detailed Description

Partition function and equilibrium probability implementation for the sliding window algorithm.

This file contains the implementation for sliding window partition function and equilibrium probabilities. It also provides the unpaired probability implementation from Bernhart et al. 2011 [4]

### 17.56.2 Function Documentation

#### 17.56.2.1 [init\\_pf\\_foldLP\(\)](#)

```
void init_pf_foldLP (
    int length )
```

Dunno if this function was ever used by external programs linking to RNAlib, but it was declared PUBLIC before. Anyway, never use this function as it will be removed soon and does nothing at all

## 17.57 ViennaRNA/MEA.h File Reference

Computes a MEA (maximum expected accuracy) structure.

Include dependency graph for MEA.h:



## Functions

- float [MEA](#) ([plist](#) \*p, char \*structure, double gamma)  
*Computes a MEA (maximum expected accuracy) structure.*

### 17.57.1 Detailed Description

Computes a MEA (maximum expected accuracy) structure.

## 17.58 ViennaRNA/mfe.h File Reference

Compute Minimum Free energy (MFE) and backtrace corresponding secondary structures from RNA sequence data.

Include dependency graph for mfe.h:

This graph shows which files directly or indirectly include this file:

## Functions

### Basic global MFE prediction interface

- float [vrna\\_mfe](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, char \*structure)  
*Compute minimum free energy and an appropriate secondary structure of an RNA sequence, or RNA sequence alignment.*
- float [vrna\\_mfe\\_dimer](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, char \*structure)  
*Compute the minimum free energy of two interacting RNA molecules.*

### Simplified global MFE prediction using sequence(s) or multiple sequence alignment(s)

- float [vrna\\_fold](#) (const char \*sequence, char \*structure)  
*Compute Minimum Free Energy (MFE), and a corresponding secondary structure for an RNA sequence.*
- float [vrna\\_circfold](#) (const char \*sequence, char \*structure)  
*Compute Minimum Free Energy (MFE), and a corresponding secondary structure for a circular RNA sequence.*
- float [vrna\\_alifold](#) (const char \*\*sequences, char \*structure)  
*Compute Minimum Free Energy (MFE), and a corresponding consensus secondary structure for an RNA sequence alignment using a comparative method.*
- float [vrna\\_circalifold](#) (const char \*\*sequences, char \*structure)  
*Compute Minimum Free Energy (MFE), and a corresponding consensus secondary structure for a sequence alignment of circular RNAs using a comparative method.*
- float [vrna\\_cofold](#) (const char \*sequence, char \*structure)  
*Compute Minimum Free Energy (MFE), and a corresponding secondary structure for two dimerized RNA sequences.*

### 17.58.1 Detailed Description

Compute Minimum Free energy (MFE) and backtrace corresponding secondary structures from RNA sequence data.

, This file includes (almost) all function declarations within the RNALib that are related to MFE folding...

## 17.59 ViennaRNA/mfe\_window.h File Reference

Compute local Minimum Free Energy (MFE) using a sliding window approach and backtrace corresponding secondary structures.

Include dependency graph for mfe\_window.h:

This graph shows which files directly or indirectly include this file:

### Typedefs

- typedef void() [vrna\\_mfe\\_window\\_callback](#)(int start, int end, const char \*structure, float en, void \*data)

*The default callback for sliding window MFE structure predictions.*

### Functions

#### Basic local (sliding window) MFE prediction interface

- float [vrna\\_mfe\\_window](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, FILE \*file)  
*Local MFE prediction using a sliding window approach.*
- float [vrna\\_mfe\\_window\\_cb](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_mfe\\_window\\_callback](#) \*cb, void \*data)
- float [vrna\\_mfe\\_window\\_zscore](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, double min\_z, FILE \*file)  
*Local MFE prediction using a sliding window approach (with z-score cut-off)*
- float [vrna\\_mfe\\_window\\_zscore\\_cb](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, double min\_z, [vrna\\_mfe\\_window\\_zscore\\_callback](#) \*cb, void \*data)

#### Simplified local MFE prediction using sequence(s) or multiple sequence alignment(s)

- float [vrna\\_Lfold](#) (const char \*string, int window\_size, FILE \*file)  
*Local MFE prediction using a sliding window approach (simplified interface)*
- float [vrna\\_Lfold\\_cb](#) (const char \*string, int window\_size, [vrna\\_mfe\\_window\\_callback](#) \*cb, void \*data)
- float [vrna\\_Lfoldz](#) (const char \*string, int window\_size, double min\_z, FILE \*file)  
*Local MFE prediction using a sliding window approach with z-score cut-off (simplified interface)*
- float [vrna\\_Lfoldz\\_cb](#) (const char \*string, int window\_size, double min\_z, [vrna\\_mfe\\_window\\_zscore\\_callback](#) \*cb, void \*data)
- float [vrna\\_alifold](#) (const char \*\*alignment, int maxdist, FILE \*fp)
- float [vrna\\_alifold\\_cb](#) (const char \*\*alignment, int maxdist, [vrna\\_mfe\\_window\\_callback](#) \*cb, void \*data)

### 17.59.1 Detailed Description

Compute local Minimum Free Energy (MFE) using a sliding window approach and backtrace corresponding secondary structures.

, This file includes the interface to all functions related to predicting locally stable secondary structures.

## 17.60 ViennaRNA/mm.h File Reference

Several Maximum Matching implementations.

### Functions

- int [vrna\\_maximum\\_matching](#) ([vrna\\_fold\\_compound\\_t](#) \*fc)
- int [vrna\\_maximum\\_matching\\_simple](#) (const char \*sequence)

### 17.60.1 Detailed Description

Several Maximum Matching implementations.

This file contains the declarations for several maximum matching implementations

### 17.60.2 Function Documentation

#### 17.60.2.1 vrna\_maximum\_matching()

```
int vrna_maximum_matching (  
    vrna\_fold\_compound\_t * fc )
```

**SWIG Wrapper Notes** This function is attached as method **maximum\_matching()** to objects of type `fold_compound` (i.e. [vrna\\_fold\\_compound\\_t](#)).

#### 17.60.2.2 vrna\_maximum\_matching\_simple()

```
int vrna_maximum_matching_simple (  
    const char * sequence )
```

**SWIG Wrapper Notes** This function is available as global function **maximum\_matching()**.

## 17.61 ViennaRNA/model.h File Reference

The model details data structure and its corresponding modifiers.

This graph shows which files directly or indirectly include this file:

### Data Structures

- struct [vrna\\_md\\_s](#)

*The data structure that contains the complete model details used throughout the calculations. [More...](#)*

### Macros

- `#define VRNA_MODEL_DEFAULT_TEMPERATURE 37.0`  
*Default temperature for structure prediction and free energy evaluation in °C*
- `#define VRNA_MODEL_DEFAULT_PF_SCALE -1`  
*Default scaling factor for partition function computations.*
- `#define VRNA_MODEL_DEFAULT_BETA_SCALE 1.`  
*Default scaling factor for absolute thermodynamic temperature in Boltzmann factors.*
- `#define VRNA_MODEL_DEFAULT_DANGLES 2`  
*Default dangling end model.*
- `#define VRNA_MODEL_DEFAULT_SPECIAL_HP 1`  
*Default model behavior for lookup of special tri-, tetra-, and hexa-loops.*
- `#define VRNA_MODEL_DEFAULT_NO_LP 0`  
*Default model behavior for so-called 'lonely pairs'.*
- `#define VRNA_MODEL_DEFAULT_NO_GU 0`  
*Default model behavior for G-U base pairs.*
- `#define VRNA_MODEL_DEFAULT_NO_GU_CLOSURE 0`  
*Default model behavior for G-U base pairs closing a loop.*
- `#define VRNA_MODEL_DEFAULT_CIRC 0`  
*Default model behavior to treat a molecule as a circular RNA (DNA)*
- `#define VRNA_MODEL_DEFAULT_GQUAD 0`  
*Default model behavior regarding the treatment of G-Quadruplexes.*
- `#define VRNA_MODEL_DEFAULT_UNIQ_ML 0`  
*Default behavior of the model regarding unique multi-branch loop decomposition.*
- `#define VRNA_MODEL_DEFAULT_ENERGY_SET 0`  
*Default model behavior on which energy set to use.*
- `#define VRNA_MODEL_DEFAULT_BACKTRACK 1`  
*Default model behavior with regards to backtracking of structures.*
- `#define VRNA_MODEL_DEFAULT_BACKTRACK_TYPE 'F'`  
*Default model behavior on what type of backtracking to perform.*
- `#define VRNA_MODEL_DEFAULT_COMPUTE_BPP 1`  
*Default model behavior with regards to computing base pair probabilities.*
- `#define VRNA_MODEL_DEFAULT_MAX_BP_SPAN -1`  
*Default model behavior for the allowed maximum base pair span.*
- `#define VRNA_MODEL_DEFAULT_WINDOW_SIZE -1`  
*Default model behavior for the sliding window approach.*

- `#define VRNA_MODEL_DEFAULT_LOG_ML 0`  
*Default model behavior on how to evaluate the energy contribution of multi-branch loops.*
- `#define VRNA_MODEL_DEFAULT_ALI_OLD_EN 0`  
*Default model behavior for consensus structure energy evaluation.*
- `#define VRNA_MODEL_DEFAULT_ALI_RIBO 0`  
*Default model behavior for consensus structure co-variance contribution assessment.*
- `#define VRNA_MODEL_DEFAULT_ALI_CV_FACT 1.`  
*Default model behavior for weighting the co-variance score in consensus structure prediction.*
- `#define VRNA_MODEL_DEFAULT_ALI_NC_FACT 1.`  
*Default model behavior for weighting the nucleotide conservation? in consensus structure prediction.*
- `#define MAXALPHA 20`  
*Maximal length of alphabet.*

## Typedefs

- `typedef struct vrna_md_s vrna_md_t`  
*Typename for the model details data structure [vrna\\_md\\_s](#).*

## Functions

- `void vrna_md_set_default (vrna_md_t *md)`  
*Apply default model details to a provided [vrna\\_md\\_t](#) data structure.*
- `void vrna_md_update (vrna_md_t *md)`  
*Update the model details data structure.*
- `vrna_md_t * vrna_md_copy (vrna_md_t *md_to, const vrna_md_t *md_from)`  
*Copy/Clone a [vrna\\_md\\_t](#) model.*
- `char * vrna_md_option_string (vrna_md_t *md)`  
*Get a corresponding commandline parameter string of the options in a [vrna\\_md\\_t](#).*
- `void vrna_md_defaults_reset (vrna_md_t *md_p)`  
*Reset the global default model details to a specific set of parameters, or their initial values.*
- `void vrna_md_defaults_temperature (double T)`  
*Set default temperature for energy evaluation of loops.*
- `double vrna_md_defaults_temperature_get (void)`  
*Get default temperature for energy evaluation of loops.*
- `void vrna_md_defaults_betaScale (double b)`  
*Set default scaling factor of thermodynamic temperature in Boltzmann factors.*
- `double vrna_md_defaults_betaScale_get (void)`  
*Get default scaling factor of thermodynamic temperature in Boltzmann factors.*
- `void vrna_md_defaults_dangles (int d)`  
*Set default dangle model for structure prediction.*
- `int vrna_md_defaults_dangles_get (void)`  
*Get default dangle model for structure prediction.*
- `void vrna_md_defaults_special_hp (int flag)`  
*Set default behavior for lookup of tabulated free energies for special hairpin loops, such as Tri-, Tetra-, or Hexa-loops.*
- `int vrna_md_defaults_special_hp_get (void)`  
*Get default behavior for lookup of tabulated free energies for special hairpin loops, such as Tri-, Tetra-, or Hexa-loops.*
- `void vrna_md_defaults_noLP (int flag)`  
*Set default behavior for prediction of canonical secondary structures.*
- `int vrna_md_defaults_noLP_get (void)`

- Get default behavior for prediction of canonical secondary structures.*

  - void [vrna\\_md\\_defaults\\_noGU](#) (int flag)
- Set default behavior for treatment of G-U wobble pairs.*

  - int [vrna\\_md\\_defaults\\_noGU\\_get](#) (void)
- Get default behavior for treatment of G-U wobble pairs.*

  - void [vrna\\_md\\_defaults\\_noGUclosure](#) (int flag)
- Set default behavior for G-U pairs as closing pair for loops.*

  - int [vrna\\_md\\_defaults\\_noGUclosure\\_get](#) (void)
- Get default behavior for G-U pairs as closing pair for loops.*

  - void [vrna\\_md\\_defaults\\_logML](#) (int flag)
- Set default behavior recomputing free energies of multi-branch loops using a logarithmic model.*

  - int [vrna\\_md\\_defaults\\_logML\\_get](#) (void)
- Get default behavior recomputing free energies of multi-branch loops using a logarithmic model.*

  - void [vrna\\_md\\_defaults\\_circ](#) (int flag)
- Set default behavior whether input sequences are circularized.*

  - int [vrna\\_md\\_defaults\\_circ\\_get](#) (void)
- Get default behavior whether input sequences are circularized.*

  - void [vrna\\_md\\_defaults\\_gquad](#) (int flag)
- Set default behavior for treatment of G-Quadruplexes.*

  - int [vrna\\_md\\_defaults\\_gquad\\_get](#) (void)
- Get default behavior for treatment of G-Quadruplexes.*

  - void [vrna\\_md\\_defaults\\_uniq\\_ML](#) (int flag)
- Set default behavior for creating additional matrix for unique multi-branch loop prediction.*

  - int [vrna\\_md\\_defaults\\_uniq\\_ML\\_get](#) (void)
- Get default behavior for creating additional matrix for unique multi-branch loop prediction.*

  - void [vrna\\_md\\_defaults\\_energy\\_set](#) (int e)
- Set default energy set.*

  - int [vrna\\_md\\_defaults\\_energy\\_set\\_get](#) (void)
- Get default energy set.*

  - void [vrna\\_md\\_defaults\\_backtrack](#) (int flag)
- Set default behavior for whether to backtrack secondary structures.*

  - int [vrna\\_md\\_defaults\\_backtrack\\_get](#) (void)
- Get default behavior for whether to backtrack secondary structures.*

  - void [vrna\\_md\\_defaults\\_backtrack\\_type](#) (char t)
- Set default backtrack type, i.e. which DP matrix is used.*

  - char [vrna\\_md\\_defaults\\_backtrack\\_type\\_get](#) (void)
- Get default backtrack type, i.e. which DP matrix is used.*

  - void [vrna\\_md\\_defaults\\_compute\\_bpp](#) (int flag)
- Set the default behavior for whether to compute base pair probabilities after partition function computation.*

  - int [vrna\\_md\\_defaults\\_compute\\_bpp\\_get](#) (void)
- Get the default behavior for whether to compute base pair probabilities after partition function computation.*

  - void [vrna\\_md\\_defaults\\_max\\_bp\\_span](#) (int span)
- Set default maximal base pair span.*

  - int [vrna\\_md\\_defaults\\_max\\_bp\\_span\\_get](#) (void)
- Get default maximal base pair span.*

  - void [vrna\\_md\\_defaults\\_min\\_loop\\_size](#) (int size)
- Set default minimal loop size.*

  - int [vrna\\_md\\_defaults\\_min\\_loop\\_size\\_get](#) (void)
- Get default minimal loop size.*

  - void [vrna\\_md\\_defaults\\_window\\_size](#) (int size)
- Set default window size for sliding window structure prediction approaches.*

- int [vrna\\_md\\_defaults\\_window\\_size\\_get](#) (void)  
*Get default window size for sliding window structure prediction approaches.*
- void [vrna\\_md\\_defaults\\_oldAliEn](#) (int flag)  
*Set default behavior for whether to use old energy model for comparative structure prediction.*
- int [vrna\\_md\\_defaults\\_oldAliEn\\_get](#) (void)  
*Get default behavior for whether to use old energy model for comparative structure prediction.*
- void [vrna\\_md\\_defaults\\_ribo](#) (int flag)  
*Set default behavior for whether to use Ribosum Scoring in comparative structure prediction.*
- int [vrna\\_md\\_defaults\\_ribo\\_get](#) (void)  
*Get default behavior for whether to use Ribosum Scoring in comparative structure prediction.*
- void [vrna\\_md\\_defaults\\_cv\\_fact](#) (double factor)  
*Set the default co-variance scaling factor used in comparative structure prediction.*
- double [vrna\\_md\\_defaults\\_cv\\_fact\\_get](#) (void)  
*Get the default co-variance scaling factor used in comparative structure prediction.*
- void [vrna\\_md\\_defaults\\_nc\\_fact](#) (double factor)
- double [vrna\\_md\\_defaults\\_nc\\_fact\\_get](#) (void)
- void [vrna\\_md\\_defaults\\_sfact](#) (double factor)  
*Set the default scaling factor used to avoid under-/overflows in partition function computation.*
- double [vrna\\_md\\_defaults\\_sfact\\_get](#) (void)  
*Get the default scaling factor used to avoid under-/overflows in partition function computation.*
- void [set\\_model\\_details](#) ([vrna\\_md\\_t](#) \*md)  
*Set default model details.*

## Variables

- double [temperature](#)  
*Rescale energy parameters to a temperature in degC.*
- double [pf\\_scale](#)  
*A scaling factor used by [pf\\_fold\(\)](#) to avoid overflows.*
- int [dangles](#)  
*Switch the energy model for dangling end contributions (0, 1, 2, 3)*
- int [tetra\\_loop](#)  
*Include special stabilizing energies for some tri-, tetra- and hexa-loops;.*
- int [noLonelyPairs](#)  
*Global switch to avoid/allow helices of length 1.*
- int [noGU](#)  
*Global switch to forbid/allow GU base pairs at all.*
- int [no\\_closingGU](#)  
*GU allowed only inside stacks if set to 1.*
- int [circ](#)  
*backward compatibility variable.. this does not effect anything*
- int [gquad](#)  
*Allow G-quadruplex formation.*
- int [uniq\\_ML](#)  
*do ML decomposition uniquely (for subopt)*
- int [energy\\_set](#)  
*0 = BP; 1=any with GC; 2=any with AU-parameter*
- int [do\\_backtrack](#)  
*do backtracking, i.e. compute secondary structures or base pair probabilities*
- char [backtrack\\_type](#)

- A backtrack array marker for [inverse\\_fold\(\)](#)
- char \* [nonstandards](#)  
*contains allowed non standard base pairs*
- int [max\\_bp\\_span](#)  
*Maximum allowed base pair span.*
- int [oldAliEn](#)  
*use old alifold energies (with gaps)*
- int [ribo](#)  
*use ribosum matrices*
- int [logML](#)  
*if nonzero use logarithmic ML energy in energy\_of\_struct*

### 17.61.1 Detailed Description

The model details data structure and its corresponding modifiers.

## 17.62 ViennaRNA/multibranch\_loops.h File Reference

Use [ViennaRNA/loops/multibranch.h](#) instead.

Include dependency graph for multibranch\_loops.h:

### 17.62.1 Detailed Description

Use [ViennaRNA/loops/multibranch.h](#) instead.

**Deprecated** Use [ViennaRNA/loops/multibranch.h](#) instead

## 17.63 ViennaRNA/naview.h File Reference

Use [ViennaRNA/plotting/naview.h](#) instead.

Include dependency graph for naview.h:

### 17.63.1 Detailed Description

Use [ViennaRNA/plotting/naview.h](#) instead.

**Deprecated** Use [ViennaRNA/plotting/naview.h](#) instead



## 17.64 ViennaRNA/plotting/naview.h File Reference

This graph shows which files directly or indirectly include this file:

## 17.65 ViennaRNA/neighbor.h File Reference

Methods to compute the neighbors of an RNA secondary structure.

Include dependency graph for neighbor.h:

This graph shows which files directly or indirectly include this file:

### Data Structures

- struct [vrna\\_move\\_s](#)  
An atomic representation of the transition / move from one structure to its neighbor. [More...](#)

### Macros

- #define [VRNA\\_MOVESET\\_INSERTION](#) 4  
Option flag indicating insertion move.
- #define [VRNA\\_MOVESET\\_DELETION](#) 8  
Option flag indicating deletion move.
- #define [VRNA\\_MOVESET\\_SHIFT](#) 16  
Option flag indicating shift move.
- #define [VRNA\\_MOVESET\\_NO\\_LP](#) 32  
Option flag indicating moves without lonely base pairs.
- #define [VRNA\\_MOVESET\\_DEFAULT](#) ([VRNA\\_MOVESET\\_INSERTION](#) | [VRNA\\_MOVESET\\_DELETION](#))  
Option flag indicating default move set, i.e. insertions/deletion of a base pair.

### Functions

- void [vrna\\_move\\_list\\_free](#) ([vrna\\_move\\_t](#) \*moves)
- void [vrna\\_move\\_apply](#) (short \*pt, const [vrna\\_move\\_t](#) \*m)  
Apply a particular move / transition to a secondary structure, i.e. transform a structure.
- void [vrna\\_loopidx\\_update](#) (int \*loopidx, const short \*pt, int length, const [vrna\\_move\\_t](#) \*m)  
Alters the loopIndices array that was constructed with [vrna\\_loopidx\\_from\\_ptable\(\)](#).
- [vrna\\_move\\_t](#) \* [vrna\\_neighbors](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const short \*pt, unsigned int options)  
Generate neighbors of a secondary structure.
- [vrna\\_move\\_t](#) \* [vrna\\_neighbors\\_successive](#) (const [vrna\\_fold\\_compound\\_t](#) \*vc, const [vrna\\_move\\_t](#) \*curr↔  
\_move, const short \*prev\_pt, const [vrna\\_move\\_t](#) \*prev\_neighbors, int size\_prev\_neighbors, int \*size\_↔  
neighbors, unsigned int options)  
Generate neighbors of a secondary structure (the fast way)

### 17.65.1 Detailed Description

Methods to compute the neighbors of an RNA secondary structure.

## 17.66 ViennaRNA/params.h File Reference

Use [ViennaRNA/params/basic.h](#) instead.

Include dependency graph for params.h:

### 17.66.1 Detailed Description

Use [ViennaRNA/params/basic.h](#) instead.

**Deprecated** Use [ViennaRNA/params/basic.h](#) instead

## 17.67 ViennaRNA/params/1.8.4\_epars.h File Reference

Free energy parameters for parameter file conversion.

### 17.67.1 Detailed Description

Free energy parameters for parameter file conversion.

This file contains the free energy parameters used in ViennaRNAPackage 1.8.4. They are summarized in:

D.H.Mathews, J. Sabina, M. ZUker, D.H. Turner "Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure" JMB, 288, pp 911-940, 1999

Enthalpies taken from:

A. Walter, D Turner, J Kim, M Lytle, P M"uller, D Mathews, M Zuker "Coaxial stckaing of helices enhances binding of oligoribonucleotides.." PNAS, 91, pp 9218-9222, 1994

D.H. Turner, N. Sugimoto, and S.M. Freier. "RNA Structure Prediction", Ann. Rev. Biophys. Biophys. Chem. 17, 167-192, 1988.

John A.Jaeger, Douglas H.Turner, and Michael Zuker. "Improved predictions of secondary structures for RNA", PNAS, 86, 7706-7710, October 1989.

L. He, R. Kierzek, J. SantaLucia, A.E. Walter, D.H. Turner "Nearest-Neughbor Parameters for GU Mismatches..." Biochemistry 1991, 30 11124-11132

A.E. Peritz, R. Kierzek, N, Sugimoto, D.H. Turner "Thermodynamic Study of Internal Loops in Oligoribonucleotides..." Biochemistry 1991, 30, 6428-6435

## 17.68 ViennaRNA/params/1.8.4\_intloops.h File Reference

Free energy parameters for interior loop contributions needed by the parameter file conversion functions.

### 17.68.1 Detailed Description

Free energy parameters for interior loop contributions needed by the parameter file conversion functions.

## 17.69 ViennaRNA/params/basic.h File Reference

Functions to deal with sets of energy parameters.

Include dependency graph for basic.h:

This graph shows which files directly or indirectly include this file:

### Data Structures

- struct [vrna\\_param\\_s](#)  
*The datastructure that contains temperature scaled energy parameters. [More...](#)*
- struct [vrna\\_exp\\_param\\_s](#)  
*The data structure that contains temperature scaled Boltzmann weights of the energy parameters. [More...](#)*

### Typedefs

- typedef struct [vrna\\_param\\_s](#) [vrna\\_param\\_t](#)  
*Typename for the free energy parameter data structure [vrna\\_params](#).*
- typedef struct [vrna\\_exp\\_param\\_s](#) [vrna\\_exp\\_param\\_t](#)  
*Typename for the Boltzmann factor data structure [vrna\\_exp\\_params](#).*
- typedef struct [vrna\\_param\\_s](#) paramT  
*Old typename of [vrna\\_param\\_s](#).*
- typedef struct [vrna\\_exp\\_param\\_s](#) pf\_paramT  
*Old typename of [vrna\\_exp\\_param\\_s](#).*

## Functions

- `vrna_param_t * vrna_params (vrna_md_t *md)`  
*Get a data structure containing prescaled free energy parameters.*
- `vrna_param_t * vrna_params_copy (vrna_param_t *par)`  
*Get a copy of the provided free energy parameters.*
- `vrna_exp_param_t * vrna_exp_params (vrna_md_t *md)`  
*Get a data structure containing prescaled free energy parameters already transformed to Boltzmann factors.*
- `vrna_exp_param_t * vrna_exp_params_comparative (unsigned int n_seq, vrna_md_t *md)`  
*Get a data structure containing prescaled free energy parameters already transformed to Boltzmann factors (alifold version)*
- `vrna_exp_param_t * vrna_exp_params_copy (vrna_exp_param_t *par)`  
*Get a copy of the provided free energy parameters (provided as Boltzmann factors)*
- `void vrna_params_subst (vrna_fold_compound_t *vc, vrna_param_t *par)`  
*Update/Reset energy parameters data structure within a `vrna_fold_compound_t`.*
- `void vrna_exp_params_subst (vrna_fold_compound_t *vc, vrna_exp_param_t *params)`  
*Update the energy parameters for subsequent partition function computations.*
- `void vrna_exp_params_rescale (vrna_fold_compound_t *vc, double *mfe)`  
*Rescale Boltzmann factors for partition function computations.*
- `void vrna_params_reset (vrna_fold_compound_t *vc, vrna_md_t *md_p)`  
*Reset free energy parameters within a `vrna_fold_compound_t` according to provided, or default model details.*
- `void vrna_exp_params_reset (vrna_fold_compound_t *vc, vrna_md_t *md_p)`  
*Reset Boltzmann factors for partition function computations within a `vrna_fold_compound_t` according to provided, or default model details.*
- `vrna_exp_param_t * get_scaled_pf_parameters (void)`
- `vrna_exp_param_t * get_boltzmann_factors (double temperature, double betaScale, vrna_md_t md, double pf_scale)`  
*Get precomputed Boltzmann factors of the loop type dependent energy contributions with independent thermodynamic temperature.*
- `vrna_exp_param_t * get_boltzmann_factor_copy (vrna_exp_param_t *parameters)`  
*Get a copy of already precomputed Boltzmann factors.*
- `vrna_exp_param_t * get_scaled_alipf_parameters (unsigned int n_seq)`  
*Get precomputed Boltzmann factors of the loop type dependent energy contributions (alifold variant)*
- `vrna_exp_param_t * get_boltzmann_factors_ali (unsigned int n_seq, double temperature, double betaScale, vrna_md_t md, double pf_scale)`  
*Get precomputed Boltzmann factors of the loop type dependent energy contributions (alifold variant) with independent thermodynamic temperature.*
- `vrna_param_t * scale_parameters (void)`  
*Get precomputed energy contributions for all the known loop types.*
- `vrna_param_t * get_scaled_parameters (double temperature, vrna_md_t md)`  
*Get precomputed energy contributions for all the known loop types.*

### 17.69.1 Detailed Description

Functions to deal with sets of energy parameters.

## 17.70 ViennaRNA/constraints/basic.h File Reference

Functions and data structures for constraining secondary structure predictions and evaluation.

Include dependency graph for basic.h:

This graph shows which files directly or indirectly include this file:

### Macros

- `#define VRNA_CONSTRAINT_FILE 0`  
*Flag for `vrna_constraints_add()` to indicate that constraints are present in a text file.*
- `#define VRNA_CONSTRAINT_SOFT_MFE 0`  
*Indicate generation of constraints for MFE folding.*
- `#define VRNA_CONSTRAINT_SOFT_PF VRNA_OPTION_PF`  
*Indicate generation of constraints for partition function computation.*
- `#define VRNA_DECOMP_PAIR_HP (unsigned char)1`  
*Flag passed to generic softt constraints callback to indicate hairpin loop decomposition step.*
- `#define VRNA_DECOMP_PAIR_IL (unsigned char)2`  
*Indicator for interior loop decomposition step.*
- `#define VRNA_DECOMP_PAIR_ML (unsigned char)3`  
*Indicator for multibranch loop decomposition step.*
- `#define VRNA_DECOMP_ML_ML_ML (unsigned char)5`  
*Indicator for decomposition of multibranch loop part.*
- `#define VRNA_DECOMP_ML_STEM (unsigned char)6`  
*Indicator for decomposition of multibranch loop part.*
- `#define VRNA_DECOMP_ML_ML (unsigned char)7`  
*Indicator for decomposition of multibranch loop part.*
- `#define VRNA_DECOMP_ML_UP (unsigned char)8`  
*Indicator for decomposition of multibranch loop part.*
- `#define VRNA_DECOMP_ML_ML_STEM (unsigned char)9`  
*Indicator for decomposition of multibranch loop part.*
- `#define VRNA_DECOMP_ML_COAXIAL (unsigned char)10`  
*Indicator for decomposition of multibranch loop part.*
- `#define VRNA_DECOMP_ML_COAXIAL_ENC (unsigned char)11`  
*Indicator for decomposition of multibranch loop part.*
- `#define VRNA_DECOMP_EXT_EXT (unsigned char)12`  
*Indicator for decomposition of exterior loop part.*
- `#define VRNA_DECOMP_EXT_UP (unsigned char)13`  
*Indicator for decomposition of exterior loop part.*
- `#define VRNA_DECOMP_EXT_STEM (unsigned char)14`  
*Indicator for decomposition of exterior loop part.*
- `#define VRNA_DECOMP_EXT_EXT_EXT (unsigned char)15`  
*Indicator for decomposition of exterior loop part.*
- `#define VRNA_DECOMP_EXT_STEM_EXT (unsigned char)16`  
*Indicator for decomposition of exterior loop part.*

- #define [VRNA\\_DECOMP\\_EXT\\_STEM\\_OUTSIDE](#) (unsigned char)17  
*Indicator for decomposition of exterior loop part.*
- #define [VRNA\\_DECOMP\\_EXT\\_EXT\\_STEM](#) (unsigned char)18  
*Indicator for decomposition of exterior loop part.*
- #define [VRNA\\_DECOMP\\_EXT\\_EXT\\_STEM1](#) (unsigned char)19  
*Indicator for decomposition of exterior loop part.*

## Functions

- void [vrna\\_constraints\\_add](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*constraint, unsigned int options)  
*Add constraints to a [vrna\\_fold\\_compound\\_t](#) data structure.*

### 17.70.1 Detailed Description

Functions and data structures for constraining secondary structure predictions and evaluation.

## 17.71 ViennaRNA/utils/basic.h File Reference

General utility- and helper-functions used throughout the *ViennaRNA Package*.

