

RNAlib-2.4.17

Generated by Doxygen 1.8.20



<b>1 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 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 Concepts and Algorithms</b>	<b>13</b>
3.1 RNA Structure	14
3.1.1 RNA Structures	14
3.1.2 Levels of Structure Abstraction	14
3.1.2.1 Primary Structure	14
3.1.2.2 Secondary Structure	14
3.1.2.3 Tertiary Structure	14

3.1.2.4 Quarternary Structure . . . . .	14
3.1.2.5 Pseudo-Knots . . . . .	14
3.2 Distance Measures . . . . .	14
3.2.1 Functions for Tree Edit Distances . . . . .	15
3.2.2 Functions for String Alignment . . . . .	16
3.2.3 Functions for Comparison of Base Pair Probabilities . . . . .	16
3.3 Free Energy of Secondary Structures . . . . .	16
3.3.1 Secondary Structure Loop Decomposition . . . . .	17
3.3.1.1 Free Energy Evaluation API . . . . .	18
3.3.2 Free Energy Parameters . . . . .	18
3.3.2.1 Free Energy Parameters Modification API . . . . .	18
3.3.3 Fine-tuning of the Energy Evaluation Model . . . . .	18
3.4 Secondary Structure Folding Grammar . . . . .	18
3.4.1 Secondary Structure Folding Recurrences . . . . .	19
3.4.2 Additional Structural Domains . . . . .	19
3.4.2.1 Structured Domains . . . . .	20
3.4.2.2 Unstructured Domains . . . . .	20
3.4.2.3 Domain Extension API . . . . .	21
3.4.3 Constraints on the Folding Grammar . . . . .	21
3.4.3.1 Hard Constraints API . . . . .	21
3.4.3.2 Soft Constraints API . . . . .	22
3.5 RNA Secondary Structure Landscapes . . . . .	22
3.5.1 The Neighborhood of a Secondary Structure . . . . .	22
3.5.2 The Secondary Structure Landscape API . . . . .	22
3.6 Minimum Free Energy Algorithm(s) . . . . .	22
3.6.1 Zuker's Algorithm . . . . .	22
3.6.2 MFE for circular RNAs . . . . .	22
3.6.3 MFE Algorithm API . . . . .	22
3.7 Partition Function and Equilibrium Probability Algorithm(s) . . . . .	23
3.7.1 Equilibrium Ensemble Statistics . . . . .	23
3.7.2 Partition Function and Equilibrium Probability API . . . . .	23
3.8 Suboptimals and (other) Representative Structures . . . . .	24
3.8.1 Suboptimal Secondary Structures . . . . .	24
3.8.2 Sampling Secondary Structures from the Ensemble . . . . .	24
3.8.3 Structure Enumeration and Sampling API . . . . .	24
3.9 RNA-RNA Interaction . . . . .	24
3.9.1 rip_intro . . . . .	24
3.9.2 Concatenating RNA sequences . . . . .	24
3.9.3 RNA-RNA interaction as a Stepwise Process . . . . .	24
3.9.4 RNA-RNA Interaction API . . . . .	25
3.10 Locally Stable Secondary Structures . . . . .	25
3.10.1 local_intro . . . . .	25

3.10.2 local_mfe . . . . .	25
3.10.3 local_pf . . . . .	25
3.10.4 Locally Stable Secondary Structure API . . . . .	25
3.11 Comparative Structure Prediction . . . . .	25
3.11.1 Incorporate Evolutionary Information . . . . .	25
3.11.2 Comparative Structure Prediction API . . . . .	25
3.12 Classified DP variations . . . . .	25
3.12.1 The Idea of Classified Dynamic Programming . . . . .	25
3.12.2 Distance Class Partitioning . . . . .	25
3.12.3 Density of States (DOS) . . . . .	26
3.12.4 Classified DP API . . . . .	26
3.13 RNA Sequence Design . . . . .	26
3.13.1 Generate Sequences that fold into particular Secondary Structures . . . . .	26
3.13.2 RNA Sequence Design API . . . . .	26
3.14 Experimental Structure Probing Data . . . . .	26
3.14.1 Guide the Structure Prediction using Experimental Data . . . . .	26
3.14.1.1 SHAPE reactivities . . . . .	26
3.14.2 Structure Probing Data API . . . . .	26
3.15 Ligand Binding . . . . .	26
3.15.1 Small Molecules and Proteins that bind to specific RNA Structures . . . . .	26
3.15.2 ligand_binding_api . . . . .	26
3.16 (Tertiary) Structure Motifs . . . . .	27
3.16.1 Incorporating Higher-Order (Tertiary) Structure Motifs . . . . .	27
3.16.2 RNA G-Quadruplexes . . . . .	27
3.16.3 (Tertiary) Structure Motif API . . . . .	27
<b>4 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	41
<b>5 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 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?	45
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 Additional Utilities</b>	<b>59</b>
<b>8 Examples</b>	<b>61</b>
8.1 C Examples	61
8.1.1 Hello World Examples	61
8.1.2 First Steps with the Fold Compound	63
8.1.3 Writing Callback Functions	64
8.1.4 Application of Soft Constraints	64
8.1.5 Other Examples	65
8.1.6 Deprecated Examples	66
8.2 Perl5 Examples	67
8.3 Python Examples	68
<b>9 Contributing</b>	<b>71</b>
<b>10 Changelog</b>	<b>73</b>

<b>11 Deprecated List</b>	<b>105</b>
<b>12 Bug List</b>	<b>117</b>
<b>13 Module Index</b>	<b>119</b>
13.1 The RNAlib API . . . . .	119
<b>14 Data Structure Index</b>	<b>123</b>
14.1 Data Structures . . . . .	123
<b>15 File Index</b>	<b>125</b>
15.1 File List . . . . .	125
<b>16 Module Documentation</b>	<b>131</b>
16.1 Free Energy Evaluation . . . . .	131
16.1.1 Detailed Description . . . . .	131
16.1.2 Function Documentation . . . . .	135
16.1.2.1 vrna_eval_structure() . . . . .	135
16.1.2.2 vrna_eval_covar_structure() . . . . .	135
16.1.2.3 vrna_eval_structure_verbose() . . . . .	136
16.1.2.4 vrna_eval_structure_v() . . . . .	137
16.1.2.5 vrna_eval_structure_pt() . . . . .	138
16.1.2.6 vrna_eval_structure_pt_verbose() . . . . .	138
16.1.2.7 vrna_eval_structure_pt_v() . . . . .	139
16.1.2.8 vrna_eval_structure_simple() . . . . .	140
16.1.2.9 vrna_eval_circ_structure() . . . . .	140
16.1.2.10 vrna_eval_gquad_structure() . . . . .	141
16.1.2.11 vrna_eval_circ_gquad_structure() . . . . .	142
16.1.2.12 vrna_eval_structure_simple_verbose() . . . . .	142
16.1.2.13 vrna_eval_structure_simple_v() . . . . .	143
16.1.2.14 vrna_eval_circ_structure_v() . . . . .	144
16.1.2.15 vrna_eval_gquad_structure_v() . . . . .	144
16.1.2.16 vrna_eval_circ_gquad_structure_v() . . . . .	145
16.1.2.17 vrna_eval_consensus_structure_simple() . . . . .	146
16.1.2.18 vrna_eval_circ_consensus_structure() . . . . .	147
16.1.2.19 vrna_eval_gquad_consensus_structure() . . . . .	147
16.1.2.20 vrna_eval_circ_gquad_consensus_structure() . . . . .	148
16.1.2.21 vrna_eval_consensus_structure_simple_verbose() . . . . .	149
16.1.2.22 vrna_eval_consensus_structure_simple_v() . . . . .	150
16.1.2.23 vrna_eval_circ_consensus_structure_v() . . . . .	151
16.1.2.24 vrna_eval_gquad_consensus_structure_v() . . . . .	151
16.1.2.25 vrna_eval_circ_gquad_consensus_structure_v() . . . . .	152
16.1.2.26 vrna_eval_structure_pt_simple() . . . . .	153
16.1.2.27 vrna_eval_structure_pt_simple_verbose() . . . . .	154

16.1.2.28 vrna_eval_structure_pt_simple_v()	154
16.1.2.29 vrna_eval_consensus_structure_pt_simple()	155
16.2 Energy Evaluation for Individual Loops	157
16.2.1 Detailed Description	157
16.2.2 Function Documentation	158
16.2.2.1 vrna_eval_loop_pt()	158
16.2.2.2 vrna_eval_loop_pt_v()	159
16.3 Energy Evaluation for Atomic Moves	160
16.3.1 Detailed Description	160
16.3.2 Function Documentation	160
16.3.2.1 vrna_eval_move()	161
16.3.2.2 vrna_eval_move_pt()	161
16.4 Deprecated Interface for Free Energy Evaluation	163
16.4.1 Detailed Description	163
16.4.2 Function Documentation	164
16.4.2.1 energy_of_structure()	164
16.4.2.2 energy_of_struct_par()	165
16.4.2.3 energy_of_circ_structure()	165
16.4.2.4 energy_of_circ_struct_par()	166
16.4.2.5 energy_of_structure_pt()	166
16.4.2.6 energy_of_struct_pt_par()	167
16.4.2.7 energy_of_move()	168
16.4.2.8 energy_of_move_pt()	169
16.4.2.9 loop_energy()	169
16.4.2.10 energy_of_struct()	170
16.4.2.11 energy_of_struct_pt()	171
16.4.2.12 energy_of_circ_struct()	171
16.4.2.13 E_Stem()	172
16.4.2.14 exp_E_ExtLoop()	173
16.4.2.15 exp_E_Stem()	174
16.4.2.16 E_IntLoop()	174
16.4.2.17 exp_E_IntLoop()	176
16.5 The RNA Folding Grammar	178
16.5.1 Detailed Description	178
16.5.2 Data Structure Documentation	179
16.5.2.1 struct vrna_gr_aux_s	179
16.6 Fine-tuning of the Implemented Models	180
16.6.1 Detailed Description	180
16.6.2 Data Structure Documentation	184
16.6.2.1 struct vrna_md_s	184
16.6.3 Macro Definition Documentation	187
16.6.3.1 VRNA_MODEL_DEFAULT_TEMPERATURE	187



16.6.3.2 VRNA_MODEL_DEFAULT_PF_SCALE . . . . .	188
16.6.3.3 VRNA_MODEL_DEFAULT_BETA_SCALE . . . . .	188
16.6.3.4 VRNA_MODEL_DEFAULT_DANGLES . . . . .	188
16.6.3.5 VRNA_MODEL_DEFAULT_SPECIAL_HP . . . . .	188
16.6.3.6 VRNA_MODEL_DEFAULT_NO_LP . . . . .	189
16.6.3.7 VRNA_MODEL_DEFAULT_NO_GU . . . . .	189
16.6.3.8 VRNA_MODEL_DEFAULT_NO_GU_CLOSURE . . . . .	189
16.6.3.9 VRNA_MODEL_DEFAULT_CIRC . . . . .	189
16.6.3.10 VRNA_MODEL_DEFAULT_GQUAD . . . . .	190
16.6.3.11 VRNA_MODEL_DEFAULT_UNIQ_ML . . . . .	190
16.6.3.12 VRNA_MODEL_DEFAULT_ENERGY_SET . . . . .	190
16.6.3.13 VRNA_MODEL_DEFAULT_BACKTRACK . . . . .	190
16.6.3.14 VRNA_MODEL_DEFAULT_BACKTRACK_TYPE . . . . .	191
16.6.3.15 VRNA_MODEL_DEFAULT_COMPUTE_BPP . . . . .	191
16.6.3.16 VRNA_MODEL_DEFAULT_MAX_BP_SPAN . . . . .	191
16.6.3.17 VRNA_MODEL_DEFAULT_WINDOW_SIZE . . . . .	191
16.6.3.18 VRNA_MODEL_DEFAULT_LOG_ML . . . . .	192
16.6.3.19 VRNA_MODEL_DEFAULT_ALI_OLD_EN . . . . .	192
16.6.3.20 VRNA_MODEL_DEFAULT_ALI_RIBO . . . . .	192
16.6.3.21 VRNA_MODEL_DEFAULT_ALI_CV_FACT . . . . .	192
16.6.3.22 VRNA_MODEL_DEFAULT_ALI_NC_FACT . . . . .	193
16.6.4 Function Documentation . . . . .	193
16.6.4.1 vrna_md_set_default() . . . . .	193
16.6.4.2 vrna_md_update() . . . . .	193
16.6.4.3 vrna_md_copy() . . . . .	194
16.6.4.4 vrna_md_option_string() . . . . .	194
16.6.4.5 vrna_md_defaults_reset() . . . . .	194
16.6.4.6 vrna_md_defaults_temperature() . . . . .	195
16.6.4.7 vrna_md_defaults_temperature_get() . . . . .	195
16.6.4.8 vrna_md_defaults_betaScale() . . . . .	196
16.6.4.9 vrna_md_defaults_betaScale_get() . . . . .	196
16.6.4.10 vrna_md_defaults_dangles() . . . . .	197
16.6.4.11 vrna_md_defaults_dangles_get() . . . . .	197
16.6.4.12 vrna_md_defaults_special_hp() . . . . .	197
16.6.4.13 vrna_md_defaults_special_hp_get() . . . . .	198
16.6.4.14 vrna_md_defaults_noLP() . . . . .	198
16.6.4.15 vrna_md_defaults_noLP_get() . . . . .	198
16.6.4.16 vrna_md_defaults_noGU() . . . . .	199
16.6.4.17 vrna_md_defaults_noGU_get() . . . . .	199
16.6.4.18 vrna_md_defaults_noGUclosure() . . . . .	200
16.6.4.19 vrna_md_defaults_noGUclosure_get() . . . . .	200
16.6.4.20 vrna_md_defaults_logML() . . . . .	200

16.6.4.21	<a href="#">vrna_md_defaults_logML_get()</a>	201
16.6.4.22	<a href="#">vrna_md_defaults_circ()</a>	201
16.6.4.23	<a href="#">vrna_md_defaults_circ_get()</a>	201
16.6.4.24	<a href="#">vrna_md_defaults_gquad()</a>	202
16.6.4.25	<a href="#">vrna_md_defaults_gquad_get()</a>	202
16.6.4.26	<a href="#">vrna_md_defaults_uniq_ML()</a>	203
16.6.4.27	<a href="#">vrna_md_defaults_uniq_ML_get()</a>	203
16.6.4.28	<a href="#">vrna_md_defaults_energy_set()</a>	203
16.6.4.29	<a href="#">vrna_md_defaults_energy_set_get()</a>	204
16.6.4.30	<a href="#">vrna_md_defaults_backtrack()</a>	204
16.6.4.31	<a href="#">vrna_md_defaults_backtrack_get()</a>	204
16.6.4.32	<a href="#">vrna_md_defaults_backtrack_type()</a>	205
16.6.4.33	<a href="#">vrna_md_defaults_backtrack_type_get()</a>	205
16.6.4.34	<a href="#">vrna_md_defaults_compute_bpp()</a>	206
16.6.4.35	<a href="#">vrna_md_defaults_compute_bpp_get()</a>	206
16.6.4.36	<a href="#">vrna_md_defaults_max_bp_span()</a>	206
16.6.4.37	<a href="#">vrna_md_defaults_max_bp_span_get()</a>	207
16.6.4.38	<a href="#">vrna_md_defaults_min_loop_size()</a>	207
16.6.4.39	<a href="#">vrna_md_defaults_min_loop_size_get()</a>	207
16.6.4.40	<a href="#">vrna_md_defaults_window_size()</a>	208
16.6.4.41	<a href="#">vrna_md_defaults_window_size_get()</a>	208
16.6.4.42	<a href="#">vrna_md_defaults_oldAliEn()</a>	209
16.6.4.43	<a href="#">vrna_md_defaults_oldAliEn_get()</a>	209
16.6.4.44	<a href="#">vrna_md_defaults_ribo()</a>	209
16.6.4.45	<a href="#">vrna_md_defaults_ribo_get()</a>	210
16.6.4.46	<a href="#">vrna_md_defaults_cv_fact()</a>	210
16.6.4.47	<a href="#">vrna_md_defaults_cv_fact_get()</a>	210
16.6.4.48	<a href="#">vrna_md_defaults_nc_fact()</a>	211
16.6.4.49	<a href="#">vrna_md_defaults_nc_fact_get()</a>	211
16.6.4.50	<a href="#">vrna_md_defaults_sfact()</a>	212
16.6.4.51	<a href="#">vrna_md_defaults_sfact_get()</a>	212
16.6.4.52	<a href="#">set_model_details()</a>	212
16.6.5	<a href="#">Variable Documentation</a>	213
16.6.5.1	<a href="#">temperature</a>	213
16.6.5.2	<a href="#">pf_scale</a>	213
16.6.5.3	<a href="#">dangles</a>	214
16.6.5.4	<a href="#">tetra_loop</a>	214
16.6.5.5	<a href="#">noLonelyPairs</a>	214
16.6.5.6	<a href="#">energy_set</a>	214
16.6.5.7	<a href="#">do_backtrack</a>	215
16.6.5.8	<a href="#">backtrack_type</a>	215
16.6.5.9	<a href="#">nonstandards</a>	215

16.6.5.10 max_bp_span . . . . .	215
16.7 Energy Parameters . . . . .	216
16.7.1 Detailed Description . . . . .	216
16.7.2 Data Structure Documentation . . . . .	218
16.7.2.1 struct vrna_param_s . . . . .	218
16.7.2.2 struct vrna_exp_param_s . . . . .	218
16.7.3 Typedef Documentation . . . . .	219
16.7.3.1 paramT . . . . .	219
16.7.3.2 pf_paramT . . . . .	220
16.7.4 Function Documentation . . . . .	220
16.7.4.1 vrna_params() . . . . .	220
16.7.4.2 vrna_params_copy() . . . . .	220
16.7.4.3 vrna_exp_params() . . . . .	221
16.7.4.4 vrna_exp_params_comparative() . . . . .	222
16.7.4.5 vrna_exp_params_copy() . . . . .	222
16.7.4.6 vrna_params_subst() . . . . .	223
16.7.4.7 vrna_exp_params_subst() . . . . .	223
16.7.4.8 vrna_exp_params_rescale() . . . . .	224
16.7.4.9 vrna_params_reset() . . . . .	225
16.7.4.10 vrna_exp_params_reset() . . . . .	226
16.7.4.11 get_scaled_pf_parameters() . . . . .	226
16.7.4.12 get_boltzmann_factors() . . . . .	227
16.7.4.13 get_boltzmann_factor_copy() . . . . .	227
16.7.4.14 get_scaled_alipf_parameters() . . . . .	228
16.7.4.15 get_boltzmann_factors_ali() . . . . .	228
16.7.4.16 scale_parameters() . . . . .	229
16.7.4.17 get_scaled_parameters() . . . . .	229
16.8 Extending the Folding Grammar with Additional Domains . . . . .	231
16.8.1 Detailed Description . . . . .	231
16.9 Unstructured Domains . . . . .	232
16.9.1 Detailed Description . . . . .	232
16.9.2 Data Structure Documentation . . . . .	234
16.9.2.1 struct vrna_unstructured_domain_s . . . . .	234
16.9.3 Typedef Documentation . . . . .	235
16.9.3.1 vrna_callback_ud_energy . . . . .	235
16.9.3.2 vrna_callback_ud_exp_energy . . . . .	236
16.9.3.3 vrna_callback_ud_production . . . . .	236
16.9.3.4 vrna_callback_ud_exp_production . . . . .	237
16.9.3.5 vrna_callback_ud_probs_add . . . . .	237
16.9.3.6 vrna_callback_ud_probs_get . . . . .	237
16.9.4 Function Documentation . . . . .	237
16.9.4.1 vrna_ud_motifs_centroid() . . . . .	237

16.9.4.2 vrna_ud_motifs_MEA()	238
16.9.4.3 vrna_ud_motifs_MFE()	238
16.9.4.4 vrna_ud_add_motif()	239
16.9.4.5 vrna_ud_remove()	240
16.9.4.6 vrna_ud_set_data()	240
16.9.4.7 vrna_ud_set_prod_rule_cb()	241
16.9.4.8 vrna_ud_set_exp_prod_rule_cb()	242
16.10 Structured Domains	244
16.10.1 Detailed Description	244
16.11 Constraining the RNA Folding Grammar	245
16.11.1 Detailed Description	245
16.11.2 Macro Definition Documentation	248
16.11.2.1 VRNA_CONSTRAINT_FILE	248
16.11.2.2 VRNA_CONSTRAINT_SOFT_MFE	249
16.11.2.3 VRNA_CONSTRAINT_SOFT_PF	249
16.11.2.4 VRNA_DECOMP_PAIR_HP	249
16.11.2.5 VRNA_DECOMP_PAIR_IL	250
16.11.2.6 VRNA_DECOMP_PAIR_ML	250
16.11.2.7 VRNA_DECOMP_ML_ML_ML	251
16.11.2.8 VRNA_DECOMP_ML_STEM	252
16.11.2.9 VRNA_DECOMP_ML_ML	252
16.11.2.10 VRNA_DECOMP_ML_UP	253
16.11.2.11 VRNA_DECOMP_ML_ML_STEM	253
16.11.2.12 VRNA_DECOMP_ML_COAXIAL	254
16.11.2.13 VRNA_DECOMP_ML_COAXIAL_ENC	254
16.11.2.14 VRNA_DECOMP_EXT_EXT	255
16.11.2.15 VRNA_DECOMP_EXT_UP	255
16.11.2.16 VRNA_DECOMP_EXT_STEM	256
16.11.2.17 VRNA_DECOMP_EXT_EXT_EXT	256
16.11.2.18 VRNA_DECOMP_EXT_STEM_EXT	257
16.11.2.19 VRNA_DECOMP_EXT_EXT_STEM	257
16.11.2.20 VRNA_DECOMP_EXT_EXT_STEM1	258
16.11.3 Function Documentation	258
16.11.3.1 vrna_constraints_add()	258
16.11.3.2 vrna_message_constraint_options()	259
16.11.3.3 vrna_message_constraint_options_all()	261
16.12 Hard Constraints	262
16.12.1 Detailed Description	262
16.12.2 Data Structure Documentation	264
16.12.2.1 struct vrna_hc_s	264
16.12.2.2 struct vrna_hc_up_s	265
16.12.3 Macro Definition Documentation	265

16.12.3.1 VRNA_CONSTRAINT_DB . . . . .	265
16.12.3.2 VRNA_CONSTRAINT_DB_ENFORCE_BP . . . . .	265
16.12.3.3 VRNA_CONSTRAINT_DB_PIPE . . . . .	266
16.12.3.4 VRNA_CONSTRAINT_DB_DOT . . . . .	266
16.12.3.5 VRNA_CONSTRAINT_DB_X . . . . .	266
16.12.3.6 VRNA_CONSTRAINT_DB_RND_BRACK . . . . .	267
16.12.3.7 VRNA_CONSTRAINT_DB_INTRAMOL . . . . .	267
16.12.3.8 VRNA_CONSTRAINT_DB_INTERMOL . . . . .	267
16.12.3.9 VRNA_CONSTRAINT_DB_GQUAD . . . . .	268
16.12.3.10 VRNA_CONSTRAINT_DB_WUSS . . . . .	268
16.12.3.11 VRNA_CONSTRAINT_DB_DEFAULT . . . . .	268
16.12.4 Typedef Documentation . . . . .	269
16.12.4.1 vrna_callback_hc_evaluate . . . . .	269
16.12.5 Function Documentation . . . . .	269
16.12.5.1 vrna_hc_init() . . . . .	270
16.12.5.2 vrna_hc_add_up() . . . . .	270
16.12.5.3 vrna_hc_add_up_batch() . . . . .	271
16.12.5.4 vrna_hc_add_bp() . . . . .	271
16.12.5.5 vrna_hc_add_bp_nonspecific() . . . . .	271
16.12.5.6 vrna_hc_free() . . . . .	272
16.12.5.7 vrna_hc_add_from_db() . . . . .	272
16.13 Soft Constraints . . . . .	274
16.13.1 Detailed Description . . . . .	274
16.13.2 Data Structure Documentation . . . . .	275
16.13.2.1 struct vrna_sc_s . . . . .	275
16.13.3 Typedef Documentation . . . . .	277
16.13.3.1 vrna_callback_sc_energy . . . . .	277
16.13.3.2 vrna_callback_sc_exp_energy . . . . .	278
16.13.3.3 vrna_callback_sc_backtrack . . . . .	279
16.13.4 Function Documentation . . . . .	279
16.13.4.1 vrna_sc_init() . . . . .	279
16.13.4.2 vrna_sc_set_bp() . . . . .	280
16.13.4.3 vrna_sc_add_bp() . . . . .	281
16.13.4.4 vrna_sc_set_up() . . . . .	281
16.13.4.5 vrna_sc_add_up() . . . . .	282
16.13.4.6 vrna_sc_remove() . . . . .	283
16.13.4.7 vrna_sc_free() . . . . .	283
16.13.4.8 vrna_sc_add_data() . . . . .	283
16.13.4.9 vrna_sc_add_f() . . . . .	284
16.13.4.10 vrna_sc_add_bt() . . . . .	284
16.13.4.11 vrna_sc_add_exp_f() . . . . .	285
16.14 The RNA Secondary Structure Landscape . . . . .	286

16.14.1 Detailed Description	286
16.15 Minimum Free Energy (MFE) Algorithms	287
16.15.1 Detailed Description	287
16.16 Partition Function and Equilibrium Properties	288
16.16.1 Detailed Description	288
16.16.2 Function Documentation	289
16.16.2.1 vrna_pf_float_precision()	289
16.17 Global MFE Prediction	290
16.17.1 Detailed Description	290
16.17.2 Function Documentation	291
16.17.2.1 vrna_mfe()	291
16.17.2.2 vrna_mfe_dimer()	292
16.17.2.3 vrna_fold()	292
16.17.2.4 vrna_circfold()	293
16.17.2.5 vrna_alifold()	294
16.17.2.6 vrna_circalifold()	295
16.17.2.7 vrna_cofold()	295
16.18 Local (sliding window) MFE Prediction	297
16.18.1 Detailed Description	297
16.18.2 Typedef Documentation	298
16.18.2.1 vrna_mfe_window_callback	298
16.18.3 Function Documentation	299
16.18.3.1 vrna_mfe_window()	299
16.18.3.2 vrna_mfe_window_zscore()	300
16.18.3.3 vrna_Lfold()	300
16.18.3.4 vrna_Lfoldz()	301
16.19 Backtracking MFE structures	302
16.19.1 Detailed Description	302
16.19.2 Function Documentation	302
16.19.2.1 vrna_backtrack5()	303
16.19.2.2 vrna_BT_hp_loop()	303
16.19.2.3 vrna_BT_mb_loop()	304
16.20 Global Partition Function and Equilibrium Probabilities	305
16.20.1 Detailed Description	305
16.20.2 Data Structure Documentation	306
16.20.2.1 struct vrna_dimer_pf_s	306
16.20.3 Function Documentation	307
16.20.3.1 vrna_pf()	307
16.20.3.2 vrna_pf_dimer()	307
16.20.3.3 vrna_pf_fold()	308
16.20.3.4 vrna_pf_circfold()	309
16.20.3.5 vrna_pf_alifold()	310

16.20.3.6	<a href="#">vrna_pf_circalifold()</a>	310
16.20.3.7	<a href="#">vrna_plist_from_probs()</a>	311
16.20.3.8	<a href="#">vrna_pf_co_fold()</a>	312
16.21	<a href="#">Local (sliding window) Partition Function and Equilibrium Probabilities</a>	313
16.21.1	<a href="#">Detailed Description</a>	313
16.21.2	<a href="#">Macro Definition Documentation</a>	314
16.21.2.1	<a href="#">VRNA_PROBS_WINDOW_BPP</a>	314
16.21.2.2	<a href="#">VRNA_PROBS_WINDOW_UP</a>	315
16.21.2.3	<a href="#">VRNA_PROBS_WINDOW_STACKP</a>	315
16.21.2.4	<a href="#">VRNA_PROBS_WINDOW_UP_SPLIT</a>	315
16.21.2.5	<a href="#">VRNA_PROBS_WINDOW_PF</a>	316
16.21.3	<a href="#">Typedef Documentation</a>	316
16.21.3.1	<a href="#">vrna_probs_window_callback</a>	316
16.21.4	<a href="#">Function Documentation</a>	317
16.21.4.1	<a href="#">vrna_probs_window()</a>	317
16.21.4.2	<a href="#">vrna_pfl_fold()</a>	318
16.21.4.3	<a href="#">vrna_pfl_fold_cb()</a>	319
16.21.4.4	<a href="#">vrna_pfl_fold_up()</a>	319
16.21.4.5	<a href="#">vrna_pfl_fold_up_cb()</a>	320
16.22	<a href="#">Suboptimals and Representative Structures</a>	323
16.22.1	<a href="#">Detailed Description</a>	323
16.23	<a href="#">Suboptimal Structures sensu Stiegler et al. 1984 / Zuker et al. 1989</a>	324
16.23.1	<a href="#">Detailed Description</a>	324
16.23.2	<a href="#">Function Documentation</a>	324
16.23.2.1	<a href="#">vrna_subopt_zuker()</a>	324
16.23.2.2	<a href="#">zukersubopt()</a>	325
16.23.2.3	<a href="#">zukersubopt_par()</a>	325
16.24	<a href="#">Suboptimal Structures within an Energy Band around the MFE</a>	326
16.24.1	<a href="#">Detailed Description</a>	326
16.24.2	<a href="#">Typedef Documentation</a>	326
16.24.2.1	<a href="#">vrna_subopt_callback</a>	327
16.24.3	<a href="#">Function Documentation</a>	327
16.24.3.1	<a href="#">vrna_subopt()</a>	327
16.24.3.2	<a href="#">vrna_subopt_cb()</a>	328
16.24.3.3	<a href="#">subopt()</a>	329
16.24.3.4	<a href="#">subopt_circ()</a>	329
16.25	<a href="#">Random Structure Samples from the Ensemble</a>	331
16.25.1	<a href="#">Detailed Description</a>	331
16.25.2	<a href="#">Macro Definition Documentation</a>	332
16.25.2.1	<a href="#">VRNA_PBACKTRACK_DEFAULT</a>	332
16.25.2.2	<a href="#">VRNA_PBACKTRACK_NON_REDUNDANT</a>	333
16.25.3	<a href="#">Typedef Documentation</a>	333

16.25.3.1 vrna_boltzmann_sampling_callback . . . . .	333
16.25.3.2 vrna_pbacktrack_mem_t . . . . .	333
16.25.4 Function Documentation . . . . .	334
16.25.4.1 vrna_pbacktrack5() . . . . .	334
16.25.4.2 vrna_pbacktrack5_num() . . . . .	335
16.25.4.3 vrna_pbacktrack5_cb() . . . . .	336
16.25.4.4 vrna_pbacktrack5_resume() . . . . .	337
16.25.4.5 vrna_pbacktrack5_resume_cb() . . . . .	339
16.25.4.6 vrna_pbacktrack() . . . . .	341
16.25.4.7 vrna_pbacktrack_num() . . . . .	342
16.25.4.8 vrna_pbacktrack_cb() . . . . .	343
16.25.4.9 vrna_pbacktrack_resume() . . . . .	344
16.25.4.10 vrna_pbacktrack_resume_cb() . . . . .	345
16.25.4.11 vrna_pbacktrack_mem_free() . . . . .	347
16.26 Compute the Structure with Maximum Expected Accuracy (MEA) . . . . .	348
16.26.1 Detailed Description . . . . .	348
16.26.2 Function Documentation . . . . .	348
16.26.2.1 vrna_MEA() . . . . .	348
16.26.2.2 vrna_MEA_from_plist() . . . . .	349
16.26.2.3 MEA() . . . . .	350
16.27 Compute the Centroid Structure . . . . .	351
16.27.1 Detailed Description . . . . .	351
16.27.2 Function Documentation . . . . .	351
16.27.2.1 vrna_centroid() . . . . .	351
16.27.2.2 vrna_centroid_from_plist() . . . . .	352
16.27.2.3 vrna_centroid_from_probs() . . . . .	352
16.28 RNA-RNA Interaction . . . . .	354
16.28.1 Detailed Description . . . . .	354
16.29 Classified Dynamic Programming Variants . . . . .	355
16.29.1 Detailed Description . . . . .	355
16.30 Distance Based Partitioning of the Secondary Structure Space . . . . .	356
16.30.1 Detailed Description . . . . .	356
16.31 Computing MFE representatives of a Distance Based Partitioning . . . . .	357
16.31.1 Detailed Description . . . . .	357
16.31.2 Data Structure Documentation . . . . .	358
16.31.2.1 struct vrna_sol_TwoD_t . . . . .	358
16.31.2.2 struct TwoDfold_vars . . . . .	359
16.31.3 Typedef Documentation . . . . .	360
16.31.3.1 vrna_sol_TwoD_t . . . . .	360
16.31.3.2 TwoDfold_vars . . . . .	360
16.31.4 Function Documentation . . . . .	360
16.31.4.1 vrna_mfe_TwoD() . . . . .	361



16.31.4.2 vrna_backtrack5_TwoD()	361
16.31.4.3 get_TwoDfold_variables()	362
16.31.4.4 destroy_TwoDfold_variables()	363
16.31.4.5 TwoDfoldList()	363
16.31.4.6 TwoDfold_backtrack_f5()	364
16.32 Computing Partition Functions of a Distance Based Partitioning	365
16.32.1 Detailed Description	365
16.32.2 Data Structure Documentation	365
16.32.2.1 struct vrna_sol_TwoD_pf_t	365
16.32.3 Typedef Documentation	366
16.32.3.1 vrna_sol_TwoD_pf_t	366
16.32.4 Function Documentation	366
16.32.4.1 vrna_pf_TwoD()	366
16.33 Stochastic Backtracking of Structures from Distance Based Partitioning	368
16.33.1 Detailed Description	368
16.33.2 Function Documentation	368
16.33.2.1 vrna_pbacktrack_TwoD()	368
16.33.2.2 vrna_pbacktrack5_TwoD()	369
16.34 Predicting various thermodynamic properties	371
16.34.1 Detailed Description	371
16.34.2 Data Structure Documentation	373
16.34.2.1 struct vrna_heat_capacity_s	373
16.34.3 Typedef Documentation	373
16.34.3.1 vrna_heat_capacity_callback	373
16.34.3.2 vrna_heat_capacity_t	374
16.34.4 Function Documentation	374
16.34.4.1 vrna_mean_bp_distance_pr()	374
16.34.4.2 vrna_mean_bp_distance()	374
16.34.4.3 vrna_ensemble_defect()	375
16.34.4.4 vrna_positional_entropy()	376
16.34.4.5 vrna_stack_prob()	377
16.34.4.6 vrna_pf_dimer_probs()	377
16.34.4.7 vrna_pr_structure()	378
16.34.4.8 vrna_pr_energy()	378
16.34.4.9 vrna_heat_capacity()	379
16.34.4.10 vrna_heat_capacity_cb()	379
16.34.4.11 vrna_heat_capacity_simple()	380
16.35 Compute the Density of States	382
16.35.1 Detailed Description	382
16.35.2 Variable Documentation	382
16.35.2.1 density_of_states	382
16.36 Inverse Folding (Design)	383

16.36.1 Detailed Description . . . . .	383
16.36.2 Function Documentation . . . . .	383
16.36.2.1 <code>inverse_fold()</code> . . . . .	383
16.36.2.2 <code>inverse_pf_fold()</code> . . . . .	384
16.36.3 Variable Documentation . . . . .	384
16.36.3.1 <code>final_cost</code> . . . . .	384
16.36.3.2 <code>give_up</code> . . . . .	385
16.36.3.3 <code>inv_verbose</code> . . . . .	385
16.37 Neighborhood Relation and Move Sets for Secondary Structures . . . . .	386
16.37.1 Detailed Description . . . . .	386
16.37.2 Data Structure Documentation . . . . .	389
16.37.2.1 <code>struct vrna_move_s</code> . . . . .	389
16.37.3 Macro Definition Documentation . . . . .	390
16.37.3.1 <code>VRNA_MOVESET_INSERTION</code> . . . . .	390
16.37.3.2 <code>VRNA_MOVESET_DELETION</code> . . . . .	390
16.37.3.3 <code>VRNA_MOVESET_SHIFT</code> . . . . .	390
16.37.3.4 <code>VRNA_MOVESET_NO_LP</code> . . . . .	391
16.37.3.5 <code>VRNA_MOVESET_DEFAULT</code> . . . . .	391
16.37.3.6 <code>VRNA_NEIGHBOR_CHANGE</code> . . . . .	391
16.37.3.7 <code>VRNA_NEIGHBOR_INVALID</code> . . . . .	391
16.37.3.8 <code>VRNA_NEIGHBOR_NEW</code> . . . . .	392
16.37.4 Typedef Documentation . . . . .	392
16.37.4.1 <code>vrna_callback_move_update</code> . . . . .	392
16.37.5 Function Documentation . . . . .	392
16.37.5.1 <code>vrna_move_init()</code> . . . . .	392
16.37.5.2 <code>vrna_move_list_free()</code> . . . . .	393
16.37.5.3 <code>vrna_move_apply()</code> . . . . .	393
16.37.5.4 <code>vrna_move_is_removal()</code> . . . . .	393
16.37.5.5 <code>vrna_move_is_insertion()</code> . . . . .	394
16.37.5.6 <code>vrna_move_is_shift()</code> . . . . .	394
16.37.5.7 <code>vrna_move_compare()</code> . . . . .	395
16.37.5.8 <code>vrna_loopidx_update()</code> . . . . .	395
16.37.5.9 <code>vrna_neighbors()</code> . . . . .	396
16.37.5.10 <code>vrna_neighbors_successive()</code> . . . . .	396
16.37.5.11 <code>vrna_move_neighbor_diff_cb()</code> . . . . .	397
16.37.5.12 <code>vrna_move_neighbor_diff()</code> . . . . .	398
16.38 (Re-)folding Paths, Saddle Points, Energy Barriers, and Local Minima . . . . .	400
16.38.1 Detailed Description . . . . .	400
16.38.2 Data Structure Documentation . . . . .	401
16.38.2.1 <code>struct vrna_path_s</code> . . . . .	401
16.38.3 Macro Definition Documentation . . . . .	402
16.38.3.1 <code>VRNA_PATH_TYPE_DOT_BRACKET</code> . . . . .	403