Include dependency graph for basic.h:

This graph shows which files directly or indirectly include this file:

## Macros

- #define [VRNA\\_INPUT\\_ERROR](#) 1U  
*Output flag of [get\\_input\\_line\(\)](#): "An ERROR has occurred, maybe EOF".*
- #define [VRNA\\_INPUT\\_QUIT](#) 2U  
*Output flag of [get\\_input\\_line\(\)](#): "the user requested quitting the program".*
- #define [VRNA\\_INPUT\\_MISC](#) 4U  
*Output flag of [get\\_input\\_line\(\)](#): "something was read".*
- #define [VRNA\\_INPUT\\_FASTA\\_HEADER](#) 8U  
*Input/Output flag of [get\\_input\\_line\(\)](#):  
if used as input option this tells [get\\_input\\_line\(\)](#) that the data to be read should comply with the FASTA format.*
- #define [VRNA\\_INPUT\\_CONSTRAINT](#) 32U  
*Input flag for [get\\_input\\_line\(\)](#):  
Tell [get\\_input\\_line\(\)](#) that we assume to read a structure constraint.*
- #define [VRNA\\_INPUT\\_NO\\_TRUNCATION](#) 256U  
*Input switch for [get\\_input\\_line\(\)](#): "do not truncate the line by eliminating white spaces at end of line".*
- #define [VRNA\\_INPUT\\_NO\\_REST](#) 512U  
*Input switch for [vrna\\_file\\_fasta\\_read\\_record\(\)](#): "do fill rest array".*
- #define [VRNA\\_INPUT\\_NO\\_SPAN](#) 1024U

- Input switch for `vrna_file_fasta_read_record()`: "never allow data to span more than one line".*
- #define `VRNA_INPUT_NOSKIP_BLANK_LINES` 2048U
- Input switch for `vrna_file_fasta_read_record()`: "do not skip empty lines".*
- #define `VRNA_INPUT_BLANK_LINE` 4096U
- Output flag for `vrna_file_fasta_read_record()`: "read an empty line".*
- #define `VRNA_INPUT_NOSKIP_COMMENTS` 128U
- Input switch for `get_input_line()`: "do not skip comment lines".*
- #define `VRNA_INPUT_COMMENT` 8192U
- Output flag for `vrna_file_fasta_read_record()`: "read a comment".*
- #define `MIN2(A, B) ((A) < (B) ? (A) : (B))`
- Get the minimum of two comparable values.*
- #define `MAX2(A, B) ((A) > (B) ? (A) : (B))`
- Get the maximum of two comparable values.*
- #define `MIN3(A, B, C) (MIN2((MIN2((A), (B))), (C)))`
- Get the minimum of three comparable values.*
- #define `MAX3(A, B, C) (MAX2((MAX2((A), (B))), (C)))`
- Get the maximum of three comparable values.*

## Functions

- void \* `vrna_alloc` (unsigned size)
- Allocate space safely.*
- void \* `vrna_realloc` (void \*p, unsigned size)
- Reallocate space safely.*
- void `vrna_init_rand` (void)
- Initialize seed for random number generator.*
- double `vrna_urn` (void)
- get a random number from [0..1]*
- int `vrna_int_urn` (int from, int to)
- Generates a pseudo random integer in a specified range.*
- char \* `vrna_time_stamp` (void)
- Get a timestamp.*
- unsigned int `get_input_line` (char \*\*string, unsigned int options)
- int \* `vrna_idx_row_wise` (unsigned int length)
- Get an index mapper array (iindx) for accessing the energy matrices, e.g. in partition function related functions.*
- int \* `vrna_idx_col_wise` (unsigned int length)
- Get an index mapper array (indx) for accessing the energy matrices, e.g. in MFE related functions.*
- void `vrna_message_error` (const char \*format,...)
- Print an error message and die.*
- void `vrna_message_verror` (const char \*format, va\_list args)
- Print an error message and die.*
- void `vrna_message_warning` (const char \*format,...)
- Print a warning message.*
- void `vrna_message_vwarning` (const char \*format, va\_list args)
- Print a warning message.*
- void `vrna_message_info` (FILE \*fp, const char \*format,...)
- Print an info message.*
- void `vrna_message_vinfo` (FILE \*fp, const char \*format, va\_list args)
- Print an info message.*
- void `vrna_message_input_seq_simple` (void)

- *Print a line to stdout that asks for an input sequence.*
- void [vrna\\_message\\_input\\_seq](#) (const char \*s)
  - *Print a line with a user defined string and a ruler to stdout.*
- char \* [get\\_line](#) (FILE \*fp)
  - *Read a line of arbitrary length from a stream.*
- void [print\\_tty\\_input\\_seq](#) (void)
  - *Print a line to stdout that asks for an input sequence.*
- void [print\\_tty\\_input\\_seq\\_str](#) (const char \*s)
  - *Print a line with a user defined string and a ruler to stdout.*
- void [warn\\_user](#) (const char message[])
  - *Print a warning message.*
- void [nrerror](#) (const char message[])
  - *Die with an error message.*
- void \* [space](#) (unsigned size)
  - *Allocate space safely.*
- void \* [xrealloc](#) (void \*p, unsigned size)
  - *Reallocate space safely.*
- void [init\\_rand](#) (void)
  - *Make random number seeds.*
- double [urn](#) (void)
  - *get a random number from [0..1]*
- int [int\\_urn](#) (int from, int to)
  - *Generates a pseudo random integer in a specified range.*
- void [filecopy](#) (FILE \*from, FILE \*to)
  - *Inefficient cp*
- char \* [time\\_stamp](#) (void)
  - *Get a timestamp.*

## Variables

- unsigned short [xsubi](#) [3]
  - *Current 48 bit random number.*

### 17.71.1 Detailed Description

General utility- and helper-functions used throughout the *ViennaRNA Package*.

### 17.71.2 Function Documentation

#### 17.71.2.1 [get\\_line\(\)](#)

```
char* get_line (
    FILE * fp )
```

Read a line of arbitrary length from a stream.

Returns a pointer to the resulting string. The necessary memory is allocated and should be released using *free()* when the string is no longer needed.

**Deprecated** Use [vrna\\_read\\_line\(\)](#) as a substitute!



### Parameters

<i>fp</i>	A file pointer to the stream where the function should read from
-----------	--

### Returns

A pointer to the resulting string

#### 17.71.2.2 `print_tty_input_seq()`

```
void print_tty_input_seq (
    void )
```

Print a line to *stdout* that asks for an input sequence.

There will also be a ruler (scale line) printed that helps orientation of the sequence positions

**Deprecated** Use `vrna_message_input_seq_simple()` instead!

#### 17.71.2.3 `print_tty_input_seq_str()`

```
void print_tty_input_seq_str (
    const char * s )
```

Print a line with a user defined string and a ruler to stdout.

(usually this is used to ask for user input) There will also be a ruler (scale line) printed that helps orientation of the sequence positions

**Deprecated** Use `vrna_message_input_seq()` instead!

#### 17.71.2.4 `warn_user()`

```
void warn_user (
    const char message[ ] )
```

Print a warning message.

Print a warning message to *stderr*

**Deprecated** Use `vrna_message_warning()` instead!

#### 17.71.2.5 nrerror()

```
void nrerror (
    const char message[] )
```

Die with an error message.

**Deprecated** Use [vrna\\_message\\_error\(\)](#) instead!

#### 17.71.2.6 space()

```
void* space (
    unsigned size )
```

Allocate space safely.

**Deprecated** Use [vrna\\_alloc\(\)](#) instead!

#### 17.71.2.7 xrealloc()

```
void* xrealloc (
    void * p,
    unsigned size )
```

Reallocate space safely.

**Deprecated** Use [vrna\\_realloc\(\)](#) instead!

#### 17.71.2.8 init\_rand()

```
void init_rand (
    void )
```

Make random number seeds.

**Deprecated** Use [vrna\\_init\\_rand\(\)](#) instead!

### 17.71.2.9 urn()

```
double urn (
    void )
```

get a random number from [0..1]

**Deprecated** Use `vrna_urn()` instead!

### 17.71.2.10 int\_urn()

```
int int_urn (
    int from,
    int to )
```

Generates a pseudo random integer in a specified range.

**Deprecated** Use `vrna_int_urn()` instead!

### 17.71.2.11 filecopy()

```
void filecopy (
    FILE * from,
    FILE * to )
```

Inefficient `cp`

**Deprecated** Use `vrna_file_copy()` instead!

### 17.71.2.12 time\_stamp()

```
char* time_stamp (
    void )
```

Get a timestamp.

**Deprecated** Use `vrna_time_stamp()` instead!

## 17.72 ViennaRNA/datastructures/basic.h File Reference

Various data structures and pre-processor macros.

Include dependency graph for basic.h:

### Data Structures

- struct [vrna\\_basepair\\_s](#)  
*Base pair data structure used in subopt.c. [More...](#)*
- struct [vrna\\_cpair\\_s](#)  
*this datastructure is used as input parameter in functions of PS\_dot.c [More...](#)*
- struct [vrna\\_color\\_s](#)
- struct [vrna\\_data\\_linear\\_s](#)
- struct [vrna\\_sect\\_s](#)  
*Stack of partial structures for backtracking. [More...](#)*
- struct [vrna\\_bp\\_stack\\_s](#)  
*Base pair stack element. [More...](#)*
- struct [pu\\_contrib](#)  
*contributions to p\_u [More...](#)*
- struct [interact](#)  
*interaction data structure for RNAup [More...](#)*
- struct [pu\\_out](#)  
*Collection of all free\_energy of beeing unpaired values for output. [More...](#)*
- struct [constrain](#)  
*constraints for cofolding [More...](#)*
- struct [duplexT](#)  
*Data structure for RNAduplex. [More...](#)*
- struct [node](#)  
*Data structure for RNAsnoop (fold energy list) [More...](#)*
- struct [snoopT](#)  
*Data structure for RNAsnoop. [More...](#)*
- struct [dupVar](#)  
*Data structure used in RNApkplex. [More...](#)*

### Typedefs

- typedef struct [vrna\\_basepair\\_s](#) [vrna\\_basepair\\_t](#)  
*Typename for the base pair representing data structure [vrna\\_basepair\\_s](#).*
- typedef struct [vrna\\_elem\\_prob\\_s](#) [vrna\\_plist\\_t](#)  
*Typename for the base pair list representing data structure [vrna\\_elem\\_prob\\_s](#).*
- typedef struct [vrna\\_bp\\_stack\\_s](#) [vrna\\_bp\\_stack\\_t](#)  
*Typename for the base pair stack representing data structure [vrna\\_bp\\_stack\\_s](#).*
- typedef struct [vrna\\_cpair\\_s](#) [vrna\\_cpair\\_t](#)  
*Typename for data structure [vrna\\_cpair\\_s](#).*
- typedef struct [vrna\\_sect\\_s](#) [vrna\\_sect\\_t](#)  
*Typename for stack of partial structures [vrna\\_sect\\_s](#).*

- typedef double [FLT\\_OR\\_DBL](#)  
*Typename for floating point number in partition function computations.*
- typedef struct [vrna\\_basepair\\_s](#) PAIR  
*Old typename of [vrna\\_basepair\\_s](#).*
- typedef struct [vrna\\_elem\\_prob\\_s](#) plist  
*Old typename of [vrna\\_elem\\_prob\\_s](#).*
- typedef struct [vrna\\_cpair\\_s](#) cpair  
*Old typename of [vrna\\_cpair\\_s](#).*
- typedef struct [vrna\\_sect\\_s](#) sect  
*Old typename of [vrna\\_sect\\_s](#).*
- typedef struct [vrna\\_bp\\_stack\\_s](#) bondT  
*Old typename of [vrna\\_bp\\_stack\\_s](#).*
- typedef struct [pu\\_contrib](#) pu\_contrib  
*contributions to  $p_u$*
- typedef struct [interact](#) interact  
*interaction data structure for RNAup*
- typedef struct [pu\\_out](#) pu\_out  
*Collection of all free\_energy of beeing unpaired values for output.*
- typedef struct [constrain](#) constrain  
*constraints for cofolding*
- typedef struct [node](#) folden  
*Data structure for RNAsnoop (fold energy list)*
- typedef struct [dupVar](#) dupVar  
*Data structure used in RNAplex.*

## Functions

- void [vrna\\_C11\\_features](#) (void)  
*Dummy symbol to check whether the library was build using C11/C++11 features.*

### 17.72.1 Detailed Description

Various data structures and pre-processor macros.

## 17.73 ViennaRNA/params/constants.h File Reference

Energy parameter constants.

Include dependency graph for constants.h:

This graph shows which files directly or indirectly include this file:

## Macros

- `#define GASCONST 1.98717 /* in [cal/K] */`
- `#define K0 273.15`
- `#define INF 10000000 /* (INT_MAX/10) */`
- `#define FORBIDDEN 9999`
- `#define BONUS 10000`
- `#define NBPAIRS 7`
- `#define TURN 3`
- `#define MAXLOOP 30`

### 17.73.1 Detailed Description

Energy parameter constants.

### 17.73.2 Macro Definition Documentation

#### 17.73.2.1 GASCONST

```
#define GASCONST 1.98717 /* in [cal/K] */
```

The gas constant

#### 17.73.2.2 K0

```
#define K0 273.15
```

0 deg Celsius in Kelvin

#### 17.73.2.3 INF

```
#define INF 10000000 /* (INT_MAX/10) */
```

Infinity as used in minimization routines

#### 17.73.2.4 FORBIDDEN

```
#define FORBIDDEN 9999
```

forbidden

### 17.73.2.5 BONUS

```
#define BONUS 10000
```

bonus contribution

### 17.73.2.6 NBPAIRS

```
#define NBPAIRS 7
```

The number of distinguishable base pairs

### 17.73.2.7 TURN

```
#define TURN 3
```

The minimum loop length

### 17.73.2.8 MAXLOOP

```
#define MAXLOOP 30
```

The maximum loop length

## 17.74 ViennaRNA/params/convert.h File Reference

Functions and definitions for energy parameter file format conversion.

This graph shows which files directly or indirectly include this file:

### Macros

- `#define VRNA_CONVERT_OUTPUT_ALL 1U`
- `#define VRNA_CONVERT_OUTPUT_HP 2U`
- `#define VRNA_CONVERT_OUTPUT_STACK 4U`
- `#define VRNA_CONVERT_OUTPUT_MM_HP 8U`
- `#define VRNA_CONVERT_OUTPUT_MM_INT 16U`
- `#define VRNA_CONVERT_OUTPUT_MM_INT_1N 32U`
- `#define VRNA_CONVERT_OUTPUT_MM_INT_23 64U`
- `#define VRNA_CONVERT_OUTPUT_MM_MULTI 128U`
- `#define VRNA_CONVERT_OUTPUT_MM_EXT 256U`
- `#define VRNA_CONVERT_OUTPUT_DANGLE5 512U`
- `#define VRNA_CONVERT_OUTPUT_DANGLE3 1024U`
- `#define VRNA_CONVERT_OUTPUT_INT_11 2048U`
- `#define VRNA_CONVERT_OUTPUT_INT_21 4096U`
- `#define VRNA_CONVERT_OUTPUT_INT_22 8192U`
- `#define VRNA_CONVERT_OUTPUT_BULGE 16384U`
- `#define VRNA_CONVERT_OUTPUT_INT 32768U`
- `#define VRNA_CONVERT_OUTPUT_ML 65536U`
- `#define VRNA_CONVERT_OUTPUT_MISC 131072U`
- `#define VRNA_CONVERT_OUTPUT_SPECIAL_HP 262144U`
- `#define VRNA_CONVERT_OUTPUT_VANILLA 524288U`
- `#define VRNA_CONVERT_OUTPUT_NINIO 1048576U`
- `#define VRNA_CONVERT_OUTPUT_DUMP 2097152U`

## Functions

- void [convert\\_parameter\\_file](#) (const char \*iname, const char \*oname, unsigned int options)

### 17.74.1 Detailed Description

Functions and definitions for energy parameter file format conversion.

## 17.75 ViennaRNA/params/io.h File Reference

Read and write energy parameter files.

This graph shows which files directly or indirectly include this file:

## Functions

- const char \* [last\\_parameter\\_file](#) (void)  
*Get the file name of the parameter file that was most recently loaded.*
- void [read\\_parameter\\_file](#) (const char fname[])  
*Read energy parameters from a file.*
- void [write\\_parameter\\_file](#) (const char fname[])  
*Write energy parameters to a file.*

### 17.75.1 Detailed Description

Read and write energy parameter files.

## 17.76 ViennaRNA/part\_func.h File Reference

Partition function implementations.

Include dependency graph for part\_func.h:

This graph shows which files directly or indirectly include this file:

## Data Structures

- struct [vrna\\_dimer\\_pf\\_s](#)  
*Data structure returned by [vrna\\_pf\\_dimer\(\)](#) [More...](#)*



## Typedefs

- typedef struct [vrna\\_dimer\\_pf\\_s](#) [vrna\\_dimer\\_pf\\_t](#)  
 Typename for the data structure that stores the dimer partition functions, [vrna\\_dimer\\_pf\\_s](#), as returned by [vrna\\_pf\\_dimer\(\)](#)
- typedef struct [vrna\\_dimer\\_pf\\_s](#) [cofoldF](#)  
 Backward compatibility typedef for [vrna\\_dimer\\_pf\\_s](#).

## Functions

- int [vrna\\_pf\\_float\\_precision](#) (void)  
 Find out whether partition function computations are using single precision floating points.
- float [pf\\_fold\\_par](#) (const char \*sequence, char \*structure, [vrna\\_exp\\_param\\_t](#) \*parameters, int calculate\_bp, int is\_constrained, int is\_circular)  
 Compute the partition function  $Q$  for a given RNA sequence.
- float [pf\\_fold](#) (const char \*sequence, char \*structure)  
 Compute the partition function  $Q$  of an RNA sequence.
- float [pf\\_circ\\_fold](#) (const char \*sequence, char \*structure)  
 Compute the partition function of a circular RNA sequence.
- char \* [pbacktrack](#) (char \*sequence)  
 Sample a secondary structure from the Boltzmann ensemble according its probability.
- char \* [pbacktrack\\_circ](#) (char \*sequence)  
 Sample a secondary structure of a circular RNA from the Boltzmann ensemble according its probability.
- void [free\\_pf\\_arrays](#) (void)  
 Free arrays for the partition function recursions.
- void [update\\_pf\\_params](#) (int length)  
 Recalculate energy parameters.
- void [update\\_pf\\_params\\_par](#) (int length, [vrna\\_exp\\_param\\_t](#) \*parameters)  
 Recalculate energy parameters.
- [FLT\\_OR\\_DBL](#) \* [export\\_bp](#) (void)  
 Get a pointer to the base pair probability array.
- int [get\\_pf\\_arrays](#) (short \*\*S\_p, short \*\*S1\_p, char \*\*ptype\_p, [FLT\\_OR\\_DBL](#) \*\*qb\_p, [FLT\\_OR\\_DBL](#) \*\*qm\_p, [FLT\\_OR\\_DBL](#) \*\*q1k\_p, [FLT\\_OR\\_DBL](#) \*\*qln\_p)  
 Get the pointers to (almost) all relevant computation arrays used in partition function computation.
- double [get\\_subseq\\_F](#) (int i, int j)  
 Get the free energy of a subsequence from the  $q[]$  array.
- double [mean\\_bp\\_distance](#) (int length)  
 Get the mean base pair distance of the last partition function computation.
- double [mean\\_bp\\_distance\\_pr](#) (int length, [FLT\\_OR\\_DBL](#) \*pr)  
 Get the mean base pair distance in the thermodynamic ensemble.
- [vrna\\_ep\\_t](#) \* [stackProb](#) (double cutoff)  
 Get the probability of stacks.
- void [init\\_pf\\_fold](#) (int length)  
 Allocate space for [pf\\_fold\(\)](#)
- char \* [centroid](#) (int length, double \*dist)
- char \* [get\\_centroid\\_struct\\_gquad\\_pr](#) (int length, double \*dist)
- double [mean\\_bp\\_dist](#) (int length)
- double [expLoopEnergy](#) (int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short sq1)
- double [expHairpinEnergy](#) (int u, int type, short si1, short sj1, const char \*string)

## Basic global partition function interface

- float [vrna\\_pf](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, char \*structure)  
*Compute the partition function  $Q$  for a given RNA sequence, or sequence alignment.*
- [vrna\\_dimer\\_pf\\_t vrna\\_pf\\_dimer](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, char \*structure)  
*Calculate partition function and base pair probabilities of nucleic acid/nucleic acid dimers.*

### Simplified global partition function computation using sequence(s) or multiple sequence alignment(s)

- float [vrna\\_pf\\_fold](#) (const char \*sequence, char \*structure, [vrna\\_ep\\_t](#) \*\*pl)  
*Compute Partition function  $Q$  (and base pair probabilities) for an RNA sequence using a comparative method.*
- float [vrna\\_pf\\_circfold](#) (const char \*sequence, char \*structure, [vrna\\_ep\\_t](#) \*\*pl)  
*Compute Partition function  $Q$  (and base pair probabilities) for a circular RNA sequences using a comparative method.*
- float [vrna\\_pf\\_alifold](#) (const char \*\*sequences, char \*structure, [vrna\\_ep\\_t](#) \*\*pl)  
*Compute Partition function  $Q$  (and base pair probabilities) for an RNA sequence alignment using a comparative method.*
- float [vrna\\_pf\\_circalifold](#) (const char \*\*sequences, char \*structure, [vrna\\_ep\\_t](#) \*\*pl)  
*Compute Partition function  $Q$  (and base pair probabilities) for an alignment of circular RNA sequences using a comparative method.*
- [vrna\\_dimer\\_pf\\_t vrna\\_pf\\_co\\_fold](#) (const char \*seq, char \*structure, [vrna\\_ep\\_t](#) \*\*pl)  
*Calculate partition function and base pair probabilities of nucleic acid/nucleic acid dimers.*

### Variables

- int [st\\_back](#)  
*Flag indicating that auxiliary arrays are needed throughout the computations. This is essential for stochastic back-tracking.*

## 17.76.1 Detailed Description

Partition function implementations.

, This file includes (almost) all function declarations within the **RNALib** that are related to Partion function computations

## 17.76.2 Function Documentation

### 17.76.2.1 centroid()

```
char* centroid (
    int length,
    double * dist )
```

**Deprecated** This function is deprecated and should not be used anymore as it is not threadsafe!

See also

[get\\_centroid\\_struct\\_pl\(\)](#), [get\\_centroid\\_struct\\_pr\(\)](#)

### 17.76.2.2 `get_centroid_struct_gquad_pr()`

```
char* get_centroid_struct_gquad_pr (
    int length,
    double * dist )
```

**Deprecated** This function is deprecated and should not be used anymore as it is not threadsafe!

See also

[vrna\\_centroid\(\)](#), [vrna\\_centroid\\_from\\_probs\(\)](#), [vrna\\_centroid\\_from\\_plist\(\)](#)

### 17.76.2.3 `mean_bp_dist()`

```
double mean_bp_dist (
    int length )
```

get the mean pair distance of ensemble

**Deprecated** This function is not threadsafe and should not be used anymore. Use [mean\\_bp\\_distance\(\)](#) instead!

### 17.76.2.4 `expLoopEnergy()`

```
double expLoopEnergy (
    int u1,
    int u2,
    int type,
    int type2,
    short sil,
    short sj1,
    short spl,
    short sql )
```

**Deprecated** Use [exp\\_E\\_IntLoop\(\)](#) from [loop\\_energies.h](#) instead

### 17.76.2.5 `expHairpinEnergy()`

```
double expHairpinEnergy (
    int u,
    int type,
    short sil,
    short sj1,
    const char * string )
```

**Deprecated** Use [exp\\_E\\_Hairpin\(\)](#) from [loop\\_energies.h](#) instead

## 17.77 ViennaRNA/part\_func\_co.h File Reference

Partition function for two RNA sequences.

Include dependency graph for part\_func\_co.h:

### Functions

- [vrna\\_dimer\\_pf\\_t co\\_pf\\_fold](#) (char \*sequence, char \*structure)  
*Calculate partition function and base pair probabilities.*
- [vrna\\_dimer\\_pf\\_t co\\_pf\\_fold\\_par](#) (char \*sequence, char \*structure, [vrna\\_exp\\_param\\_t](#) \*parameters, int calculate\_bppm, int is\_constrained)  
*Calculate partition function and base pair probabilities.*
- [vrna\\_ep\\_t](#) \* [get\\_plist](#) ([vrna\\_ep\\_t](#) \*pl, int length, double cut\_off)
- void [compute\\_probabilities](#) (double FAB, double FEA, double FEB, [vrna\\_ep\\_t](#) \*prAB, [vrna\\_ep\\_t](#) \*prA, [vrna\\_ep\\_t](#) \*prB, int Alength)  
*Compute Boltzmann probabilities of dimerization without homodimers.*
- void [init\\_co\\_pf\\_fold](#) (int length)
- [FLT\\_OR\\_DBL](#) \* [export\\_co\\_bppm](#) (void)  
*Get a pointer to the base pair probability array.*
- void [free\\_co\\_pf\\_arrays](#) (void)  
*Free the memory occupied by [co\\_pf\\_fold\(\)](#)*
- void [update\\_co\\_pf\\_params](#) (int length)  
*Recalculate energy parameters.*
- void [update\\_co\\_pf\\_params\\_par](#) (int length, [vrna\\_exp\\_param\\_t](#) \*parameters)  
*Recalculate energy parameters.*

### Variables

- int [mirnatog](#)  
*Toggles no intrabp in 2nd mol.*
- double [F\\_monomer](#) [2]  
*Free energies of the two monomers.*

#### 17.77.1 Detailed Description

Partition function for two RNA sequences.

#### 17.77.2 Function Documentation

17.77.2.1 `get_plist()`

```
vrna_ep_t* get_plist (
    vrna_ep_t * pl,
    int length,
    double cut_off )
```

DO NOT USE THIS FUNCTION ANYMORE

**Deprecated** { This function is deprecated and will be removed soon!} use [assign\\_plist\\_from\\_pr\(\)](#) instead!

## 17.78 ViennaRNA/part\_func\_up.h File Reference

Implementations for accessibility and RNA-RNA interaction as a stepwise process.

Include dependency graph for `part_func_up.h`:

### Functions

- [pu\\_contrib](#) \* [pf\\_unstru](#) (char \*sequence, int max\_w)  
*Calculate the partition function over all unpaired regions of a maximal length.*
- [interact](#) \* [pf\\_interact](#) (const char \*s1, const char \*s2, [pu\\_contrib](#) \*p\_c, [pu\\_contrib](#) \*p\_c2, int max\_w, char \*cstruc, int incr3, int incr5)  
*Calculates the probability of a local interaction between two sequences.*
- void [free\\_interact](#) ([interact](#) \*pin)  
*Frees the output of function [pf\\_interact\(\)](#).*
- void [free\\_pu\\_contrib\\_struct](#) ([pu\\_contrib](#) \*pu)  
*Frees the output of function [pf\\_unstru\(\)](#).*

### 17.78.1 Detailed Description

Implementations for accessibility and RNA-RNA interaction as a stepwise process.

## 17.79 ViennaRNA/part\_func\_window.h File Reference

Partition function and equilibrium probability implementation for the sliding window algorithm.

Include dependency graph for `part_func_window.h`:

This graph shows which files directly or indirectly include this file:

## Macros

- `#define VRNA_EXT_LOOP 1U`  
*Exterior loop.*
- `#define VRNA_HP_LOOP 2U`  
*Hairpin loop.*
- `#define VRNA_INT_LOOP 4U`  
*Internal loop.*
- `#define VRNA_MB_LOOP 8U`  
*Multibranch loop.*
- `#define VRNA_ANY_LOOP (VRNA_EXT_LOOP | VRNA_HP_LOOP | VRNA_INT_LOOP | VRNA_MB_LOOP)`  
*Any loop.*
- `#define VRNA_PROBS_WINDOW_BPP 4096U`  
*Trigger base pairing probabilities.*
- `#define VRNA_PROBS_WINDOW_UP 8192U`  
*Trigger unpaired probabilities.*
- `#define VRNA_PROBS_WINDOW_STACKP 16384U`  
*Trigger base pair stack probabilities.*
- `#define VRNA_PROBS_WINDOW_UP_SPLIT 32768U`  
*Trigger detailed unpaired probabilities split up into different loop type contexts.*
- `#define VRNA_PROBS_WINDOW_PF 65536U`  
*Trigger partition function.*

## Typedefs

- `typedef void() vrna_probs_window_callback(FLT_OR_DBL *pr, int pr_size, int i, int max, unsigned int type, void *data)`  
*Sliding window probability computation callback.*

## Functions

### Basic local partition function interface

- `int vrna_probs_window(vrna_fold_compound_t *fc, int ulength, unsigned int options, vrna_probs_window_callback *cb, void *data)`  
*Compute various equilibrium probabilities under a sliding window approach.*

### Simplified global partition function computation using sequence(s) or multiple sequence alignment(s)

- `vrna_ep_t * vrna_pfl_fold (const char *sequence, int window_size, int max_bp_span, float cutoff)`  
*Compute base pair probabilities using a sliding-window approach.*
- `int vrna_pfl_fold_cb (const char *sequence, int window_size, int max_bp_span, vrna_probs_window_callback *cb, void *data)`  
*Compute base pair probabilities using a sliding-window approach (callback version)*
- `double ** vrna_pfl_fold_up (const char *sequence, int ulength, int window_size, int max_bp_span)`  
*Compute probability of contiguous unpaired segments.*
- `int vrna_pfl_fold_up_cb (const char *sequence, int ulength, int window_size, int max_bp_span, vrna_probs_window_callback *cb, void *data)`  
*Compute probability of contiguous unpaired segments.*

### 17.79.1 Detailed Description

Partition function and equilibrium probability implementation for the sliding window algorithm.

, This file contains the implementation for sliding window partition function and equilibrium probabilities. It also provides the unpaired probability implementation from Bernhart et al. 2011 [4]

## 17.80 ViennaRNA/perturbation\_fold.h File Reference

Find a vector of perturbation energies that minimizes the discrepancies between predicted and observed pairing probabilities and the amount of necessary adjustments.