16.38.3.2 VRNA_PATH_TYPE_MOVES . . . . .	403
16.38.4 Function Documentation . . . . .	403
16.38.4.1 vrna_path_free() . . . . .	403
16.38.4.2 vrna_path_options_free() . . . . .	404
16.39 Direct Refolding Paths between two Secondary Structures . . . . .	405
16.39.1 Detailed Description . . . . .	405
16.39.2 Function Documentation . . . . .	405
16.39.2.1 vrna_path_findpath_saddle() . . . . .	406
16.39.2.2 vrna_path_findpath_saddle_ub() . . . . .	406
16.39.2.3 vrna_path_findpath() . . . . .	407
16.39.2.4 vrna_path_findpath_ub() . . . . .	408
16.39.2.5 vrna_path_options_findpath() . . . . .	409
16.39.2.6 vrna_path_direct() . . . . .	410
16.39.2.7 vrna_path_direct_ub() . . . . .	410
16.40 Folding Paths that start at a single Secondary Structure . . . . .	412
16.40.1 Detailed Description . . . . .	412
16.40.2 Macro Definition Documentation . . . . .	412
16.40.2.1 VRNA_PATH_STEEPEST_DESCENT . . . . .	413
16.40.2.2 VRNA_PATH_RANDOM . . . . .	413
16.40.2.3 VRNA_PATH_NO_TRANSITION_OUTPUT . . . . .	413
16.40.2.4 VRNA_PATH_DEFAULT . . . . .	413
16.40.3 Function Documentation . . . . .	414
16.40.3.1 vrna_path() . . . . .	414
16.40.3.2 vrna_path_gradient() . . . . .	415
16.40.3.3 vrna_path_random() . . . . .	416
16.41 Experimental Structure Probing Data . . . . .	417
16.41.1 Detailed Description . . . . .	417
16.42 SHAPE Reactivity Data . . . . .	418
16.42.1 Detailed Description . . . . .	418
16.42.2 Function Documentation . . . . .	418
16.42.2.1 vrna_sc_add_SHAPE_deigan() . . . . .	419
16.42.2.2 vrna_sc_add_SHAPE_deigan_ali() . . . . .	419
16.42.2.3 vrna_sc_add_SHAPE_zarringhalam() . . . . .	420
16.42.2.4 vrna_sc_SHAPE_to_pr() . . . . .	421
16.43 Generate Soft Constraints from Data . . . . .	422
16.43.1 Detailed Description . . . . .	422
16.43.2 Macro Definition Documentation . . . . .	423
16.43.2.1 VRNA_OBJECTIVE_FUNCTION_QUADRATIC . . . . .	423
16.43.2.2 VRNA_OBJECTIVE_FUNCTION_ABSOLUTE . . . . .	423
16.43.2.3 VRNA_MINIMIZER_CONJUGATE_FR . . . . .	423
16.43.2.4 VRNA_MINIMIZER_CONJUGATE_PR . . . . .	424
16.43.2.5 VRNA_MINIMIZER_VECTOR_BFGS . . . . .	424

16.43.2.6 VRNA_MINIMIZER_VECTOR_BFGS2	424
16.43.2.7 VRNA_MINIMIZER_STEEPEST_DESCENT	424
16.43.3 Typedef Documentation	424
16.43.3.1 progress_callback	424
16.43.4 Function Documentation	425
16.43.4.1 vrna_sc_minimize_pertubation()	425
16.44 Ligands Binding to RNA Structures	427
16.44.1 Detailed Description	427
16.45 Ligands Binding to Unstructured Domains	428
16.46 Incorporating Ligands Binding to Specific Sequence/Structure Motifs using Soft Constraints	429
16.46.1 Detailed Description	429
16.46.2 Function Documentation	429
16.46.2.1 vrna_sc_add_hi_motif()	429
16.47 Complex Structured Modules	431
16.47.1 Detailed Description	431
16.48 G-Quadruplexes	432
16.48.1 Detailed Description	432
16.48.2 Function Documentation	432
16.48.2.1 get_gquad_matrix()	432
16.48.2.2 parse_gquad()	433
16.48.2.3 backtrack_GQuad_IntLoop()	433
16.48.2.4 backtrack_GQuad_IntLoop_L()	434
16.49 Utilities	435
16.49.1 Detailed Description	435
16.49.2 Macro Definition Documentation	438
16.49.2.1 VRNA_INPUT_FASTA_HEADER	438
16.49.3 Function Documentation	438
16.49.3.1 vrna_alloc()	438
16.49.3.2 vrna_realloc()	439
16.49.3.3 vrna_urn()	439
16.49.3.4 vrna_int_urn()	439
16.49.3.5 vrna_time_stamp()	440
16.49.3.6 get_input_line()	440
16.49.3.7 vrna_idx_row_wise()	441
16.49.3.8 vrna_idx_col_wise()	441
16.49.4 Variable Documentation	442
16.49.4.1 xsubi	442
16.50 Exterior Loops	443
16.50.1 Detailed Description	443
16.50.2 Typedef Documentation	444
16.50.2.1 vrna_mx_pf_aux_el_t	444
16.50.3 Function Documentation	444

16.50.3.1 vrna_E_ext_stem()	444
16.50.3.2 vrna_E_ext_loop()	445
16.50.3.3 vrna_exp_E_ext_stem()	445
16.51 Hairpin Loops	447
16.51.1 Detailed Description	447
16.51.2 Function Documentation	448
16.51.2.1 vrna_E_hp_loop()	448
16.51.2.2 vrna_E_ext_hp_loop()	448
16.51.2.3 vrna_eval_hp_loop()	449
16.51.2.4 E_Hairpin()	449
16.51.2.5 exp_E_Hairpin()	450
16.51.2.6 vrna_exp_E_hp_loop()	451
16.52 Internal Loops	452
16.52.1 Detailed Description	452
16.52.2 Function Documentation	452
16.52.2.1 vrna_eval_int_loop()	453
16.53 Multibranch Loops	454
16.53.1 Detailed Description	454
16.53.2 Typedef Documentation	455
16.53.2.1 vrna_mx_pf_aux_ml_t	455
16.53.3 Function Documentation	455
16.53.3.1 vrna_E_mb_loop_stack()	455
16.54 Partition Function for Two Hybridized Sequences	456
16.54.1 Detailed Description	456
16.54.2 Function Documentation	457
16.54.2.1 vrna_pf_co_fold()	457
16.54.2.2 vrna_pf_dimer_concentrations()	458
16.55 Partition Function for two Hybridized Sequences as a Stepwise Process	459
16.55.1 Detailed Description	459
16.55.2 Function Documentation	459
16.55.2.1 pf_unstru()	460
16.55.2.2 pf_interact()	460
16.56 Reading/Writing Energy Parameter Sets from/to File	462
16.56.1 Detailed Description	462
16.56.2 Macro Definition Documentation	463
16.56.2.1 VRNA_PARAMETER_FORMAT_DEFAULT	463
16.56.3 Function Documentation	463
16.56.3.1 vrna_params_load()	463
16.56.3.2 vrna_params_save()	464
16.56.3.3 vrna_params_load_from_string()	464
16.56.3.4 vrna_params_load_defaults()	465
16.56.3.5 vrna_params_load_RNA_Turner2004()	466

16.56.3.6 vrna_params_load_RNA_Turner1999()	466
16.56.3.7 vrna_params_load_RNA_Andronescu2007()	467
16.56.3.8 vrna_params_load_RNA_Langdon2018()	467
16.56.3.9 vrna_params_load_RNA_misc_special_hairpins()	468
16.56.3.10 vrna_params_load_DNA_Mathews2004()	468
16.56.3.11 vrna_params_load_DNA_Mathews1999()	469
16.56.3.12 last_parameter_file()	469
16.56.3.13 read_parameter_file()	469
16.56.3.14 write_parameter_file()	470
16.57 Converting Energy Parameter Files	471
16.57.1 Detailed Description	471
16.57.2 Macro Definition Documentation	472
16.57.2.1 VRNA_CONVERT_OUTPUT_ALL	472
16.57.2.2 VRNA_CONVERT_OUTPUT_HP	472
16.57.2.3 VRNA_CONVERT_OUTPUT_STACK	472
16.57.2.4 VRNA_CONVERT_OUTPUT_MM_HP	472
16.57.2.5 VRNA_CONVERT_OUTPUT_MM_INT	472
16.57.2.6 VRNA_CONVERT_OUTPUT_MM_INT_1N	473
16.57.2.7 VRNA_CONVERT_OUTPUT_MM_INT_23	473
16.57.2.8 VRNA_CONVERT_OUTPUT_MM_MULTI	473
16.57.2.9 VRNA_CONVERT_OUTPUT_MM_EXT	473
16.57.2.10 VRNA_CONVERT_OUTPUT_DANGLE5	473
16.57.2.11 VRNA_CONVERT_OUTPUT_DANGLE3	473
16.57.2.12 VRNA_CONVERT_OUTPUT_INT_11	474
16.57.2.13 VRNA_CONVERT_OUTPUT_INT_21	474
16.57.2.14 VRNA_CONVERT_OUTPUT_INT_22	474
16.57.2.15 VRNA_CONVERT_OUTPUT_BULGE	474
16.57.2.16 VRNA_CONVERT_OUTPUT_INT	474
16.57.2.17 VRNA_CONVERT_OUTPUT_ML	474
16.57.2.18 VRNA_CONVERT_OUTPUT_MISC	475
16.57.2.19 VRNA_CONVERT_OUTPUT_SPECIAL_HP	475
16.57.2.20 VRNA_CONVERT_OUTPUT_VANILLA	475
16.57.2.21 VRNA_CONVERT_OUTPUT_NINIO	475
16.57.2.22 VRNA_CONVERT_OUTPUT_DUMP	475
16.57.3 Function Documentation	476
16.57.3.1 convert_parameter_file()	476
16.58 Utilities to deal with Nucleotide Alphabets	477
16.58.1 Detailed Description	477
16.58.2 Data Structure Documentation	478
16.58.2.1 struct vrna_sequence_s	478
16.58.2.2 struct vrna_alignment_s	478
16.58.3 Enumeration Type Documentation	479

16.58.3.1 vrna_seq_type_e . . . . .	479
16.58.4 Function Documentation . . . . .	479
16.58.4.1 vrna_ptypes() . . . . .	479
16.58.4.2 vrna_seq_encode() . . . . .	479
16.58.4.3 vrna_nucleotide_encode() . . . . .	480
16.58.4.4 vrna_nucleotide_decode() . . . . .	480
16.59 (Nucleic Acid Sequence) String Utilites . . . . .	482
16.59.1 Detailed Description . . . . .	482
16.59.2 Macro Definition Documentation . . . . .	483
16.59.2.1 FILENAME_MAX_LENGTH . . . . .	483
16.59.2.2 FILENAME_ID_LENGTH . . . . .	483
16.59.3 Function Documentation . . . . .	483
16.59.3.1 vrna_strdup_printf() . . . . .	484
16.59.3.2 vrna_strdup_vprintf() . . . . .	484
16.59.3.3 vrna_strcat_printf() . . . . .	485
16.59.3.4 vrna_strcat_vprintf() . . . . .	485
16.59.3.5 vrna_strsplit() . . . . .	486
16.59.3.6 vrna_random_string() . . . . .	487
16.59.3.7 vrna_hamming_distance() . . . . .	487
16.59.3.8 vrna_hamming_distance_bound() . . . . .	488
16.59.3.9 vrna_seq_toRNA() . . . . .	488
16.59.3.10 vrna_seq_toupper() . . . . .	489
16.59.3.11 vrna_seq_ungapped() . . . . .	489
16.59.3.12 vrna_cut_point_insert() . . . . .	489
16.59.3.13 vrna_cut_point_remove() . . . . .	490
16.60 Secondary Structure Utilities . . . . .	491
16.60.1 Detailed Description . . . . .	491
16.60.2 Function Documentation . . . . .	492
16.60.2.1 vrna_bp_distance() . . . . .	492
16.60.2.2 vrna_refBPcnt_matrix() . . . . .	492
16.60.2.3 vrna_refBPdist_matrix() . . . . .	493
16.60.2.4 vrna_db_from_bp_stack() . . . . .	493
16.61 Dot-Bracket Notation of Secondary Structures . . . . .	494
16.61.1 Detailed Description . . . . .	494
16.61.2 Macro Definition Documentation . . . . .	495
16.61.2.1 VRNA_BRACKETS_ALPHA . . . . .	495
16.61.2.2 VRNA_BRACKETS_RND . . . . .	495
16.61.2.3 VRNA_BRACKETS_CLY . . . . .	495
16.61.2.4 VRNA_BRACKETS_ANG . . . . .	496
16.61.2.5 VRNA_BRACKETS_SQR . . . . .	496
16.61.2.6 VRNA_BRACKETS_DEFAULT . . . . .	496
16.61.2.7 VRNA_BRACKETS_ANY . . . . .	497

16.61.3 Function Documentation	497
16.61.3.1 vrna_db_pack()	497
16.61.3.2 vrna_db_unpack()	498
16.61.3.3 vrna_db_flatten()	498
16.61.3.4 vrna_db_flatten_to()	499
16.61.3.5 vrna_db_from_ptable()	499
16.61.3.6 vrna_db_from_WUSS()	500
16.61.3.7 vrna_db_from_plist()	500
16.61.3.8 vrna_db_to_element_string()	501
16.61.3.9 vrna_db_pk_remove()	501
16.62 Pair Table Representation of Secondary Structures	503
16.62.1 Detailed Description	503
16.62.2 Function Documentation	503
16.62.2.1 vrna_ptable()	503
16.62.2.2 vrna_ptable_from_string()	504
16.62.2.3 vrna_pt_pk_get()	504
16.62.2.4 vrna_ptable_copy()	505
16.62.2.5 vrna_pt_snoop_get()	505
16.62.2.6 vrna_pt_pk_remove()	506
16.63 Pair List Representation of Secondary Structures	507
16.63.1 Detailed Description	507
16.63.2 Data Structure Documentation	508
16.63.2.1 struct vrna_elem_prob_s	508
16.63.3 Function Documentation	508
16.63.3.1 vrna_plist()	508
16.64 Helix List Representation of Secondary Structures	510
16.64.1 Detailed Description	510
16.64.2 Data Structure Documentation	510
16.64.2.1 struct vrna_hx_s	510
16.64.3 Function Documentation	510
16.64.3.1 vrna_hx_from_ptable()	510
16.65 Tree Representation of Secondary Structures	512
16.65.1 Detailed Description	512
16.65.2 Macro Definition Documentation	512
16.65.2.1 VRNA_STRUCTURE_TREE_HIT	513
16.65.2.2 VRNA_STRUCTURE_TREE_SHAPIRO_SHORT	513
16.65.2.3 VRNA_STRUCTURE_TREE_SHAPIRO	513
16.65.2.4 VRNA_STRUCTURE_TREE_SHAPIRO_EXT	513
16.65.2.5 VRNA_STRUCTURE_TREE_SHAPIRO_WEIGHT	514
16.65.2.6 VRNA_STRUCTURE_TREE_EXPANDED	514
16.65.3 Function Documentation	514
16.65.3.1 vrna_db_to_tree_string()	514



16.65.3.2 vrna_tree_string_unweight()	515
16.65.3.3 vrna_tree_string_to_db()	515
16.66 Multiple Sequence Alignment Utilities	517
16.66.1 Detailed Description	517
16.66.2 Data Structure Documentation	518
16.66.2.1 struct vrna_pinfo_s	518
16.66.3 Macro Definition Documentation	519
16.66.3.1 VRNA_MEASURE_SHANNON_ENTROPY	519
16.66.4 Function Documentation	519
16.66.4.1 vrna_aln_mpi()	519
16.66.4.2 vrna_aln_pinfo()	520
16.66.4.3 vrna_aln_slice()	520
16.66.4.4 vrna_aln_free()	521
16.66.4.5 vrna_aln_uppercase()	521
16.66.4.6 vrna_aln_toRNA()	522
16.66.4.7 vrna_aln_copy()	522
16.66.4.8 vrna_aln_conservation_struct()	523
16.66.4.9 vrna_aln_conservation_col()	523
16.66.4.10 vrna_aln_consensus_sequence()	524
16.66.4.11 vrna_aln_consensus_mis()	524
16.67 Files and I/O	526
16.67.1 Detailed Description	526
16.67.2 Function Documentation	527
16.67.2.1 vrna_read_line()	527
16.67.2.2 vrna_filename_sanitize()	528
16.67.2.3 vrna_file_exists()	529
16.68 Nucleic Acid Sequences and Structures	530
16.68.1 Detailed Description	530
16.68.2 Macro Definition Documentation	531
16.68.2.1 VRNA_OPTION_MULTILINE	531
16.68.2.2 VRNA_CONSTRAINT_MULTILINE	531
16.68.3 Function Documentation	531
16.68.3.1 vrna_file_helixlist()	531
16.68.3.2 vrna_file_connect()	532
16.68.3.3 vrna_file_bpseq()	533
16.68.3.4 vrna_file_json()	533
16.68.3.5 vrna_file_fasta_read_record()	533
16.68.3.6 vrna_extract_record_rest_structure()	535
16.68.3.7 vrna_file_SHAPE_read()	535
16.68.3.8 vrna_extract_record_rest_constraint()	536
16.68.3.9 read_record()	537
16.69 Multiple Sequence Alignments	538

16.69.1 Detailed Description	538
16.69.2 Macro Definition Documentation	539
16.69.2.1 VRNA_FILE_FORMAT_MSA_CLUSTAL	539
16.69.2.2 VRNA_FILE_FORMAT_MSA_STOCKHOLM	539
16.69.2.3 VRNA_FILE_FORMAT_MSA_FASTA	540
16.69.2.4 VRNA_FILE_FORMAT_MSA_MAF	540
16.69.2.5 VRNA_FILE_FORMAT_MSA_MIS	540
16.69.2.6 VRNA_FILE_FORMAT_MSA_DEFAULT	541
16.69.2.7 VRNA_FILE_FORMAT_MSA_NOCHECK	541
16.69.2.8 VRNA_FILE_FORMAT_MSA_UNKNOWN	541
16.69.2.9 VRNA_FILE_FORMAT_MSA_APPEND	542
16.69.2.10 VRNA_FILE_FORMAT_MSA_QUIET	542
16.69.2.11 VRNA_FILE_FORMAT_MSA_SILENT	542
16.69.3 Function Documentation	542
16.69.3.1 vrna_file_msa_read()	543
16.69.3.2 vrna_file_msa_read_record()	544
16.69.3.3 vrna_file_msa_detect_format()	545
16.69.3.4 vrna_file_msa_write()	546
16.70 Command Files	548
16.70.1 Detailed Description	548
16.70.2 Macro Definition Documentation	549
16.70.2.1 VRNA_CMD_PARSE_HC	549
16.70.2.2 VRNA_CMD_PARSE_SC	549
16.70.2.3 VRNA_CMD_PARSE_UD	550
16.70.2.4 VRNA_CMD_PARSE_SD	550
16.70.2.5 VRNA_CMD_PARSE_DEFAULTS	550
16.70.3 Function Documentation	550
16.70.3.1 vrna_file_commands_read()	550
16.70.3.2 vrna_file_commands_apply()	551
16.70.3.3 vrna_commands_apply()	551
16.70.3.4 vrna_commands_free()	552
16.71 Plotting	553
16.71.1 Detailed Description	553
16.71.2 Data Structure Documentation	555
16.71.2.1 struct vrna_dotplot_auxdata_t	555
16.71.3 Function Documentation	555
16.71.3.1 PS_dot_plot_list()	555
16.71.3.2 PS_dot_plot()	556
16.71.3.3 vrna_file_PS_rnaplot()	556
16.71.3.4 vrna_file_PS_rnaplot_a()	557
16.71.3.5 gmlRNA()	557
16.71.3.6 ssv_rna_plot()	558

16.71.3.7 <code>svg_rna_plot()</code>	559
16.71.3.8 <code>xrna_plot()</code>	559
16.71.3.9 <code>PS_rna_plot()</code>	559
16.71.3.10 <code>PS_rna_plot_a()</code>	560
16.71.3.11 <code>PS_rna_plot_a_gquad()</code>	560
16.72 Layouts and Coordinates	561
16.72.1 Detailed Description	561
16.72.2 Data Structure Documentation	563
16.72.2.1 <code>struct vrna_plot_layout_s</code>	563
16.72.2.2 <code>struct vrna_plot_options_puzzler_t</code>	563
16.72.3 Macro Definition Documentation	563
16.72.3.1 <code>VRNA_PLOT_TYPE_SIMPLE</code>	563
16.72.3.2 <code>VRNA_PLOT_TYPE_NAVIEW</code>	563
16.72.3.3 <code>VRNA_PLOT_TYPE_CIRCULAR</code>	564
16.72.4 Typedef Documentation	564
16.72.4.1 <code>vrna_plot_layout_t</code>	564
16.72.5 Function Documentation	564
16.72.5.1 <code>vrna_plot_layout()</code>	565
16.72.5.2 <code>vrna_plot_layout_simple()</code>	565
16.72.5.3 <code>vrna_plot_layout_naview()</code>	566
16.72.5.4 <code>vrna_plot_layout_circular()</code>	567
16.72.5.5 <code>vrna_plot_layout_turtle()</code>	567
16.72.5.6 <code>vrna_plot_layout_puzzler()</code>	568
16.72.5.7 <code>vrna_plot_layout_free()</code>	568
16.72.5.8 <code>vrna_plot_coords()</code>	569
16.72.5.9 <code>vrna_plot_coords_pt()</code>	570
16.72.5.10 <code>vrna_plot_coords_simple()</code>	571
16.72.5.11 <code>vrna_plot_coords_simple_pt()</code>	572
16.72.5.12 <code>vrna_plot_coords_circular()</code>	572
16.72.5.13 <code>vrna_plot_coords_circular_pt()</code>	573
16.72.5.14 <code>vrna_plot_coords_naview()</code>	574
16.72.5.15 <code>vrna_plot_coords_naview_pt()</code>	575
16.72.5.16 <code>vrna_plot_coords_puzzler()</code>	575
16.72.5.17 <code>vrna_plot_coords_puzzler_pt()</code>	576
16.72.5.18 <code>vrna_plot_options_puzzler()</code>	577
16.72.5.19 <code>vrna_plot_options_puzzler_free()</code>	577
16.72.5.20 <code>vrna_plot_coords_turtle()</code>	578
16.72.5.21 <code>vrna_plot_coords_turtle_pt()</code>	579
16.73 Annotation	580
16.73.1 Detailed Description	580
16.74 Alignment Plots	581
16.74.1 Detailed Description	581

16.74.2 Function Documentation	581
16.74.2.1 vrna_file_PS_aln()	581
16.74.2.2 vrna_file_PS_aln_slice()	582
16.75 Search Algorithms	584
16.75.1 Detailed Description	584
16.75.2 Function Documentation	584
16.75.2.1 vrna_search_BMH_num()	585
16.75.2.2 vrna_search_BMH()	585
16.75.2.3 vrna_search_BM_BCT_num()	586
16.75.2.4 vrna_search_BM_BCT()	587
16.76 Combinatorics Algorithms	588
16.76.1 Detailed Description	588
16.76.2 Function Documentation	588
16.76.2.1 vrna_enumerate_necklaces()	589
16.76.2.2 vrna_rotational_symmetry_num()	589
16.76.2.3 vrna_rotational_symmetry_pos_num()	590
16.76.2.4 vrna_rotational_symmetry()	591
16.76.2.5 vrna_rotational_symmetry_pos()	591
16.76.2.6 vrna_rotational_symmetry_db()	592
16.76.2.7 vrna_rotational_symmetry_db_pos()	593
16.77 (Abstract) Data Structures	595
16.77.1 Detailed Description	595
16.77.2 Data Structure Documentation	597
16.77.2.1 struct vrna_basepair_s	597
16.77.2.2 struct vrna_cpair_s	597
16.77.2.3 struct vrna_color_s	597
16.77.2.4 struct vrna_data_linear_s	597
16.77.2.5 struct vrna_sect_s	598
16.77.2.6 struct vrna_bp_stack_s	598
16.77.2.7 struct pu_contrib	598
16.77.2.8 struct interact	598
16.77.2.9 struct pu_out	599
16.77.2.10 struct constrain	599
16.77.2.11 struct duplexT	599
16.77.2.12 struct node	599
16.77.2.13 struct snoopT	599
16.77.2.14 struct dupVar	600
16.77.3 Typedef Documentation	600
16.77.3.1 PAIR	600
16.77.3.2 plist	600
16.77.3.3 cpair	600
16.77.3.4 sect	601

16.77.3.5 bondT	601
16.77.4 Function Documentation	601
16.77.4.1 vrna_C11_features()	601
16.78 Messages	602
16.78.1 Detailed Description	602
16.78.2 Function Documentation	602
16.78.2.1 vrna_message_error()	603
16.78.2.2 vrna_message_verror()	603
16.78.2.3 vrna_message_warning()	604
16.78.2.4 vrna_message_vwarning()	604
16.78.2.5 vrna_message_info()	604
16.78.2.6 vrna_message_vinfo()	605
16.78.2.7 vrna_message_input_seq_simple()	605
16.78.2.8 vrna_message_input_seq()	606
16.79 Unit Conversion	607
16.79.1 Detailed Description	607
16.79.2 Enumeration Type Documentation	608
16.79.2.1 vrna_unit_energy_e	608
16.79.2.2 vrna_unit_temperature_e	608
16.79.3 Function Documentation	609
16.79.3.1 vrna_convert_energy()	609
16.79.3.2 vrna_convert_temperature()	609
16.79.3.3 vrna_convert_kcal_to_dcal()	610
16.79.3.4 vrna_convert_dcal_to_kcal()	611
16.80 The Fold Compound	612
16.80.1 Detailed Description	612
16.80.2 Data Structure Documentation	613
16.80.2.1 struct vrna_fc_s	613
16.80.3 Macro Definition Documentation	621
16.80.3.1 VRNA_STATUS_MFE_PRE	621
16.80.3.2 VRNA_STATUS_MFE_POST	622
16.80.3.3 VRNA_STATUS_PF_PRE	622
16.80.3.4 VRNA_STATUS_PF_POST	622
16.80.3.5 VRNA_OPTION_MFE	622
16.80.3.6 VRNA_OPTION_PF	623
16.80.3.7 VRNA_OPTION_EVAL_ONLY	623
16.80.4 Typedef Documentation	623
16.80.4.1 vrna_callback_free_auxdata	623
16.80.4.2 vrna_callback_recursion_status	624
16.80.5 Enumeration Type Documentation	624
16.80.5.1 vrna_fc_type_e	624
16.80.6 Function Documentation	625

16.80.6.1 vrna_fold_compound()	625
16.80.6.2 vrna_fold_compound_comparative()	626
16.80.6.3 vrna_fold_compound_free()	627
16.80.6.4 vrna_fold_compound_add_auxdata()	627
16.80.6.5 vrna_fold_compound_add_callback()	628
16.81 The Dynamic Programming Matrices	629
16.81.1 Detailed Description	629
16.81.2 Data Structure Documentation	630
16.81.2.1 struct vrna_mx_mfe_s	630
16.81.2.2 struct vrna_mx_pf_s	630
16.81.3 Enumeration Type Documentation	631
16.81.3.1 vrna_mx_type_e	631
16.81.4 Function Documentation	631
16.81.4.1 vrna_mx_add()	631
16.81.4.2 vrna_mx_mfe_free()	632
16.81.4.3 vrna_mx_pf_free()	632
16.82 Hash Tables	634
16.82.1 Detailed Description	634
16.82.2 Data Structure Documentation	635
16.82.2.1 struct vrna_ht_entry_db_t	635
16.82.3 Typedef Documentation	636
16.82.3.1 vrna_hash_table_t	636
16.82.3.2 vrna_callback_ht_compare_entries	636
16.82.3.3 vrna_callback_ht_hash_function	636
16.82.3.4 vrna_callback_ht_free_entry	637
16.82.4 Function Documentation	637
16.82.4.1 vrna_ht_init()	637
16.82.4.2 vrna_ht_size()	638
16.82.4.3 vrna_ht_collisions()	638
16.82.4.4 vrna_ht_get()	639
16.82.4.5 vrna_ht_insert()	639
16.82.4.6 vrna_ht_remove()	640
16.82.4.7 vrna_ht_clear()	640
16.82.4.8 vrna_ht_free()	641
16.82.4.9 vrna_ht_db_comp()	641
16.82.4.10 vrna_ht_db_hash_func()	642
16.82.4.11 vrna_ht_db_free_entry()	642
16.83 Heaps	644
16.83.1 Detailed Description	644
16.83.2 Typedef Documentation	645
16.83.2.1 vrna_heap_t	645
16.83.2.2 vrna_callback_heap_cmp	645

16.83.2.3 vrna_callback_heap_get_pos . . . . .	646
16.83.2.4 vrna_callback_heap_set_pos . . . . .	646
16.83.3 Function Documentation . . . . .	646
16.83.3.1 vrna_heap_init() . . . . .	646
16.83.3.2 vrna_heap_free() . . . . .	647
16.83.3.3 vrna_heap_size() . . . . .	648
16.83.3.4 vrna_heap_insert() . . . . .	648
16.83.3.5 vrna_heap_pop() . . . . .	649
16.83.3.6 vrna_heap_top() . . . . .	649
16.83.3.7 vrna_heap_remove() . . . . .	650
16.83.3.8 vrna_heap_update() . . . . .	650
16.84 Buffers . . . . .	652
16.84.1 Detailed Description . . . . .	652
16.84.2 Typedef Documentation . . . . .	653
16.84.2.1 vrna_callback_stream_output . . . . .	653
16.84.3 Function Documentation . . . . .	653
16.84.3.1 vrna_cstr() . . . . .	653
16.84.3.2 vrna_cstr_free() . . . . .	654
16.84.3.3 vrna_cstr_close() . . . . .	654
16.84.3.4 vrna_cstr_fflush() . . . . .	654
16.84.3.5 vrna_ostream_init() . . . . .	655
16.84.3.6 vrna_ostream_free() . . . . .	656
16.84.3.7 vrna_ostream_request() . . . . .	656
16.84.3.8 vrna_ostream_provide() . . . . .	656
16.85 Deprecated Interface for Global MFE Prediction . . . . .	658
16.85.1 Detailed Description . . . . .	658
16.85.2 Function Documentation . . . . .	659
16.85.2.1 alifold() . . . . .	659
16.85.2.2 cofold() . . . . .	660
16.85.2.3 cofold_par() . . . . .	660
16.85.2.4 free_co_arrays() . . . . .	661
16.85.2.5 update_cofold_params() . . . . .	661
16.85.2.6 update_cofold_params_par() . . . . .	661
16.85.2.7 export_cofold_arrays_gq() . . . . .	662
16.85.2.8 export_cofold_arrays() . . . . .	662
16.85.2.9 get_monomere_mfes() . . . . .	663
16.85.2.10 initialize_cofold() . . . . .	664
16.85.2.11 fold_par() . . . . .	664
16.85.2.12 fold() . . . . .	665
16.85.2.13 circfold() . . . . .	666
16.85.2.14 free_arrays() . . . . .	666
16.85.2.15 update_fold_params() . . . . .	667

16.85.2.16	update_fold_params_par()	667
16.85.2.17	export_fold_arrays()	667
16.85.2.18	export_fold_arrays_par()	668
16.85.2.19	export_circfold_arrays()	668
16.85.2.20	export_circfold_arrays_par()	668
16.85.2.21	LoopEnergy()	669
16.85.2.22	HairpinE()	669
16.85.2.23	initialize_fold()	669
16.85.2.24	circalifold()	670
16.85.2.25	free_alifold_arrays()	670
16.86	Deprecated Interface for Local (Sliding Window) MFE Prediction	671
16.86.1	Detailed Description	671
16.86.2	Function Documentation	671
16.86.2.1	Lfold()	671
16.86.2.2	Lfoldz()	672
16.87	Deprecated Interface for Global Partition Function Computation	673
16.87.1	Detailed Description	673
16.87.2	Function Documentation	674
16.87.2.1	alipf_fold_par()	675
16.87.2.2	pf_fold_par()	675
16.87.2.3	pf_fold()	676
16.87.2.4	pf_circ_fold()	677
16.87.2.5	free_pf_arrays()	678
16.87.2.6	update_pf_params()	679
16.87.2.7	update_pf_params_par()	679
16.87.2.8	export_bppm()	679
16.87.2.9	get_pf_arrays()	680
16.87.2.10	mean_bp_distance()	680
16.87.2.11	mean_bp_distance_pr()	681
16.87.2.12	stackProb()	681
16.87.2.13	init_pf_fold()	682
16.87.2.14	co_pf_fold()	682
16.87.2.15	co_pf_fold_par()	682
16.87.2.16	compute_probabilities()	683
16.87.2.17	init_co_pf_fold()	684
16.87.2.18	export_co_bppm()	684
16.87.2.19	free_co_pf_arrays()	685
16.87.2.20	update_co_pf_params()	685
16.87.2.21	update_co_pf_params_par()	685
16.87.2.22	assign_plist_from_db()	686
16.87.2.23	assign_plist_from_pr()	686
16.87.2.24	alipf_fold()	687



16.87.2.25	<a href="#">alipf_circ_fold()</a>	687
16.87.2.26	<a href="#">export_ali_bppm()</a>	688
16.87.2.27	<a href="#">free_alipf_arrays()</a>	689
16.87.2.28	<a href="#">alipbacktrack()</a>	689
16.87.2.29	<a href="#">get_alipf_arrays()</a>	689
16.88	<a href="#">Deprecated Interface for Local (Sliding Window) Partition Function Computation</a>	691
16.88.1	<a href="#">Detailed Description</a>	691
16.88.2	<a href="#">Function Documentation</a>	691
16.88.2.1	<a href="#">update_pf_paramsLP()</a>	691
16.88.2.2	<a href="#">pfl_fold()</a>	692
16.88.2.3	<a href="#">putoutpU_prob()</a>	693
16.88.2.4	<a href="#">putoutpU_prob_bin()</a>	693
16.89	<a href="#">Deprecated Interface for Stochastic Backtracking</a>	694
16.89.1	<a href="#">Detailed Description</a>	694
16.89.2	<a href="#">Function Documentation</a>	694
16.89.2.1	<a href="#">pbacktrack()</a>	694
16.89.2.2	<a href="#">pbacktrack_circ()</a>	695
16.89.3	<a href="#">Variable Documentation</a>	695
16.89.3.1	<a href="#">st_back</a>	695
16.90	<a href="#">Deprecated Interface for Multiple Sequence Alignment Utilities</a>	697
16.90.1	<a href="#">Detailed Description</a>	697
16.90.2	<a href="#">Typedef Documentation</a>	697
16.90.2.1	<a href="#">pair_info</a>	697
16.90.3	<a href="#">Function Documentation</a>	698
16.90.3.1	<a href="#">get_mpi()</a>	698
16.90.3.2	<a href="#">encode_ali_sequence()</a>	698
16.90.3.3	<a href="#">alloc_sequence_arrays()</a>	699
16.90.3.4	<a href="#">free_sequence_arrays()</a>	699
16.91	<a href="#">Deprecated Interface for Secondary Structure Utilities</a>	701
16.91.1	<a href="#">Detailed Description</a>	701
16.91.2	<a href="#">Function Documentation</a>	702
16.91.2.1	<a href="#">b2HIT()</a>	702
16.91.2.2	<a href="#">b2C()</a>	703
16.91.2.3	<a href="#">b2Shapiro()</a>	703
16.91.2.4	<a href="#">add_root()</a>	704
16.91.2.5	<a href="#">expand_Shapiro()</a>	704
16.91.2.6	<a href="#">expand_Full()</a>	705
16.91.2.7	<a href="#">unexpand_Full()</a>	705
16.91.2.8	<a href="#">unweight()</a>	705
16.91.2.9	<a href="#">unexpand_aligned_F()</a>	706
16.91.2.10	<a href="#">parse_structure()</a>	706
16.91.2.11	<a href="#">pack_structure()</a>	706

16.91.2.12 unpack_structure()	707
16.91.2.13 make_pair_table()	707
16.91.2.14 copy_pair_table()	708
16.91.2.15 alimake_pair_table()	708
16.91.2.16 make_pair_table_snoop()	709
16.91.2.17 bp_distance()	709
16.91.2.18 make_referenceBP_array()	709
16.91.2.19 compute_BPdifferences()	710
16.91.2.20 parenthesis_structure()	710
16.91.2.21 parenthesis_zuker()	711
16.91.2.22 bppm_to_structure()	711
16.91.2.23 bppm_symbol()	711
16.92 Deprecated Interface for Plotting Utilities	712
16.92.1 Detailed Description	712
16.92.2 Data Structure Documentation	712
16.92.2.1 struct COORDINATE	712
16.92.3 Function Documentation	712
16.92.3.1 PS_color_aln()	713
16.92.3.2 aliPS_color_aln()	713
16.92.3.3 simple_xy_coordinates()	713
16.92.3.4 simple_circplot_coordinates()	714
16.92.3.5 naview_xy_coordinates()	715
16.92.4 Variable Documentation	715
16.92.4.1 rna_plot_type	715
16.93 Deprecated Interface for (Re-)folding Paths, Saddle Points, and Energy Barriers	716
16.93.1 Detailed Description	716
16.93.2 Typedef Documentation	716
16.93.2.1 path_t	716
16.93.3 Function Documentation	717
16.93.3.1 find_saddle()	717
16.93.3.2 free_path()	717
16.93.3.3 get_path()	718
<b>17 Data Structure Documentation</b>	<b>719</b>
17.1 _struct_en Struct Reference	719
17.1.1 Detailed Description	719
17.2 LIST Struct Reference	719
17.3 LST_BUCKET Struct Reference	720
17.4 Postorder_list Struct Reference	720
17.4.1 Detailed Description	720
17.5 swString Struct Reference	720
17.5.1 Detailed Description	720

17.6 Tree Struct Reference . . . . .	721
17.6.1 Detailed Description . . . . .	721
17.7 TwoDpfold_vars Struct Reference . . . . .	721
17.7.1 Detailed Description . . . . .	722
17.8 vrna_dimer_conc_s Struct Reference . . . . .	722
17.8.1 Detailed Description . . . . .	723
17.9 vrna_sc_bp_storage_t Struct Reference . . . . .	723
17.9.1 Detailed Description . . . . .	723
17.10 vrna_sc_motif_s Struct Reference . . . . .	723
17.11 vrna_structured_domains_s Struct Reference . . . . .	723
17.12 vrna_subopt_sol_s Struct Reference . . . . .	724
17.12.1 Detailed Description . . . . .	724
17.13 vrna_unstructured_domain_motif_s Struct Reference . . . . .	724
<b>18 File Documentation . . . . .</b>	<b>725</b>
18.1 ViennaRNA/2Dfold.h File Reference . . . . .	725
18.1.1 Detailed Description . . . . .	726
18.2 ViennaRNA/2Dpfold.h File Reference . . . . .	726
18.2.1 Detailed Description . . . . .	727
18.2.2 Function Documentation . . . . .	727
18.2.2.1 get_TwoDpfold_variables() . . . . .	727
18.2.2.2 destroy_TwoDpfold_variables() . . . . .	728
18.2.2.3 TwoDpfoldList() . . . . .	728
18.2.2.4 TwoDpfold_pbacktrack() . . . . .	729
18.2.2.5 TwoDpfold_pbacktrack5() . . . . .	730
18.3 ViennaRNA/alifold.h File Reference . . . . .	731
18.3.1 Detailed Description . . . . .	732
18.3.2 Function Documentation . . . . .	732
18.3.2.1 energy_of_alistruct() . . . . .	732
18.3.2.2 update_alifold_params() . . . . .	732
18.3.3 Variable Documentation . . . . .	733
18.3.3.1 cv_fact . . . . .	733
18.3.3.2 nc_fact . . . . .	733
18.4 ViennaRNA/aln_util.h File Reference . . . . .	733
18.4.1 Detailed Description . . . . .	734
18.5 ViennaRNA/alphabet.h File Reference . . . . .	734
18.5.1 Detailed Description . . . . .	735
18.6 ViennaRNA/boltzmann_sampling.h File Reference . . . . .	735
18.6.1 Detailed Description . . . . .	736
18.7 ViennaRNA/centroid.h File Reference . . . . .	737
18.7.1 Detailed Description . . . . .	737
18.7.2 Function Documentation . . . . .	738