Include dependency graph for perturbation\_fold.h:

### Macros

- `#define VRNA_OBJECTIVE_FUNCTION_QUADRATIC 0`  
*Use the sum of squared aberrations as objective function.*
- `#define VRNA_OBJECTIVE_FUNCTION_ABSOLUTE 1`  
*Use the sum of absolute aberrations as objective function.*
- `#define VRNA_MINIMIZER_DEFAULT 0`  
*Use a custom implementation of the gradient descent algorithm to minimize the objective function.*
- `#define VRNA_MINIMIZER_CONJUGATE_FR 1`  
*Use the GNU Scientific Library implementation of the Fletcher-Reeves conjugate gradient algorithm to minimize the objective function.*
- `#define VRNA_MINIMIZER_CONJUGATE_PR 2`  
*Use the GNU Scientific Library implementation of the Polak-Ribiere conjugate gradient algorithm to minimize the objective function.*
- `#define VRNA_MINIMIZER_VECTOR_BFGS 3`  
*Use the GNU Scientific Library implementation of the vector Broyden-Fletcher-Goldfarb-Shanno algorithm to minimize the objective function.*
- `#define VRNA_MINIMIZER_VECTOR_BFGS2 4`  
*Use the GNU Scientific Library implementation of the vector Broyden-Fletcher-Goldfarb-Shanno algorithm to minimize the objective function.*
- `#define VRNA_MINIMIZER_STEEPEST_DESCENT 5`  
*Use the GNU Scientific Library implementation of the steepest descent algorithm to minimize the objective function.*

### Typedefs

- `typedef void(* progress_callback) (int iteration, double score, double *epsilon)`  
*Callback for following the progress of the minimization process.*

### Functions

- `void vrna_sc_minimize_pertubation (vrna_fold_compound_t *vc, const double *q_prob_unpaired, int objective_function, double sigma_squared, double tau_squared, int algorithm, int sample_size, double *epsilon, double initialStepSize, double minStepSize, double minImprovement, double minimizerTolerance, progress_callback callback)`  
*Find a vector of perturbation energies that minimizes the discrepancies between predicted and observed pairing probabilities and the amount of necessary adjustments.*

### 17.80.1 Detailed Description

Find a vector of perturbation energies that minimizes the discrepancies between predicted and observed pairing probabilities and the amount of necessary adjustments.

## 17.81 ViennaRNA/plot\_aln.h File Reference

Use [ViennaRNA/plotting/alignments.h](#) instead.

Include dependency graph for plot\_aln.h:

### 17.81.1 Detailed Description

Use [ViennaRNA/plotting/alignments.h](#) instead.

**Deprecated** Use [ViennaRNA/plotting/alignments.h](#) instead

## 17.82 ViennaRNA/plot\_layouts.h File Reference

Use [ViennaRNA/plotting/layouts.h](#) instead.

Include dependency graph for plot\_layouts.h:

### 17.82.1 Detailed Description

Use [ViennaRNA/plotting/layouts.h](#) instead.

**Deprecated** Use [ViennaRNA/plotting/layouts.h](#) instead

## 17.83 ViennaRNA/plot\_structure.h File Reference

Use [ViennaRNA/plotting/structures.h](#) instead.

Include dependency graph for plot\_structure.h:



### 17.83.1 Detailed Description

Use [ViennaRNA/plotting/structures.h](#) instead.

**Deprecated** Use [ViennaRNA/plotting/structures.h](#) instead

## 17.84 ViennaRNA/plot\_utils.h File Reference

Use [ViennaRNA/plotting/utls.h](#) instead.

Include dependency graph for plot\_utils.h:

### 17.84.1 Detailed Description

Use [ViennaRNA/plotting/utls.h](#) instead.

**Deprecated** Use [ViennaRNA/plotting/utls.h](#) instead

## 17.85 ViennaRNA/plotting/alignments.h File Reference

Various functions for plotting Sequence / Structure Alignments.

This graph shows which files directly or indirectly include this file:

### Functions

- int [PS\\_color\\_aln](#) (const char \*structure, const char \*filename, const char \*seqs[], const char \*names[])  
*Produce PostScript sequence alignment color-annotated by consensus structure.*
- int [vrna\\_file\\_PS\\_aln](#) (const char \*filename, const char \*\*seqs, const char \*\*names, const char \*structure, int columns)
- int [vrna\\_file\\_PS\\_aln\\_sub](#) (const char \*filename, const char \*\*seqs, const char \*\*names, const char \*structure, int start, int end, int columns)
- int [aliPS\\_color\\_aln](#) (const char \*structure, const char \*filename, const char \*seqs[], const char \*names[])

### 17.85.1 Detailed Description

Various functions for plotting Sequence / Structure Alignments.

## 17.86 ViennaRNA/utils/alignments.h File Reference

Various utility- and helper-functions for sequence alignments and comparative structure prediction.

Include dependency graph for alignments.h:

This graph shows which files directly or indirectly include this file:

### Data Structures

- struct [vrna\\_pinfo\\_s](#)  
*A base pair info structure. [More...](#)*

### Macros

- #define [VRNA\\_ALN\\_DEFAULT](#) 0U  
*Use default alignment settings.*
- #define [VRNA\\_ALN\\_RNA](#) 1U  
*Convert to RNA alphabet.*
- #define [VRNA\\_ALN\\_DNA](#) 2U  
*Convert to DNA alphabet.*
- #define [VRNA\\_ALN\\_UPPERCASE](#) 4U  
*Convert to uppercase nucleotide letters.*
- #define [VRNA\\_ALN\\_LOWERCASE](#) 8U  
*Convert to lowercase nucleotide letters.*
- #define [VRNA\\_MEASURE\\_SHANNON\\_ENTROPY](#) 1U  
*Flag indicating Shannon Entropy measure.*

### Typedefs

- typedef struct [vrna\\_pinfo\\_s](#) [vrna\\_pinfo\\_t](#)  
*Typename for the base pair info representing data structure [vrna\\_pinfo\\_s](#).*
- typedef struct [vrna\\_pinfo\\_s](#) [pair\\_info](#)  
*Old typename of [vrna\\_pinfo\\_s](#).*

## Functions

- int [vrna\\_aln\\_mpi](#) (const char \*\*alignment)  
*Get the mean pairwise identity in steps from ?to?(ident)*
- [vrna\\_pinfo\\_t](#) \* [vrna\\_aln\\_pinfo](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*structure, double threshold)  
*Retrieve an array of [vrna\\_pinfo\\_t](#) structures from precomputed pair probabilities.*
- char \*\* [vrna\\_aln\\_slice](#) (const char \*\*alignment, unsigned int i, unsigned int j)  
*Slice out a subalignment from a larger alignment.*
- void [vrna\\_aln\\_free](#) (char \*\*alignment)  
*Free memory occupied by a set of aligned sequences.*
- char \*\* [vrna\\_aln\\_uppercase](#) (const char \*\*alignment)  
*Create a copy of an alignment with only uppercase letters in the sequences.*
- char \*\* [vrna\\_aln\\_toRNA](#) (const char \*\*alignment)  
*Create a copy of an alignment where DNA alphabet is replaced by RNA alphabet.*
- char \*\* [vrna\\_aln\\_copy](#) (const char \*\*alignment, unsigned int options)  
*Make a copy of a multiple sequence alignment.*
- float \* [vrna\\_aln\\_conservation\\_struct](#) (const char \*\*alignment, const char \*structure, const [vrna\\_md\\_t](#) \*md)  
*Compute base pair conservation of a consensus structure.*
- float \* [vrna\\_aln\\_conservation\\_col](#) (const char \*\*alignment, const [vrna\\_md\\_t](#) \*md\_p, unsigned int options)  
*Compute nucleotide conservation in an alignment.*
- char \* [vrna\\_aln\\_consensus\\_sequence](#) (const char \*\*alignment, const [vrna\\_md\\_t](#) \*md\_p)  
*Compute the consensus sequence for a given multiple sequence alignment.*
- char \* [vrna\\_aln\\_consensus\\_mis](#) (const char \*\*alignment, const [vrna\\_md\\_t](#) \*md\_p)  
*Compute the Most Informative Sequence (MIS) for a given multiple sequence alignment.*
- int [get\\_mpi](#) (char \*Aseq[], int n\_seq, int length, int \*mini)  
*Get the mean pairwise identity in steps from ?to?(ident)*
- void [encode\\_aln\\_sequence](#) (const char \*sequence, short \*S, short \*s5, short \*s3, char \*ss, unsigned short \*as, int circ)  
*Get arrays with encoded sequence of the alignment.*
- void [alloc\\_sequence\\_arrays](#) (const char \*\*sequences, short \*\*\*S, short \*\*\*S5, short \*\*\*S3, unsigned short \*\*\*a2s, char \*\*\*Ss, int circ)  
*Allocate memory for sequence array used to deal with aligned sequences.*
- void [free\\_sequence\\_arrays](#) (unsigned int n\_seq, short \*\*\*S, short \*\*\*S5, short \*\*\*S3, unsigned short \*\*\*a2s, char \*\*\*Ss)  
*Free the memory of the sequence arrays used to deal with aligned sequences.*

### 17.86.1 Detailed Description

Various utility- and helper-functions for sequence alignments and comparative structure prediction.

,

## 17.87 ViennaRNA/plotting/layouts.h File Reference

Secondary structure plot layout algorithms.

Include dependency graph for layouts.h:

This graph shows which files directly or indirectly include this file:

## Data Structures

- struct [COORDINATE](#)

*this is a workaround for the SWIG Perl Wrapper RNA plot function that returns an array of type [COORDINATE](#) [More...](#)*

## Macros

- #define [VRNA\\_PLOT\\_TYPE\\_SIMPLE](#) 0  
*Definition of Plot type simple*
- #define [VRNA\\_PLOT\\_TYPE\\_NAVIEW](#) 1  
*Definition of Plot type Naview*
- #define [VRNA\\_PLOT\\_TYPE\\_CIRCULAR](#) 2  
*Definition of Plot type Circular*

## Functions

- int [simple\\_xy\\_coordinates](#) (short \*pair\_table, float \*X, float \*Y)  
*Calculate nucleotide coordinates for secondary structure plot the Simple way*
- int [simple\\_circplot\\_coordinates](#) (short \*pair\_table, float \*x, float \*y)  
*Calculate nucleotide coordinates for Circular Plot*

## Variables

- int [rna\\_plot\\_type](#)  
*Switch for changing the secondary structure layout algorithm.*

### 17.87.1 Detailed Description

Secondary structure plot layout algorithms.

## 17.88 ViennaRNA/plotting/probabilities.h File Reference

Various functions for plotting RNA secondary structures, dot-plots and other visualizations.

Include dependency graph for probabilities.h:

This graph shows which files directly or indirectly include this file:

## Data Structures

- struct [vrna\\_dotplot\\_auxdata\\_t](#)

## Functions

- int [PS\\_dot\\_plot\\_list](#) (char \*seq, char \*filename, [plist](#) \*pl, [plist](#) \*mf, char \*comment)  
*Produce a postscript dot-plot from two pair lists.*
- int [PS\\_dot\\_plot](#) (char \*string, char \*file)  
*Produce postscript dot-plot.*

### 17.88.1 Detailed Description

Various functions for plotting RNA secondary structures, dot-plots and other visualizations.

## 17.89 ViennaRNA/plotting/structures.h File Reference

Various functions for plotting RNA secondary structures.

Include dependency graph for structures.h:

This graph shows which files directly or indirectly include this file:

## Functions

- int [vrna\\_file\\_PS\\_rnaplot](#) (const char \*seq, const char \*structure, const char \*file, [vrna\\_md\\_t](#) \*md\_p)  
*Produce a secondary structure graph in PostScript and write it to 'filename'.*
- int [vrna\\_file\\_PS\\_rnaplot\\_a](#) (const char \*seq, const char \*structure, const char \*file, const char \*pre, const char \*post, [vrna\\_md\\_t](#) \*md\_p)  
*Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename'.*
- int [gmlRNA](#) (char \*string, char \*structure, char \*ssfile, char option)  
*Produce a secondary structure graph in Graph Meta Language (gml) and write it to a file.*
- int [ssv\\_rna\\_plot](#) (char \*string, char \*structure, char \*ssfile)  
*Produce a secondary structure graph in SStructView format.*
- int [svg\\_rna\\_plot](#) (char \*string, char \*structure, char \*ssfile)  
*Produce a secondary structure plot in SVG format and write it to a file.*
- int [xrna\\_plot](#) (char \*string, char \*structure, char \*ssfile)  
*Produce a secondary structure plot for further editing in XRNA.*
- int [PS\\_rna\\_plot](#) (char \*string, char \*structure, char \*file)  
*Produce a secondary structure graph in PostScript and write it to 'filename'.*
- int [PS\\_rna\\_plot\\_a](#) (char \*string, char \*structure, char \*file, char \*pre, char \*post)  
*Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename'.*
- int [PS\\_rna\\_plot\\_a\\_gquad](#) (char \*string, char \*structure, char \*ssfile, char \*pre, char \*post)  
*Produce a secondary structure graph in PostScript including additional annotation macros and write it to 'filename' (detect and draw g-quadruplexes)*

### 17.89.1 Detailed Description

Various functions for plotting RNA secondary structures.

## 17.90 ViennaRNA/utls/structures.h File Reference

Various utility- and helper-functions for secondary structure parsing, converting, etc.

Include dependency graph for structures.h:

This graph shows which files directly or indirectly include this file:

### Data Structures

- struct [vrna\\_elem\\_prob\\_s](#)  
Data structure representing a single entry of an element probability list (e.g. list of pair probabilities) [More...](#)
- struct [vrna\\_hx\\_s](#)  
Data structure representing an entry of a helix list. [More...](#)

### Macros

- #define [VRNA\\_BRACKETS\\_ALPHA](#) 4U  
Bitflag to indicate secondary structure notations using uppercase/lowercase letters from the latin alphabet.
- #define [VRNA\\_BRACKETS\\_RND](#) 8U  
Bitflag to indicate secondary structure notations using round brackets (parenthesis), ( )
- #define [VRNA\\_BRACKETS\\_CLY](#) 16U  
Bitflag to indicate secondary structure notations using curly brackets, { }
- #define [VRNA\\_BRACKETS\\_ANG](#) 32U  
Bitflag to indicate secondary structure notations using angular brackets, < >
- #define [VRNA\\_BRACKETS\\_SQR](#) 64U  
Bitflag to indicate secondary structure notations using square brackets, [ ]
- #define [VRNA\\_BRACKETS\\_DEFAULT](#)  
Default bitmask to indicate secondary structure notation using any pair of brackets.
- #define [VRNA\\_PLIST\\_TYPE\\_BASEPAIR](#) 0  
A Base Pair element.
- #define [VRNA\\_PLIST\\_TYPE\\_GQUAD](#) 1  
A G-Quadruplex element.
- #define [VRNA\\_PLIST\\_TYPE\\_H\\_MOTIF](#) 2  
A Hairpin loop motif element.
- #define [VRNA\\_PLIST\\_TYPE\\_I\\_MOTIF](#) 3  
An Internal loop motif element.
- #define [VRNA\\_PLIST\\_TYPE\\_UD\\_MOTIF](#) 4  
An Unstructured Domain motif element.

- `#define VRNA_PLIST_TYPE_STACK 5`  
*A Base Pair stack element.*
- `#define VRNA_STRUCTURE_TREE_HIT 1U`  
*Homeomorphically Irreducible [Tree](#) (HIT) representation of a secondary structure.*
- `#define VRNA_STRUCTURE_TREE_SHAPIRO_SHORT 2U`  
*(short) Coarse Grained representation of a secondary structure*
- `#define VRNA_STRUCTURE_TREE_SHAPIRO 3U`  
*(full) Coarse Grained representation of a secondary structure*
- `#define VRNA_STRUCTURE_TREE_SHAPIRO_EXT 4U`  
*(extended) Coarse Grained representation of a secondary structure*
- `#define VRNA_STRUCTURE_TREE_SHAPIRO_WEIGHT 5U`  
*(weighted) Coarse Grained representation of a secondary structure*
- `#define VRNA_STRUCTURE_TREE_EXPANDED 6U`  
*Expanded [Tree](#) representation of a secondary structure.*

## Typedefs

- `typedef struct vrna_hx_s vrna_hx_t`  
*Convenience typedef for data structure [vrna\\_hx\\_s](#).*
- `typedef struct vrna_elem_prob_s vrna_ep_t`  
*Convenience typedef for data structure [vrna\\_elem\\_prob\\_s](#).*

## Functions

- `char * vrna_db_pack (const char *struc)`  
*Pack secondary secondary structure, 5:1 compression using base 3 encoding.*
- `char * vrna_db_unpack (const char *packed)`  
*Unpack secondary structure previously packed with [vrna\\_db\\_pack\(\)](#)*
- `void vrna_db_flatten (char *structure, unsigned int options)`  
*Substitute pairs of brackets in a string with parenthesis.*
- `void vrna_db_flatten_to (char *string, const char target[3], unsigned int options)`  
*Substitute pairs of brackets in a string with another type of pair characters.*
- `char * vrna_db_from_ptable (short *pt)`  
*Convert a pair table into dot-parenthesis notation.*
- `char * vrna_db_from_WUSS (const char *wuss)`  
*Convert a WUSS annotation string to dot-bracket format.*
- `char * vrna_db_from_plist (vrna_ep_t *pairs, unsigned int n)`  
*Convert a list of base pairs into dot-bracket notation.*
- `char * vrna_db_to_element_string (const char *structure)`  
*Convert a secondary structure in dot-bracket notation to a nucleotide annotation of loop contexts.*
- `short * vrna_ptable (const char *structure)`  
*Create a pair table from a dot-bracket notation of a secondary structure.*
- `short * vrna_ptable_from_string (const char *string, unsigned int options)`  
*Create a pair table for a secondary structure string.*
- `short * vrna_pt_pk_get (const char *structure)`  
*Create a pair table of a secondary structure (pseudo-knot version)*
- `short * vrna_ptable_copy (const short *pt)`  
*Get an exact copy of a pair table.*
- `short * vrna_pt_ali_get (const char *structure)`

- Create a pair table of a secondary structure (snoop align version)*

  - short \* [vrna\\_pt\\_snoop\\_get](#) (const char \*structure)
- Create a pair table of a secondary structure (snoop version)*

  - [vrna\\_ep\\_t](#) \* [vrna\\_plist](#) (const char \*struc, float pr)
- Create a [vrna\\_ep\\_t](#) from a dot-bracket string.*

  - [vrna\\_ep\\_t](#) \* [vrna\\_plist\\_from\\_probs](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, double cut\_off)
- Create a [vrna\\_ep\\_t](#) from base pair probability matrix.*

  - [vrna\\_hx\\_t](#) \* [vrna\\_hx\\_from\\_ptable](#) (short \*pt)
- Convert a pair table representation of a secondary structure into a helix list.*

  - [vrna\\_hx\\_t](#) \* [vrna\\_hx\\_merge](#) (const [vrna\\_hx\\_t](#) \*list, int maxdist)
- Create a merged helix list from another helix list.*

  - int \* [vrna\\_loopidx\\_from\\_ptable](#) (const short \*pt)
- Get a loop index representation of a structure.*

  - int [vrna\\_bp\\_distance](#) (const char \*str1, const char \*str2)
- Compute the "base pair" distance between two secondary structures s1 and s2.*

  - unsigned int \* [vrna\\_refBPcnt\\_matrix](#) (const short \*reference\_pt, unsigned int turn)
- Make a reference base pair count matrix.*

  - unsigned int \* [vrna\\_refBPdist\\_matrix](#) (const short \*pt1, const short \*pt2, unsigned int turn)
- Make a reference base pair distance matrix.*

  - char \* [vrna\\_db\\_from\\_probs](#) (const [FLT\\_OR\\_DBL](#) \*pr, unsigned int length)
- Create a dot-bracket like structure string from base pair probability matrix.*

  - char [vrna\\_bpp\\_symbol](#) (const float \*x)
- Get a pseudo dot bracket notation for a given probability information.*

  - char \* [vrna\\_db\\_from\\_bp\\_stack](#) ([vrna\\_bp\\_stack\\_t](#) \*bp, unsigned int length)
- Create a dot-bracket/parenthesis structure from backtracking stack.*

  - char \* [vrna\\_db\\_to\\_tree\\_string](#) (const char \*structure, unsigned int type)
- Convert a Dot-Bracket structure string into tree string representation.*

  - char \* [vrna\\_tree\\_string\\_unweight](#) (const char \*structure)
- Remove weights from a linear string tree representation of a secondary structure.*

  - char \* [vrna\\_tree\\_string\\_to\\_db](#) (const char \*tree)
- Convert a linear tree string representation of a secondary structure back to Dot-Bracket notation.*

  - void [assign\\_plist\\_from\\_db](#) ([vrna\\_ep\\_t](#) \*\*pl, const char \*struc, float pr)
- Create a [vrna\\_ep\\_t](#) from a dot-bracket string.*

  - char \* [pack\\_structure](#) (const char \*struc)
- Pack secondary secondary structure, 5:1 compression using base 3 encoding.*

  - char \* [unpack\\_structure](#) (const char \*packed)
- Unpack secondary structure previously packed with [pack\\_structure\(\)](#)*

  - short \* [make\\_pair\\_table](#) (const char \*structure)
- Create a pair table of a secondary structure.*

  - short \* [copy\\_pair\\_table](#) (const short \*pt)
- Get an exact copy of a pair table.*

  - short \* [alimake\\_pair\\_table](#) (const char \*structure)
  - short \* [make\\_pair\\_table\\_snoop](#) (const char \*structure)
  - int [bp\\_distance](#) (const char \*str1, const char \*str2)
- Compute the "base pair" distance between two secondary structures s1 and s2.*

  - unsigned int \* [make\\_referenceBP\\_array](#) (short \*reference\_pt, unsigned int turn)
- Make a reference base pair count matrix.*

  - unsigned int \* [compute\\_BPdifferences](#) (short \*pt1, short \*pt2, unsigned int turn)
- Make a reference base pair distance matrix.*

  - void [assign\\_plist\\_from\\_pr](#) ([vrna\\_ep\\_t](#) \*\*pl, [FLT\\_OR\\_DBL](#) \*probs, int length, double cutoff)
- Create a [vrna\\_ep\\_t](#) from a probability matrix.*



- void [parenthesis\\_structure](#) (char \*structure, [vrna\\_bp\\_stack\\_t](#) \*bp, int length)  
*Create a dot-bracket/parenthesis structure from backtracking stack.*
- void [parenthesis\\_zuker](#) (char \*structure, [vrna\\_bp\\_stack\\_t](#) \*bp, int length)  
*Create a dot-bracket/parenthesis structure from backtracking stack obtained by zuker suboptimal calculation in cofold.c.*
- void [bppm\\_to\\_structure](#) (char \*structure, [FLT\\_OR\\_DBL](#) \*pr, unsigned int length)  
*Create a dot-bracket like structure string from base pair probability matrix.*
- char [bppm\\_symbol](#) (const float \*x)  
*Get a pseudo dot bracket notation for a given probability information.*

### 17.90.1 Detailed Description

Various utility- and helper-functions for secondary structure parsing, converting, etc.

## 17.91 ViennaRNA/profiledist.h File Reference

Include dependency graph for profiledist.h:

### Functions

- float [profile\\_edit\\_distance](#) (const float \*T1, const float \*T2)  
*Align the 2 probability profiles T1, T2*
- float \* [Make\\_bp\\_profile\\_bppm](#) ([FLT\\_OR\\_DBL](#) \*bppm, int length)  
*condense pair probability matrix into a vector containing probabilities for unpaired, upstream paired and downstream paired.*
- void [print\\_bppm](#) (const float \*T)  
*print string representation of probability profile*
- void [free\\_profile](#) (float \*T)  
*free space allocated in Make\_bp\_profile*
- float \* [Make\\_bp\\_profile](#) (int length)

### 17.91.1 Function Documentation

#### 17.91.1.1 [profile\\_edit\\_distance\(\)](#)

```
float profile_edit_distance (
    const float * T1,
    const float * T2 )
```

Align the 2 probability profiles T1, T2

.

This is like a Needleman-Wunsch alignment, we should really use affine gap-costs ala Gotoh

### 17.91.1.2 Make\_bp\_profile\_bppm()

```
float* Make_bp_profile_bppm (
    FLT_OR_DBL * bppm,
    int length )
```

condense pair probability matrix into a vector containing probabilities for unpaired, upstream paired and downstream paired.

This resulting probability profile is used as input for `profile_edit_distance`

#### Parameters

<i>bppm</i>	A pointer to the base pair probability matrix
<i>length</i>	The length of the sequence

#### Returns

The bp profile

### 17.91.1.3 free\_profile()

```
void free_profile (
    float * T )
```

free space allocated in `Make_bp_profile`

Backward compatibility only. You can just use plain `free()`

### 17.91.1.4 Make\_bp\_profile()

```
float* Make_bp_profile (
    int length )
```

#### Note

This function is NOT threadsafe

#### See also

[Make\\_bp\\_profile\\_bppm\(\)](#)

**Deprecated** This function is deprecated and will be removed soon! See [Make\\_bp\\_profile\\_bppm\(\)](#) for a replacement

## 17.92 ViennaRNA/PS\_dot.h File Reference

Use [ViennaRNA/plotting/probabilities.h](#) instead.

Include dependency graph for PS\_dot.h:

### 17.92.1 Detailed Description

Use [ViennaRNA/plotting/probabilities.h](#) instead.

**Deprecated** Use [ViennaRNA/plotting/probabilities.h](#) instead

## 17.93 ViennaRNA/read\_epars.h File Reference

Use [ViennaRNA/params/io.h](#) instead.

Include dependency graph for read\_epars.h:

### 17.93.1 Detailed Description

Use [ViennaRNA/params/io.h](#) instead.

**Deprecated** Use [ViennaRNA/params/io.h](#) instead

## 17.94 ViennaRNA/ribo.h File Reference

Parse RiboSum Scoring Matrices for Covariance Scoring of Alignments.

This graph shows which files directly or indirectly include this file:

### Functions

- float \*\* [get\\_ribosum](#) (const char \*\*Alseq, int n\_seq, int length)  
*Retrieve a RiboSum Scoring Matrix for a given Alignment.*
- float \*\* [readribosum](#) (char \*name)  
*Read a RiboSum or other user-defined Scoring Matrix and Store into global Memory.*

### 17.94.1 Detailed Description

Parse RiboSum Scoring Matrices for Covariance Scoring of Alignments.

## 17.95 ViennaRNA/RNAstruct.h File Reference

Parsing and Coarse Graining of Structures.

### Functions

- char \* [b2HIT](#) (const char \*structure)  
*Converts the full structure from bracket notation to the HIT notation including root.*
- char \* [b2C](#) (const char \*structure)  
*Converts the full structure from bracket notation to the a coarse grained notation using the 'H' 'B' 'I' 'M' and 'R' identifiers.*
- char \* [b2Shapiro](#) (const char \*structure)  
*Converts the full structure from bracket notation to the weighted coarse grained notation using the 'H' 'B' 'I' 'M' 'S' 'E' and 'R' identifiers.*
- char \* [add\\_root](#) (const char \*structure)  
*Adds a root to an un-rooted tree in any except bracket notation.*
- char \* [expand\\_Shapiro](#) (const char \*coarse)  
*Inserts missing 'S' identifiers in unweighted coarse grained structures as obtained from [b2C\(\)](#).*
- char \* [expand\\_Full](#) (const char \*structure)  
*Convert the full structure from bracket notation to the expanded notation including root.*
- char \* [unexpand\\_Full](#) (const char \*ffull)  
*Restores the bracket notation from an expanded full or HIT tree, that is any tree using only identifiers 'U' 'P' and 'R'.*
- char \* [unweight](#) (const char \*wcoarse)  
*Strip weights from any weighted tree.*
- void [unexpand\\_aligned\\_F](#) (char \*align[2])  
*Converts two aligned structures in expanded notation.*
- void [parse\\_structure](#) (const char \*structure)  
*Collects a statistic of structure elements of the full structure in bracket notation.*

### Variables

- int [loop\\_size](#) [2000]  
*contains a list of all loop sizes. `loop_size[0]` contains the number of external bases.*
- int [helix\\_size](#) [2000]  
*contains a list of all stack sizes.*
- int [loop\\_degree](#) [2000]  
*contains the corresponding list of loop degrees.*
- int [loops](#)  
*contains the number of loops ( and therefore of stacks ).*
- int [unpaired](#)  
*contains the number of unpaired bases.*
- int [pairs](#)  
*contains the number of base pairs in the last parsed structure.*

### 17.95.1 Detailed Description

Parsing and Coarse Graining of Structures.

Example:

```
*  .((..(((....)))..((...))).  is the bracket or full tree
*  becomes expanded: - expand_Full() -
*  ((U)((U)(U)((U)(U)(U)P)P)P)(U)(U)((U)(U)P)P)P)(U)R)
*  HIT: - b2HIT() -
*  ((U1)((U2)((U3)P3)(U2)((U2)P2)P2)(U1)R)
*  Coarse: - b2C() -
*  ((H)((H)M)R)
*  becomes expanded: - expand_Shapiro() -
*  (((((H)S)((H)S)M)S)R)
*  weighted Shapiro: - b2Shapiro() -
*  (((((H3)S3)((H2)S2)M4)S2)E2)R)
*
```

## 17.96 ViennaRNA/search/BoyerMoore.h File Reference

Variants of the Boyer-Moore string search algorithm.

### Functions

- `const unsigned int * vrna_search_BMH_num` (`const unsigned int *needle`, `size_t needle_size`, `const unsigned int *haystack`, `size_t haystack_size`, `size_t start`, `size_t *badchars`, `unsigned char cyclic`)  
*Search for a string of elements in a larger string of elements using the Boyer-Moore-Horspool algorithm.*
- `const char * vrna_search_BMH` (`const char *needle`, `size_t needle_size`, `const char *haystack`, `size_t haystack_size`, `size_t start`, `size_t *badchars`, `unsigned char cyclic`)  
*Search for an ASCII pattern within a larger ASCII string using the Boyer-Moore-Horspool algorithm.*
- `size_t * vrna_search_BM_BCT_num` (`const unsigned int *pattern`, `size_t pattern_size`, `unsigned int num_max`)  
*Retrieve a Boyer-Moore Bad Character Table for a pattern of elements represented by natural numbers.*
- `size_t * vrna_search_BM_BCT` (`const char *pattern`)  
*Retrieve a Boyer-Moore Bad Character Table for a NULL-terminated pattern of ASCII characters.*

### 17.96.1 Detailed Description

Variants of the Boyer-Moore string search algorithm.

,

## 17.97 ViennaRNA/sequence.h File Reference

Functions and data structures related to sequence representations ,.

This graph shows which files directly or indirectly include this file:

## Data Structures

- struct [vrna\\_sequence\\_s](#)

*Data structure representing a nucleotide sequence. [More...](#)*

## Typedefs

- typedef struct [vrna\\_sequence\\_s](#) [vrna\\_seq\\_t](#)

*Typename for nucleotide sequence representation data structure [vrna\\_sequence\\_s](#).*

## Enumerations

- enum [vrna\\_seq\\_type\\_e](#) { [VRNA\\_SEQ\\_UNKNOWN](#), [VRNA\\_SEQ\\_RNA](#), [VRNA\\_SEQ\\_DNA](#) }

*A enumerator used in [vrna\\_sequence\\_s](#) to distinguish different nucleotide sequences.*

### 17.97.1 Detailed Description

Functions and data structures related to sequence representations ,.

## 17.98 ViennaRNA/stream\_output.h File Reference

Use [ViennaRNA/datastructures/stream\\_output.h](#) instead.

Include dependency graph for stream\_output.h:

### 17.98.1 Detailed Description

Use [ViennaRNA/datastructures/stream\\_output.h](#) instead.

**Deprecated** Use [ViennaRNA/datastructures/stream\\_output.h](#) instead

## 17.99 ViennaRNA/datastructures/stream\_output.h File Reference

An implementation of a buffered, ordered stream output data structure.

This graph shows which files directly or indirectly include this file:

## Typedefs

- typedef struct vrna\_ordered\_stream\_s \* [vrna\\_ostream\\_t](#)  
*An ordered output stream structure with unordered insert capabilities.*
- typedef void() [vrna\\_callback\\_stream\\_output](#)(void \*auxdata, unsigned int i, void \*data)  
*Ordered stream processing callback.*

## Functions

- [vrna\\_ostream\\_t](#) [vrna\\_ostream\\_init](#) ([vrna\\_callback\\_stream\\_output](#) \*output, void \*auxdata)  
*Get an initialized ordered output stream.*
- void [vrna\\_ostream\\_free](#) ([vrna\\_ostream\\_t](#) dat)  
*Free an initialized ordered output stream.*
- void [vrna\\_ostream\\_request](#) ([vrna\\_ostream\\_t](#) dat, unsigned int num)  
*Request index in ordered output stream.*
- void [vrna\\_ostream\\_provide](#) ([vrna\\_ostream\\_t](#) dat, unsigned int i, void \*data)  
*Provide output stream data for a particular index.*

### 17.99.1 Detailed Description

An implementation of a buffered, ordered stream output data structure.

,

## 17.100 ViennaRNA/string\_utils.h File Reference

Use [ViennaRNA/utls/strings.h](#) instead.

Include dependency graph for string\_utils.h:

### 17.100.1 Detailed Description

Use [ViennaRNA/utls/strings.h](#) instead.

**Deprecated** Use [ViennaRNA/utls/strings.h](#) instead

## 17.101 ViennaRNA/stringdist.h File Reference

Functions for String Alignment.

Include dependency graph for stringdist.h:

## Functions

- `swString * Make_swString` (`char *string`)  
Convert a structure into a format suitable for `string_edit_distance()`.
- `float string_edit_distance` (`swString *T1`, `swString *T2`)  
Calculate the string edit distance of `T1` and `T2`.

### 17.101.1 Detailed Description

Functions for String Alignment.

### 17.101.2 Function Documentation

#### 17.101.2.1 Make\_swString()

```
swString* Make_swString (
    char * string )
```

Convert a structure into a format suitable for `string_edit_distance()`.

##### Parameters

<code>string</code>	
---------------------	--

##### Returns

#### 17.101.2.2 string\_edit\_distance()

```
float string_edit_distance (
    swString * T1,
    swString * T2 )
```

Calculate the string edit distance of `T1` and `T2`.

##### Parameters

<code>T1</code>	
<code>T2</code>	



Returns

## 17.102 ViennaRNA/structure\_utils.h File Reference

Use [ViennaRNA/utis/structures.h](#) instead.

Include dependency graph for structure\_utils.h:

### 17.102.1 Detailed Description

Use [ViennaRNA/utis/structures.h](#) instead.

**Deprecated** Use [ViennaRNA/utis/structures.h](#) instead

## 17.103 ViennaRNA/structured\_domains.h File Reference

This module provides interfaces that deal with additional structured domains in the folding grammar.

This graph shows which files directly or indirectly include this file:

### Data Structures

- struct [vrna\\_structured\\_domains\\_s](#)

### 17.103.1 Detailed Description

This module provides interfaces that deal with additional structured domains in the folding grammar.

## 17.104 ViennaRNA/subopt.h File Reference

RNAsubopt and density of states declarations.

Include dependency graph for subopt.h:

## Data Structures

- struct [vrna\\_subopt\\_sol\\_s](#)  
*Solution element from subopt.c.*

## Macros

- #define [MAXDOS](#) 1000  
*Maximum density of states discretization for subopt.*

## Typedefs

- typedef struct [vrna\\_subopt\\_sol\\_s](#) [vrna\\_subopt\\_solution\\_t](#)  
*Typename for the subopt solution list representing data structure [vrna\\_subopt\\_sol\\_s](#).*
- typedef void() [vrna\\_subopt\\_callback](#)(const char \*structure, float energy, void \*data)  
*Callback for [vrna\\_subopt\\_cb\(\)](#)*
- typedef struct [vrna\\_subopt\\_sol\\_s](#) SOLUTION  
*Backward compatibility typedef for [vrna\\_subopt\\_sol\\_s](#).*

## Functions

- [vrna\\_subopt\\_solution\\_t](#) \* [vrna\\_subopt](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int delta, int sorted, FILE \*fp)  
*Returns list of subopt structures or writes to fp.*
- void [vrna\\_subopt\\_cb](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int delta, [vrna\\_subopt\\_callback](#) \*cb, void \*data)  
*Generate suboptimal structures within an energy band around the MFE.*
- [vrna\\_subopt\\_solution\\_t](#) \* [vrna\\_subopt\\_zuker](#) ([vrna\\_fold\\_compound\\_t](#) \*vc)  
*Compute Zuker type suboptimal structures.*
- SOLUTION \* [subopt](#) (char \*seq, char \*structure, int delta, FILE \*fp)  
*Returns list of subopt structures or writes to fp.*
- SOLUTION \* [subopt\\_par](#) (char \*seq, char \*structure, [vrna\\_param\\_t](#) \*parameters, int delta, int is\_↔ constrained, int is\_circular, FILE \*fp)  
*Returns list of subopt structures or writes to fp.*
- SOLUTION \* [subopt\\_circ](#) (char \*seq, char \*sequence, int delta, FILE \*fp)  
*Returns list of circular subopt structures or writes to fp.*
- SOLUTION \* [zukersubopt](#) (const char \*string)  
*Compute Zuker type suboptimal structures.*
- SOLUTION \* [zukersubopt\\_par](#) (const char \*string, [vrna\\_param\\_t](#) \*parameters)  
*Compute Zuker type suboptimal structures.*

## Variables

- double [print\\_energy](#)  
*printing threshold for use with logML*
- int [subopt\\_sorted](#)  
*Sort output by energy.*
- int [density\\_of\\_states](#) [MAXDOS+1]  
*The Density of States.*

### 17.104.1 Detailed Description

RNAsubopt and density of states declarations.

### 17.104.2 Typedef Documentation

#### 17.104.2.1 SOLUTION

```
typedef struct vrna_subopt_sol_s SOLUTION
```

Backward compatibility typedef for [vrna\\_subopt\\_sol\\_s](#).

**Deprecated** Use [vrna\\_subopt\\_solution\\_t](#) instead!

## 17.105 ViennaRNA/svm\_utils.h File Reference

Use [ViennaRNA/utis/svm.h](#) instead.

Include dependency graph for svm\_utils.h:

### 17.105.1 Detailed Description

Use [ViennaRNA/utis/svm.h](#) instead.

**Deprecated** Use [ViennaRNA/utis/svm.h](#) instead

## 17.106 ViennaRNA/treedist.h File Reference

Functions for [Tree](#) Edit Distances.

Include dependency graph for treedist.h:

## Functions

- `Tree * make_tree (char *struc)`  
Constructs a *Tree* ( essentially the postorder list ) of the structure 'struc', for use in `tree_edit_distance()`.
- `float tree_edit_distance (Tree *T1, Tree *T2)`  
Calculates the edit distance of the two trees.
- `void print_tree (Tree *t)`  
Print a tree (mainly for debugging)
- `void free_tree (Tree *t)`  
Free the memory allocated for *Tree* t.

### 17.106.1 Detailed Description

Functions for *Tree* Edit Distances.

### 17.106.2 Function Documentation

#### 17.106.2.1 make\_tree()

```
Tree* make_tree (
    char * struc )
```

Constructs a *Tree* ( essentially the postorder list ) of the structure 'struc', for use in `tree_edit_distance()`.

##### Parameters

<i>struc</i>	may be any rooted structure representation.
--------------	---

##### Returns

#### 17.106.2.2 tree\_edit\_distance()

```
float tree_edit_distance (
    Tree * T1,
    Tree * T2 )
```

Calculates the edit distance of the two trees.

##### Parameters

<i>T1</i>	
<i>T2</i>	

## Returns

## 17.106.2.3 free\_tree()

```
void free_tree (
    Tree * t )
```

Free the memory allocated for [Tree](#) t.

## Parameters

<i>t</i>	
----------	--

## 17.107 ViennaRNA/units.h File Reference

Physical Units and Functions to convert them into each other.

## Enumerations

- enum [vrna\\_unit\\_energy\\_e](#) {  
[VRNA\\_UNIT\\_J](#), [VRNA\\_UNIT\\_KJ](#), [VRNA\\_UNIT\\_CAL\\_IT](#), [VRNA\\_UNIT\\_DACAL\\_IT](#),  
[VRNA\\_UNIT\\_KCAL\\_IT](#), [VRNA\\_UNIT\\_CAL](#), [VRNA\\_UNIT\\_DACAL](#), [VRNA\\_UNIT\\_KCAL](#),  
[VRNA\\_UNIT\\_G\\_TNT](#), [VRNA\\_UNIT\\_KG\\_TNT](#), [VRNA\\_UNIT\\_T\\_TNT](#), [VRNA\\_UNIT\\_EV](#),  
[VRNA\\_UNIT\\_WH](#), [VRNA\\_UNIT\\_KWH](#) }  
*Energy / Work Units.*
- enum [vrna\\_unit\\_temperature\\_e](#) {  
[VRNA\\_UNIT\\_K](#), [VRNA\\_UNIT\\_DEG\\_C](#), [VRNA\\_UNIT\\_DEG\\_F](#), [VRNA\\_UNIT\\_DEG\\_R](#),  
[VRNA\\_UNIT\\_DEG\\_N](#), [VRNA\\_UNIT\\_DEG\\_DE](#), [VRNA\\_UNIT\\_DEG\\_RE](#), [VRNA\\_UNIT\\_DEG\\_RO](#) }  
*Temperature Units.*

## Functions

- double [vrna\\_convert\\_energy](#) (double energy, [vrna\\_unit\\_energy\\_e](#) from, [vrna\\_unit\\_energy\\_e](#) to)  
*Convert between energy / work units.*
- double [vrna\\_convert\\_temperature](#) (double temp, [vrna\\_unit\\_temperature\\_e](#) from, [vrna\\_unit\\_temperature\\_e](#) to)  
*Convert between temperature units.*

## 17.107.1 Detailed Description

Physical Units and Functions to convert them into each other.

,

## 17.108 ViennaRNA/unstructured\_domains.h File Reference

Functions to modify unstructured domains, e.g. to incorporate ligands binding to unpaired stretches.

Include dependency graph for unstructured\_domains.h:

This graph shows which files directly or indirectly include this file:

### Data Structures

- struct [vrna\\_unstructured\\_domain\\_s](#)  
*Data structure to store all functionality for ligand binding. [More...](#)*
- struct [vrna\\_unstructured\\_domain\\_motif\\_s](#)

### Macros

- `#define VRNA_UNSTRUCTURED_DOMAIN_EXT_LOOP 1U`  
*Flag to indicate ligand bound to unpaired stretch in the exterior loop.*
- `#define VRNA_UNSTRUCTURED_DOMAIN_HP_LOOP 2U`  
*Flag to indicate ligand bound to unpaired stretch in a hairpin loop.*
- `#define VRNA_UNSTRUCTURED_DOMAIN_INT_LOOP 4U`  
*Flag to indicate ligand bound to unpaired stretch in an interior loop.*
- `#define VRNA_UNSTRUCTURED_DOMAIN_MB_LOOP 8U`  
*Flag to indicate ligand bound to unpaired stretch in a multibranch loop.*
- `#define VRNA_UNSTRUCTURED_DOMAIN_MOTIF 16U`  
*Flag to indicate ligand binding without additional unbound nucleotides (motif-only)*
- `#define VRNA_UNSTRUCTURED_DOMAIN_ALL_LOOPS`  
*Flag to indicate ligand bound to unpaired stretch in any loop (convenience macro)*

### Typedefs

- typedef struct [vrna\\_unstructured\\_domain\\_s](#) [vrna\\_ud\\_t](#)  
*Typename for the ligand binding extension data structure [vrna\\_unstructured\\_domain\\_s](#).*
- typedef int([vrna\\_callback\\_ud\\_energy](#))([vrna\\_fold\\_compound\\_t](#) \*vc, int i, int j, unsigned int loop\_type, void \*data)  
*Callback to retrieve binding free energy of a ligand bound to an unpaired sequence segment.*
- typedef FLT\_OR\_DBL([vrna\\_callback\\_ud\\_exp\\_energy](#))([vrna\\_fold\\_compound\\_t](#) \*vc, int i, int j, unsigned int loop\_type, void \*data)  
*Callback to retrieve Boltzmann factor of the binding free energy of a ligand bound to an unpaired sequence segment.*
- typedef void([vrna\\_callback\\_ud\\_production](#))([vrna\\_fold\\_compound\\_t](#) \*vc, void \*data)  
*Callback for pre-processing the production rule of the ligand binding to unpaired stretches feature.*
- typedef void([vrna\\_callback\\_ud\\_exp\\_production](#))([vrna\\_fold\\_compound\\_t](#) \*vc, void \*data)  
*Callback for pre-processing the production rule of the ligand binding to unpaired stretches feature (partition function variant)*
- typedef void([vrna\\_callback\\_ud\\_probs\\_add](#))([vrna\\_fold\\_compound\\_t](#) \*vc, int i, int j, unsigned int loop\_type, FLT\_OR\_DBL exp\_energy, void \*data)  
*Callback to store/add equilibrium probability for a ligand bound to an unpaired sequence segment.*
- typedef FLT\_OR\_DBL([vrna\\_callback\\_ud\\_probs\\_get](#))([vrna\\_fold\\_compound\\_t](#) \*vc, int i, int j, unsigned int loop\_type, int motif, void \*data)  
*Callback to retrieve equilibrium probability for a ligand bound to an unpaired sequence segment.*

## Functions

- [vrna\\_ud\\_motif\\_t](#) \* [vrna\\_ud\\_motifs\\_centroid](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, const char \*structure)  
*Detect unstructured domains in centroid structure.*
- [vrna\\_ud\\_motif\\_t](#) \* [vrna\\_ud\\_motifs\\_MEA](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, const char \*structure, [vrna\\_ep\\_t](#) \*probability\_list)  
*Detect unstructured domains in MEA structure.*
- [vrna\\_ud\\_motif\\_t](#) \* [vrna\\_ud\\_motifs\\_MFE](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, const char \*structure)  
*Detect unstructured domains in MFE structure.*
- void [vrna\\_ud\\_add\\_motif](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*motif, double motif\_en, const char \*motif←\_name, unsigned int loop\_type)  
*Add an unstructured domain motif, e.g. for ligand binding.*
- int \* [vrna\\_ud\\_get\\_motif\\_size\\_at](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int i, unsigned int loop\_type)  
*Get a list of unique motif sizes that start at a certain position within the sequence.*
- void [vrna\\_ud\\_remove](#) ([vrna\\_fold\\_compound\\_t](#) \*vc)  
*Remove ligand binding to unpaired stretches.*
- void [vrna\\_ud\\_set\\_data](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, void \*data, [vrna\\_callback\\_free\\_auxdata](#) \*free\_cb)  
*Attach an auxiliary data structure.*
- void [vrna\\_ud\\_set\\_prod\\_rule\\_cb](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_callback\\_ud\\_production](#) \*pre\_cb, [vrna\\_callback\\_ud\\_energy](#) \*e\_cb)  
*Attach production rule callbacks for free energies computations.*
- void [vrna\\_ud\\_set\\_exp\\_prod\\_rule\\_cb](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_callback\\_ud\\_exp\\_production](#) \*pre←\_cb, [vrna\\_callback\\_ud\\_exp\\_energy](#) \*exp\_e\_cb)  
*Attach production rule for partition function.*
- void [vrna\\_ud\\_set\\_prob\\_cb](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_callback\\_ud\\_probs\\_add](#) \*setter, [vrna\\_callback\\_ud\\_probs\\_get](#) \*getter)

### 17.108.1 Detailed Description

Functions to modify unstructured domains, e.g. to incorporate ligands binding to unpaired stretches.

### 17.108.2 Function Documentation

#### 17.108.2.1 [vrna\\_ud\\_set\\_prob\\_cb\(\)](#)

```
void vrna_ud_set_prob_cb (
    vrna_fold_compound_t * vc,
    vrna_callback_ud_probs_add * setter,
    vrna_callback_ud_probs_get * getter )
```

**SWIG Wrapper Notes** This function is attached as method [ud\\_set\\_prob\\_cb\(\)](#) to objects of type *fold\_compound*

## 17.109 ViennaRNA/utils.h File Reference

Use [ViennaRNA/utils/basic.h](#) instead.

Include dependency graph for utils.h:

### 17.109.1 Detailed Description

Use [ViennaRNA/utils/basic.h](#) instead.

**Deprecated** Use [ViennaRNA/utils/basic.h](#) instead

## 17.110 ViennaRNA/io/utils.h File Reference

Several utilities for file handling.

This graph shows which files directly or indirectly include this file:

### Functions

- void [vrna\\_file\\_copy](#) (FILE \*from, FILE \*to)  
*Inefficient 'cp'.*
- char \* [vrna\\_read\\_line](#) (FILE \*fp)  
*Read a line of arbitrary length from a stream.*
- int [vrna\\_mkdir\\_p](#) (const char \*path)  
*Recursively create a directory tree.*
- char \* [vrna\\_basename](#) (const char \*path)  
*Extract the filename from a file path.*
- char \* [vrna\\_dirname](#) (const char \*path)  
*Extract the directory part of a file path.*
- char \* [vrna\\_filename\\_sanitize](#) (const char \*name, const char \*replacement)  
*Sanitize a file name.*
- int [vrna\\_file\\_exists](#) (const char \*filename)  
*Check if a file already exists in the file system.*

### 17.110.1 Detailed Description

Several utilities for file handling.

,



## 17.111 ViennaRNA/plotting/utils.h File Reference

Various utilities to assist in plotting secondary structures and consensus structures.

This graph shows which files directly or indirectly include this file:

### Functions

- char \*\* [vrna\\_annotate\\_covar\\_struct](#) (const char \*\*alignment, const char \*structure, [vrna\\_md\\_t](#) \*md)  
*Produce covariance annotation for an alignment given a secondary structure.*
- [vrna\\_cpair\\_t](#) \* [vrna\\_annotate\\_covar\\_pairs](#) (const char \*\*alignment, [vrna\\_ep\\_t](#) \*pl, [vrna\\_ep\\_t](#) \*mfel, double threshold, [vrna\\_md\\_t](#) \*md)  
*Produce covariance annotation for an alignment given a set of base pairs.*

### 17.111.1 Detailed Description

Various utilities to assist in plotting secondary structures and consensus structures.

,

## 17.112 ViennaRNA/utils/strings.h File Reference

General utility- and helper-functions for RNA sequence and structure strings used throughout the ViennaRNA Package.

Include dependency graph for strings.h:

This graph shows which files directly or indirectly include this file:

### Macros

- #define [XSTR](#)(s) [STR](#)(s)  
*Stringify a macro after expansion.*
- #define [STR](#)(s) #s  
*Stringify a macro argument.*
- #define [FILENAME\\_MAX\\_LENGTH](#) 80  
*Maximum length of filenames that are generated by our programs.*
- #define [FILENAME\\_ID\\_LENGTH](#) 42  
*Maximum length of id taken from fasta header for filename generation.*

## Functions

- char \* [vrna\\_strdup\\_printf](#) (const char \*format,...)  
*Safely create a formatted string.*
- char \* [vrna\\_strdup\\_vprintf](#) (const char \*format, va\_list argp)  
*Safely create a formatted string.*
- int [vrna\\_strcat\\_printf](#) (char \*\*dest, const char \*format,...)  
*Safely append a formatted string to another string.*
- int [vrna\\_strcat\\_vprintf](#) (char \*\*dest, const char \*format, va\_list args)  
*Safely append a formatted string to another string.*
- char \*\* [vrna\\_strsplit](#) (const char \*string, const char \*delimiter)  
*Split a string into tokens using a delimiting character.*
- char \* [vrna\\_random\\_string](#) (int l, const char symbols[])  
*Create a random string using characters from a specified symbol set.*
- int [vrna\\_hamming\\_distance](#) (const char \*s1, const char \*s2)  
*Calculate hamming distance between two sequences.*
- int [vrna\\_hamming\\_distance\\_bound](#) (const char \*s1, const char \*s2, int n)  
*Calculate hamming distance between two sequences up to a specified length.*
- void [vrna\\_seq\\_toRNA](#) (char \*sequence)  
*Convert an input sequence (possibly containing DNA alphabet characters) to RNA alphabet.*
- void [vrna\\_seq\\_toupper](#) (char \*sequence)  
*Convert an input sequence to uppercase.*
- char \* [vrna\\_seq\\_ungapped](#) (const char \*seq)  
*Remove gap characters from a nucleotide sequence.*
- char \* [vrna\\_cut\\_point\\_insert](#) (const char \*string, int cp)  
*Add a separating '&' character into a string according to cut-point position.*
- char \* [vrna\\_cut\\_point\\_remove](#) (const char \*string, int \*cp)  
*Remove a separating '&' character from a string.*
- void [str\\_uppercase](#) (char \*sequence)  
*Convert an input sequence to uppercase.*
- void [str\\_DNA2RNA](#) (char \*sequence)  
*Convert a DNA input sequence to RNA alphabet.*
- char \* [random\\_string](#) (int l, const char symbols[])  
*Create a random string using characters from a specified symbol set.*
- int [hamming](#) (const char \*s1, const char \*s2)  
*Calculate hamming distance between two sequences.*
- int [hamming\\_bound](#) (const char \*s1, const char \*s2, int n)  
*Calculate hamming distance between two sequences up to a specified length.*

### 17.112.1 Detailed Description

General utility- and helper-functions for RNA sequence and structure strings used throughout the ViennaRNA Package.

### 17.112.2 Function Documentation

### 17.112.2.1 str\_uppercase()

```
void str_uppercase (
    char * sequence )
```

Convert an input sequence to uppercase.

**Deprecated** Use [vrna\\_seq\\_toupper\(\)](#) instead!

### 17.112.2.2 str\_DNA2RNA()

```
void str_DNA2RNA (
    char * sequence )
```

Convert a DNA input sequence to RNA alphabet.

**Deprecated** Use [vrna\\_seq\\_toRNA\(\)](#) instead!

### 17.112.2.3 random\_string()

```
char* random_string (
    int l,
    const char symbols[] )
```

Create a random string using characters from a specified symbol set.

**Deprecated** Use [vrna\\_random\\_string\(\)](#) instead!

### 17.112.2.4 hamming()

```
int hamming (
    const char * s1,
    const char * s2 )
```

Calculate hamming distance between two sequences.

**Deprecated** Use [vrna\\_hamming\\_distance\(\)](#) instead!

### 17.112.2.5 hamming\_bound()

```
int hamming_bound (
    const char * s1,
    const char * s2,
    int n )
```

Calculate hamming distance between two sequences up to a specified length.

**Deprecated** Use [vrna\\_hamming\\_distance\\_bound\(\)](#) instead!

## 17.113 ViennaRNA/walk.h File Reference

Methods to generate particular paths such as gradient or random walks through the energy landscape of an RNA sequence.

Include dependency graph for walk.h:

### Macros

- `#define VRNA_PATH_STEEPEST_DESCENT 128`  
*Option flag to request a steepest descent / gradient path.*
- `#define VRNA_PATH_RANDOM 256`  
*Option flag to request a random walk path.*
- `#define VRNA_PATH_NO_TRANSITION_OUTPUT 512`  
*Option flag to omit returning the transition path.*
- `#define VRNA_PATH_DEFAULT (VRNA_PATH_STEEPEST_DESCENT | VRNA_MOVESET_DEFAULT)`  
*Option flag to request defaults (steepest descent / default move set)*

### Functions

- `vrna_move_t * vrna_path (vrna_fold_compound_t *vc, short *pt, unsigned int steps, unsigned int options)`  
*Compute a path, store the final structure, and return a list of transition moves from the start to the final structure.*
- `vrna_move_t * vrna_path_gradient (vrna_fold_compound_t *vc, short *pt, unsigned int options)`  
*Compute a steepest descent / gradient path, store the final structure, and return a list of transition moves from the start to the final structure.*
- `vrna_move_t * vrna_path_random (vrna_fold_compound_t *vc, short *pt, unsigned int steps, unsigned int options)`  
*Generate a random walk / path of a given length, store the final structure, and return a list of transition moves from the start to the final structure.*

### 17.113.1 Detailed Description

Methods to generate particular paths such as gradient or random walks through the energy landscape of an RNA sequence.

# Bibliography

- [1] S.H. Bernhart, I.L. Hofacker, S. Will, A.R. Gruber, and P.F. Stadler. RNAalifold: Improved consensus structure prediction for RNA alignments. *BMC bioinformatics*, 9(1):474, 2008. [26](#)
- [2] S.H. Bernhart, H. Tafer, U. Mückstein, C. Flamm, P.F. Stadler, and I.L. Hofacker. Partition function and base pairing probabilities of RNA heterodimers. *Algorithms for Molecular Biology*, 1(1):3, 2006. [411](#)
- [3] Stephan H Bernhart, Ivo L Hofacker, and Peter F Stadler. Local RNA base pairing probabilities in large sequences. *Bioinformatics*, 22(5):614–615, 2005. [278](#)
- [4] Stephan H Bernhart, Ullrike Mückstein, and Ivo L Hofacker. RNA accessibility in cubic time. *Algorithms for Molecular Biology*, 6(1):3, 2011. [279](#), [658](#), [689](#)
- [5] R.E. Brucoleri and G. Heinrich. An improved algorithm for nucleic acid secondary structure display. *Computer applications in the biosciences: CABIOS*, 4(1):167–173, 1988. [532](#)
- [6] Katherine E. Deigan, Tian W. Li, David H. Mathews, and Kevin M. Weeks. Accurate SHAPE-directed RNA structure determination. *PNAS*, 106:97–102, 2009. [340](#)
- [7] Christoph Flamm, Ivo L Hofacker, Sebastian Maurer-Stroh, Peter F Stadler, and Martin Zehl. Design of multi-stable RNA molecules. *RNA*, 7(02):254–265, 2001. [426](#), [427](#), [428](#), [429](#)
- [8] W. Fontana, P.F. Stadler, E.G. Bornberg-Bauer, T. Griesmacher, I.L. Hofacker, M. Tacker, P. Tarazona, E.D. Weinberger, and P. Schuster. RNA folding and combinatorial landscapes. *Physical review E*, 47(3):2083, 1993. [31](#), [32](#), [466](#)
- [9] Eva Freyhult, Vincent Moulton, and Paul Gardner. Predicting RNA structure using mutual information. *Applied bioinformatics*, 4(1):53–59, 2005. [490](#)
- [10] I.L. Hofacker, M. Fekete, and P.F. Stadler. Secondary structure prediction for aligned RNA sequences. *Journal of molecular biology*, 319(5):1059–1066, 2002. [26](#)
- [11] I.L. Hofacker, W. Fontana, P.F. Stadler, L.S. Bonhoeffer, M. Tacker, and P. Schuster. Fast folding and comparison of RNA secondary structures. *Monatshefte für Chemie/Chemical Monthly*, 125(2):167–188, 1994. [1](#)
- [12] I.L. Hofacker and P.F. Stadler. Memory efficient folding algorithms for circular RNA secondary structures. *Bioinformatics*, 22(10):1172–1176, 2006. [24](#), [254](#), [255](#), [273](#), [274](#)
- [13] Ronny Lorenz, Stephan H. Bernhart, Christian Höner zu Siederdissen, Hakim Tafer, Christoph Flamm, Peter F. Stadler, and Ivo L. Hofacker. ViennaRNA package 2.0. *Algorithms for Molecular Biology*, 6(1):26, 2011. [1](#)
- [14] Ronny Lorenz, Christoph Flamm, and Ivo L. Hofacker. 2d projections of RNA folding landscapes. In Ivo Grosse, Steffen Neumann, Stefan Posch, Falk Schreiber, and Peter F. Stadler, editors, *German Conference on Bioinformatics 2009*, volume 157 of *Lecture Notes in Informatics*, pages 11–20, Bonn, September 2009. Gesellschaft f. Informatik. [27](#), [307](#)
- [15] Ronny Lorenz, Ivo L. Hofacker, and Peter F. Stadler. RNA folding with hard and soft constraints. *Algorithms for Molecular Biology*, 11(1):1–13, 2016. [22](#)
- [16] Ronny Lorenz, Dominik Luntzer, Ivo L. Hofacker, Peter F. Stadler, and Michael T. Wolfinger. Shape directed rna folding. *Bioinformatics*, 32(1):145–147, 2016. [339](#)

- [17] D.H. Mathews, M.D. Disney, J.L. Childs, S.J. Schroeder, M. Zuker, and D.H. Turner. Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of RNA secondary structure. *Proceedings of the National Academy of Sciences of the United States of America*, 101(19):7287, 2004. [417](#)
- [18] J.S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29(6-7):1105–1119, 1990. [24](#), [25](#)
- [19] Joe Sawada. A fast algorithm to generate necklaces with fixed content. *Theoretical Computer Science*, 301(1):477–489, 2003. [538](#)
- [20] B.A. Shapiro. An algorithm for comparing multiple RNA secondary structures. *Computer applications in the biosciences: CABIOS*, 4(3):387–393, 1988. [31](#), [32](#), [466](#), [467](#)
- [21] B.A. Shapiro and K. Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Computer applications in the biosciences: CABIOS*, 6(4):309–318, 1990. [17](#)
- [22] D.H. Turner and D.H. Mathews. NNDB: The nearest neighbor parameter database for predicting stability of nucleic acid secondary structure. *Nucleic Acids Research*, 38(suppl 1):D280–D282, 2010. [417](#)
- [23] Stefan Washietl, Ivo L. Hofacker, Peter F. Stadler, and Manolis Kellis. RNA folding with soft constraints: reconciliation of probing data and thermodynamics secondary structure prediction. *Nucleic Acids Research*, 40(10):4261–4272, 2012. [346](#)
- [24] S. Wuchty, W. Fontana, I. L. Hofacker, and P. Schuster. Complete suboptimal folding of RNA and the stability of secondary structures. *Biopolymers*, 49(2):145–165, February 1999. [291](#), [292](#)
- [25] Kourosh Zarringhalam, Michelle M. Meyer, Ivan Dotu, Jeffrey H. Chuang, and Peter Clote. Integrating chemical footprinting data into RNA secondary structure prediction. *PLOS ONE*, 7(10), 2012. [341](#)
- [26] M. Zuker. On finding all suboptimal foldings of an RNA molecule. *Science*, 244(4900):48–52, April 1989. [288](#)
- [27] M. Zuker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic acids research*, 9(1):133–148, 1981. [24](#)