18.7.2.1 <code>get_centroid_struct_pl()</code>	738
18.7.2.2 <code>get_centroid_struct_pr()</code>	738
18.8 <code>ViennaRNA/char_stream.h</code> File Reference	738
18.8.1 Detailed Description	739
18.9 <code>ViennaRNA/datastructures/char_stream.h</code> File Reference	739
18.9.1 Detailed Description	740
18.10 <code>ViennaRNA/cofold.h</code> File Reference	740
18.10.1 Detailed Description	741
18.11 <code>ViennaRNA/combinatorics.h</code> File Reference	741
18.11.1 Detailed Description	741
18.12 <code>ViennaRNA/commands.h</code> File Reference	742
18.12.1 Detailed Description	742
18.13 <code>ViennaRNA/concentrations.h</code> File Reference	743
18.13.1 Detailed Description	744
18.13.2 Function Documentation	744
18.13.2.1 <code>get_concentrations()</code>	744
18.14 <code>ViennaRNA/constraints.h</code> File Reference	744
18.14.1 Detailed Description	745
18.15 <code>ViennaRNA/constraints/hard.h</code> File Reference	745
18.15.1 Detailed Description	748
18.15.2 Macro Definition Documentation	748
18.15.2.1 <code>VRNA_CONSTRAINT_NO_HEADER</code>	748
18.15.2.2 <code>VRNA_CONSTRAINT_DB_ANG_BRACK</code>	748
18.15.3 Enumeration Type Documentation	748
18.15.3.1 <code>vrna_hc_type_e</code>	748
18.15.4 Function Documentation	749
18.15.4.1 <code>vrna_hc_add_data()</code>	749
18.15.4.2 <code>print_tty_constraint()</code>	749
18.15.4.3 <code>print_tty_constraint_full()</code>	750
18.15.4.4 <code>constrain_ptypes()</code>	750
18.16 <code>ViennaRNA/constraints/ligand.h</code> File Reference	750
18.16.1 Detailed Description	751
18.16.2 Typedef Documentation	752
18.16.2.1 <code>vrna_sc_motif_t</code>	752
18.17 <code>ViennaRNA/constraints/SHAPE.h</code> File Reference	752
18.17.1 Detailed Description	753
18.17.2 Function Documentation	753
18.17.2.1 <code>vrna_sc_SHAPE_parse_method()</code>	753
18.18 <code>ViennaRNA/constraints/soft.h</code> File Reference	754
18.18.1 Detailed Description	755
18.18.2 Enumeration Type Documentation	755
18.18.2.1 <code>vrna_sc_type_e</code>	755

18.19 ViennaRNA/constraints_hard.h File Reference . . . . .	756
18.19.1 Detailed Description . . . . .	756
18.20 ViennaRNA/constraints_ligand.h File Reference . . . . .	756
18.20.1 Detailed Description . . . . .	757
18.21 ViennaRNA/constraints_SHAPE.h File Reference . . . . .	757
18.21.1 Detailed Description . . . . .	757
18.22 ViennaRNA/constraints_soft.h File Reference . . . . .	757
18.22.1 Detailed Description . . . . .	758
18.23 ViennaRNA/convert_epars.h File Reference . . . . .	758
18.23.1 Detailed Description . . . . .	758
18.24 ViennaRNA/data_structures.h File Reference . . . . .	758
18.24.1 Detailed Description . . . . .	759
18.25 ViennaRNA/datastructures/hash_tables.h File Reference . . . . .	759
18.25.1 Detailed Description . . . . .	760
18.26 ViennaRNA/datastructures/heap.h File Reference . . . . .	760
18.26.1 Detailed Description . . . . .	760
18.27 ViennaRNA/dist_vars.h File Reference . . . . .	761
18.27.1 Detailed Description . . . . .	761
18.27.2 Variable Documentation . . . . .	761
18.27.2.1 edit_backtrack . . . . .	762
18.27.2.2 cost_matrix . . . . .	762
18.28 ViennaRNA/dp_matrices.h File Reference . . . . .	762
18.28.1 Detailed Description . . . . .	763
18.29 ViennaRNA/duplex.h File Reference . . . . .	763
18.29.1 Detailed Description . . . . .	763
18.30 ViennaRNA/edit_cost.h File Reference . . . . .	764
18.30.1 Detailed Description . . . . .	764
18.31 ViennaRNA/energy_const.h File Reference . . . . .	764
18.31.1 Detailed Description . . . . .	764
18.32 ViennaRNA/energy_par.h File Reference . . . . .	765
18.32.1 Detailed Description . . . . .	765
18.33 ViennaRNA/equilibrium_probs.h File Reference . . . . .	765
18.33.1 Detailed Description . . . . .	766
18.34 ViennaRNA/eval.h File Reference . . . . .	767
18.34.1 Detailed Description . . . . .	770
18.35 ViennaRNA/exterior_loops.h File Reference . . . . .	770
18.35.1 Detailed Description . . . . .	770
18.36 ViennaRNA/file_formats.h File Reference . . . . .	771
18.36.1 Detailed Description . . . . .	771
18.37 ViennaRNA/io/file_formats.h File Reference . . . . .	771
18.37.1 Detailed Description . . . . .	772
18.38 ViennaRNA/file_formats_msa.h File Reference . . . . .	772

18.38.1 Detailed Description	773
18.39 ViennaRNA/io/file_formats_msa.h File Reference	773
18.39.1 Detailed Description	775
18.40 ViennaRNA/file_utils.h File Reference	775
18.40.1 Detailed Description	775
18.41 ViennaRNA/findpath.h File Reference	776
18.41.1 Detailed Description	776
18.42 ViennaRNA/landscape/findpath.h File Reference	776
18.42.1 Detailed Description	777
18.43 ViennaRNA/fold.h File Reference	777
18.43.1 Detailed Description	778
18.44 ViennaRNA/fold_compound.h File Reference	778
18.44.1 Detailed Description	780
18.45 ViennaRNA/fold_vars.h File Reference	780
18.45.1 Detailed Description	781
18.45.2 Variable Documentation	781
18.45.2.1 RibosumFile	781
18.45.2.2 james_rule	781
18.45.2.3 logML	781
18.45.2.4 cut_point	782
18.45.2.5 base_pair	782
18.45.2.6 pr	782
18.45.2.7 iindx	782
18.46 ViennaRNA/gquad.h File Reference	783
18.46.1 Detailed Description	783
18.47 ViennaRNA/grammar.h File Reference	783
18.47.1 Detailed Description	784
18.48 ViennaRNA/hairpin_loops.h File Reference	784
18.48.1 Detailed Description	784
18.49 ViennaRNA/heat_capacity.h File Reference	784
18.49.1 Detailed Description	785
18.50 ViennaRNA/interior_loops.h File Reference	785
18.50.1 Detailed Description	786
18.51 ViennaRNA/inverse.h File Reference	786
18.51.1 Detailed Description	786
18.52 ViennaRNA/landscape/move.h File Reference	786
18.52.1 Detailed Description	787
18.53 ViennaRNA/landscape/paths.h File Reference	788
18.53.1 Detailed Description	789
18.54 ViennaRNA/Lfold.h File Reference	789
18.54.1 Detailed Description	790
18.55 ViennaRNA/loop_energies.h File Reference	790

18.55.1 Detailed Description . . . . .	790
18.56 ViennaRNA/loops/all.h File Reference . . . . .	790
18.56.1 Detailed Description . . . . .	791
18.57 ViennaRNA/loops/external.h File Reference . . . . .	791
18.57.1 Detailed Description . . . . .	793
18.58 ViennaRNA/loops/hairpin.h File Reference . . . . .	793
18.58.1 Detailed Description . . . . .	794
18.59 ViennaRNA/loops/internal.h File Reference . . . . .	794
18.59.1 Detailed Description . . . . .	795
18.60 ViennaRNA/loops/multibranch.h File Reference . . . . .	796
18.60.1 Detailed Description . . . . .	797
18.61 ViennaRNA/LPfold.h File Reference . . . . .	797
18.61.1 Detailed Description . . . . .	798
18.61.2 Function Documentation . . . . .	798
18.61.2.1 init_pf_foldLP() . . . . .	798
18.62 ViennaRNA/MEA.h File Reference . . . . .	798
18.62.1 Detailed Description . . . . .	798
18.63 ViennaRNA/mfe.h File Reference . . . . .	799
18.63.1 Detailed Description . . . . .	800
18.64 ViennaRNA/mfe_window.h File Reference . . . . .	800
18.64.1 Detailed Description . . . . .	801
18.65 ViennaRNA/mm.h File Reference . . . . .	801
18.65.1 Detailed Description . . . . .	802
18.65.2 Function Documentation . . . . .	802
18.65.2.1 vrna_maximum_matching() . . . . .	802
18.65.2.2 vrna_maximum_matching_simple() . . . . .	802
18.66 ViennaRNA/model.h File Reference . . . . .	802
18.66.1 Detailed Description . . . . .	807
18.67 ViennaRNA/multibranch_loops.h File Reference . . . . .	807
18.67.1 Detailed Description . . . . .	807
18.68 ViennaRNA/naview.h File Reference . . . . .	807
18.68.1 Detailed Description . . . . .	808
18.69 ViennaRNA/plotting/naview.h File Reference . . . . .	808
18.69.1 Detailed Description . . . . .	808
18.70 ViennaRNA/neighbor.h File Reference . . . . .	809
18.70.1 Detailed Description . . . . .	809
18.71 ViennaRNA/landscape/neighbor.h File Reference . . . . .	809
18.71.1 Detailed Description . . . . .	811
18.72 ViennaRNA/params.h File Reference . . . . .	811
18.72.1 Detailed Description . . . . .	811
18.73 ViennaRNA/params/1.8.4_epars.h File Reference . . . . .	811
18.73.1 Detailed Description . . . . .	811

18.74 ViennaRNA/params/1.8.4_intloops.h File Reference . . . . .	812
18.74.1 Detailed Description . . . . .	812
18.75 ViennaRNA/params/basic.h File Reference . . . . .	812
18.75.1 Detailed Description . . . . .	813
18.76 ViennaRNA/constraints/basic.h File Reference . . . . .	814
18.76.1 Detailed Description . . . . .	815
18.77 ViennaRNA/utils/basic.h File Reference . . . . .	815
18.77.1 Detailed Description . . . . .	818
18.77.2 Function Documentation . . . . .	818
18.77.2.1 get_line() . . . . .	818
18.77.2.2 print_tty_input_seq() . . . . .	819
18.77.2.3 print_tty_input_seq_str() . . . . .	819
18.77.2.4 warn_user() . . . . .	819
18.77.2.5 perror() . . . . .	819
18.77.2.6 space() . . . . .	820
18.77.2.7 xrealloc() . . . . .	820
18.77.2.8 init_rand() . . . . .	820
18.77.2.9 urn() . . . . .	820
18.77.2.10 int_urn() . . . . .	821
18.77.2.11 filecopy() . . . . .	821
18.77.2.12 time_stamp() . . . . .	821
18.78 ViennaRNA/datastructures/basic.h File Reference . . . . .	821
18.78.1 Detailed Description . . . . .	823
18.79 ViennaRNA/params/constants.h File Reference . . . . .	823
18.79.1 Detailed Description . . . . .	824
18.79.2 Macro Definition Documentation . . . . .	824
18.79.2.1 GASCONST . . . . .	824
18.79.2.2 K0 . . . . .	824
18.79.2.3 INF . . . . .	824
18.79.2.4 FORBIDDEN . . . . .	824
18.79.2.5 BONUS . . . . .	825
18.79.2.6 NBPAIRS . . . . .	825
18.79.2.7 TURN . . . . .	825
18.79.2.8 MAXLOOP . . . . .	825
18.80 ViennaRNA/params/convert.h File Reference . . . . .	825
18.80.1 Detailed Description . . . . .	826
18.81 ViennaRNA/params/io.h File Reference . . . . .	826
18.81.1 Detailed Description . . . . .	827
18.82 ViennaRNA/part_func.h File Reference . . . . .	828
18.82.1 Detailed Description . . . . .	830
18.82.2 Function Documentation . . . . .	830
18.82.2.1 centroid() . . . . .	830



18.82.2.2 <a href="#">get_centroid_struct_gquad_pr()</a>	831
18.82.2.3 <a href="#">mean_bp_dist()</a>	831
18.82.2.4 <a href="#">expLoopEnergy()</a>	831
18.82.2.5 <a href="#">expHairpinEnergy()</a>	831
18.83 <a href="#">ViennaRNA/part_func_co.h</a> File Reference	832
18.83.1 Detailed Description	832
18.83.2 Function Documentation	833
18.83.2.1 <a href="#">get_plist()</a>	833
18.84 <a href="#">ViennaRNA/part_func_up.h</a> File Reference	833
18.84.1 Detailed Description	833
18.85 <a href="#">ViennaRNA/part_func_window.h</a> File Reference	834
18.85.1 Detailed Description	835
18.86 <a href="#">ViennaRNA/perturbation_fold.h</a> File Reference	835
18.86.1 Detailed Description	836
18.87 <a href="#">ViennaRNA/plot_aln.h</a> File Reference	837
18.87.1 Detailed Description	837
18.88 <a href="#">ViennaRNA/plot_layouts.h</a> File Reference	837
18.88.1 Detailed Description	837
18.89 <a href="#">ViennaRNA/plot_structure.h</a> File Reference	838
18.89.1 Detailed Description	838
18.90 <a href="#">ViennaRNA/plot_utils.h</a> File Reference	838
18.90.1 Detailed Description	838
18.91 <a href="#">ViennaRNA/plotting/alignments.h</a> File Reference	839
18.91.1 Detailed Description	839
18.92 <a href="#">ViennaRNA/utls/alignments.h</a> File Reference	839
18.92.1 Detailed Description	841
18.93 <a href="#">ViennaRNA/plotting/layouts.h</a> File Reference	842
18.93.1 Detailed Description	843
18.94 <a href="#">ViennaRNA/plotting/probabilities.h</a> File Reference	844
18.94.1 Detailed Description	844
18.95 <a href="#">ViennaRNA/plotting/RNApuzzler/RNApuzzler.h</a> File Reference	845
18.95.1 Detailed Description	845
18.96 <a href="#">ViennaRNA/plotting/RNApuzzler/RNAturtle.h</a> File Reference	846
18.96.1 Detailed Description	846
18.97 <a href="#">ViennaRNA/plotting/structures.h</a> File Reference	846
18.97.1 Detailed Description	847
18.98 <a href="#">ViennaRNA/utls/structures.h</a> File Reference	848
18.98.1 Detailed Description	851
18.99 <a href="#">ViennaRNA/profiledist.h</a> File Reference	851
18.99.1 Detailed Description	852
18.99.2 Function Documentation	852
18.99.2.1 <a href="#">profile_edit_distance()</a>	852

18.99.2.2 Make_bp_profile_bppm()	852
18.99.2.3 free_profile()	852
18.99.2.4 Make_bp_profile()	853
18.100 ViennaRNA/PS_dot.h File Reference	853
18.100.1 Detailed Description	853
18.101 ViennaRNA/read_epars.h File Reference	853
18.101.1 Detailed Description	854
18.102 ViennaRNA/ribo.h File Reference	854
18.102.1 Detailed Description	854
18.103 ViennaRNA/RNAstruct.h File Reference	854
18.103.1 Detailed Description	855
18.104 ViennaRNA/search/BoyerMoore.h File Reference	856
18.104.1 Detailed Description	856
18.105 ViennaRNA/sequence.h File Reference	856
18.105.1 Detailed Description	857
18.106 ViennaRNA/stream_output.h File Reference	857
18.106.1 Detailed Description	857
18.107 ViennaRNA/datastructures/stream_output.h File Reference	858
18.107.1 Detailed Description	858
18.108 ViennaRNA/string_utils.h File Reference	859
18.108.1 Detailed Description	859
18.109 ViennaRNA/stringdist.h File Reference	859
18.109.1 Detailed Description	860
18.109.2 Function Documentation	860
18.109.2.1 Make_swString()	860
18.109.2.2 string_edit_distance()	860
18.110 ViennaRNA/structure_utils.h File Reference	860
18.110.1 Detailed Description	861
18.111 ViennaRNA/structured_domains.h File Reference	861
18.111.1 Detailed Description	861
18.112 ViennaRNA/subopt.h File Reference	862
18.112.1 Detailed Description	863
18.112.2 Typedef Documentation	863
18.112.2.1 SOLUTION	863
18.113 ViennaRNA/svm_utils.h File Reference	863
18.113.1 Detailed Description	864
18.114 ViennaRNA/treedist.h File Reference	864
18.114.1 Detailed Description	864
18.114.2 Function Documentation	864
18.114.2.1 make_tree()	864
18.114.2.2 tree_edit_distance()	865
18.114.2.3 free_tree()	865

18.115 ViennaRNA/units.h File Reference . . . . .	865
18.115.1 Detailed Description . . . . .	866
18.116 ViennaRNA/utls/units.h File Reference . . . . .	866
18.116.1 Detailed Description . . . . .	867
18.117 ViennaRNA/unstructured_domains.h File Reference . . . . .	867
18.117.1 Detailed Description . . . . .	869
18.117.2 Function Documentation . . . . .	869
18.117.2.1 vrna_ud_set_prob_cb() . . . . .	869
18.118 ViennaRNA/utls.h File Reference . . . . .	869
18.118.1 Detailed Description . . . . .	870
18.119 ViennaRNA/io/utls.h File Reference . . . . .	870
18.119.1 Detailed Description . . . . .	871
18.120 ViennaRNA/plotting/utls.h File Reference . . . . .	871
18.120.1 Detailed Description . . . . .	872
18.121 ViennaRNA/utls/strings.h File Reference . . . . .	872
18.121.1 Detailed Description . . . . .	874
18.121.2 Function Documentation . . . . .	874
18.121.2.1 str_uppercase() . . . . .	874
18.121.2.2 str_DNA2RNA() . . . . .	874
18.121.2.3 random_string() . . . . .	874
18.121.2.4 hamming() . . . . .	875
18.121.2.5 hamming_bound() . . . . .	875
18.122 ViennaRNA/walk.h File Reference . . . . .	875
18.122.1 Detailed Description . . . . .	875
18.123 ViennaRNA/landscape/walk.h File Reference . . . . .	876
18.123.1 Detailed Description . . . . .	877
<b>Bibliography</b>	<b>880</b>
<b>Index</b>	<b>881</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

1994-2020

#### 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 (pertubation 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

If you want to get involved in the development of the `ViennaRNA Package` yourself, please read the [Contributing page](#).

## 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-2.4.17.tar.gz
cd ViennaRNA-2.4.17
./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 SIMD (see [Streaming SIMD Extension \(SSE\) support](#)). We therefore recommend using a more recent compiler (GCC 5 or above) or to turn off one of the two features, LTO or SIMD 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 RNALfold

By default, RNALfold 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 [C Examples](#) 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 = "GAGUAGUGGAACCGGCUAUGUUUGUGACUCGCAGACUAACA";
    /* 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 [ %.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/utills/basic.h>
#include <ViennaRNA/utills/alignments.h>
int
main()
{
    /* The RNA sequence alignment */
    const char *sequences[] = {
        "CUGCCUCACAACGUUUUGUGCCUCAGUUACCCGUAGAUGUAGUGAGGGU",
        "CUGCCUCACAACAUUUUGUGCCUCAGUUACUCAUAGAUGUAGUGAGGGU",
        "---CUCGACACCACU---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 Minimum Free Energy and corresponding secondary structure */
    float mfe = vrna_alifold(sequences, structure);
    /* print consensus sequence, structure and MFE */
    printf("%s\n%s [ %6.2f ]\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/utills/basic.h>
int
main()
{
    /* The RNA sequence */
    char *seq = "GAGUAGUGGAACCAGGCUAUGUUUGUGACUCGCAGACUAACA";
    /* 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

## 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/utills/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 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(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 $seq = "GAGUAGUGGAACCAGGCUAUGUUUGUGACUCGCAGACUAACA";
# compute minimum free energy (MFE) and corresponding structure
my ($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 = (
    "CUGCCUCACAACGUUUUGUGCCUCAGUUACCCGUGAGAUGUAGUGAGGGU",
    "CUGCCUCACAACAUAUUUGUGCCUCAGUUACUUAUAGAUGUAGUGAGGGU",
    "---CUCGACACCACU---GCCUCGGUUACCAUCGGUGCAGUGCGGGU"
);
# 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 = "GAGUAGUGGAACCGGCUAUGUUUGUGACUCGCAGACUAACA";
# 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;
```

## 2.3.2 Python

### Simple MFE prediction for a given sequence

```
import RNA
# The RNA sequence
seq = "GAGUAGUGGAACCGGCUAUGUUUGUGACUCGCAGACUAACA"
# 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 = [
    "CUGCCUCACAACGUUUGUGCCUCAGUUACCCGUGAGUAGUGAGGGU",
    "CUGCCUCACAACAUUUUGUGCCUCAGUUACUAGUAGUAGUGAGGGU",
    "---CUCGACACCACU---GCCUCGGUUACCAUCGGUGCAGUGCGGGU"
]
# compute the consensus sequence
cons = RNA.consensus(sequences)
# predict Minmum 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 Minmum 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
- [Suboptimal 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, 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

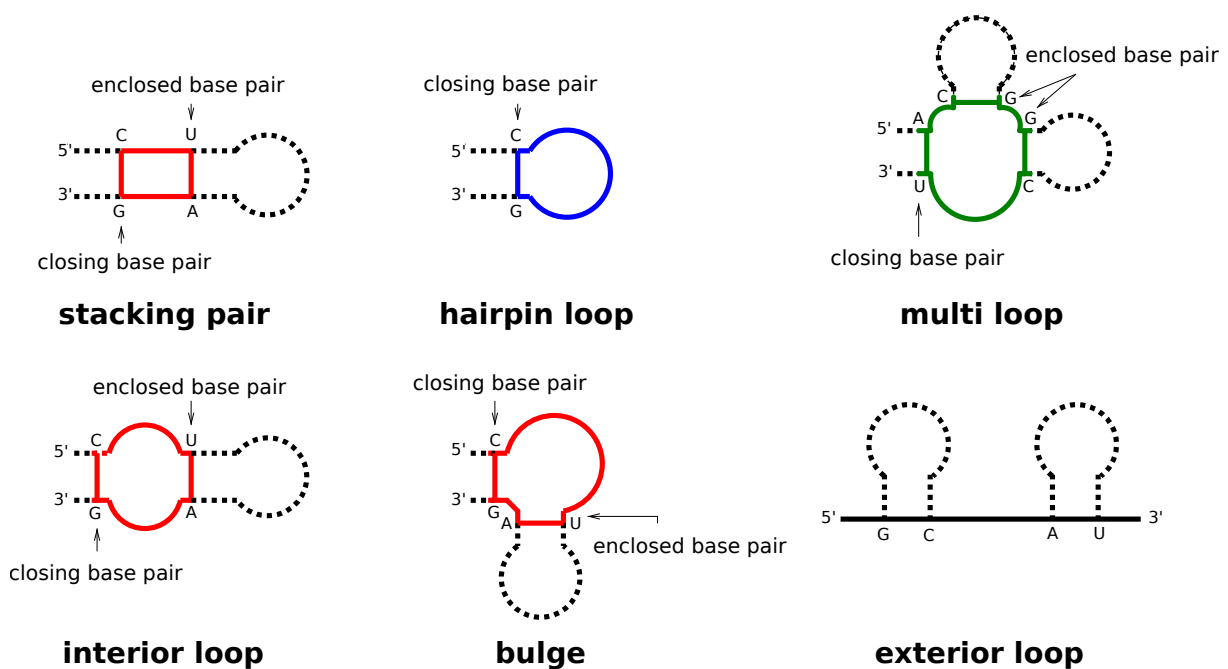
A description on how secondary structures are decomposed into individual loops to eventually evaluate their stability in terms of free energy.

### 3.3.1 Secondary Structure Loop Decomposition

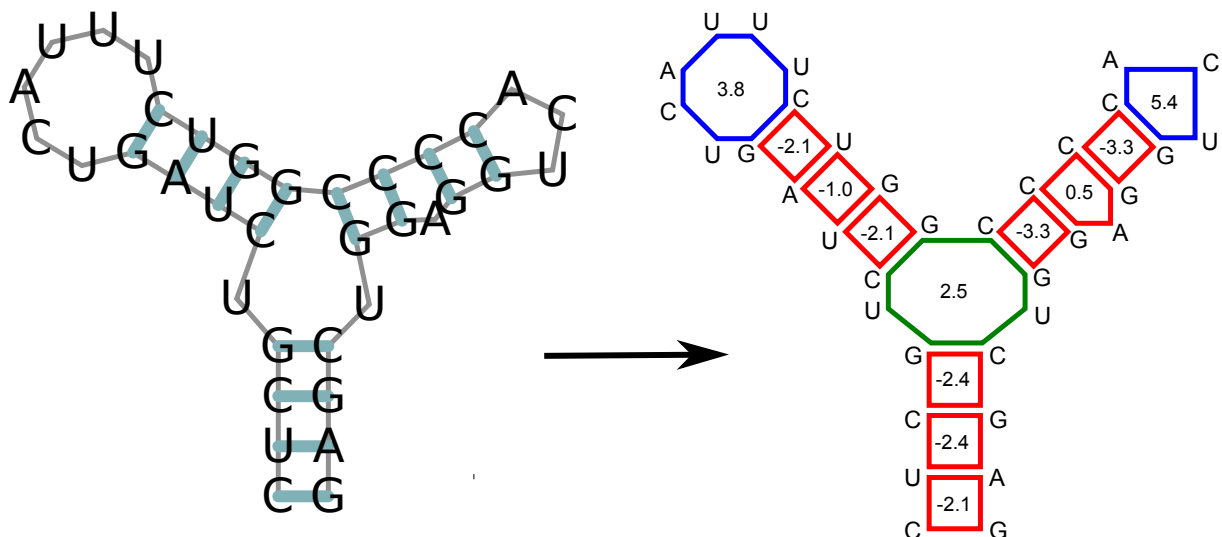
Each base pair in a secondary structure closes a loop, thereby directly enclosing unpaired nucleotides, and/or further base pairs. Our implementation distinguishes four basic types of loops:

- 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 **RNAlib** 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

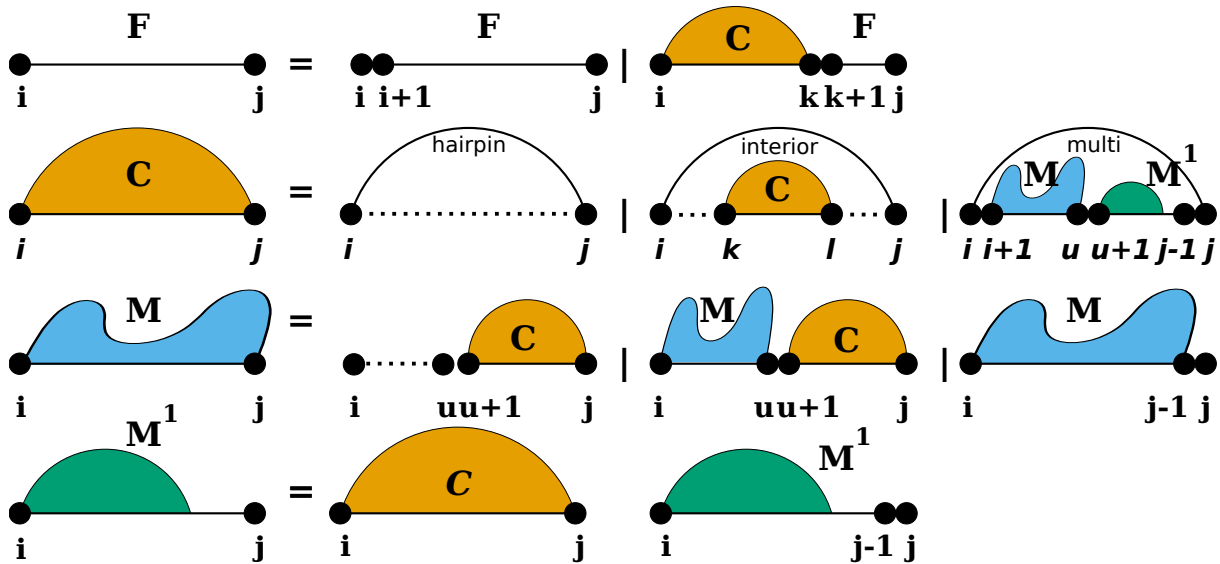
[Fine-tuning of the Implemented Models](#)

## 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

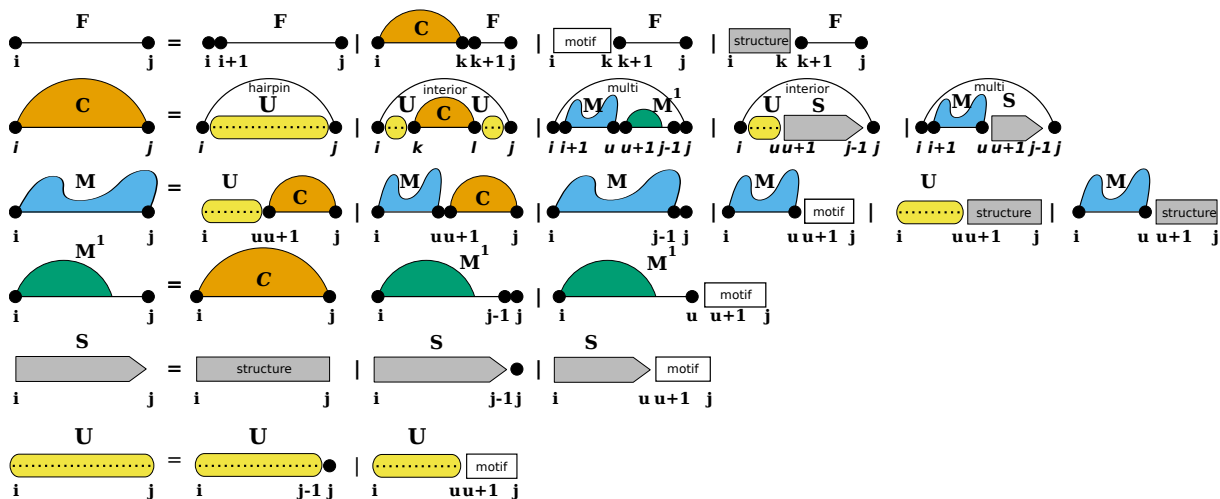
To predict secondary structures composed of the four distinguished loop types introduced before, all algorithms implemented in *RNAlib* follow a specific decomposition scheme, also known as the *RNA folding grammar*, or *Secondary Structure Folding Recurrences*.



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

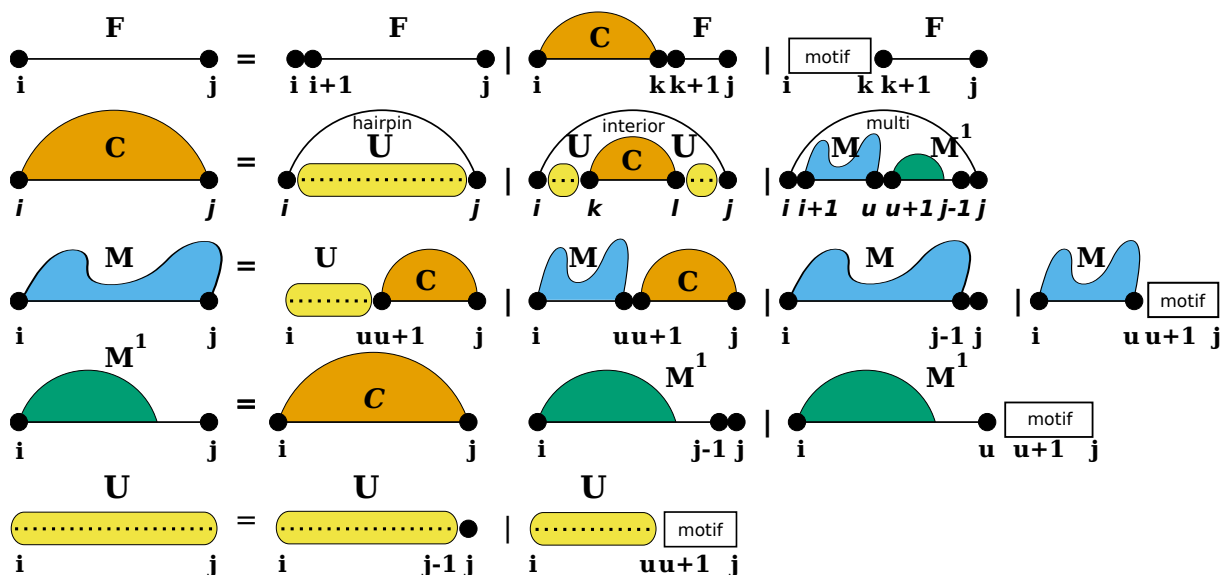
Usually, structured domains represent self-enclosed structural modules that exhibit a more or less complex base pairing pattern. This can be more or less well-defined 3D motifs, such as *G-Quadruplexes*, or loops with additional non-canonical base pair interactions, such as *kink-turns*.

#### 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, RNA-RNA Interaction, Computing MFE representatives of a Distance Based Partition](#)

## 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 [17] 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 [17]

### 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 [17]. See the corresponding submodules for specific implementation details.