# Index

- (Abstract) Data Structures, [544](#)
  - bondT, [549](#)
  - cpair, [548](#)
  - PAIR, [548](#)
  - plist, [548](#)
  - sect, [549](#)
  - vrna\_C11\_features, [549](#)
- (Nucleic Acid Sequence) String Utilities, [437](#)
  - FILENAME\_ID\_LENGTH, [438](#)
  - FILENAME\_MAX\_LENGTH, [438](#)
  - vrna\_cut\_point\_insert, [444](#)
  - vrna\_cut\_point\_remove, [444](#)
  - vrna\_hamming\_distance, [442](#)
  - vrna\_hamming\_distance\_bound, [442](#)
  - vrna\_random\_string, [441](#)
  - vrna\_seq\_toRNA, [443](#)
  - vrna\_seq\_toupper, [443](#)
  - vrna\_seq\_ungapped, [444](#)
  - vrna\_strcat\_printf, [439](#)
  - vrna\_strcat\_vprintf, [440](#)
  - vrna\_strdup\_printf, [438](#)
  - vrna\_strdup\_vprintf, [439](#)
  - vrna\_strsplit, [441](#)
- \_struct\_en, [601](#)
- 2Dpfold.h
  - destroy\_TwoDpfold\_variables, [609](#)
  - get\_TwoDpfold\_variables, [609](#)
  - TwoDpfold\_pbacktrack, [610](#)
  - TwoDpfold\_pbacktrack5, [611](#)
  - TwoDpfoldList, [610](#)
- add\_root
  - Deprecated Interface for Secondary Structure Utilities, [473](#)
- aliPS\_color\_aln
  - Plotting, [525](#)
- alifold
  - Deprecated Interface for Global MFE Prediction, [377](#)
- alifold.h
  - cv\_fact, [614](#)
  - energy\_of\_alistruct, [613](#)
  - nc\_fact, [614](#)
  - update\_alifold\_params, [614](#)
- alimake\_pair\_table
  - Deprecated Interface for Secondary Structure Utilities, [477](#)
- alipbacktrack
  - Deprecated Interface for Global Partition Function Computation, [405](#)
- alipf\_circ\_fold
  - Deprecated Interface for Global Partition Function Computation, [404](#)
- alipf\_fold
  - Deprecated Interface for Global Partition Function Computation, [403](#)
- alipf\_fold\_par
  - Deprecated Interface for Global Partition Function Computation, [391](#)
- alloc\_sequence\_arrays
  - Deprecated Interface for Multiple Sequence Alignment Utilities, [493](#)
- alpha
  - vrna\_exp\_param\_s, [180](#)
- Annotation, [533](#)
- assign\_plist\_from\_db
  - Deprecated Interface for Global Partition Function Computation, [402](#)
- assign\_plist\_from\_pr
  - Deprecated Interface for Global Partition Function Computation, [403](#)
- auxdata
  - vrna\_fc\_s, [565](#)
- b2HIT
  - Deprecated Interface for Secondary Structure Utilities, [471](#)
- b2Shapiro
  - Deprecated Interface for Secondary Structure Utilities, [472](#)
- b2C
  - Deprecated Interface for Secondary Structure Utilities, [472](#)
- BONUS
  - constants.h, [680](#)
- backtrack\_GQuad\_IntLoop
  - G-Quadruplexes, [354](#)
- backtrack\_GQuad\_IntLoop\_L
  - G-Quadruplexes, [355](#)
- backtrack\_type
  - Fine-tuning of the Implemented Models, [177](#)
- Backtracking MFE structures, [263](#)
  - vrna\_BT\_hp\_loop, [263](#)
  - vrna\_BT\_mb\_loop, [263](#)
- base\_pair
  - fold\_vars.h, [650](#)
- bondT
  - (Abstract) Data Structures, [549](#)
- bp\_distance

- Deprecated Interface for Secondary Structure Utilities, [478](#)
- bppm\_symbol
  - Deprecated Interface for Secondary Structure Utilities, [480](#)
- bppm\_to\_structure
  - Deprecated Interface for Secondary Structure Utilities, [480](#)
- bt
  - vrna\_sc\_s, [236](#)
- Buffers, [594](#)
  - vrna\_callback\_stream\_output, [595](#)
  - vrna\_cstr, [595](#)
  - vrna\_cstr\_close, [596](#)
  - vrna\_cstr\_fflush, [596](#)
  - vrna\_cstr\_free, [596](#)
  - vrna\_ostream\_free, [597](#)
  - vrna\_ostream\_init, [597](#)
  - vrna\_ostream\_provide, [598](#)
  - vrna\_ostream\_request, [598](#)
- COORDINATE, [522](#)
- centroid
  - part\_func.h, [684](#)
- centroid.h
  - get\_centroid\_struct\_pl, [617](#)
  - get\_centroid\_struct\_pr, [617](#)
- circularfold
  - Deprecated Interface for Global MFE Prediction, [387](#)
- circfold
  - Deprecated Interface for Global MFE Prediction, [383](#)
- Classified Dynamic Programming Variants, [305](#)
- co\_pf\_fold
  - Deprecated Interface for Global Partition Function Computation, [398](#)
- co\_pf\_fold\_par
  - Deprecated Interface for Global Partition Function Computation, [399](#)
- cofold
  - Deprecated Interface for Global MFE Prediction, [378](#)
- cofold\_par
  - Deprecated Interface for Global MFE Prediction, [378](#)
- Combinatorics Algorithms, [538](#)
  - vrna\_enumerate\_necklaces, [538](#)
  - vrna\_rotational\_symmetry, [540](#)
  - vrna\_rotational\_symmetry\_db, [542](#)
  - vrna\_rotational\_symmetry\_db\_pos, [542](#)
  - vrna\_rotational\_symmetry\_num, [539](#)
  - vrna\_rotational\_symmetry\_pos, [541](#)
  - vrna\_rotational\_symmetry\_pos\_num, [539](#)
- Command Files, [516](#)
  - VRNA\_CMD\_PARSE\_DEFAULTS, [518](#)
  - VRNA\_CMD\_PARSE\_HC, [517](#)
  - VRNA\_CMD\_PARSE\_SC, [517](#)
  - VRNA\_CMD\_PARSE\_SD, [517](#)
  - VRNA\_CMD\_PARSE\_UD, [517](#)
  - vrna\_commands\_apply, [519](#)
  - vrna\_commands\_free, [520](#)
  - vrna\_file\_commands\_apply, [519](#)
  - vrna\_file\_commands\_read, [518](#)
- Complex Structured Modules, [352](#)
- Compute the Centroid Structure, [301](#)
  - vrna\_centroid, [301](#)
  - vrna\_centroid\_from\_plist, [301](#)
  - vrna\_centroid\_from\_probs, [302](#)
- Compute the Density of States, [321](#)
  - density\_of\_states, [321](#)
- Compute the Structure with Maximum Expected Accuracy (MEA), [300](#)
  - MEA, [300](#)
- compute\_BPdifferences
  - Deprecated Interface for Secondary Structure Utilities, [479](#)
- compute\_probabilities
  - Deprecated Interface for Global Partition Function Computation, [400](#)
- Computing MFE representatives of a Distance Based Partitioning, [307](#)
  - destroy\_TwoDfold\_variables, [312](#)
  - get\_TwoDfold\_variables, [311](#)
  - TwoDfold\_backtrack\_f5, [313](#)
  - TwoDfold\_vars, [309](#)
  - TwoDfoldList, [312](#)
  - vrna\_backtrack5\_TwoD, [311](#)
  - vrna\_mfe\_TwoD, [310](#)
  - vrna\_sol\_TwoD\_t, [309](#)
- Computing Partition Functions of a Distance Based Partitioning, [315](#)
  - vrna\_pf\_TwoD, [316](#)
  - vrna\_sol\_TwoD\_pf\_t, [316](#)
- concentrations.h
  - get\_concentrations, [621](#)
- cons\_seq
  - vrna\_fc\_s, [568](#)
- constants.h
  - BONUS, [680](#)
  - FORBIDDEN, [680](#)
  - GASCONST, [680](#)
  - INF, [680](#)
  - K0, [680](#)
  - MAXLOOP, [681](#)
  - NBPAIRS, [681](#)
  - TURN, [681](#)
- constrain, [547](#)
- constrain\_ptypes
  - hard.h, [627](#)
- Constraining the RNA Folding Grammar, [206](#)
  - VRNA\_CONSTRAINT\_FILE, [209](#)
  - VRNA\_CONSTRAINT\_SOFT\_MFE, [209](#)
  - VRNA\_CONSTRAINT\_SOFT\_PF, [209](#)
  - VRNA\_DECOMP\_EXT\_EXT\_EXT, [217](#)
  - VRNA\_DECOMP\_EXT\_EXT\_STEM1, [218](#)
  - VRNA\_DECOMP\_EXT\_EXT\_STEM, [218](#)



- VRNA\_DECOMP\_EXT\_EXT, 215
- VRNA\_DECOMP\_EXT\_STEM\_EXT, 217
- VRNA\_DECOMP\_EXT\_STEM, 216
- VRNA\_DECOMP\_EXT\_UP, 216
- VRNA\_DECOMP\_ML\_COAXIAL\_ENC, 215
- VRNA\_DECOMP\_ML\_COAXIAL, 214
- VRNA\_DECOMP\_ML\_ML\_ML, 212
- VRNA\_DECOMP\_ML\_ML\_STEM, 214
- VRNA\_DECOMP\_ML\_ML, 213
- VRNA\_DECOMP\_ML\_STEM, 212
- VRNA\_DECOMP\_ML\_UP, 213
- VRNA\_DECOMP\_PAIR\_HP, 210
- VRNA\_DECOMP\_PAIR\_IL, 210
- VRNA\_DECOMP\_PAIR\_ML, 211
- vrna\_constraints\_add, 219
- vrna\_message\_constraint\_options, 220
- vrna\_message\_constraint\_options\_all, 221
- convert\_parameter\_file
  - Converting Energy Parameter Files, 424
- Converting Energy Parameter Files, 419
  - convert\_parameter\_file, 424
  - VRNA\_CONVERT\_OUTPUT\_ALL, 420
  - VRNA\_CONVERT\_OUTPUT\_BULGE, 422
  - VRNA\_CONVERT\_OUTPUT\_DANGLE3, 421
  - VRNA\_CONVERT\_OUTPUT\_DANGLE5, 421
  - VRNA\_CONVERT\_OUTPUT\_DUMP, 423
  - VRNA\_CONVERT\_OUTPUT\_HP, 420
  - VRNA\_CONVERT\_OUTPUT\_INT\_11, 421
  - VRNA\_CONVERT\_OUTPUT\_INT\_21, 422
  - VRNA\_CONVERT\_OUTPUT\_INT\_22, 422
  - VRNA\_CONVERT\_OUTPUT\_INT, 422
  - VRNA\_CONVERT\_OUTPUT\_MISC, 422
  - VRNA\_CONVERT\_OUTPUT\_MM\_EXT, 421
  - VRNA\_CONVERT\_OUTPUT\_MM\_HP, 420
  - VRNA\_CONVERT\_OUTPUT\_MM\_INT\_1N, 420
  - VRNA\_CONVERT\_OUTPUT\_MM\_INT\_23, 421
  - VRNA\_CONVERT\_OUTPUT\_MM\_INT, 420
  - VRNA\_CONVERT\_OUTPUT\_MM\_MULTI, 421
  - VRNA\_CONVERT\_OUTPUT\_ML, 422
  - VRNA\_CONVERT\_OUTPUT\_NINIO, 423
  - VRNA\_CONVERT\_OUTPUT\_SPECIAL\_HP, 423
  - VRNA\_CONVERT\_OUTPUT\_STACK, 420
  - VRNA\_CONVERT\_OUTPUT\_VANILLA, 423
- copy\_pair\_table
  - Deprecated Interface for Secondary Structure Utilities, 477
- cost\_matrix
  - dist\_vars.h, 635
- cpair
  - (Abstract) Data Structures, 548
- cut\_point
  - fold\_vars.h, 650
- cv\_fact
  - alifold.h, 614
- dangles
  - Fine-tuning of the Implemented Models, 175
  - vrna\_md\_s, 148
- density\_of\_states
  - Compute the Density of States, 321
- Deprecated Interface for Free Energy Evaluation, 126
  - E\_IntLoop, 137
  - E\_Stem, 135
  - energy\_of\_circ\_struct, 134
  - energy\_of\_circ\_struct\_par, 129
  - energy\_of\_circ\_structure, 128
  - energy\_of\_move, 131
  - energy\_of\_move\_pt, 132
  - energy\_of\_struct, 133
  - energy\_of\_struct\_par, 127
  - energy\_of\_struct\_pt, 134
  - energy\_of\_struct\_pt\_par, 130
  - energy\_of\_structure, 127
  - energy\_of\_structure\_pt, 129
  - exp\_E\_ExtLoop, 136
  - exp\_E\_IntLoop, 139
  - exp\_E\_Stem, 136
  - loop\_energy, 132
- Deprecated Interface for Global MFE Prediction, 376
  - alifold, 377
  - circularfold, 387
  - circfold, 383
  - cofold, 378
  - cofold\_par, 378
  - export\_circfold\_arrays, 386
  - export\_circfold\_arrays\_par, 386
  - export\_cofold\_arrays, 380
  - export\_cofold\_arrays\_gq, 379
  - export\_fold\_arrays, 385
  - export\_fold\_arrays\_par, 385
  - fold, 383
  - fold\_par, 382
  - free\_alifold\_arrays, 388
  - free\_arrays, 384
  - free\_co\_arrays, 378
  - get\_monomere\_mfes, 381
  - HairpinE, 387
  - initialize\_cofold, 382
  - initialize\_fold, 387
  - LoopEnergy, 386
  - update\_cofold\_params, 379
  - update\_cofold\_params\_par, 379
  - update\_fold\_params, 384
  - update\_fold\_params\_par, 385
- Deprecated Interface for Global Partition Function Computation, 390
  - alipbacktrack, 405
  - alipf\_circ\_fold, 404
  - alipf\_fold, 403
  - alipf\_fold\_par, 391
  - assign\_plist\_from\_db, 402
  - assign\_plist\_from\_pr, 403
  - co\_pf\_fold, 398
  - co\_pf\_fold\_par, 399
  - compute\_probabilities, 400
  - export\_ali\_bppm, 404
  - export\_bppm, 396

- export\_co\_bppm, 401
- free\_alipf\_arrays, 405
- free\_co\_pf\_arrays, 401
- free\_pf\_arrays, 395
- get\_alipf\_arrays, 406
- get\_pf\_arrays, 396
- init\_co\_pf\_fold, 400
- init\_pf\_fold, 398
- mean\_bp\_distance, 397
- mean\_bp\_distance\_pr, 397
- pf\_circ\_fold, 394
- pf\_fold, 393
- pf\_fold\_par, 392
- stackProb, 398
- update\_co\_pf\_params, 401
- update\_co\_pf\_params\_par, 402
- update\_pf\_params, 395
- update\_pf\_params\_par, 395
- Deprecated Interface for Local (Sliding Window) MF↔  
E Prediction, 389
- Lfold, 389
- Lfoldz, 389
- Deprecated Interface for Local (Sliding Window) Partition  
Function Computation, 408
- pfl\_fold, 408
- putoutpU\_prob, 409
- putoutpU\_prob\_bin, 410
- update\_pf\_paramsLP, 408
- Deprecated Interface for Multiple Sequence Alignment  
Utilities, 491
- alloc\_sequence\_arrays, 493
- encode\_ali\_sequence, 492
- free\_sequence\_arrays, 493
- get\_mpi, 491
- pair\_info, 491
- Deprecated Interface for Secondary Structure Utilities,  
470
- add\_root, 473
- alimake\_pair\_table, 477
- b2HIT, 471
- b2Shapiro, 472
- b2C, 472
- bp\_distance, 478
- bppm\_symbol, 480
- bppm\_to\_structure, 480
- compute\_BPdifferences, 479
- copy\_pair\_table, 477
- expand\_Full, 473
- expand\_Shapiro, 473
- make\_pair\_table, 476
- make\_pair\_table\_snoop, 477
- make\_referenceBP\_array, 478
- pack\_structure, 475
- parenthesis\_structure, 479
- parenthesis\_zuker, 479
- parse\_structure, 475
- unexpand\_Full, 474
- unexpand\_aligned\_F, 474
- unpack\_structure, 476
- unweight, 474
- destroy\_TwoDfold\_variables  
Computing MFE representatives of a Distance  
Based Partitioning, 312
- destroy\_TwoDpfold\_variables  
2Dpfold.h, 609
- Direct Refolding Paths between two Secondary Struc-  
tures, 425
- find\_saddle, 429
- free\_path, 430
- get\_path, 430
- path\_t, 426
- vrna\_path\_findpath, 428
- vrna\_path\_findpath\_saddle, 426
- vrna\_path\_findpath\_saddle\_ub, 427
- vrna\_path\_findpath\_ub, 428
- dist\_vars.h  
cost\_matrix, 635
- edit\_backtrack, 635
- Distance Based Partitioning of the Secondary Structure  
Space, 306
- do\_backtrack  
Fine-tuning of the Implemented Models, 176
- Dot-Bracket Notation of Secondary Structures, 450
- VRNA\_BRACKETS\_ALPHA, 450
- VRNA\_BRACKETS\_ANG, 451
- VRNA\_BRACKETS\_CLY, 451
- VRNA\_BRACKETS\_DEFAULT, 452
- VRNA\_BRACKETS\_RND, 451
- VRNA\_BRACKETS\_SQR, 451
- vrna\_db\_flatten, 453
- vrna\_db\_flatten\_to, 454
- vrna\_db\_from\_WUSS, 455
- vrna\_db\_from\_plist, 455
- vrna\_db\_from\_ptable, 454
- vrna\_db\_pack, 452
- vrna\_db\_to\_element\_string, 456
- vrna\_db\_unpack, 453
- dupVar, 548
- duplexT, 547
- E\_Hairpin  
Hairpin Loops, 370
- E\_IntLoop  
Deprecated Interface for Free Energy Evaluation,  
137
- E\_Stem  
Deprecated Interface for Free Energy Evaluation,  
135
- edit\_backtrack  
dist\_vars.h, 635
- encode\_ali\_sequence  
Deprecated Interface for Multiple Sequence Align-  
ment Utilities, 492
- energy  
vrna\_ht\_entry\_db\_t, 584
- Energy Evaluation for Atomic Moves, 124
- vrna\_eval\_move, 124

- vrna\_eval\_move\_pt, [125](#)
- Energy Evaluation for Individual Loops, [121](#)
  - vrna\_eval\_loop\_pt, [122](#)
  - vrna\_eval\_loop\_pt\_v, [122](#)
- Energy Parameters, [178](#)
  - get\_boltzmann\_factor\_copy, [188](#)
  - get\_boltzmann\_factors, [187](#)
  - get\_boltzmann\_factors\_ali, [189](#)
  - get\_scaled\_alipf\_parameters, [189](#)
  - get\_scaled\_parameters, [190](#)
  - get\_scaled\_pf\_parameters, [187](#)
  - paramT, [181](#)
  - pf\_paramT, [181](#)
  - scale\_parameters, [189](#)
  - vrna\_exp\_params, [182](#)
  - vrna\_exp\_params\_comparative, [183](#)
  - vrna\_exp\_params\_copy, [183](#)
  - vrna\_exp\_params\_rescale, [185](#)
  - vrna\_exp\_params\_reset, [186](#)
  - vrna\_exp\_params\_subst, [184](#)
  - vrna\_params, [181](#)
  - vrna\_params\_copy, [182](#)
  - vrna\_params\_reset, [186](#)
  - vrna\_params\_subst, [184](#)
- energy\_of\_alistruct
  - alifold.h, [613](#)
- energy\_of\_circ\_struct
  - Deprecated Interface for Free Energy Evaluation, [134](#)
- energy\_of\_circ\_struct\_par
  - Deprecated Interface for Free Energy Evaluation, [129](#)
- energy\_of\_circ\_structure
  - Deprecated Interface for Free Energy Evaluation, [128](#)
- energy\_of\_move
  - Deprecated Interface for Free Energy Evaluation, [131](#)
- energy\_of\_move\_pt
  - Deprecated Interface for Free Energy Evaluation, [132](#)
- energy\_of\_struct
  - Deprecated Interface for Free Energy Evaluation, [133](#)
- energy\_of\_struct\_par
  - Deprecated Interface for Free Energy Evaluation, [127](#)
- energy\_of\_struct\_pt
  - Deprecated Interface for Free Energy Evaluation, [134](#)
- energy\_of\_struct\_pt\_par
  - Deprecated Interface for Free Energy Evaluation, [130](#)
- energy\_of\_structure
  - Deprecated Interface for Free Energy Evaluation, [127](#)
- energy\_of\_structure\_pt
  - Deprecated Interface for Free Energy Evaluation, [129](#)
- energy\_set
  - Fine-tuning of the Implemented Models, [176](#)
- exp\_E\_ExtLoop
  - Deprecated Interface for Free Energy Evaluation, [136](#)
- exp\_E\_Hairpin
  - Hairpin Loops, [371](#)
- exp\_E\_IntLoop
  - Deprecated Interface for Free Energy Evaluation, [139](#)
- exp\_E\_Stem
  - Deprecated Interface for Free Energy Evaluation, [136](#)
- exp\_f
  - vrna\_sc\_s, [236](#)
- expHairpinEnergy
  - part\_func.h, [685](#)
- expLoopEnergy
  - part\_func.h, [685](#)
- expand\_Full
  - Deprecated Interface for Secondary Structure Utilities, [473](#)
- expand\_Shapiro
  - Deprecated Interface for Secondary Structure Utilities, [473](#)
- Experimental Structure Probing Data, [338](#)
- export\_ali\_bppm
  - Deprecated Interface for Global Partition Function Computation, [404](#)
- export\_bppm
  - Deprecated Interface for Global Partition Function Computation, [396](#)
- export\_circfold\_arrays
  - Deprecated Interface for Global MFE Prediction, [386](#)
- export\_circfold\_arrays\_par
  - Deprecated Interface for Global MFE Prediction, [386](#)
- export\_co\_bppm
  - Deprecated Interface for Global Partition Function Computation, [401](#)
- export\_cofold\_arrays
  - Deprecated Interface for Global MFE Prediction, [380](#)
- export\_cofold\_arrays\_gq
  - Deprecated Interface for Global MFE Prediction, [379](#)
- export\_fold\_arrays
  - Deprecated Interface for Global MFE Prediction, [385](#)
- export\_fold\_arrays\_par
  - Deprecated Interface for Global MFE Prediction, [385](#)
- Extending the Folding Grammar with Additional Domains, [192](#)
- Exterior Loops, [364](#)
  - vrna\_E\_ext\_loop, [365](#)

- vrna\_E\_ext\_stem, [365](#)
- vrna\_exp\_E\_ext\_stem, [366](#)
- vrna\_mx\_pf\_aux\_el\_t, [364](#)
- f
  - vrna\_sc\_s, [236](#)
- FILENAME\_ID\_LENGTH
  - (Nucleic Acid Sequence) String Utilites, [438](#)
- FILENAME\_MAX\_LENGTH
  - (Nucleic Acid Sequence) String Utilites, [438](#)
- FORBIDDEN
  - constants.h, [680](#)
- filecopy
  - utils/basic.h, [677](#)
- Files and I/O, [495](#)
  - vrna\_file\_exists, [497](#)
  - vrna\_filename\_sanitize, [496](#)
  - vrna\_read\_line, [496](#)
- final\_cost
  - Inverse Folding (Design), [323](#)
- find\_saddle
  - Direct Refolding Paths between two Secondary Structures, [429](#)
- Fine-tuning of the Implemented Models, [142](#)
  - backtrack\_type, [177](#)
  - dangles, [175](#)
  - do\_backtrack, [176](#)
  - energy\_set, [176](#)
  - max\_bp\_span, [177](#)
  - noLonelyPairs, [176](#)
  - nonstandards, [177](#)
  - pf\_scale, [175](#)
  - set\_model\_details, [174](#)
  - temperature, [175](#)
  - tetra\_loop, [176](#)
  - VRNA\_MODEL\_DEFAULT\_ALI\_CV\_FACT, [154](#)
  - VRNA\_MODEL\_DEFAULT\_ALI\_NC\_FACT, [154](#)
  - VRNA\_MODEL\_DEFAULT\_ALI\_OLD\_EN, [154](#)
  - VRNA\_MODEL\_DEFAULT\_ALI\_RIBO, [154](#)
  - VRNA\_MODEL\_DEFAULT\_BACKTRACK\_TYPE, [152](#)
  - VRNA\_MODEL\_DEFAULT\_BACKTRACK, [152](#)
  - VRNA\_MODEL\_DEFAULT\_BETA\_SCALE, [150](#)
  - VRNA\_MODEL\_DEFAULT\_CIRC, [151](#)
  - VRNA\_MODEL\_DEFAULT\_COMPUTE\_BPP, [153](#)
  - VRNA\_MODEL\_DEFAULT\_DANGLES, [150](#)
  - VRNA\_MODEL\_DEFAULT\_ENERGY\_SET, [152](#)
  - VRNA\_MODEL\_DEFAULT\_GQUAD, [151](#)
  - VRNA\_MODEL\_DEFAULT\_LOG\_ML, [153](#)
  - VRNA\_MODEL\_DEFAULT\_MAX\_BP\_SPAN, [153](#)
  - VRNA\_MODEL\_DEFAULT\_NO\_GU\_CLOSURE, [151](#)
  - VRNA\_MODEL\_DEFAULT\_NO\_GU, [151](#)
  - VRNA\_MODEL\_DEFAULT\_NO\_LP, [150](#)
  - VRNA\_MODEL\_DEFAULT\_PF\_SCALE, [149](#)
  - VRNA\_MODEL\_DEFAULT\_SPECIAL\_HP, [150](#)
  - VRNA\_MODEL\_DEFAULT\_TEMPERATURE, [149](#)
  - VRNA\_MODEL\_DEFAULT\_UNIQ\_ML, [152](#)
  - VRNA\_MODEL\_DEFAULT\_WINDOW\_SIZE, [153](#)
  - vrna\_md\_copy, [155](#)
  - vrna\_md\_defaults\_backtrack, [166](#)
  - vrna\_md\_defaults\_backtrack\_get, [166](#)
  - vrna\_md\_defaults\_backtrack\_type, [167](#)
  - vrna\_md\_defaults\_backtrack\_type\_get, [167](#)
  - vrna\_md\_defaults\_betaScale, [158](#)
  - vrna\_md\_defaults\_betaScale\_get, [158](#)
  - vrna\_md\_defaults\_circ, [163](#)
  - vrna\_md\_defaults\_circ\_get, [163](#)
  - vrna\_md\_defaults\_compute\_bpp, [167](#)
  - vrna\_md\_defaults\_compute\_bpp\_get, [168](#)
  - vrna\_md\_defaults\_cv\_fact, [172](#)
  - vrna\_md\_defaults\_cv\_fact\_get, [172](#)
  - vrna\_md\_defaults\_dangles, [158](#)
  - vrna\_md\_defaults\_dangles\_get, [159](#)
  - vrna\_md\_defaults\_energy\_set, [165](#)
  - vrna\_md\_defaults\_energy\_set\_get, [166](#)
  - vrna\_md\_defaults\_gquad, [164](#)
  - vrna\_md\_defaults\_gquad\_get, [164](#)
  - vrna\_md\_defaults\_logML\_get, [163](#)
  - vrna\_md\_defaults\_logML, [162](#)
  - vrna\_md\_defaults\_max\_bp\_span, [168](#)
  - vrna\_md\_defaults\_max\_bp\_span\_get, [169](#)
  - vrna\_md\_defaults\_min\_loop\_size, [169](#)
  - vrna\_md\_defaults\_min\_loop\_size\_get, [169](#)
  - vrna\_md\_defaults\_nc\_fact, [173](#)
  - vrna\_md\_defaults\_nc\_fact\_get, [173](#)
  - vrna\_md\_defaults\_noGU\_get, [161](#)
  - vrna\_md\_defaults\_noGUclosure, [161](#)
  - vrna\_md\_defaults\_noGUclosure\_get, [162](#)
  - vrna\_md\_defaults\_noGU, [161](#)
  - vrna\_md\_defaults\_noLP\_get, [160](#)
  - vrna\_md\_defaults\_noLP, [160](#)
  - vrna\_md\_defaults\_oldAliEn, [170](#)
  - vrna\_md\_defaults\_oldAliEn\_get, [171](#)
  - vrna\_md\_defaults\_reset, [156](#)
  - vrna\_md\_defaults\_ribo, [171](#)
  - vrna\_md\_defaults\_ribo\_get, [172](#)
  - vrna\_md\_defaults\_sfact, [173](#)
  - vrna\_md\_defaults\_sfact\_get, [174](#)
  - vrna\_md\_defaults\_special\_hp, [159](#)
  - vrna\_md\_defaults\_special\_hp\_get, [160](#)
  - vrna\_md\_defaults\_temperature, [157](#)
  - vrna\_md\_defaults\_temperature\_get, [157](#)
  - vrna\_md\_defaults\_uniq\_ML\_get, [165](#)
  - vrna\_md\_defaults\_uniq\_ML, [164](#)
  - vrna\_md\_defaults\_window\_size, [170](#)
  - vrna\_md\_defaults\_window\_size\_get, [170](#)
  - vrna\_md\_option\_string, [156](#)
  - vrna\_md\_set\_default, [155](#)
  - vrna\_md\_update, [155](#)
- fold
  - Deprecated Interface for Global MFE Prediction, [383](#)
- fold\_par
  - Deprecated Interface for Global MFE Prediction, [382](#)
- fold\_vars.h

- base\_pair, [650](#)
- cut\_point, [650](#)
- iindx, [650](#)
- james\_rule, [649](#)
- logML, [650](#)
- pr, [650](#)
- RibosumFile, [649](#)
- Free Energy Evaluation, [95](#)
  - vrna\_eval\_circ\_consensus\_structure, [110](#)
  - vrna\_eval\_circ\_consensus\_structure\_v, [114](#)
  - vrna\_eval\_circ\_gquad\_consensus\_structure, [112](#)
  - vrna\_eval\_circ\_gquad\_consensus\_structure\_v, [116](#)
  - vrna\_eval\_circ\_gquad\_structure, [105](#)
  - vrna\_eval\_circ\_gquad\_structure\_v, [109](#)
  - vrna\_eval\_circ\_structure, [104](#)
  - vrna\_eval\_circ\_structure\_v, [107](#)
  - vrna\_eval\_consensus\_structure\_pt\_simple, [119](#)
  - vrna\_eval\_consensus\_structure\_simple, [109](#)
  - vrna\_eval\_consensus\_structure\_simple\_v, [113](#)
  - vrna\_eval\_consensus\_structure\_simple\_verbose, [113](#)
  - vrna\_eval\_covar\_structure, [99](#)
  - vrna\_eval\_gquad\_consensus\_structure, [111](#)
  - vrna\_eval\_gquad\_consensus\_structure\_v, [115](#)
  - vrna\_eval\_gquad\_structure, [104](#)
  - vrna\_eval\_gquad\_structure\_v, [108](#)
  - vrna\_eval\_structure, [98](#)
  - vrna\_eval\_structure\_pt, [101](#)
  - vrna\_eval\_structure\_pt\_simple, [117](#)
  - vrna\_eval\_structure\_pt\_simple\_v, [118](#)
  - vrna\_eval\_structure\_pt\_simple\_verbose, [118](#)
  - vrna\_eval\_structure\_pt\_v, [102](#)
  - vrna\_eval\_structure\_pt\_verbose, [102](#)
  - vrna\_eval\_structure\_simple, [103](#)
  - vrna\_eval\_structure\_simple\_v, [106](#)
  - vrna\_eval\_structure\_simple\_verbose, [106](#)
  - vrna\_eval\_structure\_v, [100](#)
  - vrna\_eval\_structure\_verbose, [100](#)
- free\_alifold\_arrays
  - Deprecated Interface for Global MFE Prediction, [388](#)
- free\_alipf\_arrays
  - Deprecated Interface for Global Partition Function Computation, [405](#)
- free\_arrays
  - Deprecated Interface for Global MFE Prediction, [384](#)
- free\_auxdata
  - vrna\_fc\_s, [565](#)
- free\_co\_arrays
  - Deprecated Interface for Global MFE Prediction, [378](#)
- free\_co\_pf\_arrays
  - Deprecated Interface for Global Partition Function Computation, [401](#)
- free\_data
  - vrna\_hc\_s, [225](#)
- free\_path
  - Direct Refolding Paths between two Secondary Structures, [430](#)
- free\_pf\_arrays
  - Deprecated Interface for Global Partition Function Computation, [395](#)
- free\_profile
  - profiledist.h, [700](#)
- free\_sequence\_arrays
  - Deprecated Interface for Multiple Sequence Alignment Utilities, [493](#)
- free\_tree
  - treedist.h, [711](#)
- G-Quadruplexes, [353](#)
  - backtrack\_GQuad\_IntLoop, [354](#)
  - backtrack\_GQuad\_IntLoop\_L, [355](#)
  - get\_gquad\_matrix, [353](#)
  - parse\_gquad, [354](#)
- GASCONST
  - constants.h, [680](#)
- Generate Soft Constraints from Data, [343](#)
  - progress\_callback, [345](#)
  - VRNA\_MINIMIZER\_CONJUGATE\_FR, [344](#)
  - VRNA\_MINIMIZER\_CONJUGATE\_PR, [344](#)
  - VRNA\_MINIMIZER\_STEEPEST\_DESCENT, [345](#)
  - VRNA\_MINIMIZER\_VECTOR\_BFGS2, [345](#)
  - VRNA\_MINIMIZER\_VECTOR\_BFGS, [344](#)
  - VRNA\_OBJECTIVE\_FUNCTION\_ABSOLUTE, [344](#)
  - VRNA\_OBJECTIVE\_FUNCTION\_QUADRATIC, [344](#)
  - vrna\_sc\_minimize\_perturbation, [346](#)
- get\_TwoDfold\_variables
  - Computing MFE representatives of a Distance Based Partitioning, [311](#)
- get\_TwoDpfold\_variables
  - 2Dpfold.h, [609](#)
- get\_alipf\_arrays
  - Deprecated Interface for Global Partition Function Computation, [406](#)
- get\_boltzmann\_factor\_copy
  - Energy Parameters, [188](#)
- get\_boltzmann\_factors
  - Energy Parameters, [187](#)
- get\_boltzmann\_factors\_al
  - Energy Parameters, [189](#)
- get\_centroid\_struct\_gquad\_pr
  - part\_func.h, [684](#)
- get\_centroid\_struct\_pl
  - centroid.h, [617](#)
- get\_centroid\_struct\_pr
  - centroid.h, [617](#)
- get\_concentrations
  - concentrations.h, [621](#)
- get\_gquad\_matrix
  - G-Quadruplexes, [353](#)
- get\_input\_line
  - Utilities, [361](#)