See also

[Partition Function and Equilibrium Properties](#), [RNA-RNA Interaction](#), [Partition Function for two Hybridized Sequences as a Step](#), [Computing Partition Functions of a Distance Based Partitioning](#)

## 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](#), [Generate Soft Constraints from Data](#)

## 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 GCUAAUAGUUGUGAGGACCAACU
```



#### 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  $\mathbb{F}$  and  $\mathbb{W}$  commands, which remove conflicting base pairs, the  $\mathbb{A}$  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  $-5kcal/mol$  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 layouting 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 *pl,
                       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 [First Steps with the Fold Compound](#) 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_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_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_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_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_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_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_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_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_production` (`vrna_fold_compound_t *vc`, `void *data`)**

The production rule for the unstructured domain grammar extension

**Global `vrna_boltzmann_sampling_callback` (`const char *structure`, `void *data`)**

This function will be called for each secondary structure that has been successfully backtraced from the partition function DP matrices.

**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_subopt_callback` (`const char *structure`, `float energy`, `void *data`)**

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

**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_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_heat_capacity_callback` (`float temp`, `float heat_capacity`, `void *data`)**

This function will be called for each evaluated temperature in the heat capacity prediction.

**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_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()`.

## 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 [Perl5 Examples](#) and [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\\_params\\_subst](#)** (*vrna\_fold\_compound\_t* \*vc, *vrna\_param\_t* \*par)

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

When no parameter is passed, the resulting action is the same as passing *NULL* as second parameter to [vrna\\_params\\_subst\(\)](#), i.e. resetting the parameters to the global defaults.

**Global [vrna\\_heat\\_capacity\\_cb](#)** (*vrna\_fold\_compound\_t* \*fc, float T\_min, float T\_max, float T\_increment, unsigned int mpoints, *vrna\_heat\_capacity\_callback* \*cb, void \*data)

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

**Global [vrna\\_heat\\_capacity\\_simple](#)** (const char \*sequence, float T\_min, float T\_max, float T\_increment, unsigned int mpoints)

This function is available as overloaded function **heat\_capacity()**. If the optional function arguments T\_min, T\_max, T\_increment, and mpoints are omitted, they default to 0.0, 100.0, 1.0 and 2, respectively.

**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\\_backtrack5](#)** (*vrna\_fold\_compound\_t* \*fc, unsigned int length, char \*structure)

This function is attached as overloaded method **backtrack()** to objects of type *fold\_compound* with default parameter *length* equal to the total length of the RNA.

**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 [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\\_heat\\_capacity](#)** (`vrna_fold_compound_t *fc`, `float T_min`, `float T_max`, `float T_increment`, `unsigned int mpoints`)

This function is attached as overloaded method **`heat_capacity()`** to objects of type *fold\_compound*. If the optional function arguments `T_min`, `T_max`, `T_increment`, and `mpoints` are omitted, they default to 0.0, 100.0, 1.0 and 2, respectively.

Global **[vrna\\_exp\\_params\\_subst](#)** (`vrna_fold_compound_t *vc`, `vrna_exp_param_t *params`)

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

When no parameter is passed, the resulting action is the same as passing *NULL* as second parameter to [vrna\\_exp\\_params\\_subst\(\)](#), i.e. resetting the parameters to the global defaults.

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\\_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\\_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.

Global **[vrna\\_params\\_load](#)** (`const char fname[]`, `unsigned int options`)

This function is available as overloaded function **`params_load(fname="", options=VRNA_PARAMETER_FORMAT_DEFAULT)`**. Here, the empty filename string indicates to load default RNA parameters, i.e. this is equivalent to calling [vrna\\_params\\_load\\_defaults\(\)](#).

Global **[vrna\\_params\\_save](#)** (`const char fname[]`, `unsigned int options`)

This function is available as overloaded function **`params_save(fname, options=VRNA_PARAMETER_FORMAT_DEFAULT)`**.

Global **[vrna\\_params\\_load\\_from\\_string](#)** (`const char *string`, `const char *name`, `unsigned int options`)

This function is available as overloaded function **`params_load_from_string(string, name="", options=VRNA_PARAMETER_FOR`**

Global **[vrna\\_params\\_load\\_defaults](#)** (`void`)

This function is available as overloaded function **`params_load()`**.

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\\_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\\_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:

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:

Global **vrna\_params\_load\_RNA\_Turner2004** (void)

This function is available as function `params_load_RNA_Turner2004()`.

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\_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\_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*

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\_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\_path\_findpath** (vrna\_fold\_compound\_t \*fc, 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\_ub** (vrna\_fold\_compound\_t \*fc, 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_direct`** (`vrna_fold_compound_t *fc`, `const char *s1`, `const char *s2`, `vrna_path_options_t options`)

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

Global **`vrna_path_direct_ub`** (`vrna_fold_compound_t *fc`, `const char *s1`, `const char *s2`, `int maxE`, `vrna_path_options_t options`)

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

Global **`vrna_path_options_findpath`** (`int width`, `unsigned int type`)

This function is available as overloaded function `path_options_findpath()`. The optional parameter `width` defaults to 10 if omitted, while the optional parameter `type` defaults to `VRNA_PATH_TYPE_DOT_BRACKET`.

Global **`vrna_file_PS_aln`** (`const char *filename`, `const char **seqs`, `const char **names`, `const char *structure`, `unsigned int columns`)

This function is available as overloaded function `file_PS_aln()` with three additional parameters `start`, `end`, and `offset` before the `columns` argument. Thus, it resembles the `vrna_file_PS_aln_slice()` function. The last four arguments may be omitted, indicating the default of `start = 0`, `end = 0`, `offset = 0`, and `columns = 60`.

Global **`vrna_file_PS_aln_slice`** (`const char *filename`, `const char **seqs`, `const char **names`, `const char *structure`, `unsigned int start`, `unsigned int end`, `int offset`, `unsigned int columns`)

This function is available as overloaded function `file_PS_aln()` where the last four parameter may be omitted, indicating `start = 0`, `end = 0`, `offset = 0`, and `columns = 60`.

Global **`vrna_seq_encode`** (`const char *sequence`, `vrna_md_t *md`)

In the target scripting language, this function is wrapped as overloaded function `seq_encode()` where the last parameter, the `model_details` data structure, is optional. If it is omitted, default model settings are applied, i.e. default nucleotide letter conversion. The wrapped function returns a list/tuple of integer representations of the input sequence.

Global **`vrna_path_findpath_saddle_ub`** (`vrna_fold_compound_t *fc`, `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_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_db_pk_remove`** (`const char *structure`, `unsigned int options`)

This function is available as an overloaded function `db_pk_remove()` where the optional second parameter `options` defaults to `VRNA_BRACKETS_ANY`.

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_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_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\\_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\\_params\\_load\\_RNA\\_Turner1999](#)** (void)

This function is available as function [params\\_load\\_RNA\\_Turner1999\(\)](#).

Global **[vrna\\_params\\_load\\_RNA\\_Andronescu2007](#)** (void)

This function is available as function [params\\_load\\_RNA\\_Andronescu2007\(\)](#).

Global **[vrna\\_params\\_load\\_RNA\\_Langdon2018](#)** (void)

This function is available as function [params\\_load\\_RNA\\_Langdon2018\(\)](#).

Global **[vrna\\_params\\_load\\_RNA\\_misc\\_special\\_hairpins](#)** (void)

This function is available as function [params\\_load\\_RNA\\_misc\\_special\\_hairpins\(\)](#).

Global **[vrna\\_params\\_load\\_DNA\\_Mathews2004](#)** (void)

This function is available as function [params\\_load\\_DNA\\_Mathews2004\(\)](#).

Global **[vrna\\_params\\_load\\_DNA\\_Mathews1999](#)** (void)

This function is available as function [params\\_load\\_DNA\\_Mathews1999\(\)](#).

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\\_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\\_ensemble\\_defect](#)** (**[vrna\\_fold\\_compound\\_t](#)** \*fc, **const char** \*structure)

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

Global **[vrna\\_positional\\_entropy](#)** (**[vrna\\_fold\\_compound\\_t](#)** \*fc)

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

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

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

Global **[vrna\\_pr\\_energy](#)** (**[vrna\\_fold\\_compound\\_t](#)** \*vc, **double** e)

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

Global **[vrna\\_MEA](#)** (**[vrna\\_fold\\_compound\\_t](#)** \*fc, **double** gamma, **float** \*mea)

This function is attached as overloaded method **MEA**(gamma = 1.) to objects of type *fold\_compound*. Note, that it returns the MEA structure and MEA value as a tuple (MEA\_structure, MEA)

Global **[vrna\\_MEA\\_from\\_plist](#)** (**[vrna\\_ep\\_t](#)** \*plist, **const char** \*sequence, **double** gamma, **[vrna\\_md\\_t](#)** \*md, **float** \*mea)

This function is available as overloaded function **MEA\_from\_plist**(gamma = 1., md = NULL). Note, that it returns the MEA structure and MEA value as a tuple (MEA\_structure, MEA)

Global **[vrna\\_path\\_findpath\\_saddle](#)** (**[vrna\\_fold\\_compound\\_t](#)** \*fc, **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\\_rotational\\_symmetry\\_pos](#)** (**const char** \*string, unsigned int \*\*positions)

This function is available as overloaded global function **rotational\_symmetry()**. It merges the functionalities of [vrna\\_rotational\\_symmetry\(\)](#), [vrna\\_rotational\\_symmetry\\_pos\(\)](#), [vrna\\_rotational\\_symmetry\\_num\(\)](#), and [vrna\\_rotational\\_symmetry\\_pos\\_num\(\)](#). In contrast to our C-implementation, this function doesn't return the order of rotational symmetry as a single value, but returns a list of cyclic permutation shifts that result in a rotationally symmetric string. The length of the list then determines the order of rotational symmetry.

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\\_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\\_hc\\_init](#)** ([vrna\\_fold\\_compound\\_t](#) \*vc)

This function is attached as method **[hc\\_init\(\)](#)** 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\\_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\\_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\\_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\\_pbacktrack\\_cb](#)** ([vrna\\_fold\\_compound\\_t](#) \*fc, unsigned int num\_samples, [vrna\\_boltzmann\\_sampling\\_callback](#) \*cb, void \*data, unsigned int options)

This function is attached as overloaded method **[pbacktrack\(\)](#)** to objects of type *fold\_compound* where the last argument *options* is optional with default value *options* = `VRNA_PBACKTRACK_DEFAULT`. See also [Python Examples - Boltzmann Sampling](#)

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\\_maximum\\_matching\\_simple](#)** (const char \*sequence)

This function is available as global function **[maximum\\_matching\(\)](#)**.

**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_pbacktrack5` (`vrna_fold_compound_t *fc`, unsigned int length)**

This function is attached as overloaded method `pbacktrack5()` to objects of type `fold_compound`. See also [Python Examples - Boltzmann Sampling](#)

**Global `vrna_pbacktrack` (`vrna_fold_compound_t *fc`)**

This function is attached as overloaded method `pbacktrack()` to objects of type `fold_compound`. See also [Python Examples - Boltzmann Sampling](#)

**Global `vrna_pbacktrack5_num` (`vrna_fold_compound_t *fc`, unsigned int num\_samples, unsigned int length, unsigned int options)**

This function is attached as overloaded method `pbacktrack5()` to objects of type `fold_compound` where the last argument `options` is optional with default value `options = VRNA_PBACKTRACK_DEFAULT`. See also [Python Examples - Boltzmann Sampling](#)

**Global `vrna_pbacktrack_num` (`vrna_fold_compound_t *fc`, unsigned int num\_samples, unsigned int options)**

This function is attached as overloaded method `pbacktrack()` to objects of type `fold_compound` where the last argument `options` is optional with default value `options = VRNA_PBACKTRACK_DEFAULT`. See also [Python Examples - Boltzmann Sampling](#)

**Global `vrna_pbacktrack5_cb` (`vrna_fold_compound_t *fc`, unsigned int num\_samples, unsigned int length, `vrna_boltzmann_sampling_callback *cb`, void \*data, unsigned int options)**

This function is attached as overloaded method `pbacktrack5()` to objects of type `fold_compound` where the last argument `options` is optional with default value `options = VRNA_PBACKTRACK_DEFAULT`. See also [Python Examples - Boltzmann Sampling](#)

**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:

**Global `vrna_pbacktrack5_resume` (`vrna_fold_compound_t *vc`, unsigned int num\_samples, unsigned int length, `vrna_pbacktrack_mem_t *nr_mem`, unsigned int options)**

This function is attached as overloaded method `pbacktrack5()` to objects of type `fold_compound`. In addition to the list of structures, this function also returns the `nr_mem` data structure as first element. See also [Python Examples - Boltzmann Sampling](#)

**Global `vrna_pbacktrack_resume` (`vrna_fold_compound_t *fc`, unsigned int num\_samples, `vrna_pbacktrack_mem_t *nr_mem`, unsigned int options)**

This function is attached as overloaded method `pbacktrack()` to objects of type `fold_compound`. In addition to the list of structures, this function also returns the `nr_mem` data structure as first element. See also [Python Examples - Boltzmann Sampling](#)

**Global `vrna_pbacktrack5_resume_cb` (`vrna_fold_compound_t *fc`, unsigned int num\_samples, unsigned int length, `vrna_boltzmann_sampling_callback *cb`, void \*data, `vrna_pbacktrack_mem_t *nr_mem`, unsigned int options)**

This function is attached as overloaded method `pbacktrack5()` to objects of type `fold_compound`. In addition to the number of structures backtraced, this function also returns the `nr_mem` data structure as first element. See also [Python Examples - Boltzmann Sampling](#)

**Global `vrna_pbacktrack_resume_cb` (`vrna_fold_compound_t *fc`, unsigned int num\_samples, `vrna_boltzmann_sampling_callback *cb`, void \*data, `vrna_pbacktrack_mem_t *nr_mem`, unsigned int options)**

This function is attached as overloaded method `pbacktrack()` to objects of type `fold_compound`. In addition to the number of structures backtraced, this function also returns the `nr_mem` data structure as first element. See also [Python Examples - Boltzmann Sampling](#)

**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\\_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\\_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\\_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\\_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\\_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\\_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\\_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\\_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\\_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\\_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\\_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\\_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_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_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_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_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_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_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_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_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_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:

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_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_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_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_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\\_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\\_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\\_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\\_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\\_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*

## **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[] = {
        "CUGCCUCACAACGUUUUGUGCCUCAGUUACCCGUAGAUGUAGUGAGGGU",
        "CUGCCUCACAACAUUUUGUGCCUCAGUUACUCAUAGAUGUAGUGAGGGU",
        "---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 [ %6.2f ]\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 = "GAGUAGUGGAACCAGGCUAUGUUUGUGACUCGCAGACUAACA";
    /* 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 [ %.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 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(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 [ %6.2f ]\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 =
        "AGACGACAAGGUUGAAUCGCACCCACAGUCUAUGAGUCGGUGACAACAUUACGAAAGGCUGUAAAAUCAAUUACACCACAGGGGGCCCCGUGUCUAG";
    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 (%6.2f)\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 (%6.2f)\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 (%6.2f d=%6.2f)\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          *pf1, *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 $seq = "GAGUAGUGGAACCCAGGCUAUGUUUGUGACUCGCAGACUAACA";
# compute minimum free energy (MFE) and corresponding structure
my ($ss, $mfe) = RNA::fold($seq);
# print output
printf "%s\n%s [ %6.2f ]\n", $seq, $ss, $mfe;
```
- comparative MFE prediction for sequence alignments
 

```
use RNA;
# The RNA sequence alignment
my @sequences = (
    "CUGCCUCACACGUAUUGUGCCUCAGUUACCCGUAUGUAGUGAGGGU",
    "CUGCCUCACACAUUUGUGCCUCAGUUACUACAUAGAUAGUGAGGGU",
    "---CUCGACACCACU---GCCUCGGUACCAUCGGUGCAGUGCGGGU"
);
# 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;
```

#### 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 object oriented interface

- MFE prediction at different temperature and dangle model
 

```
use RNA;
# The RNA sequence
my $seq = "GAGUAGUGGAACCCAGGCUAUGUUUGUGACUCGCAGACUAACA";
# 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;
```

## 8.3 Python Examples

### MFE Prediction (flat interface)

```
import RNA
# The RNA sequence
seq = "GAGUAGUGGAACCGGCUAUGUUUGUGACUCGACAGACUAACA"
# compute minimum free energy (MFE) and corresponding structure
(ss, mfe) = RNA.fold(seq)
# print output
print "%s\n%s [ %.2f ]" % (seq, ss, mfe)
```

### MFE Prediction (object oriented interface)

```
import RNA;
sequence = "CGCAGGGGAUACCCGCG"
# 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 [ %.2f ]" % (ss, mfe)
```

### Suboptimal Structure Prediction

```
import RNA
sequence = "GGGGAAAACCCC"
# 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 [%.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 docal/mol = 5 kcal/mol arround
# the MFE and print each structure using the function above
a.subopt_cb(500, print_subopt_result, subopt_data);
```

### Boltzmann Sampling (a.k.a. Probabilistic Backtracing)

```
import RNA
sequence =
    "UGGGAAUAGUCUCUUCGAGUCUCGCGGGCGACGGGCGAUCUUCGAAAGUGGAAUCCGUACUUAUACCGCCUGUGCGGACUACUAUCCUGACCACAUAGU"
"""
A simple callback function that stores
a structure sample into a list
"""
def store_structure(s, data):
    if s:
        data.append(s)
"""
First we prepare a fold_compound object
"""
# create model details
md = RNA.md()
# activate unique multibranch loop decomposition
md.uniq_ML = 1
# create fold compound object
fc = RNA.fold_compound(sequence, md)
# compute MFE
(ss, mfe) = fc.mfe()
# rescale Boltzmann factors according to MFE
fc.exp_params_rescale(mfe)
# compute partition function to fill DP matrices
fc.pf()
"""
Now we are ready to perform Boltzmann sampling
"""
# 1. backtrace a single sub-structure of length 10
print "%s" % fc.pbacktrack5(10)
```



```

# 2. backtrack a single sub-structure of length 50
print "%s" % fc.pbacktrack5(50)
# 3. backtrack multiple sub-structures of length 10 at once
for s in fc.pbacktrack5(20, 10):
    print "%s" % s
# 4. backtrack multiple sub-structures of length 50 at once
for s in fc.pbacktrack5(100, 50):
    print "%s" % s
# 5. backtrack a single structure (full length)
print "%s" % fc.pbacktrack()
# 6. backtrack multiple structures at once
for s in fc.pbacktrack(100):
    print "%s" % s
# 7. backtrack multiple structures non-redundantly
for s in fc.pbacktrack(100, RNA.PBACKTRACK_NON_REDUNDANT):
    print "%s" % s
# 8. backtrack multiple structures non-redundantly (with resume option)
num_samples = 500
iterations = 15
d = None # pbacktrack memory object
s_list = list()
for i in range(0, iterations):
    d, ss = fc.pbacktrack(num_samples, d, RNA.PBACKTRACK_NON_REDUNDANT)
    s_list = s_list + list(ss)
for s in s_list:
    print "%s" % s
# 9. backtrack multiple sub-structures of length 50 in callback mode
ss = list()
i = fc.pbacktrack5(100, 50, store_structure, ss)
for s in ss:
    print "%s" % s
# 10. backtrack multiple full-length structures in callback mode
ss = list()
i = fc.pbacktrack(100, store_structure, ss)
for s in ss:
    print "%s" % s
# 11. non-redundantly backtrack multiple full-length structures in callback mode
ss = list()
i = fc.pbacktrack(100, store_structure, ss, RNA.PBACKTRACK_NON_REDUNDANT)
for s in ss:
    print "%s" % s
# 12. non-redundantly backtrack multiple full length structures
# in callback mode with resume option
ss = list()
d = None # pbacktrack memory object
for i in range(0, iterations):
    d, i = fc.pbacktrack(num_samples, store_structure, ss, d, RNA.PBACKTRACK_NON_REDUNDANT)
for s in ss:
    print "%s" % s

```

## RNAfold -p –MEA equivalent

```

#!/usr/bin/python
#
import RNA
seq = "AUUUCACUAGAGAAGGUCUAGAGUGUUUGUCGUUUGUCAGAGUCCCUAUUCCAGGUACGAACACGGUGGAUUAUGUUCGACGACAGGAUCGGCGCACUA"
# create fold_compound data structure (required for all subsequently applied algorithms)
fc = RNA.fold_compound(seq)
# compute MFE and MFE structure
(mfe_struct, mfe) = fc.mfe()
# rescale Boltzmann factors for partition function computation
fc.exp_params_rescale(mfe)
# compute partition function
(pp, pf) = fc.pf()
# compute centroid structure
(centroid_struct, dist) = fc.centroid()
# compute free energy of centroid structure
centroid_en = fc.eval_structure(centroid_struct)
# compute MEA structure
(MEA_struct, MEA) = fc.MEA()
# compute free energy of MEA structure
MEA_en = fc.eval_structure(MEA_struct)
# print everything like RNAfold -p --MEA
print("%s\n%s (%6.2f)" % (seq, mfe_struct, mfe))
print("%s [%6.2f]" % (pp, pf))
print("%s {%6.2f d=%6.2f}" % (centroid_struct, centroid_en, dist))
print("%s {%6.2f MEA=%6.2f}" % (MEA_struct, MEA_en, MEA))
print(" frequency of mfe structure in ensemble %g; ensemble diversity %6.2f" %
      (fc.pr_structure(mfe_struct), fc.mean_bp_distance()))