- get\_line
  - utils/basic.h, [674](#)
- get\_monomere\_mfes
  - Deprecated Interface for Global MFE Prediction, [381](#)
- get\_mpi
  - Deprecated Interface for Multiple Sequence Alignment Utilities, [491](#)
- get\_path
  - Direct Refolding Paths between two Secondary Structures, [430](#)
- get\_pf\_arrays
  - Deprecated Interface for Global Partition Function Computation, [396](#)
- get\_plist
  - part\_func\_co.h, [686](#)
- get\_scaled\_alipf\_parameters
  - Energy Parameters, [189](#)
- get\_scaled\_parameters
  - Energy Parameters, [190](#)
- get\_scaled\_pf\_parameters
  - Energy Parameters, [187](#)
- give\_up
  - Inverse Folding (Design), [323](#)
- Global MFE Prediction, [251](#)
  - vrna\_alifold, [254](#)
  - vrna\_circalifold, [255](#)
  - vrna\_circfold, [254](#)
  - vrna\_cofold, [256](#)
  - vrna\_fold, [253](#)
  - vrna\_mfe, [252](#)
  - vrna\_mfe\_dimer, [252](#)
- Global Partition Function and Equilibrium Probabilities, [265](#)
  - vrna\_ensemble\_defect, [268](#)
  - vrna\_mean\_bp\_distance, [267](#)
  - vrna\_mean\_bp\_distance\_pr, [267](#)
  - vrna\_pf, [270](#)
  - vrna\_pf\_alifold, [273](#)
  - vrna\_pf\_circalifold, [274](#)
  - vrna\_pf\_circfold, [272](#)
  - vrna\_pf\_co\_fold, [275](#)
  - vrna\_pf\_dimer, [271](#)
  - vrna\_pf\_dimer\_probs, [269](#)
  - vrna\_pf\_fold, [271](#)
  - vrna\_plist\_from\_probs, [275](#)
  - vrna\_pr\_structure, [269](#)
  - vrna\_stack\_prob, [268](#)
- gmIRNA
  - Plotting, [528](#)
- Hairpin Loops, [368](#)
  - E\_Hairpin, [370](#)
  - exp\_E\_Hairpin, [371](#)
  - vrna\_E\_ext\_hp\_loop, [369](#)
  - vrna\_E\_hp\_loop, [368](#)
  - vrna\_eval\_hp\_loop, [369](#)
  - vrna\_exp\_E\_hp\_loop, [372](#)
- HairpinE
  - Deprecated Interface for Global MFE Prediction, [387](#)
- hamming
  - strings.h, [717](#)
- hamming\_bound
  - strings.h, [717](#)
- Hard Constraints, [222](#)
  - VRNA\_CONSTRAINT\_DB\_DEFAULT, [228](#)
  - VRNA\_CONSTRAINT\_DB\_DOT, [226](#)
  - VRNA\_CONSTRAINT\_DB\_ENFORCE\_BP, [225](#)
  - VRNA\_CONSTRAINT\_DB\_GQUAD, [228](#)
  - VRNA\_CONSTRAINT\_DB\_INTERMOL, [227](#)
  - VRNA\_CONSTRAINT\_DB\_INTRAMOL, [227](#)
  - VRNA\_CONSTRAINT\_DB\_PIPE, [226](#)
  - VRNA\_CONSTRAINT\_DB\_RND\_BRACK, [227](#)
  - VRNA\_CONSTRAINT\_DB\_WUSS, [228](#)
  - VRNA\_CONSTRAINT\_DB\_X, [226](#)
  - VRNA\_CONSTRAINT\_DB, [225](#)
  - vrna\_callback\_hc\_evaluate, [229](#)
  - vrna\_hc\_add\_bp, [231](#)
  - vrna\_hc\_add\_bp\_nonspecific, [232](#)
  - vrna\_hc\_add\_from\_db, [233](#)
  - vrna\_hc\_add\_up, [230](#)
  - vrna\_hc\_add\_up\_batch, [231](#)
  - vrna\_hc\_free, [232](#)
  - vrna\_hc\_init, [230](#)
- hard.h
  - constrain\_ptypes, [627](#)
  - print\_tty\_constraint, [626](#)
  - print\_tty\_constraint\_full, [627](#)
  - VRNA\_CONSTRAINT\_DB\_ANG\_BRACK, [625](#)
  - VRNA\_CONSTRAINT\_NO\_HEADER, [625](#)
  - vrna\_hc\_add\_data, [626](#)
  - vrna\_hc\_type\_e, [626](#)
- Hash Tables, [583](#)
  - vrna\_callback\_ht\_compare\_entries, [585](#)
  - vrna\_callback\_ht\_free\_entry, [586](#)
  - vrna\_callback\_ht\_hash\_function, [585](#)
  - vrna\_hash\_table\_t, [584](#)
  - vrna\_ht\_clear, [590](#)
  - vrna\_ht\_collisions, [587](#)
  - vrna\_ht\_db\_comp, [591](#)
  - vrna\_ht\_db\_free\_entry, [592](#)
  - vrna\_ht\_db\_hash\_func, [592](#)
  - vrna\_ht\_free, [591](#)
  - vrna\_ht\_get, [589](#)
  - vrna\_ht\_init, [586](#)
  - vrna\_ht\_insert, [589](#)
  - vrna\_ht\_remove, [590](#)
  - vrna\_ht\_size, [587](#)
- Helix List Representation of Secondary Structures, [462](#)
  - vrna\_hx\_from\_ptable, [462](#)
- INF
  - constants.h, [680](#)
- id
  - vrna\_exp\_param\_s, [180](#)
- iidx
  - fold\_vars.h, [650](#)



- Incorporating Ligands Binding to Specific Sequence/↔
  - Structure Motifs using Soft Constraints, [350](#)
  - vrna\_sc\_add\_hi\_motif, [351](#)
- init\_co\_pf\_fold
  - Deprecated Interface for Global Partition Function Computation, [400](#)
- init\_pf\_fold
  - Deprecated Interface for Global Partition Function Computation, [398](#)
- init\_pf\_foldLP
  - LPfold.h, [658](#)
- init\_rand
  - utils/basic.h, [676](#)
- initialize\_cofold
  - Deprecated Interface for Global MFE Prediction, [382](#)
- initialize\_fold
  - Deprecated Interface for Global MFE Prediction, [387](#)
- int\_urn
  - utils/basic.h, [677](#)
- interact, [546](#)
- Internal Loops, [373](#)
  - vrna\_eval\_int\_loop, [373](#)
- inv\_verbose
  - Inverse Folding (Design), [324](#)
- Inverse Folding (Design), [322](#)
  - final\_cost, [323](#)
  - give\_up, [323](#)
  - inv\_verbose, [324](#)
  - inverse\_fold, [322](#)
  - inverse\_pf\_fold, [323](#)
- inverse\_fold
  - Inverse Folding (Design), [322](#)
- inverse\_pf\_fold
  - Inverse Folding (Design), [323](#)
- james\_rule
  - fold\_vars.h, [649](#)
- K0
  - constants.h, [680](#)
- LIST, [601](#)
- LPfold.h
  - init\_pf\_foldLP, [658](#)
- LST\_BUCKET, [601](#)
- last\_parameter\_file
  - Reading/Writing Energy Parameter Sets from/to File, [417](#)
- Lfold
  - Deprecated Interface for Local (Sliding Window) MFE Prediction, [389](#)
- Lfoldz
  - Deprecated Interface for Local (Sliding Window) MFE Prediction, [389](#)
- Ligands Binding to RNA Structures, [348](#)
- Ligands Binding to Unstructured Domains, [349](#)
- Local (sliding window) MFE Prediction, [258](#)
  - vrna\_Lfold, [261](#)
  - vrna\_Lfoldz, [262](#)
  - vrna\_mfe\_window, [260](#)
  - vrna\_mfe\_window\_callback, [259](#)
  - vrna\_mfe\_window\_zscore, [260](#)
- Local (sliding window) Partition Function and Equilibrium Probabilities, [277](#)
  - VRNA\_PROBS\_WINDOW\_BPP, [278](#)
  - VRNA\_PROBS\_WINDOW\_PF, [279](#)
  - VRNA\_PROBS\_WINDOW\_STACKP, [279](#)
  - VRNA\_PROBS\_WINDOW\_UP\_SPLIT, [279](#)
  - VRNA\_PROBS\_WINDOW\_UP, [278](#)
  - vrna\_pfl\_fold, [282](#)
  - vrna\_pfl\_fold\_cb, [283](#)
  - vrna\_pfl\_fold\_up, [283](#)
  - vrna\_pfl\_fold\_up\_cb, [284](#)
  - vrna\_probs\_window, [281](#)
  - vrna\_probs\_window\_callback, [280](#)
- logML
  - fold\_vars.h, [650](#)
- loop\_energy
  - Deprecated Interface for Free Energy Evaluation, [132](#)
- LoopEnergy
  - Deprecated Interface for Global MFE Prediction, [386](#)
- MAXLOOP
  - constants.h, [681](#)
- MEA
  - Compute the Structure with Maximum Expected Accuracy (MEA), [300](#)
- Make\_bp\_profile
  - profiledist.h, [700](#)
- Make\_bp\_profile\_bppm
  - profiledist.h, [699](#)
- make\_pair\_table
  - Deprecated Interface for Secondary Structure Utilities, [476](#)
- make\_pair\_table\_snoop
  - Deprecated Interface for Secondary Structure Utilities, [477](#)
- make\_referenceBP\_array
  - Deprecated Interface for Secondary Structure Utilities, [478](#)
- Make\_swString
  - stringdist.h, [706](#)
- make\_tree
  - treedist.h, [710](#)
- max\_bp\_span
  - Fine-tuning of the Implemented Models, [177](#)
- mean\_bp\_dist
  - part\_func.h, [685](#)
- mean\_bp\_distance
  - Deprecated Interface for Global Partition Function Computation, [397](#)
- mean\_bp\_distance\_pr
  - Deprecated Interface for Global Partition Function Computation, [397](#)

- Messages, 551
  - vrna\_message\_error, 551
  - vrna\_message\_info, 553
  - vrna\_message\_input\_seq, 554
  - vrna\_message\_input\_seq\_simple, 554
  - vrna\_message\_verror, 552
  - vrna\_message\_vinfo, 554
  - vrna\_message\_vwarning, 553
  - vrna\_message\_warning, 552
- min\_loop\_size
  - vrna\_md\_s, 149
- Minimum Free Energy (MFE) Algorithms, 248
- mm.h
  - vrna\_maximum\_matching, 661
  - vrna\_maximum\_matching\_simple, 661
- Multibranch Loops, 374
  - vrna\_E\_mb\_loop\_stack, 375
  - vrna\_mx\_pf\_aux\_ml\_t, 374
- Multiple Sequence Alignment Utilities, 481
  - VRNA\_MEASURE\_SHANNON\_ENTROPY, 483
  - vrna\_aln\_consensus\_mis, 490
  - vrna\_aln\_consensus\_sequence, 490
  - vrna\_aln\_conservation\_col, 489
  - vrna\_aln\_conservation\_struct, 488
  - vrna\_aln\_copy, 488
  - vrna\_aln\_free, 485
  - vrna\_aln\_mpi, 483
  - vrna\_aln\_pinfo, 483
  - vrna\_aln\_slice, 485
  - vrna\_aln\_toRNA, 487
  - vrna\_aln\_uppercase, 487
- Multiple Sequence Alignments, 506
  - VRNA\_FILE\_FORMAT\_MSA\_APPEND, 509
  - VRNA\_FILE\_FORMAT\_MSA\_CLUSTAL, 507
  - VRNA\_FILE\_FORMAT\_MSA\_DEFAULT, 508
  - VRNA\_FILE\_FORMAT\_MSA\_FASTA, 507
  - VRNA\_FILE\_FORMAT\_MSA\_MAF, 508
  - VRNA\_FILE\_FORMAT\_MSA\_MIS, 508
  - VRNA\_FILE\_FORMAT\_MSA\_NOCHECK, 509
  - VRNA\_FILE\_FORMAT\_MSA\_QUIET, 510
  - VRNA\_FILE\_FORMAT\_MSA\_SILENT, 510
  - VRNA\_FILE\_FORMAT\_MSA\_STOCKHOLM, 507
  - VRNA\_FILE\_FORMAT\_MSA\_UNKNOWN, 509
  - vrna\_file\_msa\_detect\_format, 513
  - vrna\_file\_msa\_read, 510
  - vrna\_file\_msa\_read\_record, 512
  - vrna\_file\_msa\_write, 514
- n\_seq
  - vrna\_fc\_s, 567
- NBPAIRS
  - constants.h, 681
- nc\_fact
  - alifold.h, 614
- Neighborhood Relation and Move Sets for Secondary Structures, 325
  - VRNA\_MOVESET\_DEFAULT, 329
  - VRNA\_MOVESET\_DELETION, 328
  - VRNA\_MOVESET\_INSERTION, 328
  - VRNA\_MOVESET\_NO\_LP, 329
  - VRNA\_MOVESET\_SHIFT, 328
  - vrna\_loopidx\_update, 330
  - vrna\_move\_apply, 329
  - vrna\_move\_list\_free, 329
  - vrna\_neighbors, 330
  - vrna\_neighbors\_successive, 331
- next
  - vrna\_move\_s, 328
- noLonelyPairs
  - Fine-tuning of the Implemented Models, 176
- node, 548
- nonstandards
  - Fine-tuning of the Implemented Models, 177
- nrrror
  - utils/basic.h, 675
- Nucleic Acid Sequences and Structures, 498
  - read\_record, 505
  - VRNA\_CONSTRAINT\_MULTILINE, 499
  - VRNA\_OPTION\_MULTILINE, 499
  - vrna\_extract\_record\_rest\_constraint, 504
  - vrna\_extract\_record\_rest\_structure, 503
  - vrna\_file\_SHAPE\_read, 503
  - vrna\_file\_bpseq, 500
  - vrna\_file\_connect, 500
  - vrna\_file\_fasta\_read\_record, 501
  - vrna\_file\_helixlist, 499
  - vrna\_file\_json, 501
- PAIR
  - (Abstract) Data Structures, 548
- PS\_dot\_plot
  - Plotting, 527
- PS\_dot\_plot\_list
  - Plotting, 526
- PS\_rna\_plot
  - Plotting, 530
- PS\_rna\_plot\_a
  - Plotting, 531
- PS\_rna\_plot\_a\_gquad
  - Plotting, 531
- pack\_structure
  - Deprecated Interface for Secondary Structure Utilities, 475
- Pair List Representation of Secondary Structures, 460
  - vrna\_plist, 461
- Pair Table Representation of Secondary Structures, 457
  - vrna\_pt\_pk\_get, 458
  - vrna\_pt\_snoop\_get, 459
  - vrna\_ptable, 457
  - vrna\_ptable\_copy, 459
  - vrna\_ptable\_from\_string, 458
- pair\_info
  - Deprecated Interface for Multiple Sequence Alignment Utilities, 491
- paramT
  - Energy Parameters, 181
- parenthesis\_structure



- Deprecated Interface for Secondary Structure Utilities, [479](#)
- parenthesis\_zuker
  - Deprecated Interface for Secondary Structure Utilities, [479](#)
- parse\_gquad
  - G-Quadruplexes, [354](#)
- parse\_structure
  - Deprecated Interface for Secondary Structure Utilities, [475](#)
- part\_func.h
  - centroid, [684](#)
  - expHairpinEnergy, [685](#)
  - expLoopEnergy, [685](#)
  - get\_centroid\_struct\_gquad\_pr, [684](#)
  - mean\_bp\_dist, [685](#)
- part\_func\_co.h
  - get\_plist, [686](#)
- Partition Function and Equilibrium Properties, [249](#)
  - vrna\_pf\_float\_precision, [250](#)
- Partition Function for Two Hybridized Sequences, [411](#)
  - vrna\_pf\_co\_fold, [412](#)
  - vrna\_pf\_dimer\_concentrations, [412](#)
- Partition Function for two Hybridized Sequences as a Stepwise Process, [414](#)
  - pf\_interact, [415](#)
  - pf\_unstru, [414](#)
- path\_t
  - Direct Refolding Paths between two Secondary Structures, [426](#)
- pbacktrack
  - Random Structure Samples from the Ensemble, [297](#)
- pbacktrack\_circ
  - Random Structure Samples from the Ensemble, [298](#)
- pf\_circ\_fold
  - Deprecated Interface for Global Partition Function Computation, [394](#)
- pf\_fold
  - Deprecated Interface for Global Partition Function Computation, [393](#)
- pf\_fold\_par
  - Deprecated Interface for Global Partition Function Computation, [392](#)
- pf\_interact
  - Partition Function for two Hybridized Sequences as a Stepwise Process, [415](#)
- pf\_paramT
  - Energy Parameters, [181](#)
- pf\_scale
  - Fine-tuning of the Implemented Models, [175](#)
- pf\_unstru
  - Partition Function for two Hybridized Sequences as a Stepwise Process, [414](#)
- pfl\_fold
  - Deprecated Interface for Local (Sliding Window) Partition Function Computation, [408](#)
- plist
  - (Abstract) Data Structures, [548](#)
- Plotting, [521](#)
  - aliPS\_color\_aln, [525](#)
  - gmlRNA, [528](#)
  - PS\_dot\_plot, [527](#)
  - PS\_dot\_plot\_list, [526](#)
  - PS\_rna\_plot, [530](#)
  - PS\_rna\_plot\_a, [531](#)
  - PS\_rna\_plot\_a\_gquad, [531](#)
  - rna\_plot\_type, [531](#)
  - simple\_circplot\_coordinates, [525](#)
  - simple\_xy\_coordinates, [525](#)
  - ssv\_rna\_plot, [529](#)
  - svg\_rna\_plot, [529](#)
  - VRNA\_PLOT\_TYPE\_CIRCULAR, [523](#)
  - VRNA\_PLOT\_TYPE\_NAVIEW, [523](#)
  - VRNA\_PLOT\_TYPE\_SIMPLE, [523](#)
  - vrna\_file\_PS\_aln, [524](#)
  - vrna\_file\_PS\_aln\_sub, [524](#)
  - vrna\_file\_PS\_rnaplot, [527](#)
  - vrna\_file\_PS\_rnaplot\_a, [528](#)
  - xrna\_plot, [530](#)
- pos\_3
  - vrna\_move\_s, [328](#)
- pos\_5
  - vrna\_move\_s, [328](#)
- Postorder\_list, [602](#)
- pr
  - fold\_vars.h, [650](#)
- print\_tty\_constraint
  - hard.h, [626](#)
- print\_tty\_constraint\_full
  - hard.h, [627](#)
- print\_tty\_input\_seq
  - utils/basic.h, [675](#)
- print\_tty\_input\_seq\_str
  - utils/basic.h, [675](#)
- prod\_cb
  - vrna\_unstructured\_domain\_s, [196](#)
- profile\_edit\_distance
  - profiledist.h, [699](#)
- profiledist.h
  - free\_profile, [700](#)
  - Make\_bp\_profile, [700](#)
  - Make\_bp\_profile\_bppm, [699](#)
  - profile\_edit\_distance, [699](#)
- progress\_callback
  - Generate Soft Constraints from Data, [345](#)
- pscore
  - vrna\_fc\_s, [569](#)
- pscore\_local
  - vrna\_fc\_s, [569](#)
- pscore\_pf\_compat
  - vrna\_fc\_s, [569](#)
- pctype
  - vrna\_fc\_s, [566](#)
- pctype\_pf\_compat

- vrna\_fc\_s, 566
- pu\_contrib, 546
- pu\_out, 547
- putoutU\_prob
  - Deprecated Interface for Local (Sliding Window) Partition Function Computation, 409
- putoutU\_prob\_bin
  - Deprecated Interface for Local (Sliding Window) Partition Function Computation, 410
- RNA-RNA Interaction, 304
- Random Structure Samples from the Ensemble, 295
  - pbacktrack, 297
  - pbacktrack\_circ, 298
  - st\_back, 298
  - vrna\_pbacktrack, 296
  - vrna\_pbacktrack5, 295
  - vrna\_pbacktrack\_nr, 296
- random\_string
  - strings.h, 717
- read\_parameter\_file
  - Reading/Writing Energy Parameter Sets from/to File, 417
- read\_record
  - Nucleic Acid Sequences and Structures, 505
- Reading/Writing Energy Parameter Sets from/to File, 417
  - last\_parameter\_file, 417
  - read\_parameter\_file, 417
  - write\_parameter\_file, 418
- Refolding Paths of Secondary Structures, 333
  - VRNA\_PATH\_DEFAULT, 334
  - VRNA\_PATH\_NO\_TRANSITION\_OUTPUT, 334
  - VRNA\_PATH\_RANDOM, 334
  - VRNA\_PATH\_STEEPEST\_DESCENT, 333
  - vrna\_path, 335
  - vrna\_path\_gradient, 336
  - vrna\_path\_random, 336
- RibosumFile
  - fold\_vars.h, 649
- rna\_plot\_type
  - Plotting, 531
- S
  - vrna\_fc\_s, 568
- S3
  - vrna\_fc\_s, 569
- S5
  - vrna\_fc\_s, 568
- S\_cons
  - vrna\_fc\_s, 568
- SHAPE Reactivity Data, 339
  - vrna\_sc\_SHAPE\_to\_pr, 342
  - vrna\_sc\_add\_SHAPE\_deigan, 339
  - vrna\_sc\_add\_SHAPE\_deigan\_ali, 340
  - vrna\_sc\_add\_SHAPE\_zarringham, 341
- SHAPE.h
  - vrna\_sc\_SHAPE\_parse\_method, 629
- SOLUTION
  - subopt.h, 709
- sc
  - vrna\_fc\_s, 567
- scale\_parameters
  - Energy Parameters, 189
- scs
  - vrna\_fc\_s, 570
- Search Algorithms, 534
  - vrna\_search\_BM\_BCT\_num, 536
  - vrna\_search\_BM\_BCT, 536
  - vrna\_search\_BMH\_num, 534
  - vrna\_search\_BMH, 535
- Secondary Structure Utilities, 446
  - vrna\_bp\_distance, 446
  - vrna\_db\_from\_bp\_stack, 447
  - vrna\_refBPcnt\_matrix, 447
  - vrna\_refBPdist\_matrix, 447
- sect
  - (Abstract) Data Structures, 549
- sequence
  - vrna\_fc\_s, 565
- sequence\_encoding
  - vrna\_fc\_s, 566
- sequences
  - vrna\_fc\_s, 567
- set\_model\_details
  - Fine-tuning of the Implemented Models, 174
- simple\_circplot\_coordinates
  - Plotting, 525
- simple\_xy\_coordinates
  - Plotting, 525
- snoopT, 548
- Soft Constraints, 234
  - vrna\_callback\_sc\_backtrack, 239
  - vrna\_callback\_sc\_energy, 236
  - vrna\_callback\_sc\_exp\_energy, 238
  - vrna\_sc\_add\_bp, 241
  - vrna\_sc\_add\_bt, 244
  - vrna\_sc\_add\_data, 243
  - vrna\_sc\_add\_exp\_f, 245
  - vrna\_sc\_add\_f, 244
  - vrna\_sc\_add\_up, 242
  - vrna\_sc\_free, 243
  - vrna\_sc\_init, 239
  - vrna\_sc\_remove, 243
  - vrna\_sc\_set\_bp, 240
  - vrna\_sc\_set\_up, 241
- soft.h
  - vrna\_sc\_type\_e, 631
- space
  - utils/basic.h, 676
- ssv\_rna\_plot
  - Plotting, 529
- st\_back
  - Random Structure Samples from the Ensemble, 298
- stackProb

- Deprecated Interface for Global Partition Function Computation, 398
- stat\_cb
  - vrna\_fc\_s, 565
- Stochastic Backtracking of Structures from Distance Based Partitioning, 318
  - vrna\_pbacktrack5\_TwoD, 319
  - vrna\_pbacktrack\_TwoD, 318
- str\_DNA2RNA
  - strings.h, 717
- str\_uppercase
  - strings.h, 716
- string\_edit\_distance
  - stringdist.h, 706
- stringdist.h
  - Make\_swString, 706
  - string\_edit\_distance, 706
- strings.h
  - hamming, 717
  - hamming\_bound, 717
  - random\_string, 717
  - str\_DNA2RNA, 717
  - str\_uppercase, 716
- structure
  - vrna\_ht\_entry\_db\_t, 584
- Structured Domains, 205
- subopt
  - Suboptimal Structures within an Energy Band around the MFE, 293
- subopt.h
  - SOLUTION, 709
- subopt\_circ
  - Suboptimal Structures within an Energy Band around the MFE, 293
- Suboptimal Structures sensu Stiegler et al. 1984 / Zuker et al. 1989, 288
  - vrna\_subopt\_zuker, 288
  - zukersubopt, 289
  - zukersubopt\_par, 289
- Suboptimal Structures within an Energy Band around the MFE, 290
  - subopt, 293
  - subopt\_circ, 293
  - vrna\_subopt, 291
  - vrna\_subopt\_callback, 290
  - vrna\_subopt\_cb, 292
- Suboptimals and Representative Structures, 287
- svg\_rna\_plot
  - Plotting, 529
- swString, 602
- TURN
  - constants.h, 681
- temperature
  - Fine-tuning of the Implemented Models, 175
- tetra\_loop
  - Fine-tuning of the Implemented Models, 176
- The Dynamic Programming Matrices, 578
  - vrna\_mx\_add, 581
  - vrna\_mx\_mfe\_free, 581
  - vrna\_mx\_pf\_free, 582
  - vrna\_mx\_type\_e, 580
- The Fold Compound, 561
  - VRNA\_OPTION\_EVAL\_ONLY, 572
  - VRNA\_OPTION\_MFE, 571
  - VRNA\_OPTION\_PF, 571
  - VRNA\_STATUS\_MFE\_POST, 570
  - VRNA\_STATUS\_MFE\_PRE, 570
  - VRNA\_STATUS\_PF\_POST, 571
  - VRNA\_STATUS\_PF\_PRE, 571
  - vrna\_callback\_free\_auxdata, 572
  - vrna\_callback\_recursion\_status, 573
  - vrna\_fc\_type\_e, 573
  - vrna\_fold\_compound, 574
  - vrna\_fold\_compound\_add\_auxdata, 576
  - vrna\_fold\_compound\_add\_callback, 577
  - vrna\_fold\_compound\_comparative, 575
  - vrna\_fold\_compound\_free, 576
- The RNA Folding Grammar, 141
- The RNA Secondary Structure Landscape, 247
- time\_stamp
  - utils/basic.h, 677
- Tree, 602
- Tree Representation of Secondary Structures, 464
  - VRNA\_STRUCTURE\_TREE\_EXPANDED, 465
  - VRNA\_STRUCTURE\_TREE\_HIT, 464
  - VRNA\_STRUCTURE\_TREE\_SHAPIRO\_EXT, 465
  - VRNA\_STRUCTURE\_TREE\_SHAPIRO\_SHORT, 464
  - VRNA\_STRUCTURE\_TREE\_SHAPIRO\_WEIGHT←HT, 465
  - VRNA\_STRUCTURE\_TREE\_SHAPIRO, 465
  - vrna\_db\_to\_tree\_string, 466
  - vrna\_tree\_string\_to\_db, 467
  - vrna\_tree\_string\_unweight, 467
- tree\_edit\_distance
  - treedist.h, 710
- treedist.h
  - free\_tree, 711
  - make\_tree, 710
  - tree\_edit\_distance, 710
- TwoDfold\_backtrack\_f5
  - Computing MFE representatives of a Distance Based Partitioning, 313
- TwoDfold\_vars, 308
  - Computing MFE representatives of a Distance Based Partitioning, 309
- TwoDfoldList
  - Computing MFE representatives of a Distance Based Partitioning, 312
- TwoDpfold\_pbacktrack
  - 2Dpfold.h, 610
- TwoDpfold\_pbacktrack5
  - 2Dpfold.h, 611
- TwoDpfold\_vars, 603
- TwoDpfoldList
  - 2Dpfold.h, 610

- type
  - vrna\_fc\_s, [565](#)
- unexpand\_Full
  - Deprecated Interface for Secondary Structure Utilities, [474](#)
- unexpand\_aligned\_F
  - Deprecated Interface for Secondary Structure Utilities, [474](#)
- Unit Conversion, [557](#)
  - vrna\_convert\_energy, [559](#)
  - vrna\_convert\_temperature, [559](#)
  - vrna\_unit\_energy\_e, [557](#)
  - vrna\_unit\_temperature\_e, [558](#)
- unpack\_structure
  - Deprecated Interface for Secondary Structure Utilities, [476](#)
- Unstructured Domains, [193](#)
  - vrna\_callback\_ud\_energy, [196](#)
  - vrna\_callback\_ud\_exp\_energy, [196](#)
  - vrna\_callback\_ud\_exp\_production, [197](#)
  - vrna\_callback\_ud\_probs\_add, [197](#)
  - vrna\_callback\_ud\_probs\_get, [198](#)
  - vrna\_callback\_ud\_production, [197](#)
  - vrna\_ud\_add\_motif, [200](#)
  - vrna\_ud\_motifs\_MEA, [199](#)
  - vrna\_ud\_motifs\_MFE, [199](#)
  - vrna\_ud\_motifs\_centroid, [198](#)
  - vrna\_ud\_remove, [201](#)
  - vrna\_ud\_set\_data, [201](#)
  - vrna\_ud\_set\_exp\_prod\_rule\_cb, [203](#)
  - vrna\_ud\_set\_prod\_rule\_cb, [202](#)
- unstructured\_domains.h
  - vrna\_ud\_set\_prob\_cb, [713](#)
- unweight
  - Deprecated Interface for Secondary Structure Utilities, [474](#)
- update\_alifold\_params
  - alifold.h, [614](#)
- update\_co\_pf\_params
  - Deprecated Interface for Global Partition Function Computation, [401](#)
- update\_co\_pf\_params\_par
  - Deprecated Interface for Global Partition Function Computation, [402](#)
- update\_cofold\_params
  - Deprecated Interface for Global MFE Prediction, [379](#)
- update\_cofold\_params\_par
  - Deprecated Interface for Global MFE Prediction, [379](#)
- update\_fold\_params
  - Deprecated Interface for Global MFE Prediction, [384](#)
- update\_fold\_params\_par
  - Deprecated Interface for Global MFE Prediction, [385](#)
- update\_pf\_params
  - Deprecated Interface for Global Partition Function Computation, [395](#)
- update\_pf\_params\_par
  - Deprecated Interface for Global Partition Function Computation, [395](#)
- update\_pf\_paramsLP
  - Deprecated Interface for Local (Sliding Window) Partition Function Computation, [408](#)
- urn
  - utils/basic.h, [676](#)
- Utilities, [356](#)
  - get\_input\_line, [361](#)
  - VRNA\_INPUT\_CONSTRAINT, [358](#)
  - VRNA\_INPUT\_FASTA\_HEADER, [358](#)
  - vrna\_alloc, [359](#)
  - vrna\_idx\_col\_wise, [362](#)
  - vrna\_idx\_row\_wise, [361](#)
  - vrna\_int\_urn, [360](#)
  - vrna\_realloc, [359](#)
  - vrna\_time\_stamp, [360](#)
  - vrna\_urn, [359](#)
  - xsubi, [362](#)
- Utilities to deal with Nucleotide Alphabets, [433](#)
  - vrna\_nucleotide\_decode, [435](#)
  - vrna\_nucleotide\_encode, [435](#)
  - vrna\_ptypes, [434](#)
  - vrna\_seq\_type\_e, [434](#)
- utils/basic.h
  - filecopy, [677](#)
  - get\_line, [674](#)
  - init\_rand, [676](#)
  - int\_urn, [677](#)
  - nrerror, [675](#)
  - print\_tty\_input\_seq, [675](#)
  - print\_tty\_input\_seq\_str, [675](#)
  - space, [676](#)
  - time\_stamp, [677](#)
  - urn, [676](#)
  - warn\_user, [675](#)
  - xrealloc, [676](#)
- VRNA\_BRACKETS\_ALPHA
  - Dot-Bracket Notation of Secondary Structures, [450](#)
- VRNA\_BRACKETS\_ANG
  - Dot-Bracket Notation of Secondary Structures, [451](#)
- VRNA\_BRACKETS\_CLY
  - Dot-Bracket Notation of Secondary Structures, [451](#)
- VRNA\_BRACKETS\_DEFAULT
  - Dot-Bracket Notation of Secondary Structures, [452](#)
- VRNA\_BRACKETS\_RND
  - Dot-Bracket Notation of Secondary Structures, [451](#)
- VRNA\_BRACKETS\_SQR
  - Dot-Bracket Notation of Secondary Structures, [451](#)
- VRNA\_CMD\_PARSE\_DEFAULTS
  - Command Files, [518](#)
- VRNA\_CMD\_PARSE\_HC
  - Command Files, [517](#)
- VRNA\_CMD\_PARSE\_SC
  - Command Files, [517](#)

- VRNA\_CMD\_PARSE\_SD
  - Command Files, [517](#)
- VRNA\_CMD\_PARSE\_UD
  - Command Files, [517](#)
- VRNA\_CONSTRAINT\_DB\_ANG\_BRACK
  - hard.h, [625](#)
- VRNA\_CONSTRAINT\_DB\_DEFAULT
  - Hard Constraints, [228](#)
- VRNA\_CONSTRAINT\_DB\_DOT
  - Hard Constraints, [226](#)
- VRNA\_CONSTRAINT\_DB\_ENFORCE\_BP
  - Hard Constraints, [225](#)
- VRNA\_CONSTRAINT\_DB\_GQUAD
  - Hard Constraints, [228](#)
- VRNA\_CONSTRAINT\_DB\_INTERMOL
  - Hard Constraints, [227](#)
- VRNA\_CONSTRAINT\_DB\_INTRAMOL
  - Hard Constraints, [227](#)
- VRNA\_CONSTRAINT\_DB\_PIPE
  - Hard Constraints, [226](#)
- VRNA\_CONSTRAINT\_DB\_RND\_BRACK
  - Hard Constraints, [227](#)
- VRNA\_CONSTRAINT\_DB\_WUSS
  - Hard Constraints, [228](#)
- VRNA\_CONSTRAINT\_DB\_X
  - Hard Constraints, [226](#)
- VRNA\_CONSTRAINT\_DB
  - Hard Constraints, [225](#)
- VRNA\_CONSTRAINT\_FILE
  - Constraining the RNA Folding Grammar, [209](#)
- VRNA\_CONSTRAINT\_MULTILINE
  - Nucleic Acid Sequences and Structures, [499](#)
- VRNA\_CONSTRAINT\_NO\_HEADER
  - hard.h, [625](#)
- VRNA\_CONSTRAINT\_SOFT\_MFE
  - Constraining the RNA Folding Grammar, [209](#)
- VRNA\_CONSTRAINT\_SOFT\_PF
  - Constraining the RNA Folding Grammar, [209](#)
- VRNA\_CONVERT\_OUTPUT\_ALL
  - Converting Energy Parameter Files, [420](#)
- VRNA\_CONVERT\_OUTPUT\_BULGE
  - Converting Energy Parameter Files, [422](#)
- VRNA\_CONVERT\_OUTPUT\_DANGLE3
  - Converting Energy Parameter Files, [421](#)
- VRNA\_CONVERT\_OUTPUT\_DANGLE5
  - Converting Energy Parameter Files, [421](#)
- VRNA\_CONVERT\_OUTPUT\_DUMP
  - Converting Energy Parameter Files, [423](#)
- VRNA\_CONVERT\_OUTPUT\_HP
  - Converting Energy Parameter Files, [420](#)
- VRNA\_CONVERT\_OUTPUT\_INT\_11
  - Converting Energy Parameter Files, [421](#)
- VRNA\_CONVERT\_OUTPUT\_INT\_21
  - Converting Energy Parameter Files, [422](#)
- VRNA\_CONVERT\_OUTPUT\_INT\_22
  - Converting Energy Parameter Files, [422](#)
- VRNA\_CONVERT\_OUTPUT\_INT
  - Converting Energy Parameter Files, [422](#)
- VRNA\_CONVERT\_OUTPUT\_MISC
  - Converting Energy Parameter Files, [422](#)
- VRNA\_CONVERT\_OUTPUT\_MM\_EXT
  - Converting Energy Parameter Files, [421](#)
- VRNA\_CONVERT\_OUTPUT\_MM\_HP
  - Converting Energy Parameter Files, [420](#)
- VRNA\_CONVERT\_OUTPUT\_MM\_INT\_1N
  - Converting Energy Parameter Files, [420](#)
- VRNA\_CONVERT\_OUTPUT\_MM\_INT\_23
  - Converting Energy Parameter Files, [421](#)
- VRNA\_CONVERT\_OUTPUT\_MM\_INT
  - Converting Energy Parameter Files, [420](#)
- VRNA\_CONVERT\_OUTPUT\_MM\_MULTI
  - Converting Energy Parameter Files, [421](#)
- VRNA\_CONVERT\_OUTPUT\_ML
  - Converting Energy Parameter Files, [422](#)
- VRNA\_CONVERT\_OUTPUT\_NINIO
  - Converting Energy Parameter Files, [423](#)
- VRNA\_CONVERT\_OUTPUT\_SPECIAL\_HP
  - Converting Energy Parameter Files, [423](#)
- VRNA\_CONVERT\_OUTPUT\_STACK
  - Converting Energy Parameter Files, [420](#)
- VRNA\_CONVERT\_OUTPUT\_VANILLA
  - Converting Energy Parameter Files, [423](#)
- VRNA\_DECOMP\_EXT\_EXT\_EXT
  - Constraining the RNA Folding Grammar, [217](#)
- VRNA\_DECOMP\_EXT\_EXT\_STEM1
  - Constraining the RNA Folding Grammar, [218](#)
- VRNA\_DECOMP\_EXT\_EXT\_STEM
  - Constraining the RNA Folding Grammar, [218](#)
- VRNA\_DECOMP\_EXT\_EXT
  - Constraining the RNA Folding Grammar, [215](#)
- VRNA\_DECOMP\_EXT\_STEM\_EXT
  - Constraining the RNA Folding Grammar, [217](#)
- VRNA\_DECOMP\_EXT\_STEM
  - Constraining the RNA Folding Grammar, [216](#)
- VRNA\_DECOMP\_EXT\_UP
  - Constraining the RNA Folding Grammar, [216](#)
- VRNA\_DECOMP\_ML\_COAXIAL\_ENC
  - Constraining the RNA Folding Grammar, [215](#)
- VRNA\_DECOMP\_ML\_COAXIAL
  - Constraining the RNA Folding Grammar, [214](#)
- VRNA\_DECOMP\_ML\_ML\_ML
  - Constraining the RNA Folding Grammar, [212](#)
- VRNA\_DECOMP\_ML\_ML\_STEM
  - Constraining the RNA Folding Grammar, [214](#)
- VRNA\_DECOMP\_ML\_ML
  - Constraining the RNA Folding Grammar, [213](#)
- VRNA\_DECOMP\_ML\_STEM
  - Constraining the RNA Folding Grammar, [212](#)
- VRNA\_DECOMP\_ML\_UP
  - Constraining the RNA Folding Grammar, [213](#)
- VRNA\_DECOMP\_PAIR\_HP
  - Constraining the RNA Folding Grammar, [210](#)
- VRNA\_DECOMP\_PAIR\_IL
  - Constraining the RNA Folding Grammar, [210](#)
- VRNA\_DECOMP\_PAIR\_ML
  - Constraining the RNA Folding Grammar, [211](#)

- VRNA\_FILE\_FORMAT\_MSA\_APPEND
  - Multiple Sequence Alignments, [509](#)
- VRNA\_FILE\_FORMAT\_MSA\_CLUSTAL
  - Multiple Sequence Alignments, [507](#)
- VRNA\_FILE\_FORMAT\_MSA\_DEFAULT
  - Multiple Sequence Alignments, [508](#)
- VRNA\_FILE\_FORMAT\_MSA\_FASTA
  - Multiple Sequence Alignments, [507](#)
- VRNA\_FILE\_FORMAT\_MSA\_MAF
  - Multiple Sequence Alignments, [508](#)
- VRNA\_FILE\_FORMAT\_MSA\_MIS
  - Multiple Sequence Alignments, [508](#)
- VRNA\_FILE\_FORMAT\_MSA\_NOCHECK
  - Multiple Sequence Alignments, [509](#)
- VRNA\_FILE\_FORMAT\_MSA\_QUIET
  - Multiple Sequence Alignments, [510](#)
- VRNA\_FILE\_FORMAT\_MSA\_SILENT
  - Multiple Sequence Alignments, [510](#)
- VRNA\_FILE\_FORMAT\_MSA\_STOCKHOLM
  - Multiple Sequence Alignments, [507](#)
- VRNA\_FILE\_FORMAT\_MSA\_UNKNOWN
  - Multiple Sequence Alignments, [509](#)
- VRNA\_INPUT\_CONSTRAINT
  - Utilities, [358](#)
- VRNA\_INPUT\_FASTA\_HEADER
  - Utilities, [358](#)
- VRNA\_MEASURE\_SHANNON\_ENTROPY
  - Multiple Sequence Alignment Utilities, [483](#)
- VRNA\_MINIMIZER\_CONJUGATE\_FR
  - Generate Soft Constraints from Data, [344](#)
- VRNA\_MINIMIZER\_CONJUGATE\_PR
  - Generate Soft Constraints from Data, [344](#)
- VRNA\_MINIMIZER\_STEEPEST\_DESCENT
  - Generate Soft Constraints from Data, [345](#)
- VRNA\_MINIMIZER\_VECTOR\_BFGS2
  - Generate Soft Constraints from Data, [345](#)
- VRNA\_MINIMIZER\_VECTOR\_BFGS
  - Generate Soft Constraints from Data, [344](#)
- VRNA\_MODEL\_DEFAULT\_ALI\_CV\_FACT
  - Fine-tuning of the Implemented Models, [154](#)
- VRNA\_MODEL\_DEFAULT\_ALI\_NC\_FACT
  - Fine-tuning of the Implemented Models, [154](#)
- VRNA\_MODEL\_DEFAULT\_ALI\_OLD\_EN
  - Fine-tuning of the Implemented Models, [154](#)
- VRNA\_MODEL\_DEFAULT\_ALI\_RIBO
  - Fine-tuning of the Implemented Models, [154](#)
- VRNA\_MODEL\_DEFAULT\_BACKTRACK\_TYPE
  - Fine-tuning of the Implemented Models, [152](#)
- VRNA\_MODEL\_DEFAULT\_BACKTRACK
  - Fine-tuning of the Implemented Models, [152](#)
- VRNA\_MODEL\_DEFAULT\_BETA\_SCALE
  - Fine-tuning of the Implemented Models, [150](#)
- VRNA\_MODEL\_DEFAULT\_CIRC
  - Fine-tuning of the Implemented Models, [151](#)
- VRNA\_MODEL\_DEFAULT\_COMPUTE\_BPP
  - Fine-tuning of the Implemented Models, [153](#)
- VRNA\_MODEL\_DEFAULT\_DANGLES
  - Fine-tuning of the Implemented Models, [150](#)
- VRNA\_MODEL\_DEFAULT\_ENERGY\_SET
  - Fine-tuning of the Implemented Models, [152](#)
- VRNA\_MODEL\_DEFAULT\_GQUAD
  - Fine-tuning of the Implemented Models, [151](#)
- VRNA\_MODEL\_DEFAULT\_LOG\_ML
  - Fine-tuning of the Implemented Models, [153](#)
- VRNA\_MODEL\_DEFAULT\_MAX\_BP\_SPAN
  - Fine-tuning of the Implemented Models, [153](#)
- VRNA\_MODEL\_DEFAULT\_NO\_GU\_CLOSURE
  - Fine-tuning of the Implemented Models, [151](#)
- VRNA\_MODEL\_DEFAULT\_NO\_GU
  - Fine-tuning of the Implemented Models, [151](#)
- VRNA\_MODEL\_DEFAULT\_NO\_LP
  - Fine-tuning of the Implemented Models, [150](#)
- VRNA\_MODEL\_DEFAULT\_PF\_SCALE
  - Fine-tuning of the Implemented Models, [149](#)
- VRNA\_MODEL\_DEFAULT\_SPECIAL\_HP
  - Fine-tuning of the Implemented Models, [150](#)
- VRNA\_MODEL\_DEFAULT\_TEMPERATURE
  - Fine-tuning of the Implemented Models, [149](#)
- VRNA\_MODEL\_DEFAULT\_UNIQ\_ML
  - Fine-tuning of the Implemented Models, [152](#)
- VRNA\_MODEL\_DEFAULT\_WINDOW\_SIZE
  - Fine-tuning of the Implemented Models, [153](#)
- VRNA\_MOVESET\_DEFAULT
  - Neighborhood Relation and Move Sets for Secondary Structures, [329](#)
- VRNA\_MOVESET\_DELETION
  - Neighborhood Relation and Move Sets for Secondary Structures, [328](#)
- VRNA\_MOVESET\_INSERTION
  - Neighborhood Relation and Move Sets for Secondary Structures, [328](#)
- VRNA\_MOVESET\_NO\_LP
  - Neighborhood Relation and Move Sets for Secondary Structures, [329](#)
- VRNA\_MOVESET\_SHIFT
  - Neighborhood Relation and Move Sets for Secondary Structures, [328](#)
- VRNA\_OBJECTIVE\_FUNCTION\_ABSOLUTE
  - Generate Soft Constraints from Data, [344](#)
- VRNA\_OBJECTIVE\_FUNCTION\_QUADRATIC
  - Generate Soft Constraints from Data, [344](#)
- VRNA\_OPTION\_EVAL\_ONLY
  - The Fold Compound, [572](#)
- VRNA\_OPTION\_MFE
  - The Fold Compound, [571](#)
- VRNA\_OPTION\_MULTILINE
  - Nucleic Acid Sequences and Structures, [499](#)
- VRNA\_OPTION\_PF
  - The Fold Compound, [571](#)
- VRNA\_PATH\_DEFAULT
  - Refolding Paths of Secondary Structures, [334](#)
- VRNA\_PATH\_NO\_TRANSITION\_OUTPUT
  - Refolding Paths of Secondary Structures, [334](#)
- VRNA\_PATH\_RANDOM
  - Refolding Paths of Secondary Structures, [334](#)
- VRNA\_PATH\_STEEPEST\_DESCENT



- Refolding Paths of Secondary Structures, [333](#)
- VRNA\_PLOT\_TYPE\_CIRCULAR
  - Plotting, [523](#)
- VRNA\_PLOT\_TYPE\_NAVIEW
  - Plotting, [523](#)
- VRNA\_PLOT\_TYPE\_SIMPLE
  - Plotting, [523](#)
- VRNA\_PROBS\_WINDOW\_BPP
  - Local (sliding window) Partition Function and Equilibrium Probabilities, [278](#)
- VRNA\_PROBS\_WINDOW\_PF
  - Local (sliding window) Partition Function and Equilibrium Probabilities, [279](#)
- VRNA\_PROBS\_WINDOW\_STACKP
  - Local (sliding window) Partition Function and Equilibrium Probabilities, [279](#)
- VRNA\_PROBS\_WINDOW\_UP\_SPLIT
  - Local (sliding window) Partition Function and Equilibrium Probabilities, [279](#)
- VRNA\_PROBS\_WINDOW\_UP
  - Local (sliding window) Partition Function and Equilibrium Probabilities, [278](#)
- VRNA\_STATUS\_MFE\_POST
  - The Fold Compound, [570](#)
- VRNA\_STATUS\_MFE\_PRE
  - The Fold Compound, [570](#)
- VRNA\_STATUS\_PF\_POST
  - The Fold Compound, [571](#)
- VRNA\_STATUS\_PF\_PRE
  - The Fold Compound, [571](#)
- VRNA\_STRUCTURE\_TREE\_EXPANDED
  - Tree Representation of Secondary Structures, [465](#)
- VRNA\_STRUCTURE\_TREE\_HIT
  - Tree Representation of Secondary Structures, [464](#)
- VRNA\_STRUCTURE\_TREE\_SHAPIRO\_EXT
  - Tree Representation of Secondary Structures, [465](#)
- VRNA\_STRUCTURE\_TREE\_SHAPIRO\_SHORT
  - Tree Representation of Secondary Structures, [464](#)
- VRNA\_STRUCTURE\_TREE\_SHAPIRO\_WEIGHT
  - Tree Representation of Secondary Structures, [465](#)
- VRNA\_STRUCTURE\_TREE\_SHAPIRO
  - Tree Representation of Secondary Structures, [465](#)
- ViennaRNA/2Dfold.h, [607](#)
- ViennaRNA/2Dpfold.h, [608](#)
- ViennaRNA/LPfold.h, [658](#)
- ViennaRNA/Lfold.h, [653](#)
- ViennaRNA/MEA.h, [658](#)
- ViennaRNA/PS\_dot.h, [701](#)
- ViennaRNA/RNAstruct.h, [702](#)
- ViennaRNA/alifold.h, [612](#)
- ViennaRNA/aln\_util.h, [615](#)
- ViennaRNA/alphabet.h, [615](#)
- ViennaRNA/boltzmann\_sampling.h, [616](#)
- ViennaRNA/centroid.h, [616](#)
- ViennaRNA/char\_stream.h, [617](#)
- ViennaRNA/cofold.h, [618](#)
- ViennaRNA/combinatorics.h, [619](#)
- ViennaRNA/commands.h, [620](#)
- ViennaRNA/concentrations.h, [621](#)
- ViennaRNA/constraints.h, [622](#)
- ViennaRNA/constraints/SHAPE.h, [628](#)
- ViennaRNA/constraints/basic.h, [671](#)
- ViennaRNA/constraints/hard.h, [623](#)
- ViennaRNA/constraints/ligand.h, [628](#)
- ViennaRNA/constraints/soft.h, [630](#)
- ViennaRNA/constraints\_SHAPE.h, [632](#)
- ViennaRNA/constraints\_hard.h, [632](#)
- ViennaRNA/constraints\_ligand.h, [632](#)
- ViennaRNA/constraints\_soft.h, [633](#)
- ViennaRNA/convert\_epars.h, [633](#)
- ViennaRNA/data\_structures.h, [633](#)
- ViennaRNA/datastructures/basic.h, [678](#)
- ViennaRNA/datastructures/char\_stream.h, [618](#)
- ViennaRNA/datastructures/hash\_tables.h, [634](#)
- ViennaRNA/datastructures/stream\_output.h, [704](#)
- ViennaRNA/dist\_vars.h, [635](#)
- ViennaRNA/dp\_matrices.h, [636](#)
- ViennaRNA/duplex.h, [637](#)
- ViennaRNA/edit\_cost.h, [637](#)
- ViennaRNA/energy\_const.h, [637](#)
- ViennaRNA/energy\_par.h, [638](#)
- ViennaRNA/equilibrium\_probs.h, [638](#)
- ViennaRNA/eval.h, [639](#)
- ViennaRNA/exterior\_loops.h, [642](#)
- ViennaRNA/file\_formats.h, [642](#)
- ViennaRNA/file\_formats\_msa.h, [644](#)
- ViennaRNA/file\_utils.h, [645](#)
- ViennaRNA/findpath.h, [645](#)
- ViennaRNA/fold.h, [646](#)
- ViennaRNA/fold\_compound.h, [647](#)
- ViennaRNA/fold\_vars.h, [649](#)
- ViennaRNA/gquad.h, [651](#)
- ViennaRNA/grammar.h, [651](#)
- ViennaRNA/hairpin\_loops.h, [652](#)
- ViennaRNA/interior\_loops.h, [652](#)
- ViennaRNA/inverse.h, [652](#)
- ViennaRNA/io/file\_formats.h, [643](#)
- ViennaRNA/io/file\_formats\_msa.h, [644](#)
- ViennaRNA/io/utils.h, [714](#)
- ViennaRNA/loop\_energies.h, [653](#)
- ViennaRNA/loops/all.h, [654](#)
- ViennaRNA/loops/external.h, [654](#)
- ViennaRNA/loops/hairpin.h, [655](#)
- ViennaRNA/loops/internal.h, [656](#)
- ViennaRNA/loops/multibranch.h, [657](#)
- ViennaRNA/mfe.h, [659](#)
- ViennaRNA/mfe\_window.h, [660](#)
- ViennaRNA/mm.h, [661](#)
- ViennaRNA/model.h, [662](#)
- ViennaRNA/multibranch\_loops.h, [666](#)
- ViennaRNA/naview.h, [666](#)
- ViennaRNA/neighbor.h, [667](#)
- ViennaRNA/params.h, [668](#)
- ViennaRNA/params/1.8.4\_epars.h, [668](#)
- ViennaRNA/params/1.8.4\_intloops.h, [669](#)
- ViennaRNA/params/basic.h, [669](#)

- ViennaRNA/params/constants.h, [679](#)
- ViennaRNA/params/convert.h, [681](#)
- ViennaRNA/params/io.h, [682](#)
- ViennaRNA/part\_func.h, [682](#)
- ViennaRNA/part\_func\_co.h, [686](#)
- ViennaRNA/part\_func\_up.h, [687](#)
- ViennaRNA/part\_func\_window.h, [687](#)
- ViennaRNA/perturbation\_fold.h, [689](#)
- ViennaRNA/plot\_aln.h, [690](#)
- ViennaRNA/plot\_layouts.h, [690](#)
- ViennaRNA/plot\_structure.h, [690](#)
- ViennaRNA/plot\_utils.h, [691](#)
- ViennaRNA/plotting/alignments.h, [691](#)
- ViennaRNA/plotting/layouts.h, [693](#)
- ViennaRNA/plotting/naview.h, [667](#)
- ViennaRNA/plotting/probabilities.h, [694](#)
- ViennaRNA/plotting/structures.h, [695](#)
- ViennaRNA/plotting/utils.h, [715](#)
- ViennaRNA/profiledist.h, [699](#)
- ViennaRNA/read\_epars.h, [701](#)
- ViennaRNA/ribo.h, [701](#)
- ViennaRNA/search/BoyerMoore.h, [703](#)
- ViennaRNA/sequence.h, [703](#)
- ViennaRNA/stream\_output.h, [704](#)
- ViennaRNA/string\_utils.h, [705](#)
- ViennaRNA/stringdist.h, [705](#)
- ViennaRNA/structure\_utils.h, [707](#)
- ViennaRNA/structured\_domains.h, [707](#)
- ViennaRNA/subopt.h, [707](#)
- ViennaRNA/svm\_utils.h, [709](#)
- ViennaRNA/treedist.h, [709](#)
- ViennaRNA/units.h, [711](#)
- ViennaRNA/unstructured\_domains.h, [712](#)
- ViennaRNA/utils.h, [714](#)
- ViennaRNA/utils/alignments.h, [692](#)
- ViennaRNA/utils/basic.h, [672](#)
- ViennaRNA/utils/strings.h, [715](#)
- ViennaRNA/utils/structures.h, [696](#)
- ViennaRNA/walk.h, [718](#)
- vrna\_BT\_hp\_loop
  - Backtracking MFE structures, [263](#)
- vrna\_BT\_mb\_loop
  - Backtracking MFE structures, [263](#)
- vrna\_C11\_features
  - (Abstract) Data Structures, [549](#)
- vrna\_E\_ext\_hp\_loop
  - Hairpin Loops, [369](#)
- vrna\_E\_ext\_loop
  - Exterior Loops, [365](#)
- vrna\_E\_ext\_stem
  - Exterior Loops, [365](#)
- vrna\_E\_hp\_loop
  - Hairpin Loops, [368](#)
- vrna\_E\_mb\_loop\_stack
  - Multibranch Loops, [375](#)
- vrna\_Lfold
  - Local (sliding window) MFE Prediction, [261](#)
- vrna\_Lfoldz
  - Local (sliding window) MFE Prediction, [262](#)
- vrna\_alifold
  - Global MFE Prediction, [254](#)
- vrna\_alloc
  - Utilities, [359](#)
- vrna\_aln\_consensus\_mis
  - Multiple Sequence Alignment Utilities, [490](#)
- vrna\_aln\_consensus\_sequence
  - Multiple Sequence Alignment Utilities, [490](#)
- vrna\_aln\_conservation\_col
  - Multiple Sequence Alignment Utilities, [489](#)
- vrna\_aln\_conservation\_struct
  - Multiple Sequence Alignment Utilities, [488](#)
- vrna\_aln\_copy
  - Multiple Sequence Alignment Utilities, [488](#)
- vrna\_aln\_free
  - Multiple Sequence Alignment Utilities, [485](#)
- vrna\_aln\_mpi
  - Multiple Sequence Alignment Utilities, [483](#)
- vrna\_aln\_pinfo
  - Multiple Sequence Alignment Utilities, [483](#)
- vrna\_aln\_slice
  - Multiple Sequence Alignment Utilities, [485](#)
- vrna\_aln\_toRNA
  - Multiple Sequence Alignment Utilities, [487](#)
- vrna\_aln\_uppercase
  - Multiple Sequence Alignment Utilities, [487](#)
- vrna\_backtrack5\_TwoD
  - Computing MFE representatives of a Distance Based Partitioning, [311](#)
- vrna\_basepair\_s, [546](#)
- vrna\_bp\_distance
  - Secondary Structure Utilities, [446](#)
- vrna\_bp\_stack\_s, [546](#)
- vrna\_callback\_free\_auxdata
  - The Fold Compound, [572](#)
- vrna\_callback\_hc\_evaluate
  - Hard Constraints, [229](#)
- vrna\_callback\_ht\_compare\_entries
  - Hash Tables, [585](#)
- vrna\_callback\_ht\_free\_entry
  - Hash Tables, [586](#)
- vrna\_callback\_ht\_hash\_function
  - Hash Tables, [585](#)
- vrna\_callback\_recursion\_status
  - The Fold Compound, [573](#)
- vrna\_callback\_sc\_backtrack
  - Soft Constraints, [239](#)
- vrna\_callback\_sc\_energy
  - Soft Constraints, [236](#)
- vrna\_callback\_sc\_exp\_energy
  - Soft Constraints, [238](#)
- vrna\_callback\_stream\_output
  - Buffers, [595](#)
- vrna\_callback\_ud\_energy
  - Unstructured Domains, [196](#)
- vrna\_callback\_ud\_exp\_energy
  - Unstructured Domains, [196](#)