```

## Fun with Soft Constraints

```
import RNA
```

```

seq1 = "CUCGUCGCCUUAUCCAGUGCGGGCGCUAGACAUCUAGUUAUCGCCGCAA"
# 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_ML_ML:     lambda i, j, k, l, f, p: 0,
    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_EXT:      lambda i, j, k, l, f, p: 0,
    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

# Contributing

If you wish to contribute to this project, please first discuss any proposed changes with the owners and main developers. You may do that either through making an issue at our official GitHub presence <https://github.com/ViennaRNA/ViennaRNA>, by email ( [rna@tbi.univie.ac.at](mailto:rna@tbi.univie.ac.at)), or any other personal communication with the core developer team.

Please note that we have a code of conduct. Please follow it in all your interactions with this project.

## Reporting Bugs

1. Please make an issue at GitHub or notify us by emailing to [rna@tbi.univie.ac.at](mailto:rna@tbi.univie.ac.at)
2. In your report, include as much information as possible, such that we are able to reproduce it. If possible, find a minimal example that triggers the bug.
3. Include the version number for the ViennaRNA Package you experience the bug with.
4. Include at least some minimal information regarding your operating system (Linux, Mac OS X, Windows, etc.)

## Pull Request Process

1. Ensure that you have not checked-in any files that are automatically build!
2. When contributing C source code, follow our code formatting guide lines. You may use the tool `uncrustify` together with our config located in `misc/uncrustify.cfg` to accomplish that.
3. Only expose symbols (functions, variables, etc.) to the libraries interface that are absolutely necessary! Hide all other symbols in the corresponding object file(s) by declaring them as `static`.
4. Use the prefixes `vrna_` for any symbol you add to the API of our library! Preprocessor macros in header files require the prefix in capital letters, i.e. `VRNA_`.
5. Use C-style comments at any place necessary to make sure your implementation can still be understood and followed in the future.
6. Add test cases for any new implementation! The test suite is located in the `tests` directory and is split into tests for the C-library, executable programs, and the individual scripting language interfaces.
7. Run `make check` to ensure that all other test suites still run properly with your applied changes!
8. When contributing via GitHub, make a personal fork of our project and create a separate branch for your changes. Then make a pull request to our `user-contrib` branch. Pull requests to the `master` branch will be rejected to keep its history clean.
9. Pull requests that have been successfully merged into the `user-contrib` branch usually find their way into the next release of the ViennaRNA Package. However, please note that the core developers may decide to include your changes in a later version.



# Chapter 10

## Changelog

Below, you'll find a list of notable changes for each version of the ViennaRNA Package.

### Version 2.4.x

#### Unreleased

#### Version 2.4.17 (Release date: 2020-11-25)

##### Programs

- Fix RNAup -b mode with shorter sequence first
- Add `--backtrack-global` option to RNALfold (currently only available for dangles == 2 | 0)
- Add `--zscore-pre-filter` and `--zscore-report-subsumed` options to RNALfold

##### Library

- API: Fix multiloop backtracing with soft constraints for unpaired positions in `vrna_subopt()` and `vrna_subopt_cb()`
- API: Fix parameter parse in `vrna_params_load_from_string()`
- API: Add `vrna_heat_capacity()` and `vrna_head_capacity_cb()` functions to RNAlib
- API: Add backtracing function `vrna_backtrack_window()` for global MFE structure to sliding-window predictions
- API: Add SVG support for RNApuzzler structure layouts
- API: Make `vrna_md_t` argument to `vrna_fold_compound()` a constant pointer
- API: Remove missing symbols from header file `ViennaRNA/params/default.h`
- API: Refactor z-score threshold filter handling for sliding-window MFE prediction
- SWIG: Fix typo in interface functions to load DNA parameters
- SWIG: Add python-3.9 autoconf checks
- SWIG: Add `vrna_head_capacity*()` wrappers
- SWIG: Add access to raw energy parameters
- SWIG: Add `alias` and `pair` attribute to objects of type `md`
- SWIG: Add out/varout typemaps for 2-dimensional int-like arrays
- SWIG: Add all data fields to objects of type 'param' and 'exp\_param'

## Package

- Fix Debian and Windows installer files

## Version 2.4.16 (Release date: 2020-10-09)

## Programs

- Fix backtracing errors in `RNAfold` for alignments with more than 32768 columns
- Fix backtracing errors in `RNAfold` and `RNAfold` for rare cases when two alignment columns may pair due to covariance score threshold but still yield infinite energies due to energy model
- Refactored manpages/help options for `RNAfold`, `RNAplot`, `RNApvm`, `RNAsubopt`, and `RNAup`

## Library

- API: Fix undefined behavior due to short int overflows when accessing alignment lengths with alignments larger than 32768 columns. This fixes occasional backtracing errors in `RNAfold` and `vrna_mfe_window()`
- API: Fix adding pscore to base pairs that yield INF energy in comparative global and local MFE prediction
- API: Add `vrna_convert_kcal_to_dcal()` and vice-versa function for safely converting integer to float representations of energy values
- SWIG: Add a reasonable Python interface for objects of type `vrna_path_t`
- SWIG: Add a wrapper for `vrna_seq_encode()`

## Package

- Move `units.h` include file to `ViennaRNA/Utils/units.h`

## Version 2.4.15 (Release date: 2020-08-18)

## Programs

- Fix compilation of `Kinfold` with GCC 10
- Add `--en-only` flag to `RNAsubopt` to allow for sorting by energy only
- Prevent `RNAcofold` to process input with more than two strands
- Add cutpoint marker to dot-plots created with `RNAcofold -a`
- Update `Kinfold` to version 1.4

## Library

- API: Fix removal of strand delimiter in `vrna_plot_dp_PS_list()`
- API: Fix `vrna_enumerate_necklaces()`
- API: Fix bogus backtracing for co-folded structures in `vrna_subopt()` and `vrna_subopt_cb()`
- API: Fix storing co-folded structures for sorted output in `vrna_subopt()`
- API: Fix multibranch loop component hard constraints for multi-strand cases
- API: Prevent adding internal loop energy contributions to enclosed parts with `energy=INF`
- API: Adapt `vrna_db_pack()` / `vrna_db_unpack()` functions to produce comparable strings
- API: Add sorting modes `VRNA_UNSORTED`, `VRNA_SORT_BY_ENERGY_LEXICOGRAPHIC_ASC`, and `VRNA_SORT_BY_ENERGY_ASC` to `vrna_subopt()`
- API: Add `vrna_strjoin()` function
- API: Add missing case to external loop hard constraints
- API: Make hard constraints strand-aware
- SWIG: Fix invalid memory access when using `MEA_from_plist()` in Perl 5 or Python
- SWIG: Enable keyword argument features in Python interface of constructors for `fold_compound`, `md`, `move`, `param`, and `exp_param` objects
- SWIG: Enable autodoc feature for Python interface of constructors for `fold_compound`, `md`, and `move` objects
- SWIG: Enable `toString` conversion for Python interface for objects of type `fold_compound`, `md`, `move`, `params`, `exp_params`, and `subopt_solution`
- SWIG: Add (read-only) attributes `type`, `length`, `strands`, `params`, and `exp_params` to objects of type `fold_compound`
- SWIG: Make attributes of objects of type `param` and `exp_param` read-only
- Add array of strand nicks to EPS dot plot files instead of single cutpoint
- Draw separator line for each strand nick in EPS dot-plots
- Update `libsvm` to version 3.24

## Package

- Disable Link-Time-Optimization (LTO) for third-party programs linking against `RNAlib` using `pkg-config`
- TESTS: Fix results dir path for out-of-tree builds
- TESTS: Set default timeout for library tests to 20s

## Version 2.4.14 (Release date: 2019-08-13)

## Programs

- Fix `RNApvmin` pertubation vector computation
- Add non-redundant sampling option to `RNApvmin`
- Add `RNAados` program to compute density of states
- Add `-P` DNA convenience command line parameter to most programs to quickly load DNA parameters without any input file
- MAN: Add example section to man-page of `RNAalifold`

## Library

- API: Fix memory leak in `vrna_path_gradient()`
- API: Fix release of memory for `vrna_sequence_remove_all()`
- API: Fix soft-constraints application in `vrna_sc_minimize_perturbation()` that prevented proper computation of the perturbation vector
- API: Add 5' and 3' neighbor nucleotide encoding arrays and name string to `vrna_seq_t`
- API: Add new data structure for multiple sequence alignments
- API: Add `vrna_sequence_order_update()` function
- API: Add non-redundant sampling mode to `vrna_sc_minimize_perturbation()` through passing negative sample-sizes
- API: Add v3.0 API functions for maximum expected accuracy (MEA) computation
- API: Include energy parameter sets into RNAlib and provide functions to load them at runtime
- API: Prepare sequence data in `vrna_fold_compound_t` with `vrna_sequence_add()`
- API: Use `vrna_pbacktrack_num()` instead of `vrna_pbacktrack()` in `vrna_sc_minimize_perturbation()` to speed-up sample generation
- Reduce use of global variable `cut_point` in RNAlib
- SWIG: Use `importlib` in favor of `imp` to determine Python 3 tag extension
- SWIG: Update various wrapper functions
- SWIG: Add wrappers for MEA computation with `vrna_MEA()` and `vrna_MEA_from_plist`
- SWIG: Add wrappers for `vrna_pr_structure()` and `vrna_pr_energy()`

## Package

- REFMAN: Fix LaTeX code in `units.h` that prevented proper compilation with `pdflatex`
- Add an R script to create 2D landscape plots from RNA2Dfold output
- Add `gengetopt` to configure-time requirements to build man-pages
- Add new energy parameter file `rna_misc_special_hairpins.par` with additional UV-melting derived parameters for Tri- and Tetra-loops
- Update RNA Tutorial
- Colorize final configure script message
- REFMAN: Always use `pdflatex` to compile reference manual and tutorial
- EXAMPLES: Add Python script that performs computations equivalent to `RNAfold -p --MEA`

## Version 2.4.13 (Release date: 2019-05-30)

## Programs

- Fix centroid structure prediction for RNAcofold
- Fix `--noLP` option for RNALalifold



## Library

- API: Refactor and fix collision handling in `vrna_hash_table_t`
- API: Fix one access using wrong index for odd dangles in `loops/external.c`
- API: Add two missing `MLbase` contributions for MFE prediction in `loops/multibranch.c`
- API: Refactor multiloop MFE backtracking for odd dangles
- API: Add function `vrna_backtrack5()` to allow for MFE backtracking of sub-sequences starting at the 5'-end
- API: Reduce usage of global macro `TURN` by replacing it with `min_loop_size` field of `vrna_md_t`
- API: Add functions `vrna_path_direct()` and `vrna_path_direct_ub()` that may also return move lists instead of dot-bracket lists
- API: Add functions `vrna_pt_pk_remove()` and `vrna_db_pk_remove()` that remove pseudoknots from an input structure
- API: Fix invalid memory access for lonely pair mode (`--noLP`) in comparative sliding-window MFE prediction
- SWIG: Fix access to global variable `pf_smooth` and `pf_smooth` attribute in `model_details` object
- SWIG: Fix Python reference counting for `Py_None` in `interfaces/findpath.i` wrapper
- SWIG: Refactor reference counting for all Python2 and Python3 wrappers
- REFMAN: Larger updates and restructuring of reference manual

## Package

- Install example scripts and source code files, e.g. to `$prefix/share/ViennaRNA/examples`
- Properly pass `GSL`, `PTHREADS`, and `MPFR` flags to sub-projects
- Fix `RNApuzzler` header file installation
- SWIG: Include Python 3.7 and 3.8 in list of autoconf-probed python interpreters
- SWIG: Fix wrapper building for `swig >= 4.0.0`

## Version 2.4.12 (Release date: 2019-04-16)

## Programs

- Add non-redundant stochastic backtracing option for `RNAalifold`
- Add `--noDP` option to suppress dot-plot output in `RNAfold` and `RNAalifold`
- Add `RNApuzzler` (4) and `RNAturtle` (3) secondary structure layout algorithm options to `RNAfold` and `RNAplot`
- Update help/man page of `RNAfold`
- Allow for multiple input files and parallel input processing in `RNAheat`

## Library

- API: Fix declaration of `vrna_move_apply_db()`
- API: Fix `vrna_path()` lexicographical ordering in gradient walks
- API: Enable non-redundant stochastic backtracing for comparative structure prediction
- API: Enable stochastic backtracing for circular comparative structure prediction
- API: Enable stochastic backtracing of subsequences (5' prefixes) for comparative structure prediction
- API: Add `pf_smooth` attribute to `vrna_md_t` data structure to allow for disabling Boltzmann factor energy smoothing
- API: Add functions to allow for resuming non-redundant stochastic backtracing
- API: Add functions to retrieve multiple stochastically backtraced structures (list and callback variants)
- API: Add `vrna_positional_entropy` to compute vector of positional entropies
- API: Add `RNApuzzler` and `RNAturtle` secondary structure layout algorithm (Wiegreffe et al. 2018)
- API: Add v3.0 API for secondary structure layout/coordinate algorithms
- API: Add more helper/utility functions for `vrna_move_t` data structures
- API: Add callback-based neighborhood update function for (subsequent) `vrna_move_t` application
- API: Add abstract heap data structure available as `<ViennaRNA/datastructures/heap.h>`
- API: Refactor and speed-up gradient walk implementation available as `vrna_path_gradient()`
- API: Substitute `vrna_file_PS_aln_sub()` alignment plot function by `vrna_file_PS_aln_slice()` that actually slices out a sub-alignment
- API: Rename `vrna_annotate_covar_struct()` to `vrna_annotate_covar_db()` and add new function `vrna_annotate_covar_db_extended()` to support more bracket types
- API: Calling `vrna_params_reset()` now implies a call to `vrna_exp_params_reset()` as well
- API: Move landscape implementations into separate directory, thus headers should be included as `<ViennaRNA/landscape/move.h>`, `<ViennaRNA/landscape/neighbor.h>`, etc.
- Ensure proper rescaling of energy parameters upon temperature changes
- Refactor soft constraints implementation in stochastic backtracing
- SWIG: Wrap all non-redundant stochastic backtracing functions to scripting language interface(s)
- SWIG: Refactor stochastic backtracing interface(s)
- SWIG: Add proper constructor for objects of type `vrna_ep_t`
- SWIG: Sanitize alignment plot function interface(s)

## Package

- Update Ubuntu/Debian and OpenSUSE build instructions
- Reduce intra-package dependency on non-v3.0 API

## Version 2.4.11 (Release date: 2018-12-17)

### Programs

- Add `--commands` option to `RNAsubopt`
- Add non-redundant Boltzmann sampling mode for `RNAsubopt`

### Library

- API: Fix wrong access to base pair soft constraints in equilibrium probability computations
- API: Fix behavior of `vrna_nucleotide_encode()` with lowercase characters in sequence
- API: Fix behavior of `encode_char()` with lowercase characters in sequence
- API: Fix forbidden GU pairs behavior in `pscore` computation for comparative folding
- API: Fix potential errors due to uninitialized `next` pointers in `vrna_move_t` of `vrna_eval_move_t` `shift_pt`
- API: Add AVX 512 optimized version of MFE multibranch loop decomposition
- API: Add functions for CPU SIMD feature detection
- API: Add dispatcher to automatically delegate exterior-/multibranch loop MFE decomposition to supported SIMD optimized implementation
- API: Add function `vrna_dist_mountain()` to compute mountain distance between two structures
- API: Add function `vrna_ensemble_defect()` to compute ensemble defect given a target structure
- API: Add non-redundant Boltzmann sampling
- API: Change behavior of `vrna_cstr_free()` and `vrna_cstr_close()` to always flush output before unregistering the stream
- SWIG: Add interface for `vrna_loopidx_from_ptable()`

### Package

- Activate compilation for compile-time supported SIMD optimized implementations by default
- Replace `--enable-sse` configure script option with `--disable-simd`

## Version 2.4.10 (Release date: 2018-09-26)

### Programs

- Fix wrong output filename for binary opening energies in `RNAplfold`
- Enable G-Quadruplex support for partition function computation in `RNAalifold`

## Library

- Fix broken SSE4.1 support for multibranch loop MFE computation that resulted in increased run times
- Fix redundant output issue in subopt backtracking with unusually high delta energies ( $\geq \text{INF}$ )
- Restore default behavior of '|' symbol in dot-bracket hard constraint strings that got lost with version 2.2.0
- Add faster (cache-optimized) version of Nussinov Maximum Matching algorithm
- Change default linker- and loop length computations for G-Quadruplex predictions in comparative prediction modes
- Add hard constraints warning for base pairs that violate the `min_loop_size` of the model
- Update `libsvm` to version 3.23
- API: Add functions to set auxiliary grammar extension rules
- API: Replace upper-triangular hard constraints matrix with full matrix for cache-optimized access
- API: Add G-Quadruplex prediction support for comparative partition function
- API: Remove `VRNA_GQUAD_MISMATCH_PENALTY` and `VRNA_GQUAD_MISMATCH_NUM_ALI` macros
- SWIG: Fix invalid memory access in `subopt()` method of `fold_compound` object when writing to file
- SWIG: Add wrapper for Nussinov Maximum Matching algorithm

## Package

- Add `-ftree-vectorize` compile flag by default if supported

## Version 2.4.9 (Release date: 2018-07-11)

## Programs

- Fix interactive mode behavior for multiple sequence alignment input in `RNAalifold`, `RNAalifold`
- Allow for Stockholm formatted multiple sequence alignment input in `RNAeval` and `RNAplot`
- Allow for multiple input files in `RNAeval` and `RNAplot`
- Allow for parallel processing of input batch jobs in `RNAeval` and `RNAplot`
- Add `-g` option to activate G-Quadruplex support in `RNAheat`
- Warn on unsatisfiable hard constraints from dot-bracket string input in `RNAfold`, `RNAcofold`, and `RNAalifold`

## Library

- Fix parameter order bug in `vrna_path_findpath*` functions that resulted in too large search widths
- Fix wrong application of base pair soft constraints in partition function computations
- Fix position ruler string in EPS alignment output files
- Fix MFE backtracking errors that might appear under specific hard constrained base pair patterns
- Refrain from reading anything other than `#=GC SS_cons` to retrieve structures when parsing Stockholm 1.0 format
- Complete soft constraints additions to Boltzmann sampling implementation for single sequences
- Allow for disabling alignment wrapping in `vrna_file_PS_aln*` functions
- Do not remove G-Quadruplex annotation from WUSS formatted structure strings upon calls to `vrna_db_↵  
from_WUSS`
- Enable G-Quadruplex related average loop energy correction terms in verbose output of `vrna_eval_*` functions
- Speed-up backward compatibility layer for energy evaluation functions that unnecessarily slowed down third-party tools using the old API
- Allow for passing dot-bracket strings with `"&strand-end identifier to simplevrna_eval_↵  
*functions`
- Remove `implicitexit()` calls from global MFE backtracking implementation.

## Version 2.4.8 (Release date: 2018-06-23)

## Programs

- Fix compilation of RNAforester with C++17 standard
- Fix tty input detection in RNAcofold
- Fix bad memory access with RNAcofold -p

## Library

- API: Fix incorrect unpaired probability computations in `vrna_probs_window()`
- API: Fix potential out-of-bounds access situations (for circular RNA folding) in `eval.c`
- API: Fix comparative exterior internal loop partition function computation for `circfold`
- SWIG: Fix false-positive use of uninitialized value in `Python3/file_py3.i`

## Package

- TESTS: Add tests for special features in RNAalifold
- TESTS: Add test case for RNAcofold -p

**Version 2.4.7 (Release date: 2018-06-13)**

- Allow for parallel processing across multiple input files in RNAfold
- Allow for arbitrary number of input files in RNAalifold
- Allow for parallel processing of input data in RNAalifold
- Allow for arbitrary number of input files in RNAcifold
- Allow for parallel processing of input data in RNAcifold
- Enable parallel processing in RNAfold, RNAcifold, RNAalifold for MS Windows build
- Add centroid and MEA structure computation to RNAcifold
- Add configure time check for LTO capabilities of the linker
- Include ligand binding energies in centroid and MEA structure output of RNAfold
- Refactor ct2db program to process multiple structures from single .ct file
- API: Enable processing of comparative fold\_compound with `vrna_pr_*`() functions
- API: Refactor `vrna_ostream_t` to enable NULL input in [vrna\\_ostream\\_provide\(\)](#)
- API: Major refactoring in loop energy evaluations (MFE and PF)
- API: Make `vrna_mx_pf_aux_el_t` and `vrna_mx_pf_aux_ml_s` opaque pointers
- API: Make `fold_compound` field `type` a const attribute
- API: Refactor MFE post-processing for circular RNAs
- API: Add motif name/id support for unstructured domains
- API: Remove major part of implicit `exit()` calls in RNAlib
- API: Add implementations of Boyer-Moore-Horspool search algorithm
- API: Add implementations to determine number of rotational symmetry for strings (of objects)
- API: Make `vrna_cmd_t` an opaque pointer
- API: Move headers for constraints, datastructures, io, loop energy evaluation, energy parameters, plotting, search, and utilities into separate subdirectories (backward compatibility is maintained)
- API: Add hash table data structure
- API: Fix discrepancy between comparative and single sequence `-noLP` predictions
- API: Add functions to replace 'old API' interface of [RNAstruct.h](#)
- API: Add functions to replace 'old API' interface of [aln\\_util.h](#)
- API: Add generic soft constraints support to suboptimal structure prediction sensu Wuchty et al.
- SWIG: Refactor callback execution for Python 2 / 3 interface to reduce overhead
- SWIG: Fix configure-time check for Python 3 interface build
- SWIG: Fix Python 3 IO file stream to C FILE \* conversion
- Cosmetic changes in final configure notice
- Major changes in source tree structure of the library
- Add autoconf checks for maintainer tools
- Generate C strings from static PostScript files at configure time (for structure- and dot plots)
- REFMAN: Large updates in API documentation and structure of reference manual

### Version 2.4.6 (Release date: 2018-04-19)

- Stabilize rounding of free energy output in RNAalifold
- API: Fix potential rounding errors for comparative free energies in eval.c and mfe.c
- API: Fix regression in exterior loop dangling end contributions for comparative base pair probabilities and Boltzmann sampling (introduced with v2.4.4)
- API: Fix regression with hard constrained base pairs for comparative structure prediction (introduced with v2.4.4)
- TESTS: Add basic tests for RNAalifold executable
- TESTS: Ignore 'frequency of MFE structure' in RNAcifold partition function checks

### Version 2.4.5 (Release date: 2018-04-17)

- Allow for arbitrary number of input files in RNAfold
- Allow for parallel processing of input data in RNAfold (UNIX only, no Windows support yet)
- Add SHAPE reactivity support through commandline options for RNAplfold
- Fix unstructured domain motif detection in MFE, centroid, and MEA structures computed by RNAfold
- Limit allowed set of commands in command file for RNAcifold to hard and soft constraints
- API: Add functions to compute equilibrium probability of particular secondary structures
- API: Add dynamic string stream data type and associated functions
- API: Add priority-queue like data structure with unordered fill capability and ordered output callback execution
- API: Add functions to detect unstructured domain motifs in MFE, centroid, and MEA structures
- API: Fix bug in sliding-window partition function computation with SHAPE reactivity and Deigan et al. conversion method
- API: Fix application of '<' and '>' constraint symbols in dot-bracket provided constraints (was broken since v2.4.2)
- API: Fix MEA structure computation in the presence of unstructured domains
- API: Stabilize order of probability entries in EPS dot-plot files
- Fix compiler warnings on wrong type of printf() in naview.c
- Define VRNA\_VERSION macro as string literal and add macros for major, minor, and patch numbers
- Stabilize parallel make of Mac OS X installer
- Add energy parameter set from Langdon et al. 2018
- Add autoconf checks for POSIX threads compiler/linker support
- SWIG: Fix 'next' is a perl keyword warnings for Perl5 wrapper
- SWIG: Catch errors and throw exceptions whenever scripting language provided callback functions are not applicable or fail
- SWIG: Add keyword arguments and autodoc feature for Python/Python3 wrappers

### Version 2.4.4 (Release date: 2018-03-06)

- Change verbose output for soft-constraints derived ligand binding motifs in RNAfold
- Allow for lowercase letters in ct2db input
- Fix bug in interior-like G-Quadruplex MFE computation for single sequences
- Fix autoconf switch to enable deprecation warnings
- Fix bug in eval\_int\_loop() that prevented propagation of energy evaluation for loops with nick in strands
- Fix several bugs for SHAPE reactivity related comparative partition function computations
- Fix annotation of PostScript output for soft-constraint derived ligand binding motifs in RNAfold
- Fix constraint indices for multibranch loops in unpaired probability computations of LPfold.c
- Fix dangling end contributions in comparative partition function for exterior loops
- API: Add simplified interface for [vrna\\_pf\\_dimer\(\)](#)
- API: Move concentraton dependent implementation for co-folding to separate compile unit
- API: Add new API functions for exterior loop evaluations
- API: Add simplified interfaces for energy evaluation with G-Quadruplexes and circular RNAs
- API: Add findpath functions that allow for specification of an upper bound for the saddle point
- Add configure-time linker check for Python3 interface
- Add automatic CPP suggestions for deprecated function substitutes
- Major restructuring and constraints feature additions in loop type dependent energy evaluation functions
- Major restructuring in MFE implementations
- Major restructuring in PF implementations
- Minor fixes in Boltzmann sampling implementation
- SWIG: Fix wrappers for findpath() implementation
- SWIG: Add tons of energy evaluation wrappers
- SWIG: Fix configure-time check of Perl5 interface build capabilities
- SWIG: Wrap functions from walk.c and neighbor.c
- DOC: Add some missing references to manpages of executable programs
- REFMAN: Heavy re-ordering of the RNAlib reference manual

### Version 2.4.3 (Release date: 2017-11-14)

- Fix handling of dangling end contribution at sequence boundaries for sliding window base pair probability computations
- Fix handling of base pair hard constraints in sliding-window implementations
- Fix sliding-window pair probability computations with multibranch-loop unpaired constraints
- Fix sliding-window non-specific base pair hard constraint implementation
- Fix probability computation for stochastic backtracking in RNAsubopt --stochBT\_en output
- Fix regression in comparative structure prediction for circular RNAs



- Fix LDFlags for scripting language interfaces in corresponding Makefiles
- Stabilize partition function scaling by always using sfact scaling factor from model details
- Add `-pf_scale` commandline parameter to RNAplfold
- Add constraint framework for single sequence circular RNA structure prediction
- Add RNAfold test suite to check for working implementation of constraints for circular RNAs
- Add a brief contribution guideline CONTRIBUTING.md
- Prevent RNAplfold from creating inf/-inf output when solution set is empty with particular hard constraints
- Include RNAforester v2.0.1

### Version 2.4.2 (Release date: 2017-10-13)

- Fix G-Quadruplex energy corrections in comparative structure energy evaluations
- Fix discrepancy in comparative exterior loop dangling end contribution of eval vs. MFE predictions
- Fix regression in RNAup unstructuredness and interaction energy computations
- Fix sequence length confusions when FASTA input contains carriage returns
- Fix build problems of RNALocmin with older compilers
- Fix sliding-window hard constraints where single nucleotides are prohibited from pairing
- Fix dot-bracket output string length in sliding-window MFE with G-Quadruplexes
- Fix unpaired probability computations for separate individual loop types in LPfold.c
- Fix bad memory access in RNAsubopt with dot-bracket constraint
- Add full WUSS support for `-SS_cons` constraint option in RNAalifold
- Add commandline option to RNALalifold that enables splitting of energy contributions into separate parts
- Add missing hard constraint cases to sliding-window partition function implementation
- Add CSV output option to RNAcofold
- Use the same model details for SCI computations in RNAalifold
- Abort computations in `vrna_eval_structure_v()` if structure has unexpected length
- Use original MSA in all output generated by RNAalifold and RNALalifold
- API: Add new functions to convert dot-bracket like structure annotations
- API: Add various new utility functions for alignment handling and comparative structure predictions
- API: Add function `vrna_strsplit()` to split string into tokens
- API: Do not convert sequences of input MSA to uppercase letters in `vrna_file_msa_read_record()`
- API: Rename `vrna_annotate_bp_covar()` and `vrna_annotate_pr_covar()`
- API: Add new noLP neighbor generation
- SWIG: Add wrapper for functions in `file_utils_msa.h`
- SWIG: Add wrappers for `vrna_pbacktrack()` and `vrna_pbacktrack5()`
- SWIG: Add `vrna_db_to_element_string()` to scripting language interface
- REFMAN: Fix formula to image conversion in HTML output

**Version 2.4.1 (Release date: 2017-08-23)**

- Fix memory leak in fold\_compound methods of SWIG interface
- Fix memory leaks in double \*\* returning functions of SWIG Perl5 interface
- Fix memory leak in vrna\_ep\_t to-string() function of SWIG interface
- Regression: Fix reverting pf\_scale to defaults after [vrna\\_exp\\_params\\_rescale\(\)](#)
- Regression: Fix homo-dimer partition function computation in RNAcofold
- Add unit tests for RNAcofold executable
- Add SHAPE reactivity support to RNAcofold
- Add SHAPE reactivity support to RNALalifold

**Version 2.4.0 (Release date: 2017-08-01)**

- Bump libsvm to version 3.22
- Print G-Quadruplex corrections in verbose mode of RNAeval
- Change behavior of RNAfold -outfile option to something more predictable
- Unify max\_bp\_span usage among sliding window prediction algorithms: RNAplfold, RNALfold, and RNALalifold now consider any base pair (i,j) with  $(j - i + 1) \leq \text{max\_bp\_span}$
- Add SHAPE reactivity data support to RNALfold
- Add commands-file support for RNALfold, RNAplfold (hard/soft constraints)
- Add RNALocmin - Calculate local minima from structures via gradient walks
- Add RNA Bioinformatics tutorial (PDF version)
- Add hard constraints to sliding-window MFE implementations (RNALfold, RNALalifold)
- Add hard constraints to sliding-window PF implementations (RNAplfold)
- Add soft constraints to sliding-window MFE implementation for single sequences (RNALfold)
- Add soft constraints to sliding-window PF implementations (RNAplfold)
- Add SWIG interfaces for sliding-window MFE/PF implementations
- Add proper SWIG interface for alignment and structure plotting functions
- Add proper SWIG interface for duplexfold, duplex\_subopt, and its comparative variants
- Add SWIG wrapper for [vrna\\_exp\\_params\\_rescale\(\)](#)
- Add explicit destructor for SWIG generated vrna\_md\_t objects
- Add SWIG perl5 typemap for simple nested STL vectors
- Add dummy field in [vrna\\_structured\\_domains\\_s](#)
- Add note about SSE optimized code in reference manual
- Add SWIG interface for findpath implementation
- Add prepare() functions for ptypes-arrays and vrna\_(exp\_)param\_t
- Add warnings for ignored commands in function [vrna\\_commands\\_apply\(\)](#)
- Add callback featured functions for sliding window MFE and PF implementations

- Change default behavior of adding soft constraints to a `vrna_fold_compound_t` (store only)
- Several fixes with respect to G-Quadruplex prediction in sliding-window MFE recursions (single sequence and comparative implementation)
- Replace comparative sliding-window MFE recursions (All hits are reported to callback and can be filtered in a post-processing step)
- API: Remove `E_mb_loop_stack()` and introduce new function `vrna_E_mb_loop_stack()` as a replacement
- API: change data type of all constraint bit-flags from `char` to `unsigned char`
- API: change data type of `a2s` array in comparative structure prediction from `unsigned short` to `unsigned int`
- API: Change function parameter order in `vrna_probs_window()` to follow the style of other callback-aware functions in RNALib
- Move sliding-window MFE implementations to new file `mfe_window.c`
- Fix building PDF Reference manual with non-standard executable paths
- Fix redefinition of macro `ON_SAME_STRAND()` in `subopt.c`
- Fix dangling end issues in sliding-window MFE implementations
- Fix regression for `-canonicalBPonly` switch in `RNAfold/RNAcofold/RNAsubopt`
- Fix building sliding-window MFE implementation without SVM support
- Fix parsing of STOCKHOLM 1.0 MSA files that contain MSA spanning multiple blocks
- Fix Alidot link in `RNAalifold` manpage
- Fix wrong pre-processor flags when enabling single-precision PF computations
- Fix unit testing perl5 interface by including `builddir/tests` in `PERL5LIB` path
- Fix buffer overflow in hairpin loop sequence motif extraction for circular RNAs
- Fix out-of-bounds memory access in `neighbor.c`
- Restore capability to compile stand-alone `findpath` utility
- Restore capability to use non-standard alphabets for structure prediction
- Restore old-API random number functions in SWIG interface
- Allow additional control characters in MAF MSA input that do not end a block
- Improve reference manual
- Make functions in `pair_mat.h` static inline
- Prevent users from adding out-of-range base pair soft constraints
- Inline print functions in `color_output.inc`
- Start documenting callback features in reference manual
- Re-write large portions of sliding-window PF implementation
- Introduce soft-constraint state flag
- Clean-up SWIG unit test framework
- Remove obsolete scripts `ct2b.pl` and `colorrna.pl` from `src/Utils` directory
- Remove old `RNAfold` tutorial

## Version 2.3.x

### Version 2.3.5 (Release date: 2017-04-14)

- Fix duplication of output filename prefix in RNAfold
- Add V3.0 API for sliding window partition function (a.k.a. RNAPLfold)
- Add G-Quadruplex prediction to RNALalifold
- Add SWIG wrappers for callback-based sliding window comparative MFE prediction
- Add SSE4.1 multiloop decomposition for single sequence MFE prediction
- Enable RNAfold unit tests to run in parallel
- Enable users to turn-off base pair probability computations in RNACofold with -a option
- Split move set in neighbor.c

### Version 2.3.4 (Release date: 2017-03-10)

- Fix G-Quadruplex probability computation for single sequences
- Fix double-free when using SHAPE reactivity data in RNAalifold
- Fix out-of-bounds access in strand\_number array
- Fix weighting of SHAPE reactivity data in consensus structure prediction when fewer data than sequences are present
- Fix z-score output in RNALfold
- Substitute field name 'A0'/'B0' in data structure `vrna_dimer_conc_s` by 'Ac\_start'/'Bc\_start' to avoid clashes with `termios.h` (Mac OSX Python wrapper bug)
- Minimize usage of 'unsafe' `sprintf()` calls
- Enhance auto-id feature in executable programs
- Always sanitize output file names to avoid problems due to strange FASTA headers
- Lift restrictions of FASTA header length in RNAfold, RNACofold, and RNAeval
- Add ViennaRNA/config.h with pre-processor definitions of configure time choices
- Add test-suite for RNAfold
- Add functions to produce colored EPS structure alignments
- Add function to write Stockholm 1.0 formatted alignments
- Add function to sanitize file names
- Add callback based implementation for sliding-window MFE prediction (single sequences, comparative structure prediction)
- Add fast API 3.0 implementations to generate structural neighbors and perform steepest descent / random walks (Thanks to Gregor!)
- Add parameter option to RNALalifold for colored EPS structure alignment and structure plot output
- Add parameter option to RNALalifold to write hits into Stockholm file
- Add parameter option to RNAalifold to write Stockholm 1.0 formatted output

- Add parameter option to RNAalifold to suppress stderr spam
- Add auto-id feature to RNAplot, RNALfold, RNAsubopt, RNApfold, RNAheat
- Add SHAPE reactivity derived pseudo-energies as separate output in RNAalifold
- Add colored output to RNA2Dfold, RNALalifold, RNALfold, RNAduplex, RNAheat, RNAinverse, RNApfold, and RNAsubopt
- Add command line parameters to RNAsubopt to allow for specification of input/output files

### Version 2.3.3 (Release date: 2017-01-24)

- Fix multiloop contributions for comparative partition function
- Fix building python2 extension module for OSX

### Version 2.3.2 (Release date: 2017-01-18)

- Fix pair probability plist creation with G-Quadruplexes
- Allow for specification of python2/3-config at configure time
- Fix init of vrna\_md\_t data structure after call to [set\\_model\\_details\(\)](#)
- Fix bug in consensus partition function with hard constraints that force nucleotides to be paired
- Fix compilation of functions that use ellipsis/va\_list
- Enable generic hard constraints by default
- Fix init of partition function DP matrices for unusually short RNAs
- Fix behavior of RNApfold for unusually short RNAs
- Report SCI of 0 in RNAalifold when sum of single sequence MFEs is 0
- Avoid multiple includes of [pair\\_mat.h](#)
- Add configure flag to build entirely static executables

### Version 2.3.1 (Release date: 2016-11-15)

- Add description for how to use unstructured domains through command files to reference manual and RNA↔Afold manpage
- Fix compilation issue for Windows platforms with MingW
- Add missing newline in non-TTY-color output of [vrna\\_message\\_info\(\)](#)
- Fix regression in [vrna\\_md\\_update\(\)](#) that resulted in incomplete init of reverse-basepair type array
- Extend coverage of generic hard constraints for partition function computations
- Fix scaling of secondary structure in EPS plot such that it always fits into bounding box
- Several fixes and improvements for SWIG generated scripting language interface(s)

## Version 2.3.0 (Release date: 2016-11-01)

- Add grammar extension with structured and unstructured domains
- Add default implementation for unstructured domains to allow for ligand/protein binding to unpaired structure segments (MFE and PF for single sequences)
- Introduced command files that subsume constraint definition files (currently used in RNAfold and RNAcofold)
- Replace explicit calls to `asprintf()` with portable equivalent functions in the library
- Fix configure script to deal with situations where Perl module can't be build
- Fix bug in doc/Makefile.am that prevented HTML installation due to long argument list
- Added utility functions that deal with conversion between different units
- Bugfix in SWIG wrapped generic soft constraint feature
- Add `subopt()` and `subopt_zuker()` methods to SWIG wrapped `fold_compound` objects
- Bugfix multiloop decomposition in MFE for circular RNAs
- Add separate function to compute pscore for alignments
- Renamed `VRNA_VC_TYPE_*` macros to `VRNA_FC_TYPE_*`
- Bugfix regression that prevented programs to fail on too long input sequences
- Extend EPS dot-plot in RNAfold to include motif/binding probabilities from unstructured domains
- Add variadic functions for error/warning/info message
- Add ID manipulation feature to RNAeval
- Extend API for soft constraint feature for more fine-grained control
- Add section on SWIG wrapped functions in reference manual
- Fix bug in interior loop computations when hard constraints result in non-canonical base pairs

## Version 2.2.x

### Version 2.2.10 (Release date: 2016-09-06)

- Do not 'forget' subopt results when output is not written to file handle and sorting is switched off
- Fix bad memory access in `vrna_subopt()` with sorted output
- Add SWIG wrappers for `vrna_subopt_cb()`
- Correctly show if C11 features are activated in configure status
- Fix autoconf checks to allow for cross compilation again

---

**Version 2.2.9 (Release date: 2016-09-01)**

- Fix bug in partition function scaling for backward compatibility of `ali_pf_fold()`
- Stabilize v3.0 API when building RNAlib and third party program linking against it with compilers that use different C/C++ standards
- Add details on how to link against RNAlib to the reference manual
- Fix RNAlib2.pc
- Fix bug for temperature setting in RNAplfold
- Use `-fflat-lto-objects` for static RNAlib library to allow linking without LTO
- Fix interpretation of 'P' hard constraint for single nucleotides in constraint definition files
- Add 'A' command for hard constraints
- Fix several hard constraint corner-cases in MFE and partition function computation when nucleotides must not be unpaired
- Fix order of hard constraints when read from input file
- Allow for non-canonical base pairs in MFE and partition function computations if hard constraints demand it
- Fix behavior of `--without-swig` configure script option
- Fix bug in hard constraints usage of exterior loop MFE prediction with odd dangles
- Add parsers for Clustal, Stockholm, FASTA, and MAF formatted alignment files
- Enable RNAalifold to use Clustal, Stockholm, FASTA, or MAF alignments as input
- Lift restriction of sequence number in alignments for RNAalifold
- Enable ANSI colors for TTY output in RNAfold, RNAcifold, RNAalifold, RNAsubopt, and warnings/errors issued by RNAlib
- Add various new commandline options to manipulate sequence/alignment IDs in RNAfold, RNAcifold and RNAalifold

**Version 2.2.8 (Release date: 2016-08-01)**

- Fix bad memory access in RNAalifold
- Fix regression in RNAalifold to restore covariance contribution ratio determination for circular RNA alignments
- Changed output of RNAsubopt in energy-band enumeration mode to print MFE and energy range in kcal/mol instead of 10cal/mol
- Include latest Kinfold sources that make use of v3.0 API, therefore speeding up runtime substantially
- Re-activate warnings in RNAeval when non-canonical base pairs are encountered
- Fix syntactic incompatibilities that potentially prevented compilation with compilers other than gcc
- dd function to compare nucleotides encoded in IUPAC format
- Fix regression in energy evaluation for circular RNA sequences
- Fix regression in suboptimal structure enumeration for circular RNAs
- Allow for P i-j k-l commands in constraint definition files
- Make free energy evaluation functions polymorphic

- Add free energy evaluation functions that allow for specifying verbosity level
- Secure functions in alphabet.c against NULL pointer arguments
- Fix incompatibility with swig  $\geq 3.0.9$
- Fix memory leak in swig-generated scripting language interface(s) for user-provided target language soft-constraint callbacks
- Expose additional functions to swig-generated scripting language interface(s)
- Build Python3 interface by default
- Start of more comprehensive scripting language interface documentation
- Fix linking of python2/python3 interfaces when libpython is in non-standard directory
- Restructured viennarna.spec for RPM based distributions
- Several syntactic changes in the implementation to minimize compiler warnings
- Fix `--with-*/--without-*` and `--enable-*/--disable-*` configure script behavior

### Version 2.2.7 (Release date: 2016-06-30)

- Fix partition function scaling for long sequences in RNAfold, RNAalifold, and RNAup
- Fix backtracking issue in RNAcofold when `--noLP` option is activated
- Fix hard constraints issue for circular RNAs in generating suboptimal structures
- Rebuild reference manual only when actually required

### Version 2.2.6 (Release date: 2016-06-19)

- Plugged memory leak in RNAcofold
- Fixed partition function rescaling bug in RNAup
- Fixed bug in RNALfold with window sizes larger than sequence length
- Re-added SCI parameter for RNAalifold
- Fixed backtracking issue for large G-quadruplexes in RNAalifold
- Fixed missing FASTA id in RNAeval output
- Added option to RNAalifold that allows to specify prefix for output files
- Several fixes and additional functions/methods in scripting language interface(s)
- Added version information for scripting language interface(s)
- Some changes to allow for compilation with newer compilers, such as gcc 6.1



### Version 2.2.5 (Release date: 2016-04-09)

- Fixed regression in RNAcofold that prohibited output of concentration computations
- Fixed behavior of RNAfold and RNAcofold when hard constraints create empty solution set (programs now abort with error message)
- Added optional Python 3 interface
- Added RNA::Params Perl 5 sub-package
- Update RNA::Design Perl 5 sub-package
- Simplified usage of v3.0 API with default options
- Wrap more functions of v3.0 API in SWIG generated scripting language interfaces
- Plugged some memory leaks in SWIG generated scripting language interfaces
- Changed parameters of recursion status callback in `vrna_fold_compound_t`
- Enable definition and binding of callback functions from within SWIG target language
- Added optional subpackage Kinwalker
- Added several configure options to ease building and packaging under MacOS X
- Added new utility script RNAdesign.pl

### Version 2.2.4 (Release date: 2016-02-19)

- Fixed bug in RNAsubopt that occasionally produced cofolded structures twice
- Removed debugging output in preparations of consensus structure prediction datastructures

### Version 2.2.3 (Release date: 2016-02-13)

- Added postscript annotations for found ligand motifs in RNAfold
- Added more documentation for the constraints features in RNAfold and RNAalifold
- Restore backward compatibility of `get_alipf_arrays()`

### Version 2.2.2 (Release date: 2016-02-08)

- Fix regression bug that occasionally prevented backtracking with RNAcofold `--noLP`

### Version 2.2.1 (Release date: 2016-02-06)

- Fix regression bug that made RNAcofold `-a` unusable
- Fix regression bug that prohibited RNAfold to compute the MEA structure when G-Quadruplex support was switched on
- Fix bug in Kinfold to enable loading energy parameters from file
- Fix potential use of uninitialized value in RNApdist
- Add manpage for ct2db
- Fix MEA computation when G-Quadruplex support is activated
- Allow for vendor installation of the perl interface using `INSTALLDIRS=vendor` at configure time
- Install architecture dependent and independent files of the perl and python interface to their correct file system locations

## Version 2.2.0 (Release date: 2016-01-25)

- RNAforester is now of version 2.0
- New program RNApvmin to compute pseudo-energy perturbation vector that minimizes discrepancy between observed and predicted pairing probabilities
- SHAPE reactivity support for RNAfold, RNAsubopt, and RNAalifold
- Ligand binding to hairpin- and interior-loop motif support in RNAfold
- New cmdline option to limit maximum base pair span for RNAfold, RNAsubopt, RNAcifold, and RNAalifold
- Bugfix in RNAheat to remove numerical instabilities
- Bugfix in RNAPlex to allow for computation of interactions without length limitation
- Bugfix in RNAplot for simple layouts and hairpins of size 0
- (generic) hard- and soft-constraints for MFE, partition function, base pair probabilities, stochastic backtracking, and suboptimal secondary structures of single sequences, sequence alignments, and sequence dimers
- libsvm version as required for z-scoring in RNALfold is now 3.20
- Stochastic backtracking for single sequences is faster due to usage of Boustrophedon scheme
- First polymorphic functions `vrna_mfe()`, `vrna_pf()`, and `vrna_pbacktrack()`.
- The FLT\_OR\_DBL macro is now a typedef
- New functions to convert between different secondary structure representations, such as helix lists, and RNAshapes abstractions
- First object-oriented interface for new API functions in the scripting language interfaces
- new ViennaRNA-perl submodule that augments the Perl interface to RNAlib
- Ligand binding to hairpin- and interior-loop motif support in C-library and scripting language interfaces.
- Libraries are generated using libtool
- Linking of libraries and executables defaults to use Link Time Optimization (LTO)
- Large changes in directory structure of the source code files

## Version 2.1.x

### Version 2.1.9

- Fixed integer underflow bug in RNALfold
- Added Sequence Conservation index (SCI) option to RNAalifold
- Fixed bug in energy evaluation of dangling ends / terminal mismatches of exterior loops and multibranch loops
- Fixed bug in alifold partition function for circular RNAs
- Fixed bug in alifold that scrambled backtracing with activated G-Quadruplex support
- Fixed bug in alifold backtracking for larger G-Quadruplexes

### Version 2.1.8

- Repaired incorporation of RNAinverse user provided alphabet
- Fix missing FASTA ID in RNAeval output
- prevent race condition in parallel calls of [Lfold\(\)](#)
- Fixed memory bug in [Lfold\(\)](#) that occurred using long sequences and activated G-Quad support
- Added latest version of switch.pl

### Version 2.1.7

- Fixed bug in RNALfold -z
- Python and Perl interface are compiling again under MacOSX
- Fixed handling of C arrays in Python interface
- Added latest version of switch.pl
- Make relplot.pl work with RNAcifold output

### Version 2.1.6

- New commandline switches allow for elimination of non-canonical base pairs from constraint structures in RNAfold, RNAalifold and RNAsubopt
- updated moveset functions
- final fix for discrepancy of tri-loop evaluation between partition function and mfe
- pkg-config file now includes the OpenMP linker flag if necessary
- New program ct2db allows for conversion of .ct files into dot-bracket notation (incl. pseudo-knot removal)

### Version 2.1.5

- Fix for discrepancy between special hairpin loop evaluation in partition functions and MFE

### Version 2.1.4

- Fix of G-quadruplex support in [subopt\(\)](#)
- Fix for discrepancy between special hairpin loop evaluation in partition functions and MFE

### Version 2.1.3

- RNAfold: Bugfix for ignoring user specified energy parameter files
- RNAcifold: Bugfix for crashing upon constrained folding without specifying a constraint structure
- RNAsubopt: Added G-quadruplex support
- RNAalifold: Added parameter option to specify base pair probability threshold in dotplot
- Fix of several G-quadruplex related bugs
- Added G-quadruplex support in [subopt\(\)](#)

### Version 2.1.2

- RNAfold: Bugfix for randomly missing probabilities in dot-plot during batch job execution
- RNAeval: Bugfix for misinterpreted G-quadruplex containing sequences where the quadruplex starts at nucleotide 1
- RNAsubopt: Slight changes to the output of stochastic backtracking and zucker subopt
- Fix of some memory leaks
- Bugfixes in [zukersubopt\(\)](#), [assign\\_plist\\_from\\_pr\(\)](#)
- New threadsafe variants of `putoutpU_prob*()` for `LPfold()`
- Provision of python2 interface support.

### Version 2.1.1

- Bugfix to restore backward compatibility with ViennaRNA Package 1.8.x API (this bug also affected proper usage of the perl interface)

### Version 2.1.0

- G-Quadruplex support in RNAfold, RNAcifold, RNALfold, RNAalifold, RNAeval and RNAplot
- LPfold got a new option to output its computations in split-mode
- several G-Quadruplex related functions were introduced with this release
- several functions for moves in an RNA landscape were introduced
- new function in `alipfold.c` now enables access to the partition function matrices of [alipf\\_fold\(\)](#)
- different numeric approach was implemented for concentration dependend co-folding to avoid instabilities which occurred under certain circumstances

## Version 2.0.x

### Version 2.0.7

- Bugfix for RNAplfold where segfault happened upon usage of `-O` option
- Corrected misbehavior of RNAeval and RNAplot in tty mode

### Version 2.0.6

- Bugfix for bad type casting with gcc under MacOSX (resulted in accidental "sequence too long" errors)
- Bugfix for disappearing tri-/hexaloop contributions when read in from certain parameter files
- Bugfix for RNALfold that segfaulted on short strange sequences like `AT+` repeats
- Change of RNA2Dfold output format for stochastic backtracking

---

## Version 2.0.5

- Restored z-score computation capabilities in RNALfold

## Version 2.0.4

- Bugfix for RNAcofold partition function