- `vrna_callback_ud_exp_production`
  - Unstructured Domains, [197](#)
- `vrna_callback_ud_probs_add`
  - Unstructured Domains, [197](#)
- `vrna_callback_ud_probs_get`
  - Unstructured Domains, [198](#)
- `vrna_callback_ud_production`
  - Unstructured Domains, [197](#)
- `vrna_centroid`
  - Compute the Centroid Structure, [301](#)
- `vrna_centroid_from_plist`
  - Compute the Centroid Structure, [301](#)
- `vrna_centroid_from_probs`
  - Compute the Centroid Structure, [302](#)
- `vrna_circalifold`
  - Global MFE Prediction, [255](#)
- `vrna_circfold`
  - Global MFE Prediction, [254](#)
- `vrna_cofold`
  - Global MFE Prediction, [256](#)
- `vrna_color_s`, [546](#)
- `vrna_commands_apply`
  - Command Files, [519](#)
- `vrna_commands_free`
  - Command Files, [520](#)
- `vrna_constraints_add`
  - Constraining the RNA Folding Grammar, [219](#)
- `vrna_convert_energy`
  - Unit Conversion, [559](#)
- `vrna_convert_temperature`
  - Unit Conversion, [559](#)
- `vrna_cpair_s`, [546](#)
- `vrna_cstr`
  - Buffers, [595](#)
- `vrna_cstr_close`
  - Buffers, [596](#)
- `vrna_cstr_fflush`
  - Buffers, [596](#)
- `vrna_cstr_free`
  - Buffers, [596](#)
- `vrna_cut_point_insert`
  - (Nucleic Acid Sequence) String Utilities, [444](#)
- `vrna_cut_point_remove`
  - (Nucleic Acid Sequence) String Utilities, [444](#)
- `vrna_data_linear_s`, [546](#)
- `vrna_db_flatten`
  - Dot-Bracket Notation of Secondary Structures, [453](#)
- `vrna_db_flatten_to`
  - Dot-Bracket Notation of Secondary Structures, [454](#)
- `vrna_db_from_WUSS`
  - Dot-Bracket Notation of Secondary Structures, [455](#)
- `vrna_db_from_bp_stack`
  - Secondary Structure Utilities, [447](#)
- `vrna_db_from_plist`
  - Dot-Bracket Notation of Secondary Structures, [455](#)
- `vrna_db_from_ptable`
  - Dot-Bracket Notation of Secondary Structures, [454](#)
- `vrna_db_pack`
  - Dot-Bracket Notation of Secondary Structures, [452](#)
- `vrna_db_to_element_string`
  - Dot-Bracket Notation of Secondary Structures, [456](#)
- `vrna_db_to_tree_string`
  - Tree Representation of Secondary Structures, [466](#)
- `vrna_db_unpack`
  - Dot-Bracket Notation of Secondary Structures, [453](#)
- `vrna_dimer_conc_s`, [604](#)
- `vrna_dimer_pf_s`, [266](#)
- `vrna_dotplot_auxdata_t`, [522](#)
- `vrna_elem_prob_s`, [460](#)
- `vrna_ensemble_defect`
  - Global Partition Function and Equilibrium Probabilities, [268](#)
- `vrna_enumerate_necklaces`
  - Combinatorics Algorithms, [538](#)
- `vrna_eval_circ_consensus_structure`
  - Free Energy Evaluation, [110](#)
- `vrna_eval_circ_consensus_structure_v`
  - Free Energy Evaluation, [114](#)
- `vrna_eval_circ_gquad_consensus_structure`
  - Free Energy Evaluation, [112](#)
- `vrna_eval_circ_gquad_consensus_structure_v`
  - Free Energy Evaluation, [116](#)
- `vrna_eval_circ_gquad_structure`
  - Free Energy Evaluation, [105](#)
- `vrna_eval_circ_gquad_structure_v`
  - Free Energy Evaluation, [109](#)
- `vrna_eval_circ_structure`
  - Free Energy Evaluation, [104](#)
- `vrna_eval_circ_structure_v`
  - Free Energy Evaluation, [107](#)
- `vrna_eval_consensus_structure_pt_simple`
  - Free Energy Evaluation, [119](#)
- `vrna_eval_consensus_structure_simple`
  - Free Energy Evaluation, [109](#)
- `vrna_eval_consensus_structure_simple_v`
  - Free Energy Evaluation, [113](#)
- `vrna_eval_consensus_structure_simple_verbose`
  - Free Energy Evaluation, [113](#)
- `vrna_eval_covar_structure`
  - Free Energy Evaluation, [99](#)
- `vrna_eval_gquad_consensus_structure`
  - Free Energy Evaluation, [111](#)
- `vrna_eval_gquad_consensus_structure_v`
  - Free Energy Evaluation, [115](#)
- `vrna_eval_gquad_structure`
  - Free Energy Evaluation, [104](#)
- `vrna_eval_gquad_structure_v`
  - Free Energy Evaluation, [108](#)
- `vrna_eval_hp_loop`
  - Hairpin Loops, [369](#)
- `vrna_eval_int_loop`
  - Internal Loops, [373](#)
- `vrna_eval_loop_pt`
  - Energy Evaluation for Individual Loops, [122](#)
- `vrna_eval_loop_pt_v`
  - Energy Evaluation for Individual Loops, [122](#)

- vrna\_eval\_move
  - Energy Evaluation for Atomic Moves, [124](#)
- vrna\_eval\_move\_pt
  - Energy Evaluation for Atomic Moves, [125](#)
- vrna\_eval\_structure
  - Free Energy Evaluation, [98](#)
- vrna\_eval\_structure\_pt
  - Free Energy Evaluation, [101](#)
- vrna\_eval\_structure\_pt\_simple
  - Free Energy Evaluation, [117](#)
- vrna\_eval\_structure\_pt\_simple\_v
  - Free Energy Evaluation, [118](#)
- vrna\_eval\_structure\_pt\_simple\_verbose
  - Free Energy Evaluation, [118](#)
- vrna\_eval\_structure\_pt\_v
  - Free Energy Evaluation, [102](#)
- vrna\_eval\_structure\_pt\_verbose
  - Free Energy Evaluation, [102](#)
- vrna\_eval\_structure\_simple
  - Free Energy Evaluation, [103](#)
- vrna\_eval\_structure\_simple\_v
  - Free Energy Evaluation, [106](#)
- vrna\_eval\_structure\_simple\_verbose
  - Free Energy Evaluation, [106](#)
- vrna\_eval\_structure\_v
  - Free Energy Evaluation, [100](#)
- vrna\_eval\_structure\_verbose
  - Free Energy Evaluation, [100](#)
- vrna\_exp\_E\_ext\_stem
  - Exterior Loops, [366](#)
- vrna\_exp\_E\_hp\_loop
  - Hairpin Loops, [372](#)
- vrna\_exp\_param\_s, [180](#)
  - alpha, [180](#)
  - id, [180](#)
- vrna\_exp\_params
  - Energy Parameters, [182](#)
- vrna\_exp\_params\_comparative
  - Energy Parameters, [183](#)
- vrna\_exp\_params\_copy
  - Energy Parameters, [183](#)
- vrna\_exp\_params\_rescale
  - Energy Parameters, [185](#)
- vrna\_exp\_params\_reset
  - Energy Parameters, [186](#)
- vrna\_exp\_params\_subst
  - Energy Parameters, [184](#)
- vrna\_extract\_record\_rest\_constraint
  - Nucleic Acid Sequences and Structures, [504](#)
- vrna\_extract\_record\_rest\_structure
  - Nucleic Acid Sequences and Structures, [503](#)
- vrna\_fc\_s, [562](#)
  - auxdata, [565](#)
  - cons\_seq, [568](#)
  - free\_auxdata, [565](#)
  - n\_seq, [567](#)
  - pscore, [569](#)
  - pscore\_local, [569](#)
  - pscore\_pf\_compat, [569](#)
  - ptype, [566](#)
  - ptype\_pf\_compat, [566](#)
  - S, [568](#)
  - S3, [569](#)
  - S5, [568](#)
  - S\_cons, [568](#)
  - sc, [567](#)
  - scs, [570](#)
  - sequence, [565](#)
  - sequence\_encoding, [566](#)
  - sequences, [567](#)
  - stat\_cb, [565](#)
  - type, [565](#)
- vrna\_fc\_type\_e
  - The Fold Compound, [573](#)
- vrna\_file\_PS\_aln
  - Plotting, [524](#)
- vrna\_file\_PS\_aln\_sub
  - Plotting, [524](#)
- vrna\_file\_PS\_rnaplot
  - Plotting, [527](#)
- vrna\_file\_PS\_rnaplot\_a
  - Plotting, [528](#)
- vrna\_file\_SHAPE\_read
  - Nucleic Acid Sequences and Structures, [503](#)
- vrna\_file\_bpseq
  - Nucleic Acid Sequences and Structures, [500](#)
- vrna\_file\_commands\_apply
  - Command Files, [519](#)
- vrna\_file\_commands\_read
  - Command Files, [518](#)
- vrna\_file\_connect
  - Nucleic Acid Sequences and Structures, [500](#)
- vrna\_file\_exists
  - Files and I/O, [497](#)
- vrna\_file\_fasta\_read\_record
  - Nucleic Acid Sequences and Structures, [501](#)
- vrna\_file\_helixlist
  - Nucleic Acid Sequences and Structures, [499](#)
- vrna\_file\_json
  - Nucleic Acid Sequences and Structures, [501](#)
- vrna\_file\_msa\_detect\_format
  - Multiple Sequence Alignments, [513](#)
- vrna\_file\_msa\_read
  - Multiple Sequence Alignments, [510](#)
- vrna\_file\_msa\_read\_record
  - Multiple Sequence Alignments, [512](#)
- vrna\_file\_msa\_write
  - Multiple Sequence Alignments, [514](#)
- vrna\_filename\_sanitize
  - Files and I/O, [496](#)
- vrna\_fold
  - Global MFE Prediction, [253](#)
- vrna\_fold\_compound
  - The Fold Compound, [574](#)
- vrna\_fold\_compound\_add\_auxdata
  - The Fold Compound, [576](#)

- `vrna_fold_compound_add_callback`
  - The Fold Compound, [577](#)
- `vrna_fold_compound_comparative`
  - The Fold Compound, [575](#)
- `vrna_fold_compound_free`
  - The Fold Compound, [576](#)
- `vrna_gr_aux_s`, [141](#)
- `vrna_hamming_distance`
  - (Nucleic Acid Sequence) String Utilites, [442](#)
- `vrna_hamming_distance_bound`
  - (Nucleic Acid Sequence) String Utilites, [442](#)
- `vrna_hash_table_t`
  - Hash Tables, [584](#)
- `vrna_hc_add_bp`
  - Hard Constraints, [231](#)
- `vrna_hc_add_bp_nonspecific`
  - Hard Constraints, [232](#)
- `vrna_hc_add_data`
  - `hard.h`, [626](#)
- `vrna_hc_add_from_db`
  - Hard Constraints, [233](#)
- `vrna_hc_add_up`
  - Hard Constraints, [230](#)
- `vrna_hc_add_up_batch`
  - Hard Constraints, [231](#)
- `vrna_hc_bp_storage_t`, [604](#)
- `vrna_hc_free`
  - Hard Constraints, [232](#)
- `vrna_hc_init`
  - Hard Constraints, [230](#)
- `vrna_hc_s`, [224](#)
  - `free_data`, [225](#)
- `vrna_hc_type_e`
  - `hard.h`, [626](#)
- `vrna_hc_up_s`, [225](#)
- `vrna_ht_clear`
  - Hash Tables, [590](#)
- `vrna_ht_collisions`
  - Hash Tables, [587](#)
- `vrna_ht_db_comp`
  - Hash Tables, [591](#)
- `vrna_ht_db_free_entry`
  - Hash Tables, [592](#)
- `vrna_ht_db_hash_func`
  - Hash Tables, [592](#)
- `vrna_ht_entry_db_t`, [584](#)
  - `energy`, [584](#)
  - `structure`, [584](#)
- `vrna_ht_free`
  - Hash Tables, [591](#)
- `vrna_ht_get`
  - Hash Tables, [589](#)
- `vrna_ht_init`
  - Hash Tables, [586](#)
- `vrna_ht_insert`
  - Hash Tables, [589](#)
- `vrna_ht_remove`
  - Hash Tables, [590](#)
- `vrna_ht_size`
  - Hash Tables, [587](#)
- `vrna_hx_from_ptable`
  - Helix List Representation of Secondary Structures, [462](#)
- `vrna_hx_s`, [462](#)
- `vrna_idx_col_wise`
  - Utilities, [362](#)
- `vrna_idx_row_wise`
  - Utilities, [361](#)
- `vrna_int_urn`
  - Utilities, [360](#)
- `vrna_loopidx_update`
  - Neighborhood Relation and Move Sets for Secondary Structures, [330](#)
- `vrna_maximum_matching`
  - `mm.h`, [661](#)
- `vrna_maximum_matching_simple`
  - `mm.h`, [661](#)
- `vrna_md_copy`
  - Fine-tuning of the Implemented Models, [155](#)
- `vrna_md_defaults_backtrack`
  - Fine-tuning of the Implemented Models, [166](#)
- `vrna_md_defaults_backtrack_get`
  - Fine-tuning of the Implemented Models, [166](#)
- `vrna_md_defaults_backtrack_type`
  - Fine-tuning of the Implemented Models, [167](#)
- `vrna_md_defaults_backtrack_type_get`
  - Fine-tuning of the Implemented Models, [167](#)
- `vrna_md_defaults_betaScale`
  - Fine-tuning of the Implemented Models, [158](#)
- `vrna_md_defaults_betaScale_get`
  - Fine-tuning of the Implemented Models, [158](#)
- `vrna_md_defaults_circ`
  - Fine-tuning of the Implemented Models, [163](#)
- `vrna_md_defaults_circ_get`
  - Fine-tuning of the Implemented Models, [163](#)
- `vrna_md_defaults_compute_bpp`
  - Fine-tuning of the Implemented Models, [167](#)
- `vrna_md_defaults_compute_bpp_get`
  - Fine-tuning of the Implemented Models, [168](#)
- `vrna_md_defaults_cv_fact`
  - Fine-tuning of the Implemented Models, [172](#)
- `vrna_md_defaults_cv_fact_get`
  - Fine-tuning of the Implemented Models, [172](#)
- `vrna_md_defaults_dangles`
  - Fine-tuning of the Implemented Models, [158](#)
- `vrna_md_defaults_dangles_get`
  - Fine-tuning of the Implemented Models, [159](#)
- `vrna_md_defaults_energy_set`
  - Fine-tuning of the Implemented Models, [165](#)
- `vrna_md_defaults_energy_set_get`
  - Fine-tuning of the Implemented Models, [166](#)
- `vrna_md_defaults_gquad`
  - Fine-tuning of the Implemented Models, [164](#)
- `vrna_md_defaults_gquad_get`
  - Fine-tuning of the Implemented Models, [164](#)
- `vrna_md_defaults_logML_get`

- Fine-tuning of the Implemented Models, [163](#)
- `vrna_md_defaults_logML`
  - Fine-tuning of the Implemented Models, [162](#)
- `vrna_md_defaults_max_bp_span`
  - Fine-tuning of the Implemented Models, [168](#)
- `vrna_md_defaults_max_bp_span_get`
  - Fine-tuning of the Implemented Models, [169](#)
- `vrna_md_defaults_min_loop_size`
  - Fine-tuning of the Implemented Models, [169](#)
- `vrna_md_defaults_min_loop_size_get`
  - Fine-tuning of the Implemented Models, [169](#)
- `vrna_md_defaults_nc_fact`
  - Fine-tuning of the Implemented Models, [173](#)
- `vrna_md_defaults_nc_fact_get`
  - Fine-tuning of the Implemented Models, [173](#)
- `vrna_md_defaults_noGU_get`
  - Fine-tuning of the Implemented Models, [161](#)
- `vrna_md_defaults_noGUclosure`
  - Fine-tuning of the Implemented Models, [161](#)
- `vrna_md_defaults_noGUclosure_get`
  - Fine-tuning of the Implemented Models, [162](#)
- `vrna_md_defaults_noGU`
  - Fine-tuning of the Implemented Models, [161](#)
- `vrna_md_defaults_noLP_get`
  - Fine-tuning of the Implemented Models, [160](#)
- `vrna_md_defaults_noLP`
  - Fine-tuning of the Implemented Models, [160](#)
- `vrna_md_defaults_oldAliEn`
  - Fine-tuning of the Implemented Models, [170](#)
- `vrna_md_defaults_oldAliEn_get`
  - Fine-tuning of the Implemented Models, [171](#)
- `vrna_md_defaults_reset`
  - Fine-tuning of the Implemented Models, [156](#)
- `vrna_md_defaults_ribo`
  - Fine-tuning of the Implemented Models, [171](#)
- `vrna_md_defaults_ribo_get`
  - Fine-tuning of the Implemented Models, [172](#)
- `vrna_md_defaults_sfact`
  - Fine-tuning of the Implemented Models, [173](#)
- `vrna_md_defaults_sfact_get`
  - Fine-tuning of the Implemented Models, [174](#)
- `vrna_md_defaults_special_hp`
  - Fine-tuning of the Implemented Models, [159](#)
- `vrna_md_defaults_special_hp_get`
  - Fine-tuning of the Implemented Models, [160](#)
- `vrna_md_defaults_temperature`
  - Fine-tuning of the Implemented Models, [157](#)
- `vrna_md_defaults_temperature_get`
  - Fine-tuning of the Implemented Models, [157](#)
- `vrna_md_defaults_uniq_ML_get`
  - Fine-tuning of the Implemented Models, [165](#)
- `vrna_md_defaults_uniq_ML`
  - Fine-tuning of the Implemented Models, [164](#)
- `vrna_md_defaults_window_size`
  - Fine-tuning of the Implemented Models, [170](#)
- `vrna_md_defaults_window_size_get`
  - Fine-tuning of the Implemented Models, [170](#)
- `vrna_md_option_string`
  - Fine-tuning of the Implemented Models, [156](#)
- `vrna_md_s`, [146](#)
  - `dangles`, [148](#)
  - `min_loop_size`, [149](#)
- `vrna_md_set_default`
  - Fine-tuning of the Implemented Models, [155](#)
- `vrna_md_update`
  - Fine-tuning of the Implemented Models, [155](#)
- `vrna_mean_bp_distance`
  - Global Partition Function and Equilibrium Probabilities, [267](#)
- `vrna_mean_bp_distance_pr`
  - Global Partition Function and Equilibrium Probabilities, [267](#)
- `vrna_message_constraint_options`
  - Constraining the RNA Folding Grammar, [220](#)
- `vrna_message_constraint_options_all`
  - Constraining the RNA Folding Grammar, [221](#)
- `vrna_message_error`
  - Messages, [551](#)
- `vrna_message_info`
  - Messages, [553](#)
- `vrna_message_input_seq`
  - Messages, [554](#)
- `vrna_message_input_seq_simple`
  - Messages, [554](#)
- `vrna_message_verror`
  - Messages, [552](#)
- `vrna_message_vinfo`
  - Messages, [554](#)
- `vrna_message_vwarning`
  - Messages, [553](#)
- `vrna_message_warning`
  - Messages, [552](#)
- `vrna_mfe`
  - Global MFE Prediction, [252](#)
- `vrna_mfe_TwoD`
  - Computing MFE representatives of a Distance Based Partitioning, [310](#)
- `vrna_mfe_dimer`
  - Global MFE Prediction, [252](#)
- `vrna_mfe_window`
  - Local (sliding window) MFE Prediction, [260](#)
- `vrna_mfe_window_callback`
  - Local (sliding window) MFE Prediction, [259](#)
- `vrna_mfe_window_zscore`
  - Local (sliding window) MFE Prediction, [260](#)
- `vrna_move_apply`
  - Neighborhood Relation and Move Sets for Secondary Structures, [329](#)
- `vrna_move_list_free`
  - Neighborhood Relation and Move Sets for Secondary Structures, [329](#)
- `vrna_move_s`, [327](#)
  - `next`, [328](#)
  - `pos_3`, [328](#)
  - `pos_5`, [328](#)
- `vrna_mx_add`

- The Dynamic Programming Matrices, [581](#)
- `vrna_mx_mfe_free`
  - The Dynamic Programming Matrices, [581](#)
- `vrna_mx_mfe_s`, [579](#)
- `vrna_mx_pf_aux_el_t`
  - Exterior Loops, [364](#)
- `vrna_mx_pf_aux_ml_t`
  - Multibranch Loops, [374](#)
- `vrna_mx_pf_free`
  - The Dynamic Programming Matrices, [582](#)
- `vrna_mx_pf_s`, [579](#)
- `vrna_mx_type_e`
  - The Dynamic Programming Matrices, [580](#)
- `vrna_neighbors`
  - Neighborhood Relation and Move Sets for Secondary Structures, [330](#)
- `vrna_neighbors_successive`
  - Neighborhood Relation and Move Sets for Secondary Structures, [331](#)
- `vrna_nucleotide_decode`
  - Utilities to deal with Nucleotide Alphabets, [435](#)
- `vrna_nucleotide_encode`
  - Utilities to deal with Nucleotide Alphabets, [435](#)
- `vrna_ostream_free`
  - Buffers, [597](#)
- `vrna_ostream_init`
  - Buffers, [597](#)
- `vrna_ostream_provide`
  - Buffers, [598](#)
- `vrna_ostream_request`
  - Buffers, [598](#)
- `vrna_param_s`, [179](#)
- `vrna_params`
  - Energy Parameters, [181](#)
- `vrna_params_copy`
  - Energy Parameters, [182](#)
- `vrna_params_reset`
  - Energy Parameters, [186](#)
- `vrna_params_subst`
  - Energy Parameters, [184](#)
- `vrna_path`
  - Refolding Paths of Secondary Structures, [335](#)
- `vrna_path_findpath`
  - Direct Refolding Paths between two Secondary Structures, [428](#)
- `vrna_path_findpath_saddle`
  - Direct Refolding Paths between two Secondary Structures, [426](#)
- `vrna_path_findpath_saddle_ub`
  - Direct Refolding Paths between two Secondary Structures, [427](#)
- `vrna_path_findpath_ub`
  - Direct Refolding Paths between two Secondary Structures, [428](#)
- `vrna_path_gradient`
  - Refolding Paths of Secondary Structures, [336](#)
- `vrna_path_random`
  - Refolding Paths of Secondary Structures, [336](#)
- `vrna_path_s`, [426](#)
- `vrna_pbacktrack`
  - Random Structure Samples from the Ensemble, [296](#)
- `vrna_pbacktrack5`
  - Random Structure Samples from the Ensemble, [295](#)
- `vrna_pbacktrack5_TwoD`
  - Stochastic Backtracking of Structures from Distance Based Partitioning, [319](#)
- `vrna_pbacktrack_TwoD`
  - Stochastic Backtracking of Structures from Distance Based Partitioning, [318](#)
- `vrna_pbacktrack_nr`
  - Random Structure Samples from the Ensemble, [296](#)
- `vrna_pf`
  - Global Partition Function and Equilibrium Probabilities, [270](#)
- `vrna_pf_TwoD`
  - Computing Partition Functions of a Distance Based Partitioning, [316](#)
- `vrna_pf_alifold`
  - Global Partition Function and Equilibrium Probabilities, [273](#)
- `vrna_pf_circalifold`
  - Global Partition Function and Equilibrium Probabilities, [274](#)
- `vrna_pf_circfold`
  - Global Partition Function and Equilibrium Probabilities, [272](#)
- `vrna_pf_co_fold`
  - Global Partition Function and Equilibrium Probabilities, [275](#)
  - Partition Function for Two Hybridized Sequences, [412](#)
- `vrna_pf_dimer`
  - Global Partition Function and Equilibrium Probabilities, [271](#)
- `vrna_pf_dimer_concentrations`
  - Partition Function for Two Hybridized Sequences, [412](#)
- `vrna_pf_dimer_probs`
  - Global Partition Function and Equilibrium Probabilities, [269](#)
- `vrna_pf_float_precision`
  - Partition Function and Equilibrium Properties, [250](#)
- `vrna_pf_fold`
  - Global Partition Function and Equilibrium Probabilities, [271](#)
- `vrna_pfl_fold`
  - Local (sliding window) Partition Function and Equilibrium Probabilities, [282](#)
- `vrna_pfl_fold_cb`
  - Local (sliding window) Partition Function and Equilibrium Probabilities, [283](#)
- `vrna_pfl_fold_up`
  - Local (sliding window) Partition Function and Equi-

- librium Probabilities, [283](#)
- `vrna_pfl_fold_up_cb`
  - Local (sliding window) Partition Function and Equilibrium Probabilities, [284](#)
- `vrna_pinfo_s`, [482](#)
- `vrna_plist`
  - Pair List Representation of Secondary Structures, [461](#)
- `vrna_plist_from_probs`
  - Global Partition Function and Equilibrium Probabilities, [275](#)
- `vrna_pr_structure`
  - Global Partition Function and Equilibrium Probabilities, [269](#)
- `vrna_probs_window`
  - Local (sliding window) Partition Function and Equilibrium Probabilities, [281](#)
- `vrna_probs_window_callback`
  - Local (sliding window) Partition Function and Equilibrium Probabilities, [280](#)
- `vrna_pt_pk_get`
  - Pair Table Representation of Secondary Structures, [458](#)
- `vrna_pt_snoop_get`
  - Pair Table Representation of Secondary Structures, [459](#)
- `vrna_ptable`
  - Pair Table Representation of Secondary Structures, [457](#)
- `vrna_ptable_copy`
  - Pair Table Representation of Secondary Structures, [459](#)
- `vrna_ptable_from_string`
  - Pair Table Representation of Secondary Structures, [458](#)
- `vrna_ptypes`
  - Utilities to deal with Nucleotide Alphabets, [434](#)
- `vrna_random_string`
  - (Nucleic Acid Sequence) String Utilities, [441](#)
- `vrna_read_line`
  - Files and I/O, [496](#)
- `vrna_realloc`
  - Utilities, [359](#)
- `vrna_refBPcnt_matrix`
  - Secondary Structure Utilities, [447](#)
- `vrna_refBPdist_matrix`
  - Secondary Structure Utilities, [447](#)
- `vrna_rotational_symmetry`
  - Combinatorics Algorithms, [540](#)
- `vrna_rotational_symmetry_db`
  - Combinatorics Algorithms, [542](#)
- `vrna_rotational_symmetry_db_pos`
  - Combinatorics Algorithms, [542](#)
- `vrna_rotational_symmetry_num`
  - Combinatorics Algorithms, [539](#)
- `vrna_rotational_symmetry_pos`
  - Combinatorics Algorithms, [541](#)
- `vrna_rotational_symmetry_pos_num`
  - Combinatorics Algorithms, [539](#)
- `vrna_sc_SHAPE_parse_method`
  - SHAPE.h, [629](#)
- `vrna_sc_SHAPE_to_pr`
  - SHAPE Reactivity Data, [342](#)
- `vrna_sc_add_SHAPE_deigan`
  - SHAPE Reactivity Data, [339](#)
- `vrna_sc_add_SHAPE_deigan_alii`
  - SHAPE Reactivity Data, [340](#)
- `vrna_sc_add_SHAPE_zarringhalam`
  - SHAPE Reactivity Data, [341](#)
- `vrna_sc_add_bp`
  - Soft Constraints, [241](#)
- `vrna_sc_add_bt`
  - Soft Constraints, [244](#)
- `vrna_sc_add_data`
  - Soft Constraints, [243](#)
- `vrna_sc_add_exp_f`
  - Soft Constraints, [245](#)
- `vrna_sc_add_f`
  - Soft Constraints, [244](#)
- `vrna_sc_add_hi_motif`
  - Incorporating Ligands Binding to Specific Sequence/Structure Motifs using Soft Constraints, [351](#)
- `vrna_sc_add_up`
  - Soft Constraints, [242](#)
- `vrna_sc_bp_storage_t`, [604](#)
- `vrna_sc_free`
  - Soft Constraints, [243](#)
- `vrna_sc_init`
  - Soft Constraints, [239](#)
- `vrna_sc_minimize_perturbation`
  - Generate Soft Constraints from Data, [346](#)
- `vrna_sc_motif_s`, [605](#)
- `vrna_sc_remove`
  - Soft Constraints, [243](#)
- `vrna_sc_s`, [235](#)
  - bt, [236](#)
  - exp\_f, [236](#)
  - f, [236](#)
- `vrna_sc_set_bp`
  - Soft Constraints, [240](#)
- `vrna_sc_set_up`
  - Soft Constraints, [241](#)
- `vrna_sc_type_e`
  - soft.h, [631](#)
- `vrna_search_BM_BCT_num`
  - Search Algorithms, [536](#)
- `vrna_search_BM_BCT`
  - Search Algorithms, [536](#)
- `vrna_search_BMH_num`
  - Search Algorithms, [534](#)
- `vrna_search_BMH`
  - Search Algorithms, [535](#)
- `vrna_sect_s`, [546](#)
- `vrna_seq_toRNA`
  - (Nucleic Acid Sequence) String Utilities, [443](#)

- vrna\_seq\_toupper
  - (Nucleic Acid Sequence) String Utilities, [443](#)
- vrna\_seq\_type\_e
  - Utilities to deal with Nucleotide Alphabets, [434](#)
- vrna\_seq\_ungapped
  - (Nucleic Acid Sequence) String Utilities, [444](#)
- vrna\_sequence\_s, [434](#)
- vrna\_sol\_TwoD\_pf\_t, [315](#)
  - Computing Partition Functions of a Distance Based Partitioning, [316](#)
- vrna\_sol\_TwoD\_t, [308](#)
  - Computing MFE representatives of a Distance Based Partitioning, [309](#)
- vrna\_stack\_prob
  - Global Partition Function and Equilibrium Probabilities, [268](#)
- vrna\_strcat\_printf
  - (Nucleic Acid Sequence) String Utilities, [439](#)
- vrna\_strcat\_vprintf
  - (Nucleic Acid Sequence) String Utilities, [440](#)
- vrna\_strdup\_printf
  - (Nucleic Acid Sequence) String Utilities, [438](#)
- vrna\_strdup\_vprintf
  - (Nucleic Acid Sequence) String Utilities, [439](#)
- vrna\_strsplit
  - (Nucleic Acid Sequence) String Utilities, [441](#)
- vrna\_structured\_domains\_s, [605](#)
- vrna\_subopt
  - Suboptimal Structures within an Energy Band around the MFE, [291](#)
- vrna\_subopt\_callback
  - Suboptimal Structures within an Energy Band around the MFE, [290](#)
- vrna\_subopt\_cb
  - Suboptimal Structures within an Energy Band around the MFE, [292](#)
- vrna\_subopt\_sol\_s, [605](#)
- vrna\_subopt\_zuker
  - Suboptimal Structures sensu Stiegler et al. 1984 / Zuker et al. 1989, [288](#)
- vrna\_time\_stamp
  - Utilities, [360](#)
- vrna\_tree\_string\_to\_db
  - Tree Representation of Secondary Structures, [467](#)
- vrna\_tree\_string\_unweight
  - Tree Representation of Secondary Structures, [467](#)
- vrna\_ud\_add\_motif
  - Unstructured Domains, [200](#)
- vrna\_ud\_motifs\_MEA
  - Unstructured Domains, [199](#)
- vrna\_ud\_motifs\_MFE
  - Unstructured Domains, [199](#)
- vrna\_ud\_motifs\_centroid
  - Unstructured Domains, [198](#)
- vrna\_ud\_remove
  - Unstructured Domains, [201](#)
- vrna\_ud\_set\_data
  - Unstructured Domains, [201](#)
- vrna\_ud\_set\_exp\_prod\_rule\_cb
  - Unstructured Domains, [203](#)
- vrna\_ud\_set\_prob\_cb
  - unstructured\_domains.h, [713](#)
- vrna\_ud\_set\_prod\_rule\_cb
  - Unstructured Domains, [202](#)
- vrna\_unit\_energy\_e
  - Unit Conversion, [557](#)
- vrna\_unit\_temperature\_e
  - Unit Conversion, [558](#)
- vrna\_unstructured\_domain\_motif\_s, [605](#)
- vrna\_unstructured\_domain\_s, [195](#)
  - prod\_cb, [196](#)
- vrna\_urn
  - Utilities, [359](#)
- warn\_user
  - utils/basic.h, [675](#)
- write\_parameter\_file
  - Reading/Writing Energy Parameter Sets from/to File, [418](#)
- xrealloc
  - utils/basic.h, [676](#)
- xrna\_plot
  - Plotting, [530](#)
- xsubi
  - Utilities, [362](#)
- zukersubopt
  - Suboptimal Structures sensu Stiegler et al. 1984 / Zuker et al. 1989, [289](#)
- zukersubopt\_par
  - Suboptimal Structures sensu Stiegler et al. 1984 / Zuker et al. 1989, [289](#)