- Perl wrapper compatibility to changed RNAlib has been restored
- Backward compatibility for partition function calls has been restored

## Version 2.0.3

- Bugfix for RNAalifold partition function and base pair probabilities in v2.0.3b
- Added Boltzmann factor scaling in RNAsubopt, RNAalifold, RNAplfold and RNAcofold
- Bugfix for alifold() in v2.0.3b
- Restored threadsafety of folding matrix access in LPfold.c, alifold.c, part\_func.c, part\_func\_co.c and part\_func\_up.c
- Added several new functions regarding threadsafe function calls in terms of concurrently changing the model details
- Added pkg-config file in the distribution to allow easy checks for certain RNAlib2 versions, compiler flags and linker flags.

## Version 2.0.2

- added support for Boltzmann factor scaling in RNAfold
- fixed fastaheader to filename bug
- plugged some memory leaks

## Version 2.0.1

- First official release of version 2.0
- included latest bugfixes

## History

2011-03-10 Ronny Lorenz [ronny@tbi.univie.ac.at](mailto:ronny@tbi.univie.ac.at)

- new naming scheme for all shipped energy parameter files
- fixed bugs that appear while compiling with gcc under MacOS X
- fixed bug in RNAup –interaction-first where the longer of the first two sequences was taken as target
- added full FASTA input support to RNAfold, RNAcofold, RNAheat, RNAplfold, RNALfoldz, RNAsubopt and RNALfold

2010-11-24 Ronny Lorenz [ronny@tbi.univie.ac.at](mailto:ronny@tbi.univie.ac.at)

- first full pre-release of version 2.0

2009-11-03 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- Fix memory corruption in [PS\\_color\\_aln\(\)](#)

2009-09-09 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- Fix bug in RNAplfold when -u and -L parameters are equal
- Fix double call to [free\\_arrays\(\)](#) in RNAfold.c
- Improve drawing of cofolded structures

2009-05-14 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- Fix occasional segfault in RNAalifold's [print\\_alnout\(\)](#)

2009-02-24 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- Add -MEA options to RNAfold and RNAalifold
- change [energy\\_of\\_alnstruct](#) to return float not void

2009-02-24 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- RNAfold will draw structures unless -noPS is used (no more "structure too long" messages)
- Restore the "alifold.out" output from RNAalifold -p
- RNAalifold -circ did not work due to wrong return type
- Accessibility calculation with RNAplfold would give wrong results for  $u \leq 30$

2008-12-03 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- Add zucker style suboptimals to RNAsubopt (-z)
- `get_line()` should be much faster when reading huge sequences (e.g. whole chromosomes for RNALfold)

2008-08-12 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- Add Ribosum matrices for covariance scoring in RNAalifold

2008-06-27 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- Change RNAalifold to used berni's new energy evaluation w/o gaps
- Add stochastic backtracking in RNAalifold

2008-07-04 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- modify output of RNAup (again). Program reading RNAup output will have to updated!

2008-07-02 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- RNAplfold now computes accessibilities for all regions up to a max length simultaneously. Slightly slower when only 1 value is needed, but much faster if all of them are wanted. This entails a new output format. Programs reading accessibility output from RNAplfold need to be updated!

2008-03-31 Stephan Bernhart [berni@tbi.univie.ac.at](mailto:berni@tbi.univie.ac.at)

- add cofolding to RNAsubopt

2008-01-08 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- ensure circfold works even for open chain

2007-12-13 Ulli Mueckstein [ulli@tbi.univie.ac.at](mailto:ulli@tbi.univie.ac.at)

- upate RNAup related files RNAup can now include the intramolecular structure of both molecules and handles constraints.

2007-12-05 Ronny Lorenz [ronny@tbi.univie.ac.at](mailto:ronny@tbi.univie.ac.at)

- add circfold variants in `part_func.c` `alifold.c` `subopt.c`

2007-09-19 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- compute the centroid structure of the ensemble in RNAfold -p

- fix a missing factor 2 in `mean_bp_dist()`. CAUTION ensemble diversities returned by `RNAfold -p` are now twice as large as in earlier versions.

2007-09-04 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- fix a bug in `Lfold()` where base number `n-max-4` would never pair

2007-08-26 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- add `RNAaliduplex` the alignment version of `RNA duplex`
- introduce a minimal distance between hits produced by `duplex_subopt()`

2007-07-03 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- add a `loop_energy()` function to compute energy of a single loop

2007-06-23 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- add `aliLfold()` and `RNALalifold`, alignment variant of `Lfold()`

2007-04-30 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- add `RNAup` to distribution

2007-04-15 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- fix segfault in `colorps` output (thanks to Andres Varon)

2007-03-03 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- avoid unnormalized doubles in `scale[]`, big speedup for `pf_fold()` on very long sequences

2007-02-03 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- `RNAalifold` can now produce colored structure plots and alignment plots

2007-02-01 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- Fix segfault in `RNAplfold` because of missing prototype

2006-12-01 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- `RNA duplex` would segfault when no structure base pairs are possible



---

2006-08-22 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- add computation stacking probabilities using RNAfold -p2
- add -noPS option for RNAfold to suppress drawing structures

2006-08-09 Stephan Bernhart [berni@tbi.univie.ac.at](mailto:berni@tbi.univie.ac.at)

- RNAfold can now compute probabilities of unpaired regions (scanning version of RNAup)

2006-06-14 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- compile library with -fpic (if available) for use as shared library in the Perl module.
- fix another bug when calling `Lfold()` repeatedly
- fix switch cmdline parsing in RNAalifold (-mis implied -4)
- fix bug in `cofold()` with dangles=0

2006-05-08 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- fix segfault in `Lfold()` when calling repeatedly
- fix structure parsing in RNAstruct.c (thanks to Michael Pheasant for reporting both bugs)
- add duplexfold() and `alifold()` to Perl module
- distinguish window size and max pair span in LPfold

2006-04-05 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- fix performance bug in `co_pf_fold()`
- use relative error for termination of Newton iteration

2006-03-02 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- add circular folding in `alifold()`

2006-01-18 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- cleanup berni partition cofold code, including several bug fixes

2006-01-16 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- update RNAplfold to working version
- add `PS_dot_plot_turn()` in `PS_dot.c`

2005-11-07 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- add new utilities colorna and coloraln

2005-10-11 Christoph Flamm [xtof@tbi.univie.ac.at](mailto:xtof@tbi.univie.ac.at)

- adapt [PS\\_rna\\_plot\(\)](#) for drawing co-folded structures

2005-07-24 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- fix a few memory problems in structure comparison routines

2005-04-30 Ivo Hofacker [ivo@blini.tbi.univie.ac.at](mailto:ivo@blini.tbi.univie.ac.at)

- add folding of circular RNAs

2005-03-11 Ivo Hofacker [ivo@blini.tbi.univie.ac.at](mailto:ivo@blini.tbi.univie.ac.at)

- add -mis option to RNAalifold to give "most informative sequence" as consensus

2005-02-10 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- move [alifold\(\)](#) into the library

2004-12-22 Stephan Bernhart [berni@tbi.univie.ac.at](mailto:berni@tbi.univie.ac.at)

- add partition function version of RNAcofold

2004-12-23 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- add RNApaln for fast structural alignments (RNApdist improvement)

2004-08-12 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- fix constrained folding in stochastic backtracking

2004-07-21 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- add RNAduplex, to compute hybrid structures without intra-molecular pairs

2004-02-09 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- fix bug in fold that caused segfaults when using Intel compiler

- add computation of ensemble diversity to RNAfold

2003-09-10 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- add annotation options to RNAplot

2003-08-04 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- stochastic backtracking finally works. Try e.g. RNAsubopt -p 10

2003-07-18 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- add relplot.pl and rotate\_ss.pl utilities for reliability annotation and rotation of rna structure plots

2003-01-29 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- add RNALfold program to compute locally optimal structures with maximum pair span.
- add RNAcofold for computing hybrid structure

2002-11-07 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- change [Make\\_bp\\_profile\(\)](#) and [profile\\_edit\\_distance\(\)](#) to use simple (float \*) arrays; makes Perl access much easier. RNApdist -B now works again

2002-10-28 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- Improved Perl module with pod documentation; allow to write things like (\$structure, \$energy) = RNA $\leftarrow$ ::fold(\$seq); Compatibility warning: the ptrvalue() and related functions are gone, see the pod documentation for alternatives.

2002-10-29 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- added svg structure plots in PS\_dot.c and RNAplot

2002-08-15 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- Improve reading of clustal files (alifold)
- add a sample alifold.cgi script

2001-09-18 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- moved suboptimal folding into the library, thus it's now accessible from the Perl module

2001-08-31 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- added co-folding support in [energy\\_of\\_struct\(\)](#), and thus RNAeval

2001-04-30 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- switch from handcrafted makefiles to automake and autoconf

2001-04-05 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- added PS\_rna\_plot\_a to produce structure plots with annotation

2001-03-03 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- add alifold; predict consensus structures from alignment

2000-09-28 Ivo Hofacker [ivo@tbi.univie.ac.at](mailto:ivo@tbi.univie.ac.at)

- add -d3 option to RNAfold for co-axial stacking



# Chapter 11

## Deprecated List

### Global [vrna\\_exp\\_param\\_s::id](#)

This attribute will be removed in version 3

### File [svm\\_utils.h](#)

Use [ViennaRNA/Utils/svm.h](#) instead

### File [units.h](#)

Use [ViennaRNA/Utils/units.h](#) instead

### File [utils.h](#)

Use [ViennaRNA/Utils/basic.h](#) instead

### File [walk.h](#)

Use [ViennaRNA/Landscape/walk.h](#) instead

### Global [paramT](#)

Use [vrna\\_param\\_t](#) instead!

### Global [pf\\_paramT](#)

Use [vrna\\_exp\\_param\\_t](#) instead!

### Global [get\\_scaled\\_pf\\_parameters](#) (void)

Use [vrna\\_exp\\_params\(\)](#) instead!

### Global [get\\_boltzmann\\_factors](#) (double temperature, double betaScale, vrna\_md\_t md, double pf\_scale)

Use [vrna\\_exp\\_params\(\)](#) instead!

### Global [get\\_boltzmann\\_factor\\_copy](#) (vrna\_exp\_param\_t \*parameters)

Use [vrna\\_exp\\_params\\_copy\(\)](#) instead!

### Global [get\\_scaled\\_alipf\\_parameters](#) (unsigned int n\_seq)

Use [vrna\\_exp\\_params\\_comparative\(\)](#) instead!

### Global [get\\_boltzmann\\_factors\\_alipf](#) (unsigned int n\_seq, double temperature, double betaScale, vrna\_md\_t md, double pf\_scale)

Use [vrna\\_exp\\_params\\_comparative\(\)](#) instead!

### Global [scale\\_parameters](#) (void)

Use [vrna\\_params\(\)](#) instead!

### Global [get\\_scaled\\_parameters](#) (double temperature, vrna\_md\_t md)

Use [vrna\\_params\(\)](#) instead!

### Global [zukersubopt\\_par](#) (const char \*string, vrna\_param\_t \*parameters)

use [vrna\\_zukersubopt\(\)](#) instead

Global **read\_parameter\_file** (const char fname[])

Use [vrna\\_params\\_load\(\)](#) instead!

Global **write\_parameter\_file** (const char fname[])

Use [vrna\\_params\\_save\(\)](#) 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.

Global **exp\_E\_ExtLoop** (int type, int si1, int sj1, vrna\_exp\_param\_t \*P)

Use [vrna\\_exp\\_E\\_ext\\_stem\(\)](#) instead!

Global **VRNA\_CONSTRAINT\_FILE**

Use 0 instead!

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\_CONSTRAINT\_NO\_HEADER**

This mode is not supported 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 **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.

Global **VRNA\_CONSTRAINT\_MULTILINE**

see [vrna\\_extract\\_record\\_rest\\_structure\(\)](#)

Global **vrna\_extract\_record\_rest\_constraint** (char \*\*cstruc, const char \*\*lines, unsigned int option)

Use [vrna\\_extract\\_record\\_rest\\_structure\(\)](#) instead!

File **plot\_structure.h**

Use [ViennaRNA/plotting/structures.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 **expHairpinEnergy** (int u, int type, short si1, short sj1, const char \*string)

Use [exp\\_E\\_Hairpin\(\)](#) from [loop\\_energies.h](#) 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 **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 **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 `init_co_pf_fold` (int length)**

{ This function is deprecated and will be removed soon! }

**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 `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 `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!

**File `plot_aln.h`**

Use `ViennaRNA/plotting/alignments.h` instead

**File `plot_layouts.h`**

Use `ViennaRNA/plotting/layouts.h` instead

**File `plot_utils.h`**

Use `ViennaRNA/plotting/utils.h` instead

**Global `Make_bp_profile` (int length)**

This function is deprecated and will be removed soon! See `Make_bp_profile_bppm()` for a replacement

**File `PS_dot.h`**

Use `ViennaRNA/plotting/probabilities.h` instead

**File `read_epars.h`**

Use `ViennaRNA/params/io.h` instead

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

See `vrna_db_to_tree_string()` and `VRNA_STRUCTURE_TREE_HIT` for a replacement

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

See `vrna_db_to_tree_string()` and `VRNA_STRUCTURE_TREE_SHAPIRO_SHORT` for a replacement

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

See `vrna_db_to_tree_string()` and `VRNA_STRUCTURE_TREE_SHAPIRO_WEIGHT` for a replacement

**File `stream_output.h`**

Use `ViennaRNA/datastructures/stream_output.h` instead

**File `string_utils.h`**

Use `ViennaRNA/utils/strings.h` instead

**File `structure_utils.h`**

Use `ViennaRNA/utils/structures.h` instead

**Global `SOLUTION`**

Use `vrna_subopt_solution_t` instead!

**Global `zukersubopt` (const char \*string)**

use `vrna_zukersubopt()` instead

**Global `bp_distance` (const char \*str1, const char \*str2)**

Use `vrna_bp_distance` instead

**Global `time_stamp` (void)**

Use `vrna_time_stamp()` instead!

Global **str\_uppercase** (char \*sequence)

Use [vrna\\_seq\\_toupper\(\)](#) instead!

Global **str\_DNA2RNA** (char \*sequence)

Use [vrna\\_seq\\_toRNA\(\)](#) instead!

Global **random\_string** (int l, const char symbols[])

Use [vrna\\_random\\_string\(\)](#) 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 **assign\_plist\_from\_db** (vrna\_ep\_t \*\*pl, const char \*struc, float pr)

Use [vrna\\_plist\(\)](#) instead

Global **pack\_structure** (const char \*struc)

Use [vrna\\_db\\_pack\(\)](#) as a replacement

Global **unpack\_structure** (const char \*packed)

Use [vrna\\_db\\_unpack\(\)](#) as a replacement

Global **make\_pair\_table** (const char \*structure)

Use [vrna\\_ptable\(\)](#) instead

Global **copy\_pair\_table** (const short \*pt)

Use [vrna\\_ptable\\_copy\(\)](#) instead

Global **alimake\_pair\_table** (const char \*structure)

Use [vrna\\_pt\\_ali\\_get\(\)](#) instead!

Global **make\_pair\_table\_snoop** (const char \*structure)

Use [vrna\\_pt\\_snoop\\_get\(\)](#) instead!

Global **filecopy** (FILE \*from, FILE \*to)

Use [vrna\\_file\\_copy\(\)](#) instead!

Global **make\_referenceBP\_array** (short \*reference\_pt, unsigned int turn)

Use [vrna\\_refBPcnt\\_matrix\(\)](#) instead

Global **compute\_BPdifferences** (short \*pt1, short \*pt2, unsigned int turn)

Use [vrna\\_refBPdist\\_matrix\(\)](#) 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 **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 **bppm\_to\_structure** (char \*structure, FLT\_OR\_DBL \*pr, unsigned int length)

Use [vrna\\_db\\_from\\_probs\(\)](#) instead!

Global **bppm\_symbol** (const float \*x)

Use [vrna\\_bpp\\_symbol\(\)](#) instead!

Global **PAIR**

Use [vrna\\_basepair\\_t](#) instead!

Global **plist**

Use [vrna\\_ep\\_t](#) or [vrna\\_elem\\_prob\\_s](#) instead!



**Global `cpair`**

Use `vrna_cpair_t` instead!

**Global `sect`**

Use `vrna_sect_t` instead!

**Global `bondT`**

Use `vrna_bp_stack_t` 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 `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 `find_saddle` (`const char *seq`, `const char *s1`, `const char *s2`, `int width`)**

Use `vrna_path_findpath_saddle()` instead!

**Global `free_path` (`vrna_path_t *path`)**

Use `vrna_path_free()` instead!

**Global `get_path` (`const char *seq`, `const char *s1`, `const char *s2`, `int width`)**

Use `vrna_path_findpath()` instead!

**Global `path_t`**

Use `vrna_path_t` instead!

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

Use `vrna_file_PS_aln()` instead!

**Global `aliPS_color_aln` (`const char *structure`, `const char *filename`, `const char *seqs[]`, `const char *names[]`)**

Use `vrna_file_PS_aln()` instead!

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

Consider switching to `vrna_plot_coords_simple_pt()` instead!

**Global `simple_circplot_coordinates` (`short *pair_table`, `float *x`, `float *y`)**

Consider switching to `vrna_plot_coords_circular_pt()` instead!

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

Consider using `vrna_plot_coords_naview_pt()` 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 `mean_bp_dist` (`int length`)**

This function is not threadsafe and should not be used anymore. Use `mean_bp_distance()` instead!

**Global `pair_info`**

Use `vrna_pinfo_t` instead!

**Global `get_mpi` (`char *Aseq[]`, `int n_seq`, `int length`, `int *mini`)**

Use `vrna_aln_mpi()` as a replacement

**Global `get_line` (`FILE *fp`)**

Use `vrna_read_line()` as a substitute!

**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!

Global **warn\_user** (const char message[])

Use [vrna\\_message\\_warning\(\)](#) instead!

Global **nrerror** (const char message[])

Use [vrna\\_message\\_error\(\)](#) instead!

Global **space** (unsigned size)

Use [vrna\\_alloc\(\)](#) instead!

Global **xrealloc** (void \*p, unsigned size)

Use [vrna\\_realloc\(\)](#) instead!

Global **init\_rand** (void)

Use [vrna\\_init\\_rand\(\)](#) instead!

Global **urn** (void)

Use [vrna\\_urn\(\)](#) instead!

Global **int\_urn** (int from, int to)

Use [vrna\\_int\\_urn\(\)](#) instead!

File **constraints\_ligand.h**

Use [ViennaRNA/constraints/ligand.h](#) instead

File **char\_stream.h**

Use [ViennaRNA/datastructures/char\\_stream.h](#) 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 **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 **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 **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)

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\_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 **get\_monomere\_mfes** (float \*e1, float \*e2)

{This function is obsolete and will be removed soon!}

Global **initialize\_cofold** (int length)

{This function is obsolete and will be removed soon!}

Global **get\_concentrations** (double FEAB, double FEAA, double FEBB, double FEA, double FEB, double \*startconc)

{ Use [vrna\\_pf\\_dimer\\_concentrations\(\)](#) instead!}

**File `constraints.h`**

Use [ViennaRNA/constraints/basic.h](#) instead

**File `constraints_hard.h`**

Use [ViennaRNA/constraints/hard.h](#) instead

**Global `get_centroid_struct_pr` (int length, double \*dist, FLT\_OR\_DBL \*pr)**

This function was renamed to [vrna\\_centroid\\_from\\_probs\(\)](#)

**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

**File `data_structures.h`**

Use [ViennaRNA/datastructures/basic.h](#) instead

**File `energy_const.h`**

Use [ViennaRNA/params/constants.h](#) instead

**File `energy_par.h`**

Use [ViennaRNA/params/default.h](#) 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_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_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_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!

**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_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 `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 `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 `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 `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!

**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 [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!

**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 [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 [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!

**Global [alifold](#) (const char \*\*strings, char \*structure)**

Usage of this function is discouraged! Use [vrna\\_alifold\(\)](#), or [vrna\\_mfe\(\)](#) instead!

**Global [circularfold](#) (const char \*\*strings, char \*structure)**

Usage of this function is discouraged! Use [vrna\\_alicircfold\(\)](#), and [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 [energy\\_of\\_move](#) (const char \*string, const char \*structure, int m1, int m2)**

Use [vrna\\_eval\\_move\(\)](#) instead!

**Global [cv\\_fact](#)**

See [vrna\\_md\\_t.cv\\_fact](#), and [vrna\\_mfe\(\)](#) to avoid using global variables

**Global [nc\\_fact](#)**

See [vrna\\_md\\_t.nc\\_fact](#), and [vrna\\_mfe\(\)](#) to avoid using global variables

**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

**Global [alipf\\_fold](#) (const char \*\*sequences, char \*structure, vrna\_ep\_t \*\*pl)**

Use [vrna\\_pf\(\)](#) instead

**Global [alipf\\_circ\\_fold](#) (const char \*\*sequences, char \*structure, vrna\_ep\_t \*\*pl)**

Use [vrna\\_pf\(\)](#) 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 [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 [alipbacktrack](#) (double \*prob)**

Use [vrna\\_pbacktrack\(\)](#) instead!

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 **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](#).

File **aln\_util.h**

Use [ViennaRNA/utis/alignments.h](#) instead

Global **get\_centroid\_struct\_pl** (int length, double \*dist, vrna\_ep\_t \*pl)

This function was renamed to [vrna\\_centroid\\_from\\_plist\(\)](#)

Global **st\_back**

set the *uniq\_ML* flag in [vrna\\_md\\_t](#) before passing it to [vrna\\_fold\\_compound\(\)](#).

Global **iindx**

Do not use this variable anymore!

File **hairpin\_loops.h**

Use [ViennaRNA/loops/hairpin.h](#) 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 **MEA** (plist \*p, char \*structure, double gamma)

Use [vrna\\_MEA\(\)](#) or [vrna\\_MEA\\_from\\_plist\(\)](#) instead!

Global **temperature**

Use [vrna\\_md\\_defaults\\_temperature\(\)](#), and [vrna\\_md\\_defaults\\_temperature\\_get\(\)](#) to change, and read the global default temperature settings

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!

File **multibranch\_loops.h**

Use [ViennaRNA/loops/multibranch.h](#) instead

File **naview.h**

Use [ViennaRNA/plotting/naview.h](#) instead

File **neighbor.h**

Use [ViennaRNA/landscape/neighbor.h](#) instead

File **params.h**

Use [ViennaRNA/params/basic.h](#) instead

Global **pr**

Do not use this variable anymore!

Global **pf\_fold\_par** (const char \*sequence, char \*structure, vrna\_exp\_param\_t \*parameters, int calculate\_bpmm, int is\_constrained, int is\_circular)

Use [vrna\\_pf\(\)](#) instead

Global **pf\_circ\_fold** (const char \*sequence, char \*structure)

Use [vrna\\_pf\(\)](#) instead!

Global **pbacktrack\_circ** (char \*sequence)

Use [vrna\\_pbacktrack\(\)](#) instead.

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 **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 **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!

Global **stackProb** (double cutoff)

Use [vrna\\_stack\\_prob\(\)](#) instead!

Global **init\_pf\_fold** (int length)

This function is obsolete and will be removed soon!

Global **centroid** (int length, double \*dist)

This function is deprecated and should not be used anymore as it is not threadsafe!

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 **free\_arrays** (void)

See [vrna\\_fold\(\)](#), [vrna\\_circfold\(\)](#), or [vrna\\_mfe\(\)](#) and [vrna\\_fold\\_compound\\_t](#) for the usage of the new API!

Global **energy\_of\_move\_pt** (short \*pt, short \*s, short \*s1, int m1, int m2)

Use [vrna\\_eval\\_move\\_pt\(\)](#) instead!

Global **loop\_energy** (short \*ptable, short \*s, short \*s1, int i)

Use [vrna\\_eval\\_loop\\_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\_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\_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!

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

**File `findpath.h`**

Use [ViennaRNA/landscape/findpath.h](#) instead

Global **`fold_par`** (`const char *sequence`, `char *structure`, `vrna_param_t *parameters`, `int is_constrained`, `int is_circular`)

use [vrna\\_mfe\(\)](#) instead!

Global **`fold`** (`const char *sequence`, `char *structure`)

use [vrna\\_fold\(\)](#), or [vrna\\_mfe\(\)](#) instead!

Global **`circfold`** (`const char *sequence`, `char *structure`)

Use [vrna\\_circfold\(\)](#), or [vrna\\_mfe\(\)](#) 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 **`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 **`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!

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 **`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 **`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 **`initialize_fold`** (`int length`)

See [vrna\\_mfe\(\)](#) and [vrna\\_fold\\_compound\\_t](#) for the usage of the new API!

Global **`vrna_fc_s::ptype_pf_compat`**

This attribute will vanish in the future! It's meant for backward compatibility only!

Global **`vrna_fc_s::pscore_pf_compat`**

This attribute will vanish in the future!

Global **`base_pair`**

Do not use this variable anymore!





## Chapter 12

# Bug List

### 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!

### 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.



## Chapter 13

# Module Index

### 13.1 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 . . . . .	131
Energy Evaluation for Individual Loops . . . . .	157
Exterior Loops . . . . .	443
Hairpin Loops . . . . .	447
Internal Loops . . . . .	452
Multibranch Loops . . . . .	454
Energy Evaluation for Atomic Moves . . . . .	160
Deprecated Interface for Free Energy Evaluation . . . . .	163
The RNA Folding Grammar . . . . .	178
Fine-tuning of the Implemented Models . . . . .	180
Energy Parameters . . . . .	216
Reading/Writing Energy Parameter Sets from/to File . . . . .	462
Converting Energy Parameter Files . . . . .	471
Extending the Folding Grammar with Additional Domains . . . . .	231
Unstructured Domains . . . . .	232
Structured Domains . . . . .	244
Constraining the RNA Folding Grammar . . . . .	245
Hard Constraints . . . . .	262
Soft Constraints . . . . .	274
The RNA Secondary Structure Landscape . . . . .	286
Neighborhood Relation and Move Sets for Secondary Structures . . . . .	386
(Re-)folding Paths, Saddle Points, Energy Barriers, and Local Minima . . . . .	400
Direct Refolding Paths between two Secondary Structures . . . . .	405
Folding Paths that start at a single Secondary Structure . . . . .	412
Deprecated Interface for (Re-)folding Paths, Saddle Points, and Energy Barriers . . . . .	716
Minimum Free Energy (MFE) Algorithms . . . . .	287
Global MFE Prediction . . . . .	290
Computing MFE representatives of a Distance Based Partitioning . . . . .	357
Deprecated Interface for Global MFE Prediction . . . . .	658
Local (sliding window) MFE Prediction . . . . .	297
Deprecated Interface for Local (Sliding Window) MFE Prediction . . . . .	671
Backtracking MFE structures . . . . .	302

Partition Function and Equilibrium Properties . . . . .	288
Global Partition Function and Equilibrium Probabilities . . . . .	305
Computing Partition Functions of a Distance Based Partitioning . . . . .	365
Predicting various thermodynamic properties . . . . .	371
Deprecated Interface for Global Partition Function Computation . . . . .	673
Local (sliding window) Partition Function and Equilibrium Probabilities . . . . .	313
Deprecated Interface for Local (Sliding Window) Partition Function Computation . . . . .	691
Suboptimals and Representative Structures . . . . .	323
Suboptimal Structures sensu Stiegler et al. 1984 / Zuker et al. 1989 . . . . .	324
Suboptimal Structures within an Energy Band around the MFE . . . . .	326
Random Structure Samples from the Ensemble . . . . .	331
Stochastic Backtracking of Structures from Distance Based Partitioning . . . . .	368
Deprecated Interface for Stochastic Backtracking . . . . .	694
Compute the Structure with Maximum Expected Accuracy (MEA) . . . . .	348
Compute the Centroid Structure . . . . .	351
RNA-RNA Interaction . . . . .	354
Partition Function for Two Hybridized Sequences . . . . .	456
Partition Function for two Hybridized Sequences as a Stepwise Process . . . . .	459
Classified Dynamic Programming Variants . . . . .	355
Distance Based Partitioning of the Secondary Structure Space . . . . .	356
Computing MFE representatives of a Distance Based Partitioning . . . . .	357
Computing Partition Functions of a Distance Based Partitioning . . . . .	365
Stochastic Backtracking of Structures from Distance Based Partitioning . . . . .	368
Compute the Density of States . . . . .	382
Inverse Folding (Design) . . . . .	383
Experimental Structure Probing Data . . . . .	417
SHAPE Reactivity Data . . . . .	418
Generate Soft Constraints from Data . . . . .	422
Ligands Binding to RNA Structures . . . . .	427
Ligands Binding to Unstructured Domains . . . . .	428
Incorporating Ligands Binding to Specific Sequence/Structure Motifs using Soft Constraints . . . . .	429
Complex Structured Modules . . . . .	431
G-Quadruplexes . . . . .	432
Utilities . . . . .	435
Utilities to deal with Nucleotide Alphabets . . . . .	477
(Nucleic Acid Sequence) String Utilites . . . . .	482
Secondary Structure Utilities . . . . .	491
Dot-Bracket Notation of Secondary Structures . . . . .	494
Pair Table Representation of Secondary Structures . . . . .	503
Pair List Representation of Secondary Structures . . . . .	507
Helix List Representation of Secondary Structures . . . . .	510
Tree Representation of Secondary Structures . . . . .	512
Deprecated Interface for Secondary Structure Utilities . . . . .	701
Multiple Sequence Alignment Utilities . . . . .	517
Deprecated Interface for Multiple Sequence Alignment Utilities . . . . .	697
Files and I/O . . . . .	526
Nucleic Acid Sequences and Structures . . . . .	530
Multiple Sequence Alignments . . . . .	538
Command Files . . . . .	548
Plotting . . . . .	553
Layouts and Coordinates . . . . .	561
Annotation . . . . .	580
Alignment Plots . . . . .	581
Deprecated Interface for Plotting Utilities . . . . .	712
Search Algorithms . . . . .	584

Combinatorics Algorithms . . . . .	588
(Abstract) Data Structures . . . . .	595
The Fold Compound . . . . .	612
The Dynamic Programming Matrices . . . . .	629
Hash Tables . . . . .	634
Heaps . . . . .	644
Buffers . . . . .	652
Messages . . . . .	602
Unit Conversion . . . . .	607



## Chapter 14

# Data Structure Index

### 14.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">_struct_en</a>	
Data structure for <a href="#">energy_of_move()</a>	719
<a href="#">LIST</a>	719
<a href="#">LST_BUCKET</a>	720
<a href="#">Postorder_list</a>	
Postorder data structure	720
<a href="#">swString</a>	
Some other data structure	720
<a href="#">Tree</a>	
<a href="#">Tree</a> data structure	721
<a href="#">TwoDpfold_vars</a>	
Variables compound for 2Dfold partition function folding	721
<a href="#">vrna_dimer_conc_s</a>	
Data structure for concentration dependency computations	722
<a href="#">vrna_sc_bp_storage_t</a>	
A base pair constraint	723
<a href="#">vrna_sc_motif_s</a>	723
<a href="#">vrna_structured_domains_s</a>	723
<a href="#">vrna_subopt_sol_s</a>	
Solution element from subopt.c	724
<a href="#">vrna_unstructured_domain_motif_s</a>	724





## Chapter 15

# File Index

### 15.1 File List

Here is a list of all documented files with brief descriptions:

ViennaRNA/2Dfold.h	
MFE structures for base pair distance classes . . . . .	725
ViennaRNA/2Dpfold.h	
Partition function implementations for base pair distance classes . . . . .	726
ViennaRNA/ali_plex.h . . . . .	??
ViennaRNA/alifold.h	
Functions for comparative structure prediction using RNA sequence alignments . . . . .	731
ViennaRNA/aln_util.h	
Use ViennaRNA/utls/alignments.h instead . . . . .	733
ViennaRNA/alphabet.h	
Functions to process, convert, and generally handle different nucleotide and/or base pair alphabets . . . . .	734
ViennaRNA/boltzmann_sampling.h	
Boltzmann Sampling of secondary structures from the ensemble . . . . .	735
ViennaRNA/centroid.h	
Centroid structure computation . . . . .	737
ViennaRNA/char_stream.h	
Use ViennaRNA/datastructures/char_stream.h instead . . . . .	738
ViennaRNA/cofold.h	
MFE implementations for RNA-RNA interaction . . . . .	740
ViennaRNA/combinatorics.h	
Various implementations that deal with combinatorial aspects of objects . . . . .	741
ViennaRNA/commands.h	
Parse and apply different commands that alter the behavior of secondary structure prediction and evaluation . . . . .	742
ViennaRNA/concentrations.h	
Concentration computations for RNA-RNA interactions . . . . .	743
ViennaRNA/constraints.h	
Use ViennaRNA/constraints/basic.h instead . . . . .	744
ViennaRNA/constraints_hard.h	
Use ViennaRNA/constraints/hard.h instead . . . . .	756
ViennaRNA/constraints_ligand.h	
Use ViennaRNA/constraints/ligand.h instead . . . . .	756
ViennaRNA/constraints_SHAPE.h	
Use ViennaRNA/constraints/SHAPE.h instead . . . . .	757

ViennaRNA/constraints_soft.h	
Use <a href="#">ViennaRNA/constraints/soft.h</a> instead	757
ViennaRNA/convert_epars.h	
Use <a href="#">ViennaRNA/params/convert.h</a> instead	758
ViennaRNA/data_structures.h	
Use <a href="#">ViennaRNA/datastructures/basic.h</a> instead	758
ViennaRNA/dist_vars.h	
Global variables for Distance-Package	761
ViennaRNA/dp_matrices.h	
Functions to deal with standard dynamic programming (DP) matrices	762
ViennaRNA/duplex.h	
Functions for simple RNA-RNA duplex interactions	763
ViennaRNA/edit_cost.h	
Global variables for Edit Costs included by treedist.c and stringdist.c	764
ViennaRNA/energy_const.h	
Use <a href="#">ViennaRNA/params/constants.h</a> instead	764
ViennaRNA/energy_par.h	
Use <a href="#">ViennaRNA/params/default.h</a> instead	765
ViennaRNA/equilibrium_probs.h	
Equilibrium Probability implementations	765
ViennaRNA/eval.h	
Functions and variables related to energy evaluation of sequence/structure pairs	767
ViennaRNA/exterior_loops.h	
Use <a href="#">ViennaRNA/loops/external.h</a> instead	770
ViennaRNA/file_formats.h	
Use <a href="#">ViennaRNA/io/file_formats.h</a> instead	771
ViennaRNA/file_formats_msa.h	
Use <a href="#">ViennaRNA/io/file_formats_msa.h</a> instead	772
ViennaRNA/file_utils.h	
Use <a href="#">ViennaRNA/io/utils.h</a> instead	775
ViennaRNA/findpath.h	
Use <a href="#">ViennaRNA/landscape/findpath.h</a> instead	776
ViennaRNA/fold.h	
MFE calculations for single RNA sequences	777
ViennaRNA/fold_compound.h	
The Basic Fold Compound API	778
ViennaRNA/fold_vars.h	
Here all all declarations of the global variables used throughout RNALib	780
ViennaRNA/gquad.h	
G-quadruplexes	783
ViennaRNA/grammar.h	
Implementations for the RNA folding grammar	783
ViennaRNA/hairpin_loops.h	
Use <a href="#">ViennaRNA/loops/hairpin.h</a> instead	784
ViennaRNA/heat_capacity.h	
Compute heat capacity for an RNA	784
ViennaRNA/interior_loops.h	
Use <a href="#">ViennaRNA/loops/internal.h</a> instead	785
ViennaRNA/inverse.h	
Inverse folding routines	786
ViennaRNA/Lfold.h	
Functions for locally optimal MFE structure prediction	789
ViennaRNA/loop_energies.h	
Use <a href="#">ViennaRNA/loops/all.h</a> instead	790
ViennaRNA/LPfold.h	
Partition function and equilibrium probability implementation for the sliding window algorithm	797
ViennaRNA/MEA.h	
Computes a MEA (maximum expected accuracy) structure	798

ViennaRNA/ <a href="#">mfe.h</a>	
Compute Minimum Free energy (MFE) and backtrace corresponding secondary structures from RNA sequence data	799
ViennaRNA/ <a href="#">mfe_window.h</a>	
Compute local Minimum Free Energy (MFE) using a sliding window approach and backtrace corresponding secondary structures	800
ViennaRNA/ <a href="#">mm.h</a>	
Several Maximum Matching implementations	801
ViennaRNA/ <a href="#">model.h</a>	
The model details data structure and its corresponding modifiers	802
ViennaRNA/ <a href="#">move_set.h</a>	??
ViennaRNA/ <a href="#">multibranch_loops.h</a>	
Use <a href="#">ViennaRNA/loops/multibranch.h</a> instead	807
ViennaRNA/ <a href="#">naview.h</a>	
Use <a href="#">ViennaRNA/plotting/naview.h</a> instead	807
ViennaRNA/ <a href="#">neighbor.h</a>	
Use <a href="#">ViennaRNA/landscape/neighbor.h</a> instead	809
ViennaRNA/ <a href="#">pair_mat.h</a>	??
ViennaRNA/ <a href="#">params.h</a>	
Use <a href="#">ViennaRNA/params/basic.h</a> instead	811
ViennaRNA/ <a href="#">part_func.h</a>	
Partition function implementations	828
ViennaRNA/ <a href="#">part_func_co.h</a>	
Partition function for two RNA sequences	832
ViennaRNA/ <a href="#">part_func_up.h</a>	
Implementations for accessibility and RNA-RNA interaction as a stepwise process	833
ViennaRNA/ <a href="#">part_func_window.h</a>	
Partition function and equilibrium probability implementation for the sliding window algorithm	834
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	835
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	837
ViennaRNA/ <a href="#">plot_layouts.h</a>	
Use <a href="#">ViennaRNA/plotting/layouts.h</a> instead	837
ViennaRNA/ <a href="#">plot_structure.h</a>	
Use <a href="#">ViennaRNA/plotting/structures.h</a> instead	838
ViennaRNA/ <a href="#">plot_utils.h</a>	
Use <a href="#">ViennaRNA/plotting/utils.h</a> instead	838
ViennaRNA/ <a href="#">ProfileAln.h</a>	??
ViennaRNA/ <a href="#">profiledist.h</a>	851
ViennaRNA/ <a href="#">PS_dot.h</a>	
Use <a href="#">ViennaRNA/plotting/probabilities.h</a> instead	853
ViennaRNA/ <a href="#">read_epars.h</a>	
Use <a href="#">ViennaRNA/params/io.h</a> instead	853
ViennaRNA/ <a href="#">ribo.h</a>	
Parse RiboSum Scoring Matrices for Covariance Scoring of Alignments	854
ViennaRNA/ <a href="#">RNAstruct.h</a>	
Parsing and Coarse Graining of Structures	854
ViennaRNA/ <a href="#">sequence.h</a>	
Functions and data structures related to sequence representations ,	856
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	857

ViennaRNA/string_utils.h	
Use <a href="#">ViennaRNA/utls/strings.h</a> instead	859
ViennaRNA/stringdist.h	
Functions for String Alignment	859
ViennaRNA/structure_utils.h	
Use <a href="#">ViennaRNA/utls/structures.h</a> instead	860
ViennaRNA/structured_domains.h	
This module provides interfaces that deal with additional structured domains in the folding grammar	861
ViennaRNA/subopt.h	
RNAsubopt and density of states declarations	862
ViennaRNA/svm_utils.h	
Use <a href="#">ViennaRNA/utls/svm.h</a> instead	863
ViennaRNA/treedist.h	
Functions for Tree Edit Distances	864
ViennaRNA/ugly_bt.h	??
ViennaRNA/units.h	
Use <a href="#">ViennaRNA/utls/units.h</a> instead	865
ViennaRNA/unstructured_domains.h	
Functions to modify unstructured domains, e.g. to incorporate ligands binding to unpaired stretches	867
ViennaRNA/utls.h	
Use <a href="#">ViennaRNA/utls/basic.h</a> instead	869
ViennaRNA/vrna_config.h	??
ViennaRNA/walk.h	
Use <a href="#">ViennaRNA/landscape/walk.h</a> instead	875
ViennaRNA/zscore.h	??
ViennaRNA/constraints/basic.h	
Functions and data structures for constraining secondary structure predictions and evaluation	814
ViennaRNA/constraints/hard.h	
Functions and data structures for handling of secondary structure hard constraints	745
ViennaRNA/constraints/ligand.h	
Functions for incorporation of ligands binding to hairpin and interior loop motifs using the soft constraints framework	750
ViennaRNA/constraints/SHAPE.h	
This module provides function to incorporate SHAPE reactivity data into the folding recursions by means of soft constraints	752
ViennaRNA/constraints/soft.h	
Functions and data structures for secondary structure soft constraints	754
ViennaRNA/datastructures/basic.h	
Various data structures and pre-processor macros	821
ViennaRNA/datastructures/char_stream.h	
Implementation of a dynamic, buffered character stream	739
ViennaRNA/datastructures/hash_tables.h	
Implementations of hash table functions	759
ViennaRNA/datastructures/heap.h	
Implementation of an abstract heap data structure	760
ViennaRNA/datastructures/lists.h	??
ViennaRNA/datastructures/stream_output.h	
An implementation of a buffered, ordered stream output data structure	858
ViennaRNA/io/file_formats.h	
Read and write different file formats for RNA sequences, structures	771
ViennaRNA/io/file_formats_msa.h	
Functions dealing with file formats for Multiple Sequence Alignments (MSA)	773
ViennaRNA/io/utls.h	
Several utilities for file handling	870
ViennaRNA/landscape/findpath.h	
A breadth-first search heuristic for optimal direct folding paths	776

ViennaRNA/landscape/ <a href="#">move.h</a>	
Methods to operate with structural neighbors of RNA secondary structures . . . . .	786
ViennaRNA/landscape/ <a href="#">neighbor.h</a>	
Methods to compute the neighbors of an RNA secondary structure . . . . .	809
ViennaRNA/landscape/ <a href="#">paths.h</a>	
API for computing (optimal) (re-)folding paths between secondary structures . . . . .	788
ViennaRNA/landscape/ <a href="#">walk.h</a>	
Methods to generate particular paths such as gradient or random walks through the energy landscape of an RNA sequence . . . . .	876
ViennaRNA/loops/ <a href="#">all.h</a>	
Energy evaluation for MFE and partition function calculations . . . . .	790
ViennaRNA/loops/ <a href="#">external.h</a>	
Energy evaluation of exterior loops for MFE and partition function calculations . . . . .	791
ViennaRNA/loops/ <a href="#">hairpin.h</a>	
Energy evaluation of hairpin loops for MFE and partition function calculations . . . . .	793
ViennaRNA/loops/ <a href="#">internal.h</a>	
Energy evaluation of interior loops for MFE and partition function calculations . . . . .	794
ViennaRNA/loops/ <a href="#">multibranch.h</a>	
Energy evaluation of multibranch loops for MFE and partition function calculations . . . . .	796
ViennaRNA/params/ <a href="#">1.8.4_epars.h</a>	
Free energy parameters for parameter file conversion . . . . .	811
ViennaRNA/params/ <a href="#">1.8.4_intloops.h</a>	
Free energy parameters for interior loop contributions needed by the parameter file conversion functions . . . . .	812
ViennaRNA/params/ <a href="#">basic.h</a>	
Functions to deal with sets of energy parameters . . . . .	812
ViennaRNA/params/ <a href="#">constants.h</a>	
Energy parameter constants . . . . .	823
ViennaRNA/params/ <a href="#">convert.h</a>	
Functions and definitions for energy parameter file format conversion . . . . .	825
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 . . . . .	826
ViennaRNA/plotting/ <a href="#">alignments.h</a>	
Various functions for plotting Sequence / Structure Alignments . . . . .	839
ViennaRNA/plotting/ <a href="#">layouts.h</a>	
Secondary structure plot layout algorithms . . . . .	842
ViennaRNA/plotting/ <a href="#">naview.h</a>	
Implementation of the Naview RNA secondary structure layout algorithm [5] . . . . .	808
ViennaRNA/plotting/ <a href="#">probabilities.h</a>	
Various functions for plotting RNA secondary structures, dot-plots and other visualizations . . . . .	844
ViennaRNA/plotting/ <a href="#">structures.h</a>	
Various functions for plotting RNA secondary structures . . . . .	846
ViennaRNA/plotting/ <a href="#">utils.h</a>	
Various utilities to assist in plotting secondary structures and consensus structures . . . . .	871
ViennaRNA/plotting/RNApuzzler/ <a href="#">RNApuzzler.h</a>	
Implementation of the RNApuzzler RNA secondary structure layout algorithm [23] . . . . .	845
ViennaRNA/plotting/RNApuzzler/ <a href="#">RNAturtle.h</a>	
Implementation of the RNAturtle RNA secondary structure layout algorithm [23] . . . . .	846
ViennaRNA/search/ <a href="#">BoyerMoore.h</a>	
Variants of the Boyer-Moore string search algorithm . . . . .	856

ViennaRNA/utis/ <a href="#">alignments.h</a>	
Various utility- and helper-functions for sequence alignments and comparative structure prediction . . . . .	839
ViennaRNA/utis/ <a href="#">basic.h</a>	
General utility- and helper-functions used throughout the <i>ViennaRNA Package</i> . . . . .	815
ViennaRNA/utis/ <b>cpu.h</b>	??
ViennaRNA/utis/ <b>higher_order_functions.h</b>	??
ViennaRNA/utis/ <a href="#">strings.h</a>	
General utility- and helper-functions for RNA sequence and structure strings used throughout the ViennaRNA Package . . . . .	872
ViennaRNA/utis/ <a href="#">structures.h</a>	
Various utility- and helper-functions for secondary structure parsing, converting, etc . . . . .	848
ViennaRNA/utis/ <b>svm.h</b>	??
ViennaRNA/utis/ <a href="#">units.h</a>	
Physical Units and Functions to convert them into each other . . . . .	866

## Chapter 16

# Module Documentation

### 16.1 Free Energy Evaluation

Functions and variables related to free energy evaluation of sequence/structure pairs.

#### 16.1.1 Detailed Description

Functions and variables related to free energy evaluation of sequence/structure pairs.

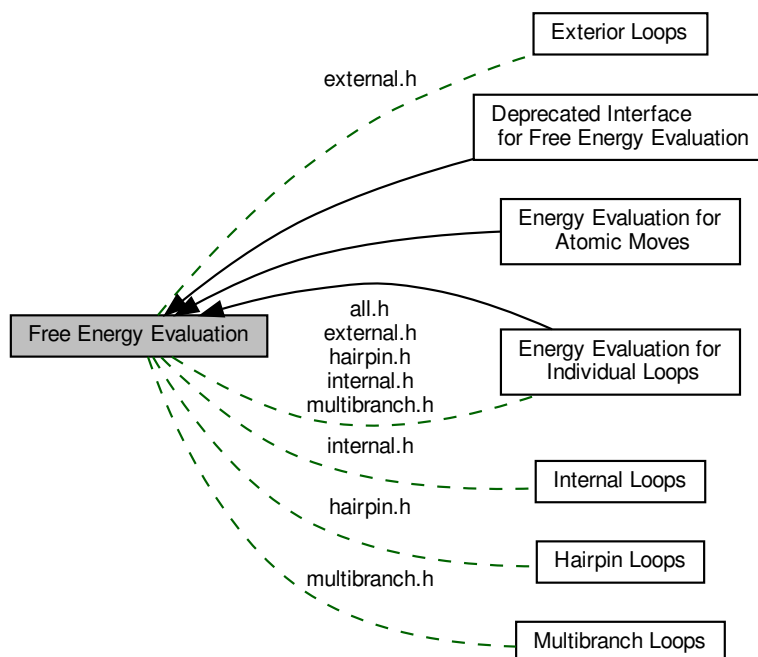
Several different functions to evaluate the free energy of a particular secondary structure under a particular set of parameters and the Nearest Neighbor Energy model are available. For most of them, two different forms of representations for the secondary structure may be used:

- 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)

## 16.1.2 Function Documentation

### 16.1.2.1 `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

<code>vc</code>	A <code>vrna_fold_compound_t</code> containing the energy parameters and model details
<code>structure</code>	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`

### 16.1.2.2 `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

<code>vc</code>	A <code>vrna_fold_compound_t</code> containing the energy parameters and model details
<code>structure</code>	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`

### 16.1.2.3 `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

<code>vc</code>	A <code>vrna_fold_compound_t</code> containing the energy parameters and model details
<code>structure</code>	Secondary structure in dot-bracket notation
<code>file</code>	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*

**16.1.2.4 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

### 16.1.2.5 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 vrna_fold_compound_t 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*

### 16.1.2.6 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`

16.1.2.7 `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

**16.1.2.8 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.

**16.1.2.9 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.

**16.1.2.10 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.

### 16.1.2.11 `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\\_VERBOSE\\_QUIET](#) and NULL, respectively.

### 16.1.2.12 `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 simplified 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\\_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>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 not available. Use [vrna\\_eval\\_structure\\_simple\\_v\(\)](#) instead!

## 16.1.2.13 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.

#### 16.1.2.14 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 <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\\_structure\(\)](#). The last two arguments for this function are optional and default to [VRNA\\_VERBOSITY\\_QUIET](#) and `NULL`, respectively.

#### 16.1.2.15 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.

#### 16.1.2.16 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 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\\_gquad\\_structure\(\)](#). The last two arguments for this function are optional and default to [VRNA\\_VERBOSITY\\_QUIET](#) and `NULL`, respectively.

**16.1.2.17 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

**16.1.2.18 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

**16.1.2.19 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

### 16.1.2.20 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

16.1.2.21 `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!

### 16.1.2.22 `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.

### 16.1.2.23 `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.

### 16.1.2.24 `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- $\leftrightarrow$  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.

#### 16.1.2.25 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.

#### 16.1.2.26 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.

**16.1.2.27 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

**16.1.2.28 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

#### 16.1.2.29 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



## 16.2 Energy Evaluation for Individual Loops

Functions to evaluate the free energy of particular types of loops.

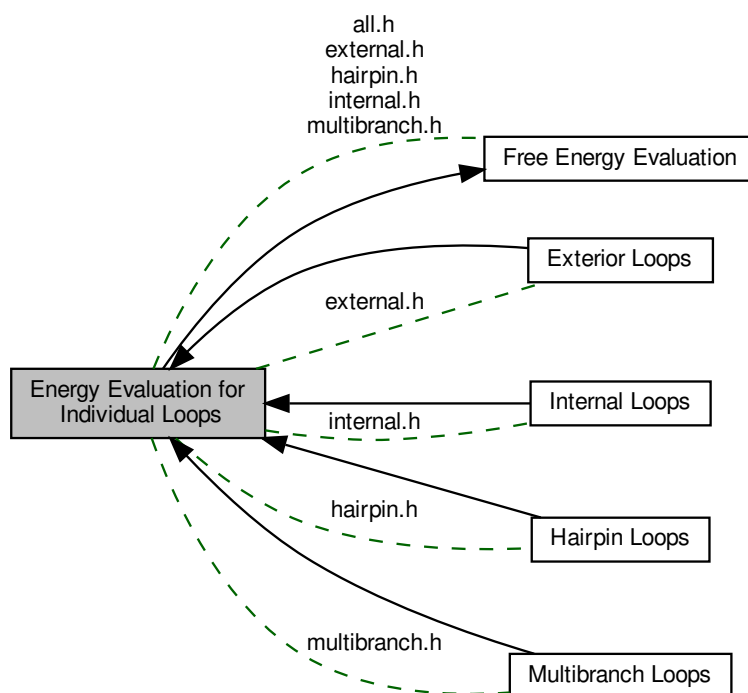
### 16.2.1 Detailed Description

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.*

## 16.2.2 Function Documentation

### 16.2.2.1 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 <a href="#">vrna_fold_compound_t</a> 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*

### 16.2.2.2 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

## 16.3 Energy Evaluation for Atomic Moves

Functions to evaluate the free energy change of a structure after application of (a set of) atomic moves.

### 16.3.1 Detailed Description

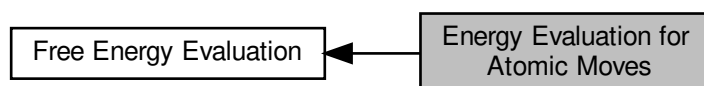
Functions to evaluate the free energy change of a structure after application of (a set of) atomic moves.

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)*

### 16.3.2 Function Documentation

### 16.3.2.1 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`

### 16.3.2.2 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*

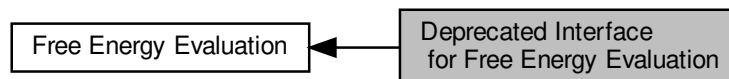
## 16.4 Deprecated Interface for Free Energy Evaluation

Deprecated Energy Evaluation functions.

### 16.4.1 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, [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*

## 16.4.2 Function Documentation

### 16.4.2.1 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



### 16.4.2.2 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

### 16.4.2.3 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

**16.4.2.4 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

**16.4.2.5 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

#### 16.4.2.6 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

#### 16.4.2.7 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

**16.4.2.8 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

**16.4.2.9 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

#### 16.4.2.10 `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

**16.4.2.11 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

**16.4.2.12 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

**16.4.2.13 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 on 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

#### 16.4.2.14 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

#### 16.4.2.15 exp\_E\_Stem()

```
FLT_OR_DBL exp_E_Stem (
    int type,
    int sil,
    int sjl,
    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

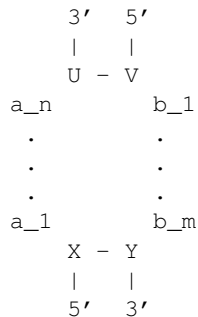
#### 16.4.2.16 E\_IntLoop()

```
PRIVATE int E_IntLoop (
    int n1,
    int n2,
    int type,
    int type_2,
    int sil,
    int sjl,
    int spl,
    int sql,
    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

<i>n1</i>	The size of the 'left'-loop (number of unpaired nucleotides)
<i>n2</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>type<sub>↔</sub> _2</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 energy parameters

**Returns**

The Free energy of the Interior-loop in dcal/mol

**16.4.2.17 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

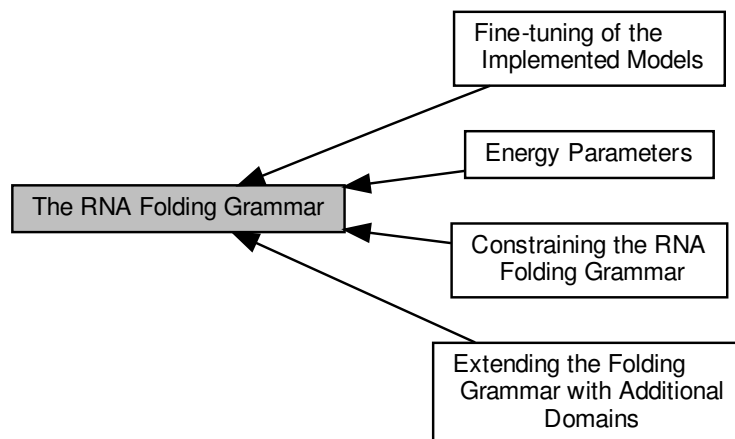
## 16.5 The RNA Folding Grammar

The RNA folding grammar as implemented in RNAlib.

### 16.5.1 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](#)

### 16.5.2 Data Structure Documentation

#### 16.5.2.1 struct vrna\_gr\_aux\_s

##### Data Fields

- vrna\_callback\_gr\_cond \* [cb\\_proc](#)  
*A callback for pre- and post-processing of auxiliary grammar rules.*

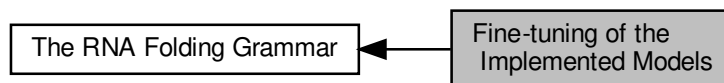
## 16.6 Fine-tuning of the Implemented Models

Functions and data structures to fine-tune the implemented secondary structure evaluation model.

### 16.6.1 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 [oldAliEn](#)  
*use old alifold energies (with gaps)*
- int [ribo](#)  
*use ribosum matrices*
- int [logML](#)  
*if nonzero use logarithmic ML energy in [energy\\_of\\_struct](#)*

## 16.6.2 Data Structure Documentation

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

## 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 `pf_smooth`  
A flag specifying whether energies in Boltzmann factors need to be smoothed.

- 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 [oldAlfEn](#)  
*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.*

### 16.6.2.1.1 Field Documentation

#### 16.6.2.1.1.1 `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.

#### 16.6.2.1.1.2 `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.

## 16.6.3 Macro Definition Documentation

### 16.6.3.1 `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  $^{\circ}C$

#### See also

[vrna\\_md\\_t.temperature](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#)

### 16.6.3.2 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\(\)](#)

### 16.6.3.3 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\(\)](#)

### 16.6.3.4 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\(\)](#)

### 16.6.3.5 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\(\)](#)



### 16.6.3.6 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\(\)](#)

### 16.6.3.7 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\(\)](#)

### 16.6.3.8 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\(\)](#)

### 16.6.3.9 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\(\)](#)

### 16.6.3.10 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\(\)](#)

### 16.6.3.11 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\(\)](#)

### 16.6.3.12 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\(\)](#)

### 16.6.3.13 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\(\)](#)

#### 16.6.3.14 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\(\)](#)

#### 16.6.3.15 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\(\)](#)

#### 16.6.3.16 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\(\)](#)

#### 16.6.3.17 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\(\)](#)

### 16.6.3.18 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\(\)](#)

### 16.6.3.19 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\(\)](#)

### 16.6.3.20 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\(\)](#)

### 16.6.3.21 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\(\)](#)

### 16.6.3.22 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\(\)](#)

## 16.6.4 Function Documentation

### 16.6.4.1 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
-----------	---

### 16.6.4.2 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\(\)](#)

#### 16.6.4.3 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`)

#### 16.6.4.4 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!

#### 16.6.4.5 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> ↔ _p	A set of model details to use as global default (if NULL is passed, factory defaults are restored)
-------------------	--

**16.6.4.6 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
----------	---------------------------

**16.6.4.7 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

**16.6.4.8 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
----------	------------------------------------

**16.6.4.9 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



**16.6.4.10 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
----------	------------------

**16.6.4.11 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

**16.6.4.12 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)
-------------	------------------------------------

**16.6.4.13 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

**16.6.4.14 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)
-------------	------------------------------------

**16.6.4.15 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

#### 16.6.4.16 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)
-------------	------------------------------------

#### 16.6.4.17 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

**16.6.4.18 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)
-------------	------------------------------------

**16.6.4.19 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

**16.6.4.20 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)
-------------	------------------------------------

**16.6.4.21 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

**16.6.4.22 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)
-------------	------------------------------------

**16.6.4.23 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

#### 16.6.4.24 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)
-------------	------------------------------------

#### 16.6.4.25 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

**16.6.4.26 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)
-------------	------------------------------------

**16.6.4.27 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

**16.6.4.28 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)
----------	-------------------------

**16.6.4.29 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

**16.6.4.30 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)
-------------	------------------------------------

**16.6.4.31 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

#### 16.6.4.32 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')
----------	-----------------------------

#### 16.6.4.33 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

**16.6.4.34 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)
-------------	------------------------------------

**16.6.4.35 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\\_](#)

Returns

The global default settings that specify whether base pair probabilities are computed together with partition function

**16.6.4.36 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
-------------	------------------------

**16.6.4.37 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

**16.6.4.38 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
-------------	--

**16.6.4.39 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

#### 16.6.4.40 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
-------------	--------------------------------

#### 16.6.4.41 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

**16.6.4.42 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)
-------------	------------------------------------

**16.6.4.43 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

**16.6.4.44 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)
-------------	------------------------------------

**16.6.4.45 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

**16.6.4.46 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
---------------	------------------------

**16.6.4.47 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\\_FACTOR](#)

Returns

The global default settings for the co-variance factor

#### 16.6.4.48 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\\_FACTOR](#)

Parameters

<i>factor</i>	
---------------	--

#### 16.6.4.49 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\\_FACTOR](#)

Returns

#### 16.6.4.50 `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)
---------------	------------------------------------

#### 16.6.4.51 `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

#### 16.6.4.52 `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
-----------	---

## 16.6.5 Variable Documentation

### 16.6.5.1 temperature

```
double temperature [extern]
```

```
#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()`

### 16.6.5.2 pf\_scale

```
double pf_scale [extern]
```

```
#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.

### 16.6.5.3 dangles

```
int dangles [extern]
```

```
#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

### 16.6.5.4 tetra\_loop

```
int tetra_loop [extern]
```

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

Include special stabilizing energies for some tri-, tetra- and hexa-loops;.

default is 1.

### 16.6.5.5 noLonelyPairs

```
int noLonelyPairs [extern]
```

```
#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.

### 16.6.5.6 energy\_set

```
int energy_set [extern]
```

```
#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.

#### 16.6.5.7 do\_backtrack

```
int do_backtrack [extern]
```

```
#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.

#### 16.6.5.8 backtrack\_type

```
char backtrack_type [extern]
```

```
#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.

#### 16.6.5.9 nonstandards

```
char* nonstandards [extern]
```

```
#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.

#### 16.6.5.10 max\_bp\_span

```
int max_bp_span [extern]
```

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

Maximum allowed base pair span.

A value of -1 indicates no restriction for distant base pairs.

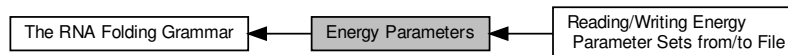
## 16.7 Energy Parameters

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).

### 16.7.1 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 files or strings.*

## 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

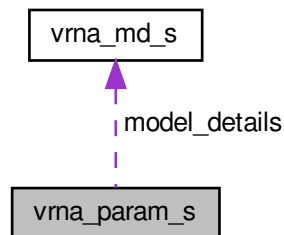
- [vrna\\_param\\_t](#) \* [vrna\\_params](#) ([vrna\\_md\\_t](#) \*md)  
*Get a data structure containing prescaled free energy parameters.*
- [vrna\\_param\\_t](#) \* [vrna\\_params\\_copy](#) ([vrna\\_param\\_t](#) \*par)  
*Get a copy of the provided free energy parameters.*
- [vrna\\_exp\\_param\\_t](#) \* [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.*

## 16.7.2 Data Structure Documentation

### 16.7.2.1 struct vrna\_param\_s

The datastructure that contains temperature scaled energy parameters.

Collaboration diagram for vrna\_param\_s:



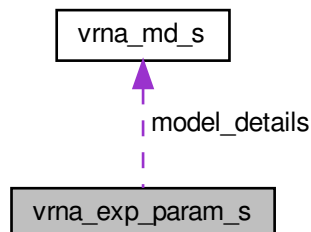
#### Data Fields

- 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.*

### 16.7.2.2 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.*

### 16.7.2.2.1 Field Documentation

**16.7.2.2.1.1** `id` `int vrna_exp_param_s::id`

An identifier for the data structure.

**Deprecated** This attribute will be removed in version 3

**16.7.2.2.1.2** `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  $e^{-E/(\alpha \cdot K \cdot T)}$

## 16.7.3 Typedef Documentation

### 16.7.3.1 `paramT`

```
typedef struct vrna_param_s paramT

#include <ViennaRNA/params/basic.h>
```

Old typename of `vrna_param_s`.

**Deprecated** Use `vrna_param_t` instead!

### 16.7.3.2 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!

## 16.7.4 Function Documentation

### 16.7.4.1 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

<code>md</code>	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

### 16.7.4.2 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

### 16.7.4.3 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

#### 16.7.4.4 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

#### 16.7.4.5 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

**16.7.4.6 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 overloaded **params\_subst()** method.

When no parameter is passed, the resulting action is the same as passing *NULL* as second parameter to [vrna\\_params\\_subst\(\)](#), i.e. resetting the parameters to the global defaults.

**16.7.4.7 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

**SWIG Wrapper Notes** This function is attached to `vrna_fc_s` objects as overloaded `exp_params_subst()` method. When no parameter is passed, the resulting action is the same as passing `NULL` as second parameter to `vrna_exp_params_subst()`, i.e. resetting the parameters to the global defaults.

**16.7.4.8 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*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  $s = \exp((sfact * MFE)/kT/length)$ , 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.

## 16.7.4.9 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.

#### 16.7.4.10 `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.

#### 16.7.4.11 `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

**16.7.4.12 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

**16.7.4.13 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

**16.7.4.14** `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!

**16.7.4.15** `get_boltzmann_factors_ali()`

```
vrna_exp_param_t* get_boltzmann_factors_ali (
    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!



#### 16.7.4.16 `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

#### 16.7.4.17 `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

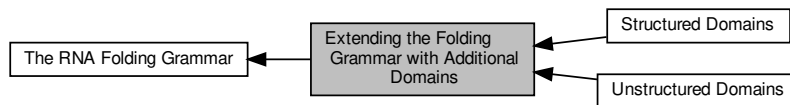
## 16.8 Extending the Folding Grammar with Additional Domains

This module covers simple and straight-forward extensions to the RNA folding grammar.

### 16.8.1 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.*

## 16.9 Unstructured Domains

Add and modify unstructured domains to the RNA folding grammar.

### 16.9.1 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.

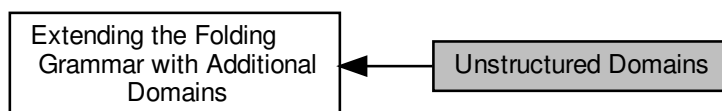
A default implementation allows one to readily use this feature by simply adding sequence motifs and corresponding binding free energies with the function `vrna_ud_add_motif()` (see also [Ligands Binding to Unstructured Domains](#)).

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 *RNAIib*, 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.*

## 16.9.2 Data Structure Documentation

### 16.9.2.1 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`

- `vrna_callback_ud_energy * energy_cb`  
Callback to ligand binding production rule, i.e. create/fill DP partition function matrices.
- `vrna_callback_ud_exp_energy * exp_energy_cb`  
Callback to evaluate free energy 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.

### 16.9.2.1.1 Field Documentation

**16.9.2.1.1.1 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.

## 16.9.3 Typedef Documentation

### 16.9.3.1 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

<code>vc</code>	The current <code>vrna_fold_compound_t</code>
<code>i</code>	The start of the unstructured domain (5' end)
<code>j</code>	The end of the unstructured domain (3' end)
<code>loop_type</code>	The loop context of the unstructured domain
<code>data</code>	Auxiliary data

**Returns**

The auxiliary energy contribution in deka-cal/mol

**16.9.3.2 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 <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 as Boltzmann factor

**16.9.3.3 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



### 16.9.3.4 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)

### 16.9.3.5 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

### 16.9.3.6 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

## 16.9.4 Function Documentation

### 16.9.4.1 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`.

16.9.4.2 `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`.

16.9.4.3 `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`

#### 16.9.4.4 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

**16.9.4.5 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*

**16.9.4.6 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

<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 $B$ production rule
<i>e_cb</i>	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`

16.9.4.8 `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  $[i, j]$ .

$$\begin{array}{c} \text{---} B \text{---} \\ i \qquad j \end{array} = \begin{array}{c} \text{---} B \text{---} \\ i \qquad j-1 \end{array} \bullet_j \mid \begin{array}{c} \text{---} B \text{---} \\ i \qquad u \end{array} \begin{array}{c} \boxed{\phantom{000}} \\ u+1 \qquad j \end{array}$$

$$f(i,j) = \begin{array}{c} \boxed{\phantom{000}} \\ i \qquad j \end{array} \mid \begin{array}{c} \text{---} B \text{---} \\ i \qquad j \end{array}$$

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`

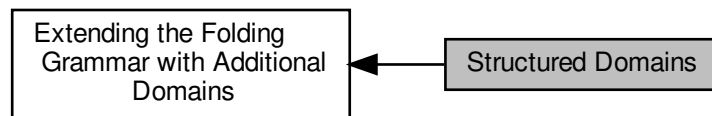
## 16.10 Structured Domains

Add and modify structured domains to the RNA folding grammar.

### 16.10.1 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.*



## 16.11 Constraining the RNA Folding Grammar

This module provides general functions that allow for an easy control of constrained secondary structure prediction and evaluation.

### 16.11.1 Detailed Description

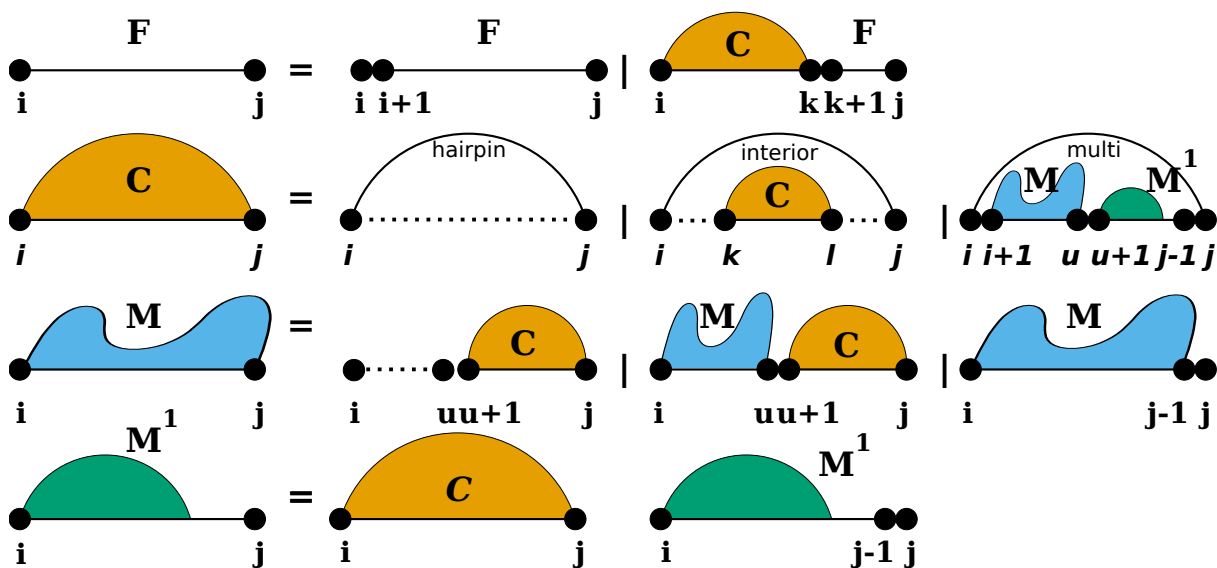
This module provides general functions that allow for an easy control of constrained secondary structure prediction and evaluation.

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.

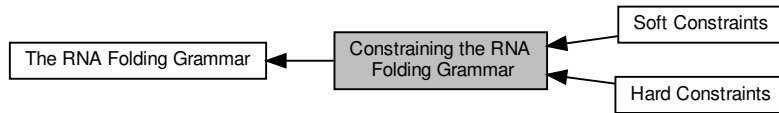
The callback will then always be notified about the type of decomposition that is happening, and the corresponding delimiting sequence positions. The following decomposition steps are distinguished, and should be captured by the user's implementation of the callback:

- [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
- [Incorporating Ligands Binding to Specific Sequence/Structure Motifs using Soft Constraints](#).

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)*

## 16.11.2 Macro Definition Documentation

### 16.11.2.1 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!

### 16.11.2.2 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!

### 16.11.2.3 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.

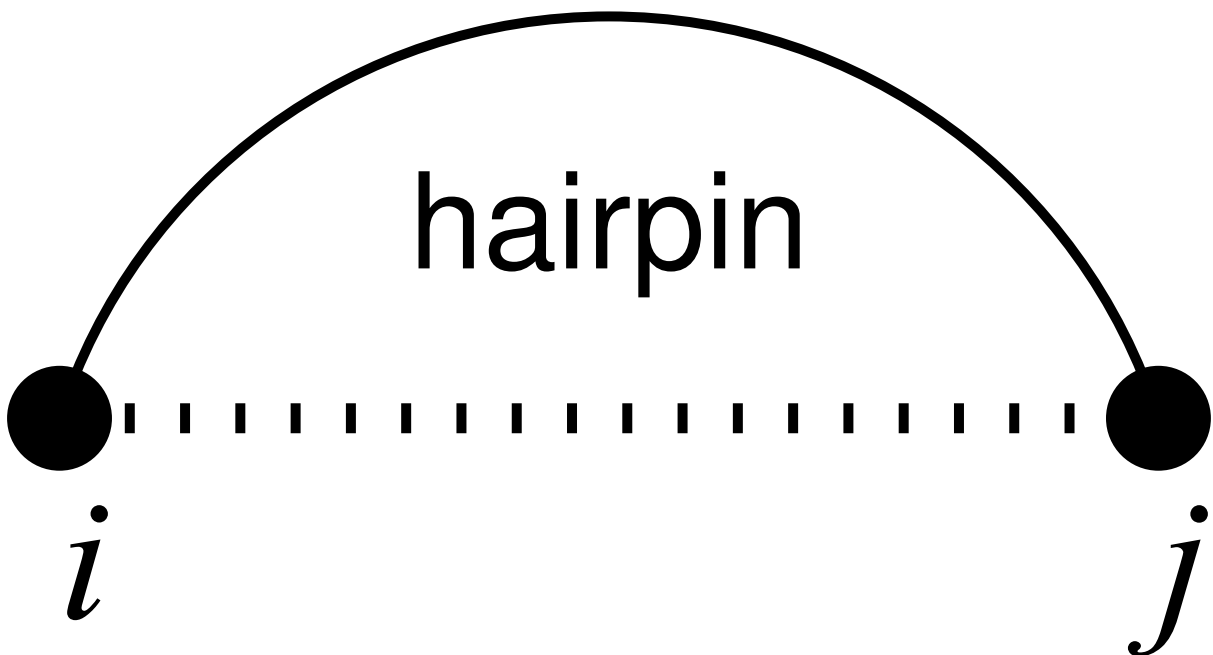
**Deprecated** Use `VRNA_OPTION_PF` instead!

### 16.11.2.4 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  $(i, j)$ .



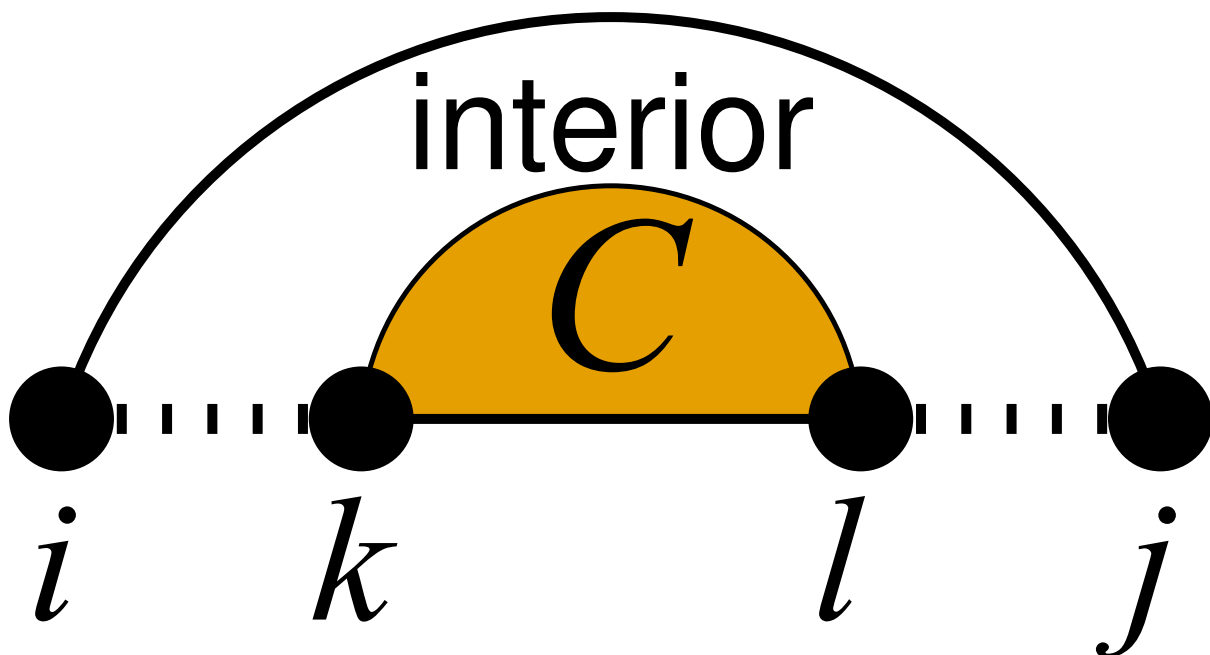
### 16.11.2.5 VRNA\_DECOMP\_PAIR\_IL

```
#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  $(k, l)$ .



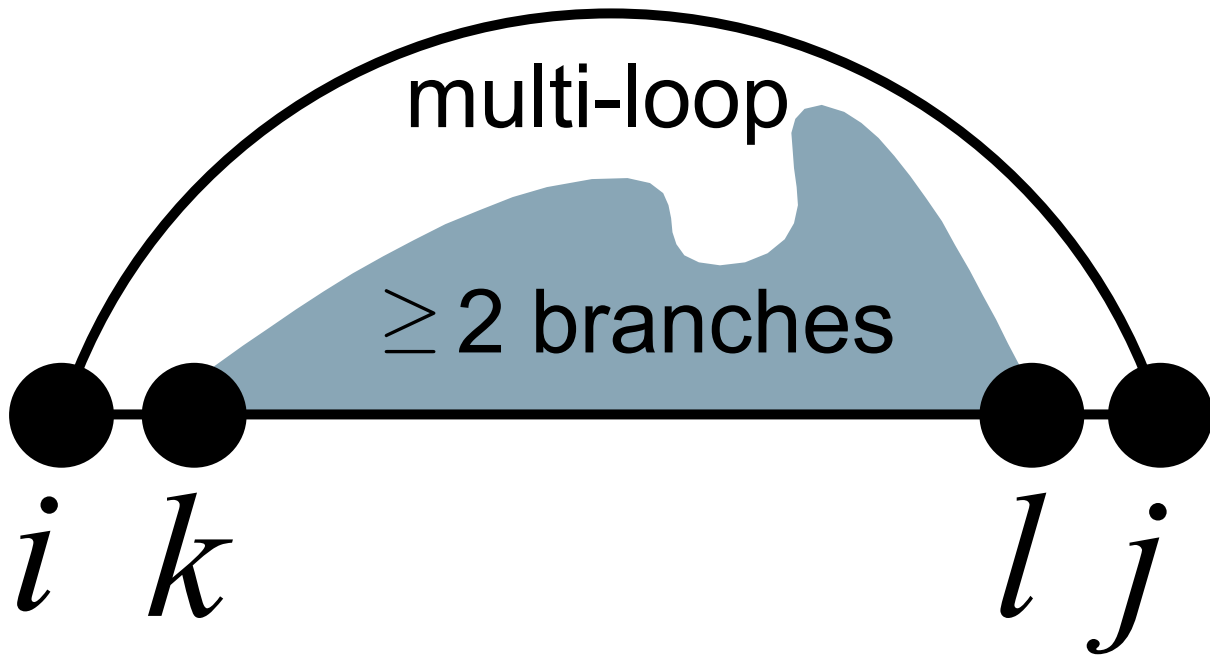
### 16.11.2.6 VRNA\_DECOMP\_PAIR\_ML

```
#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 multi-branch loop enclosed by the base pair  $(i, j)$ , and consisting of some enclosed multi loop content from  $k$  to  $l$ .



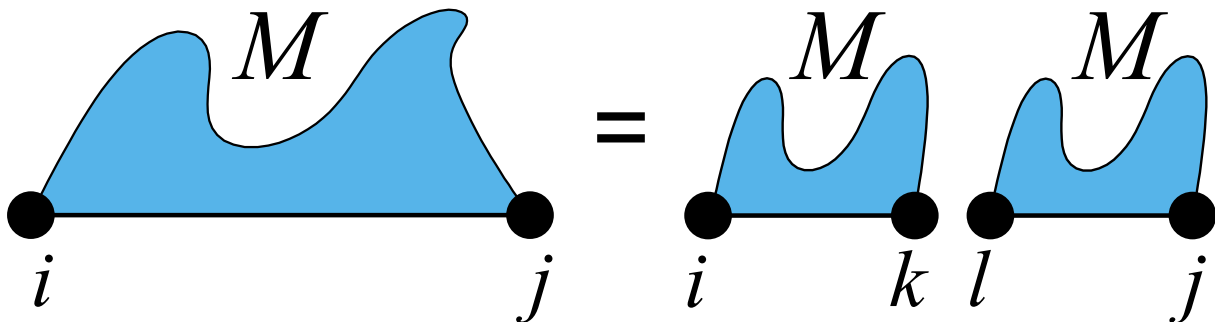
#### 16.11.2.7 VRNA\_DECOMP\_ML\_ML\_ML

```
#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  $[l : j]$ .



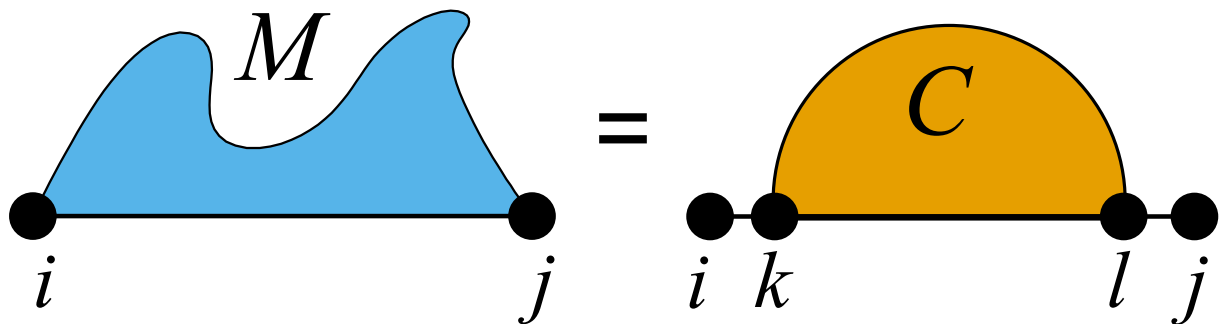
### 16.11.2.8 VRNA\_DECOMP\_ML\_STEM

```
#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  $(k, l)$ .



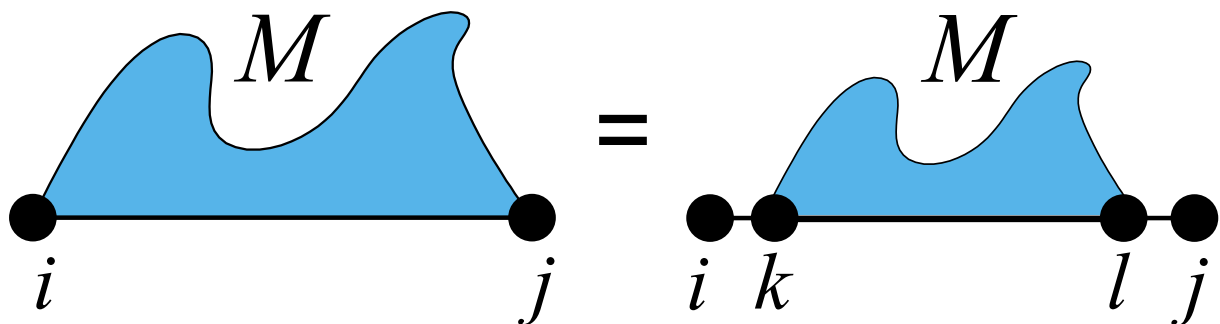
### 16.11.2.9 VRNA\_DECOMP\_ML\_ML

```
#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  $[k : l]$ .





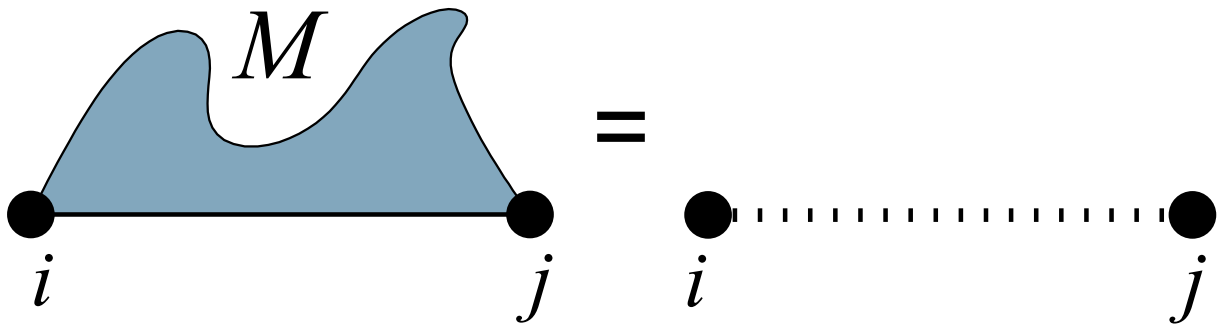
## 16.11.2.10 VRNA\_DECOMP\_ML\_UP

```
#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.



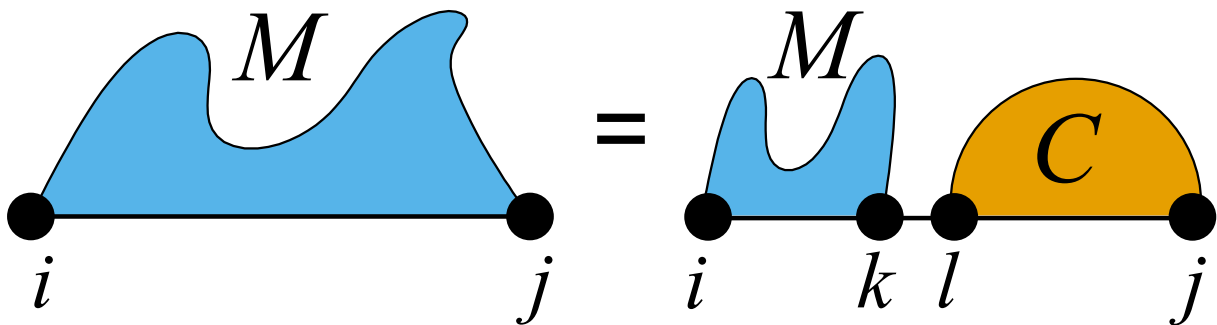
## 16.11.2.11 VRNA\_DECOMP\_ML\_ML\_STEM

```
#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  $(l, j)$ .



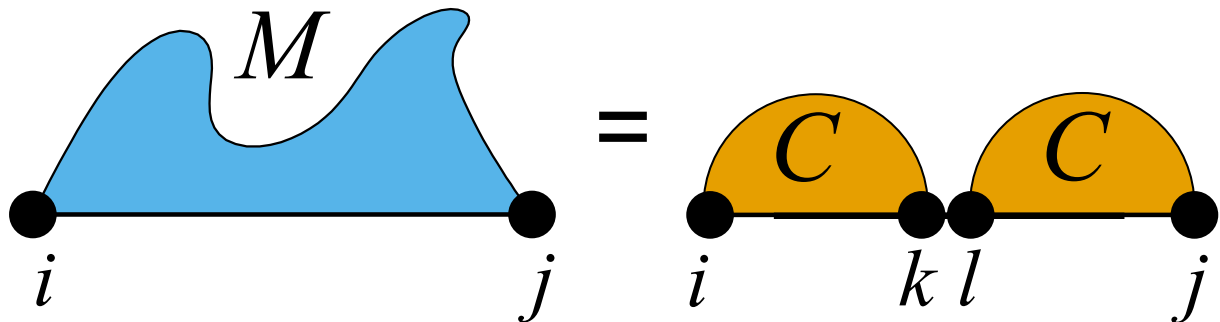
### 16.11.2.12 VRNA\_DECOMP\_ML\_COAXIAL

```
#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 multi-branch loop part in the interval  $[i : j]$ , where two stems with enclosing pairs  $(i, k)$  and  $(l, j)$  are coaxially stacking onto each other.



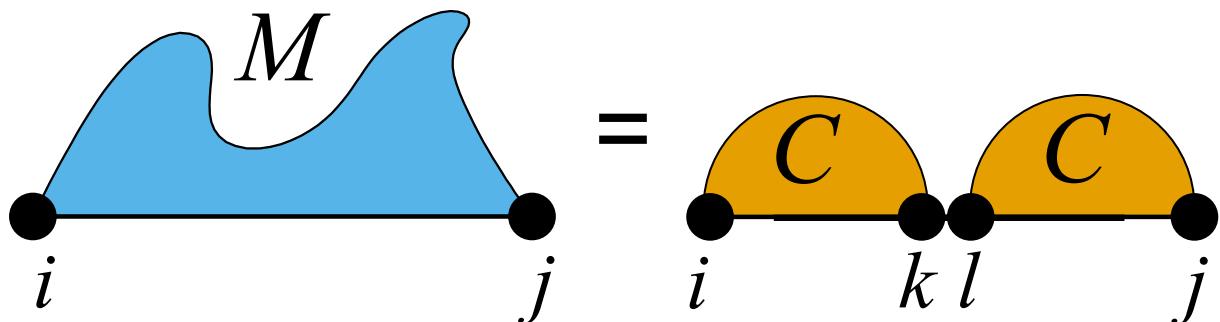
### 16.11.2.13 VRNA\_DECOMP\_ML\_COAXIAL\_ENC

```
#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 multi-branch loop part in the interval  $[i : j]$ , where two stems with enclosing pairs  $(i, k)$  and  $(l, j)$  are coaxially stacking onto each other.



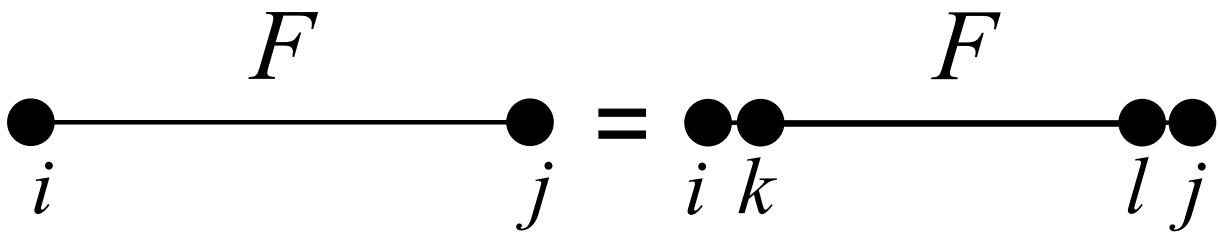
## 16.11.2.14 VRNA\_DECOMP\_EXT\_EXT

```
#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  $[k : l]$ .



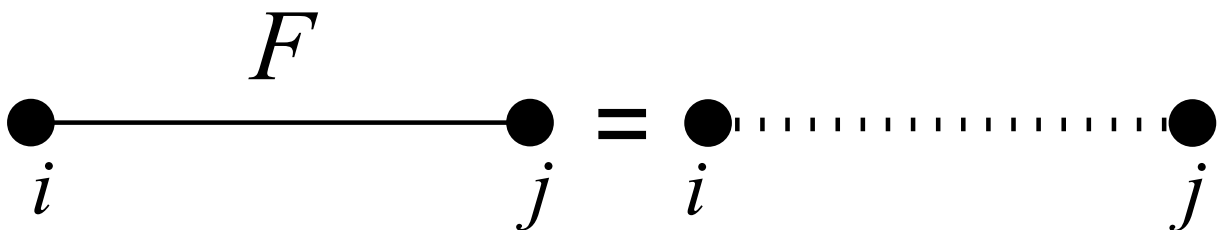
## 16.11.2.15 VRNA\_DECOMP\_EXT\_UP

```
#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.



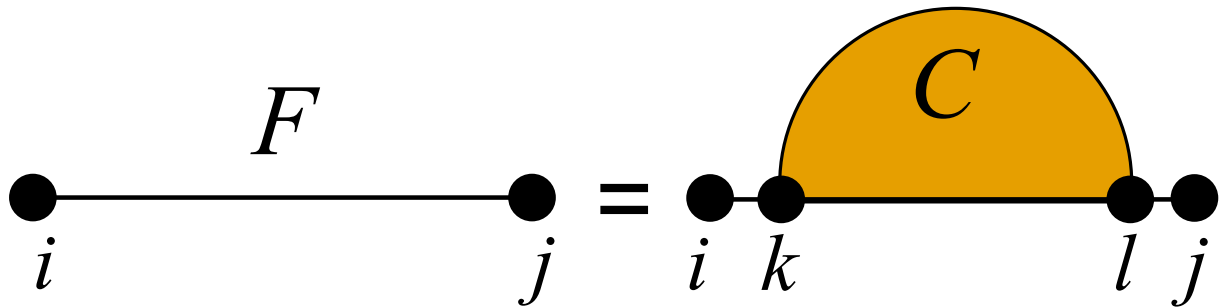
## 16.11.2.16 VRNA\_DECOMP\_EXT\_STEM

```
#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  $(k, l)$ .



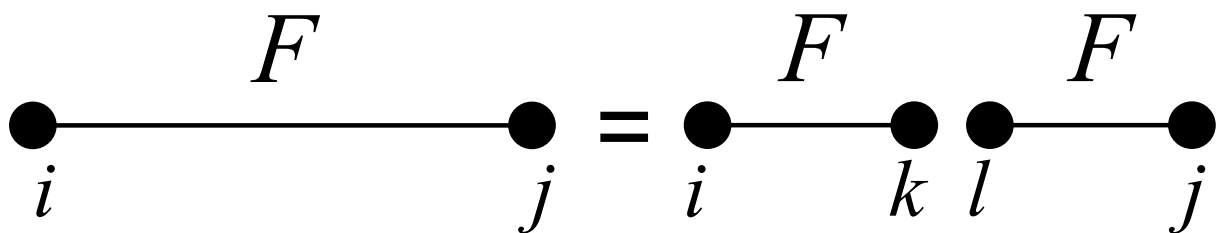
## 16.11.2.17 VRNA\_DECOMP\_EXT\_EXT\_EXT

```
#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  $[l : j]$ .



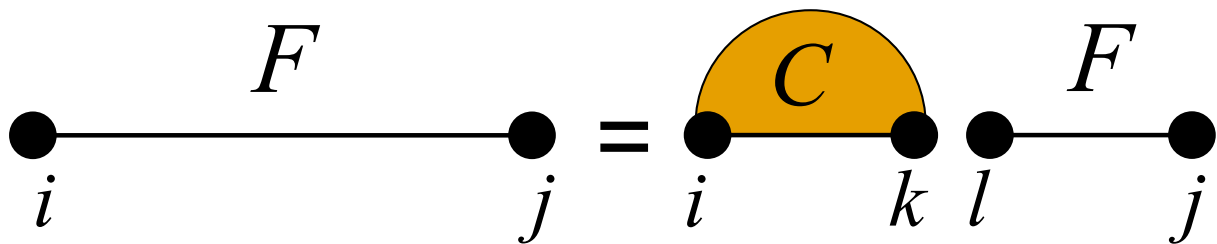
## 16.11.2.18 VRNA\_DECOMP\_EXT\_STEM\_EXT

```
#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  $[l : j]$ .



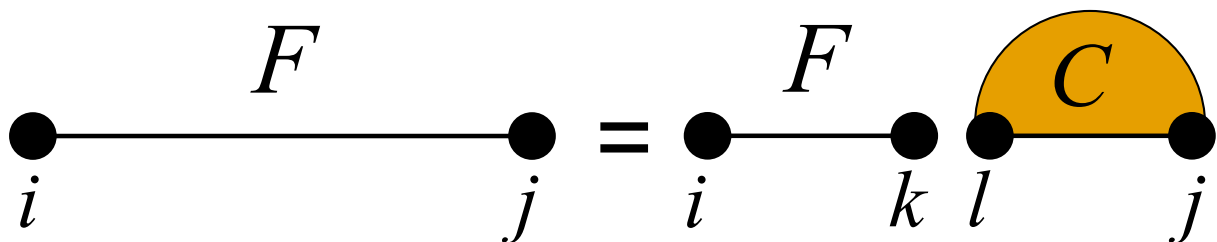
## 16.11.2.19 VRNA\_DECOMP\_EXT\_EXT\_STEM

```
#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  $(l, j)$ .



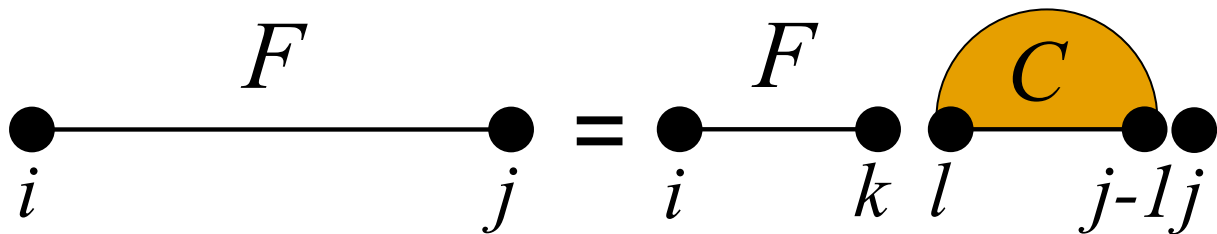
### 16.11.2.20 VRNA\_DECOMP\_EXT\_EXT\_STEM1

```
#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  $(l, j - 1)$ .



### 16.11.3 Function Documentation

#### 16.11.3.1 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 constrast 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

<code>vc</code>	The fold compound
<code>constraint</code>	A string with either the filename of the constraint definitions or a pseudo dot-bracket notation of the hard constraint. May be NULL.
<code>options</code>	The option flags

### 16.11.3.2 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
---------------	--

**16.11.3.3 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](#)

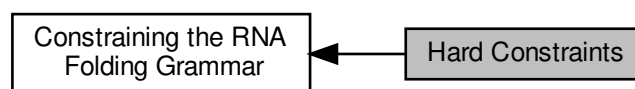
## 16.12 Hard Constraints

This module covers all functionality for hard constraints in secondary structure prediction.

### 16.12.1 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 'l' 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.*
- int `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.*

## 16.12.2 Data Structure Documentation

### 16.12.2.1 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](#)

### 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.*

#### 16.12.2.1.1 Field Documentation

#### 16.12.2.1.1 `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.

#### 16.12.2.2 `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*

### 16.12.3 Macro Definition Documentation

#### 16.12.3.1 `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()`

#### 16.12.3.2 `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()`

### 16.12.3.3 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\(\)](#)

### 16.12.3.4 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\(\)](#)

### 16.12.3.5 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\(\)](#)

### 16.12.3.6 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\(\)](#)

### 16.12.3.7 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\(\)](#)

### 16.12.3.8 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\(\)](#)

### 16.12.3.9 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.

### 16.12.3.10 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

### 16.12.3.11 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\(\)](#)



## 16.12.4 Typedef Documentation

### 16.12.4.1 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 *d* 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

## 16.12.5 Function Documentation

### 16.12.5.1 `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

<code>vc</code>	The fold compound
-----------------	-------------------

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

### 16.12.5.2 `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

<code>vc</code>	The <a href="#">vrna_fold_compound_t</a> the hard constraints are associated with
<code>i</code>	The position that needs to stay unpaired (1-based)
<code>option</code>	The options flag indicating how/where to store the hard constraints

### 16.12.5.3 `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

### 16.12.5.4 `vrna_hc_add_bp()`

```
int 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

### 16.12.5.5 `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

#### 16.12.5.6 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](#)

#### 16.12.5.7 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

<i>vc</i>	The fold compound
<i>constraint</i>	A pseudo dot-bracket notation of the hard constraint.
<i>options</i>	The option flags

**SWIG Wrapper Notes** This function is attached as method **hc\_add\_from\_db()** to objects of type *fold\_compound*

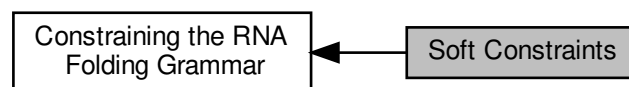
## 16.13 Soft Constraints

Functions and data structures for secondary structure soft constraints.

### 16.13.1 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

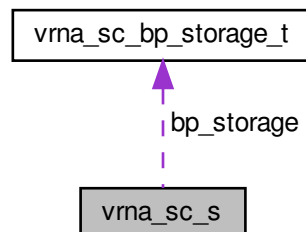
- void `vrna_sc_init` (`vrna_fold_compound_t` \*vc)  
*Initialize an empty soft constraints data structure within a `vrna_fold_compound_t`.*
- void `vrna_sc_set_bp` (`vrna_fold_compound_t` \*vc, const `FLT_OR_DBL` \*\*constraints, unsigned int options)  
*Set soft constraints for paired nucleotides.*
- 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)*

## 16.13.2 Data Structure Documentation

### 16.13.2.1 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)*

### 16.13.2.1.1 Field Documentation

#### 16.13.2.1.1.1 `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()`



#### 16.13.2.1.1.2 `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()`

#### 16.13.2.1.1.3 `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()`

### 16.13.3 Typedef Documentation

#### 16.13.3.1 `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

**16.13.3.2 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

**16.13.3.3 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

**16.13.4 Function Documentation****16.13.4.1 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_zarringhalam()`, `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`

### 16.13.4.2 `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*

#### 16.13.4.3 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.

#### 16.13.4.4 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`

**16.13.4.5 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 and may be omitted.

#### 16.13.4.6 `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`

#### 16.13.4.7 `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
-----------------	--

#### 16.13.4.8 `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*

#### 16.13.4.9 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*

#### 16.13.4.10 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*

#### 16.13.4.11 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

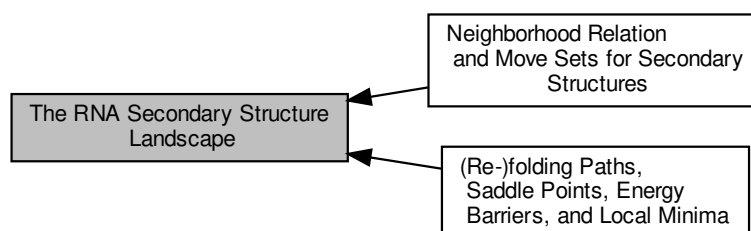
<code>vc</code>	The fold compound the generic soft constraint function should be bound to
<code>exp↔ _f</code>	A pointer to the function that evaluates the generic soft constraint feature

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

## 16.14 The RNA Secondary Structure Landscape

### 16.14.1 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.*
- [\(Re-\)folding Paths, Saddle Points, Energy Barriers, and Local Minima](#)  
*API for various RNA folding path algorithms.*

## 16.15 Minimum Free Energy (MFE) Algorithms

Predicting the Minimum Free Energy (MFE) and a corresponding (consensus) secondary structure.

### 16.15.1 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:

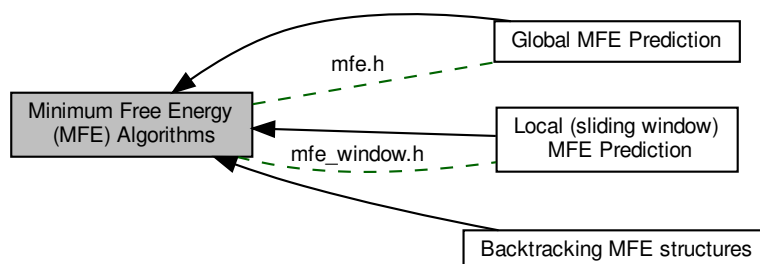
- [Global MFE Prediction](#) - to compute the MFE for the entire sequence
- [Local \(sliding window\) MFE Prediction](#) - to compute MFEs for each window using a sliding window approach

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.*

## 16.16 Partition Function and Equilibrium Properties

Compute the partition function to assess various equilibrium properties.

### 16.16.1 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:

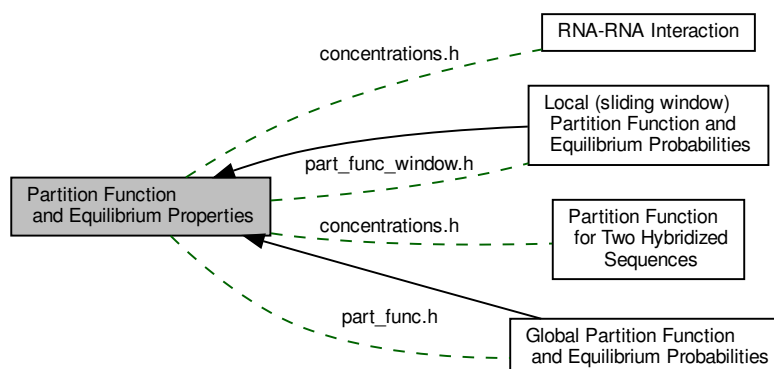
- [Global Partition Function and Equilibrium Probabilities](#) - to compute the partition function for a full length sequence
- [Local \(sliding window\) Partition Function and Equilibrium Probabilities](#) - to compute the partition function of each window using a sliding window approach

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 [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.*

### 16.16.2 Function Documentation

#### 16.16.2.1 [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

## 16.17 Global MFE Prediction

Variations of the global Minimum Free Energy (MFE) prediction algorithm.

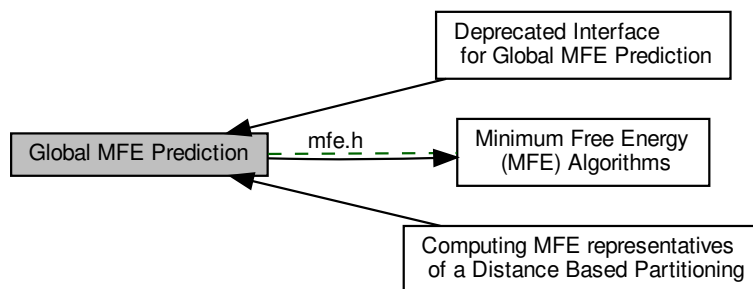
### 16.17.1 Detailed Description

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.*

## 16.17.2 Function Documentation

### 16.17.2.1 `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 NULL)

## 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*

**16.17.2.2 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*

**16.17.2.3 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

**16.17.2.4 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

**16.17.2.5 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

### 16.17.2.6 `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

### 16.17.2.7 `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

## 16.18 Local (sliding window) MFE Prediction

Variations of the local (sliding window) Minimum Free Energy (MFE) prediction algorithm.

### 16.18.1 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

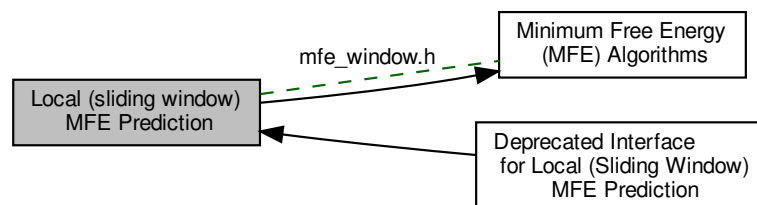
- Single sequences,
- Multiple sequence alignments (MSA), and

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\\_aliLfold](#) (const char \*\*alignment, int maxdist, FILE \*fp)
- float [vrna\\_aliLfold\\_cb](#) (const char \*\*alignment, int maxdist, [vrna\\_mfe\\_window\\_callback](#) \*cb, void \*data)

## 16.18.2 Typedef Documentation

### 16.18.2.1 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

## 16.18.3 Function Documentation

16.18.3.1 `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 <code>vrna_fold_compound_t</code> 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`

### 16.18.3.2 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)

### 16.18.3.3 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)

**16.18.3.4 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)

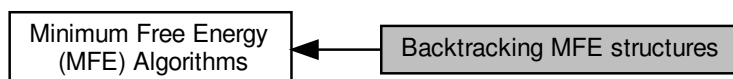
## 16.19 Backtracking MFE structures

Backtracking related interfaces.

### 16.19.1 Detailed Description

Backtracking related interfaces.

Collaboration diagram for Backtracking MFE structures:



## Functions

- float [vrna\\_backtrack5](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, unsigned int length, char \*structure)  
*Backtrack an MFE (sub)structure.*
- 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  $(i, j)$ .*

### 16.19.2 Function Documentation

### 16.19.2.1 vrna\_backtrack5()

```
float vrna_backtrack5 (
    vrna_fold_compound_t * fc,
    unsigned int length,
    char * structure )
```

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

Backtrack an MFE (sub)structure.

This function allows one to backtrack the MFE structure for a (sub)sequence

#### Note

On error, the function returns `INF / 100.` and stores the empty string in `structure`.

#### Precondition

Requires pre-filled MFE dynamic programming matrices, i.e. one has to call `vrna_mfe()` prior to calling this function

#### See also

`vrna_mfe()`, `vrna_pbacktrack5()`

#### Parameters

<i>fc</i>	fold compound
<i>length</i>	The length of the subsequence, starting from the 5' end
<i>structure</i>	A pointer to the character array where the secondary structure in dot-bracket notation will be written to. (Must have size of at least $\$length + 1$ )

#### Returns

The minimum free energy (MFE) for the specified `length` in kcal/mol and a corresponding secondary structure in dot-bracket notation (stored in `structure`)

**SWIG Wrapper Notes** This function is attached as overloaded method **backtrack()** to objects of type `fold_compound` with default parameter `length` equal to the total length of the RNA.

### 16.19.2.2 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`

### 16.19.2.3 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.

## 16.20 Global Partition Function and Equilibrium Probabilities

Variations of the global partition function algorithm.

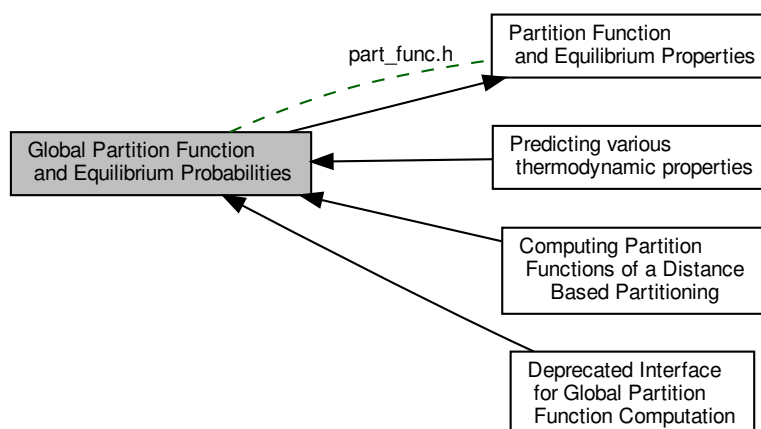
### 16.20.1 Detailed Description

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.*
- [Predicting various thermodynamic properties](#)  
*Compute various thermodynamic properties using the partition function.*
- [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

- [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.

## 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.

## 16.20.2 Data Structure Documentation

### 16.20.2.1 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

## 16.20.3 Function Documentation

### 16.20.3.1 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

<i>in, out</i>	<i>vc</i>	The fold compound data structure
<i>in, out</i>	<i>structure</i>	A pointer to the character array where position-wise pairing propensity will be stored. (Maybe NULL)

#### Returns

The ensemble free energy  $G = -RT \cdot \log(Q)$  in kcal/mol

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

### 16.20.3.2 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, FOAB 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

<i>vc</i>	the fold compound data structure
<i>structure</i>	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*

### 16.20.3.3 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 ensemble free energy  $G = -RT \cdot \log(Q)$  in kcal/mol

16.20.3.4 [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 ensemble free energy  $G = -RT \cdot \log(Q)$  in kcal/mol

**16.20.3.5 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\\_circalifold\(\)](#), [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 ensemble free energy  $G = -RT \cdot \log(Q)$  in kcal/mol

**16.20.3.6 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 <code>vrna_ep_t</code> to store pairing probabilities (Maybe NULL)

#### Returns

The ensemble free energy  $G = -RT \cdot \log(Q)$  in kcal/mol

### 16.20.3.7 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

16.20.3.8 `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 <code>vrna_ep_t</code> to store pairing probabilities (Maybe NULL)

## Returns

`vrna_dimer_pf_t` structure containing a set of energies needed for concentration computations.

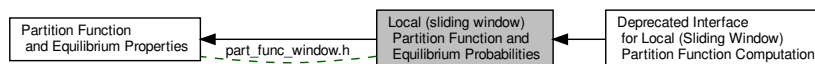
## 16.21 Local (sliding window) Partition Function and Equilibrium Probabilities

Scanning version using a sliding window approach to compute equilibrium probabilities.

### 16.21.1 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.*

## 16.21.2 Macro Definition Documentation

### 16.21.2.1 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\(\)](#)

### 16.21.2.2 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\(\)](#)

### 16.21.2.3 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\(\)](#)

### 16.21.2.4 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](#)

### 16.21.2.5 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\(\)](#)

## 16.21.3 Typedef Documentation

### 16.21.3.1 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

## 16.21.4 Function Documentation

16.21.4.1 `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

**16.21.4.2 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

**16.21.4.3 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 <code>cb</code>

**Returns**

0 on failure, non-zero on success

**16.21.4.4 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`

#### 16.21.4.5 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

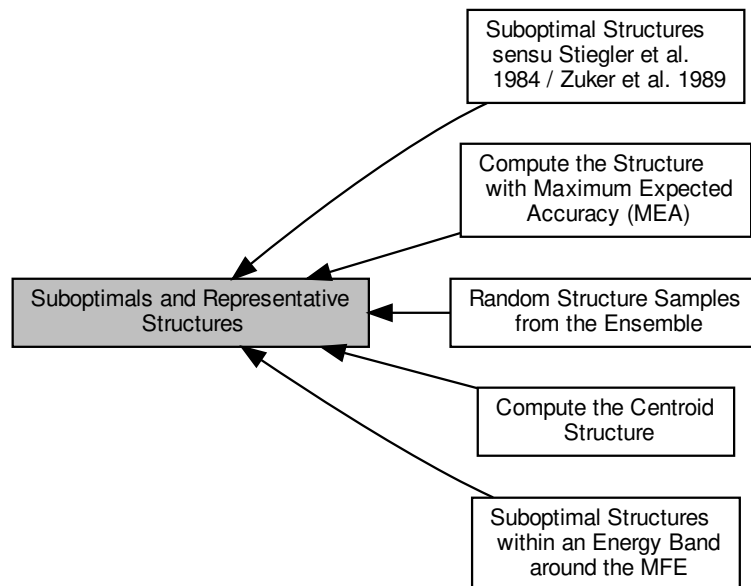
## 16.22 Suboptimals and Representative Structures

Sample and enumerate suboptimal secondary structures from RNA sequence data.

### 16.22.1 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](#)

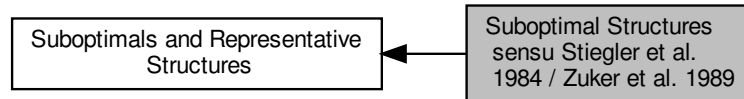
### Files

- 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.*

## 16.23 Suboptimal Structures sensu Stiegler et al. 1984 / Zuker et al. 1989

### 16.23.1 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.*

### 16.23.2 Function Documentation

#### 16.23.2.1 `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*

**16.23.2.2 zokersubopt()**

```
SOLUTION* zokersubopt (
    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_zokersubopt()` instead

## Parameters

<i>string</i>	RNA sequence
---------------	--------------

## Returns

List of Zuker suboptimal structures

**16.23.2.3 zokersubopt\_par()**

```
SOLUTION* zokersubopt_par (
    const char * string,
    vrna_param_t * parameters )

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

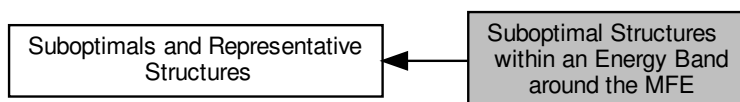
Compute Zuker type suboptimal structures.

**Deprecated** use `vrna_zokersubopt()` instead

## 16.24 Suboptimal Structures within an Energy Band around the MFE

### 16.24.1 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.*

### 16.24.2 Typedef Documentation

### 16.24.2.1 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>

## 16.24.3 Function Documentation

### 16.24.3.1 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*

16.24.3.2 **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("GGGGGGAAAAACCCCC", &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*

### 16.24.3.3 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

### 16.24.3.4 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

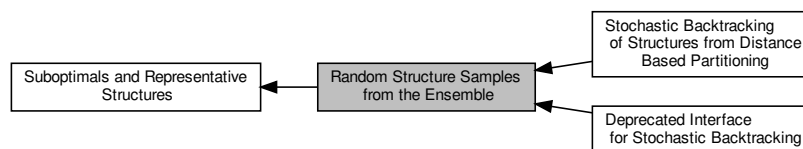
## 16.25 Random Structure Samples from the Ensemble

Functions to draw random structure samples from the ensemble according to their equilibrium probability.

### 16.25.1 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.*
- [Deprecated Interface for Stochastic Backtracking](#)

### Macros

- `#define VRNA_PBACKTRACK_DEFAULT 0`  
*Boltzmann sampling flag indicating default backtracing mode.*
- `#define VRNA_PBACKTRACK_NON_REDUNDANT 1`  
*Boltzmann sampling flag indicating non-redundant backtracing mode.*

### Typedefs

- `typedef void() vrna_boltzmann_sampling_callback(const char *structure, void *data)`  
*Callback for Boltzmann sampling.*
- `typedef struct vrna_pbacktrack_memory_s * vrna_pbacktrack_mem_t`  
*Boltzmann sampling memory data structure.*

## Functions

- `char * vrna_pbacktrack5 (vrna_fold_compound_t *fc, unsigned int length)`  
*Sample a secondary structure of a subsequence from the Boltzmann ensemble according its probability.*
- `char ** vrna_pbacktrack5_num (vrna_fold_compound_t *fc, unsigned int num_samples, unsigned int length, unsigned int options)`  
*Obtain a set of secondary structure samples for a subsequence from the Boltzmann ensemble according their probability.*
- `unsigned int vrna_pbacktrack5_cb (vrna_fold_compound_t *fc, unsigned int num_samples, unsigned int length, vrna_boltzmann_sampling_callback *cb, void *data, unsigned int options)`  
*Obtain a set of secondary structure samples for a subsequence from the Boltzmann ensemble according their probability.*
- `char ** vrna_pbacktrack5_resume (vrna_fold_compound_t *vc, unsigned int num_samples, unsigned int length, vrna_pbacktrack_mem_t *nr_mem, unsigned int options)`  
*Obtain a set of secondary structure samples for a subsequence from the Boltzmann ensemble according their probability.*
- `unsigned int vrna_pbacktrack5_resume_cb (vrna_fold_compound_t *fc, unsigned int num_samples, unsigned int length, vrna_boltzmann_sampling_callback *cb, void *data, vrna_pbacktrack_mem_t *nr_mem, unsigned int options)`  
*Obtain a set of secondary structure samples for a subsequence from the Boltzmann ensemble according their probability.*
- `char * vrna_pbacktrack (vrna_fold_compound_t *fc)`  
*Sample a secondary structure from the Boltzmann ensemble according its probability.*
- `char ** vrna_pbacktrack_num (vrna_fold_compound_t *fc, unsigned int num_samples, unsigned int options)`  
*Obtain a set of secondary structure samples from the Boltzmann ensemble according their probability.*
- `unsigned int vrna_pbacktrack_cb (vrna_fold_compound_t *fc, unsigned int num_samples, vrna_boltzmann_sampling_callback *cb, void *data, unsigned int options)`  
*Obtain a set of secondary structure samples from the Boltzmann ensemble according their probability.*
- `char ** vrna_pbacktrack_resume (vrna_fold_compound_t *fc, unsigned int num_samples, vrna_pbacktrack_mem_t *nr_mem, unsigned int options)`  
*Obtain a set of secondary structure samples from the Boltzmann ensemble according their probability.*
- `unsigned int vrna_pbacktrack_resume_cb (vrna_fold_compound_t *fc, unsigned int num_samples, vrna_boltzmann_sampling_callback *cb, void *data, vrna_pbacktrack_mem_t *nr_mem, unsigned int options)`  
*Obtain a set of secondary structure samples from the Boltzmann ensemble according their probability.*
- `void vrna_pbacktrack_mem_free (vrna_pbacktrack_mem_t s)`  
*Release memory occupied by a Boltzmann sampling memory data structure.*

## 16.25.2 Macro Definition Documentation

### 16.25.2.1 VRNA\_PBACKTRACK\_DEFAULT

```
#define VRNA_PBACKTRACK_DEFAULT 0
#include <ViennaRNA/boltzmann_sampling.h>
```

Boltzmann sampling flag indicating default backtracing mode.

See also

`vrna_pbacktrack5_num()`, `vrna_pbacktrack5_cb()`, `vrna_pbacktrack5_resume()`, `vrna_pbacktrack5_resume_cb()`, `vrna_pbacktrack_num()`, `vrna_pbacktrack_cb()`, `vrna_pbacktrack_resume()`, `vrna_pbacktrack_resume_cb()`



### 16.25.2.2 VRNA\_PBACKTRACK\_NON\_REDUNDANT

```
#define VRNA_PBACKTRACK_NON_REDUNDANT 1

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

Boltzmann sampling flag indicating non-redundant backtracing mode.

This flag will turn the Boltzmann sampling into non-redundant backtracing mode along the lines of Michalik et al. 2017 [18]

See also

[vrna\\_pbacktrack5\\_num\(\)](#), [vrna\\_pbacktrack5\\_cb\(\)](#), [vrna\\_pbacktrack5\\_resume\(\)](#), [vrna\\_pbacktrack5\\_resume\\_cb\(\)](#), [vrna\\_pbacktrack\\_num\(\)](#), [vrna\\_pbacktrack\\_cb\(\)](#), [vrna\\_pbacktrack\\_resume\(\)](#), [vrna\\_pbacktrack\\_resume\\_cb\(\)](#)

## 16.25.3 Typedef Documentation

### 16.25.3.1 vrna\_boltzmann\_sampling\_callback

```
typedef void() vrna_boltzmann_sampling_callback(const char *structure, void *data)

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

Callback for Boltzmann sampling.

**Notes on Callback Functions** This function will be called for each secondary structure that has been successfully backtraced from the partition function DP matrices.

See also

[vrna\\_pbacktrack5\\_cb\(\)](#), [vrna\\_pbacktrack\\_cb\(\)](#), [vrna\\_pbacktrack5\\_resume\\_cb\(\)](#), [vrna\\_pbacktrack\\_resume\\_cb\(\)](#)

Parameters

<i>structure</i>	The secondary structure in dot-bracket notation
<i>data</i>	Some arbitrary, auxiliary data address as provided to the calling function

### 16.25.3.2 vrna\_pbacktrack\_mem\_t

```
typedef struct vrna_pbacktrack_memory_s* vrna_pbacktrack_mem_t

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

Boltzmann sampling memory data structure.

This structure is required for properly resuming a previous sampling round in specialized Boltzmann sampling, such as non-redundant backtracking.

Initialize with `NULL` and pass its address to the corresponding functions [vrna\\_pbacktrack5\\_resume\(\)](#), etc.

#### Note

Do not forget to release memory occupied by this data structure before losing its context! Use [vrna\\_pbacktrack\\_mem\\_free\(\)](#).

#### See also

[vrna\\_pbacktrack5\\_resume\(\)](#), [vrna\\_pbacktrack\\_resume\(\)](#), [vrna\\_pbacktrack5\\_resume\\_cb\(\)](#), [vrna\\_pbacktrack\\_resume\\_cb\(\)](#), [vrna\\_pbacktrack\\_mem\\_free\(\)](#)

## 16.25.4 Function Documentation

### 16.25.4.1 vrna\_pbacktrack5()

```
char * vrna_pbacktrack5 (
    vrna_fold_compound_t * fc,
    unsigned int length )

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

Sample a secondary structure of a subsequence from the Boltzmann ensemble according to its probability.

Perform a probabilistic (stochastic) backtracing in the partition function DP arrays to obtain a secondary structure. The parameter `length` specifies the length of the substructure starting from the 5' end.

The structure  $s$  with free energy  $E(s)$  is picked from the Boltzmann distributed ensemble according to its probability

$$p(s) = \frac{\exp(-E(s)/kT)}{Z}$$

with partition function  $Z = \sum_s \exp(-E(s)/kT)$ , Boltzmann constant  $k$  and thermodynamic temperature  $T$ .

#### Precondition

Unique multiloop decomposition has to be active upon creation of `fc` 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`.

#### See also

[vrna\\_pbacktrack5\\_num\(\)](#), [vrna\\_pbacktrack5\\_cb\(\)](#), [vrna\\_pbacktrack\(\)](#)

## Parameters

<i>fc</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 `pbacktrack5()` to objects of type `fold_compound`. See also [Python Examples - Boltzmann Sampling](#)

16.25.4.2 `vrna_pbacktrack5_num()`

```
char ** vrna_pbacktrack5_num (
    vrna_fold_compound_t * fc,
    unsigned int num_samples,
    unsigned int length,
    unsigned int options )

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

Obtain a set of secondary structure samples for a subsequence from the Boltzmann ensemble according to their probability.

Perform a probabilistic (stochastic) backtracing in the partition function DP arrays to obtain a set of `num_samples` secondary structures. The parameter `length` specifies the length of the substructure starting from the 5' end.

Any structure  $s$  with free energy  $E(s)$  is picked from the Boltzmann distributed ensemble according to its probability

$$p(s) = \frac{\exp(-E(s)/kT)}{Z}$$

with partition function  $Z = \sum_s \exp(-E(s)/kT)$ , Boltzmann constant  $k$  and thermodynamic temperature  $T$ .

Using the `options` flag one can switch between regular (`VRNA_PBACKTRACK_DEFAULT`) backtracing mode, and non-redundant sampling (`VRNA_PBACKTRACK_NON_REDUNDANT`) along the lines of Michalik et al. 2017 [18].

## Precondition

Unique multiloop decomposition has to be active upon creation of `fc` 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`.

## Warning

In non-redundant sampling mode (`VRNA_PBACKTRACK_NON_REDUNDANT`), this function may not yield the full number of requested samples. This may happen if a) the number of requested structures is larger than the total number of structures in the ensemble, b) numeric instabilities prevent the backtracking function to enumerate structures with high free energies, or c) any other error occurs.

## See also

`vrna_pbacktrack5()`, `vrna_pbacktrack5_cb()`, `vrna_pbacktrack_num()`, `VRNA_PBACKTRACK_DEFAULT`, `VRNA_PBACKTRACK_NON_REDUNDANT`

## Parameters

<i>fc</i>	The fold compound data structure
<i>num_samples</i>	The size of the sample set, i.e. number of structures
<i>length</i>	The length of the subsequence to consider (starting with 5' end)
<i>options</i>	A bitwise OR-flag indicating the backtracing mode.

## Returns

A set of secondary structure samples in dot-bracket notation terminated by NULL (or NULL on error)

**SWIG Wrapper Notes** This function is attached as overloaded method [pbacktrack5\(\)](#) to objects of type *fold\_compound* where the last argument *options* is optional with default value `options = VRNA_PBACKTRACK_DEFAULT`. See also [Python Examples - Boltzmann Sampling](#)

16.25.4.3 `vrna_pbacktrack5_cb()`

```
unsigned int vrna_pbacktrack5_cb (
    vrna_fold_compound_t * fc,
    unsigned int num_samples,
    unsigned int length,
    vrna_boltzmann_sampling_callback * cb,
    void * data,
    unsigned int options )
```

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

Obtain a set of secondary structure samples for a subsequence from the Boltzmann ensemble according to their probability.

Perform a probabilistic (stochastic) backtracing in the partition function DP arrays to obtain a set of `num_samples` secondary structures. The parameter `length` specifies the length of the substructure starting from the 5' end.

Any structure  $s$  with free energy  $E(s)$  is picked from the Boltzmann distributed ensemble according to its probability

$$p(s) = \frac{\exp(-E(s)/kT)}{Z}$$

with partition function  $Z = \sum_s \exp(-E(s)/kT)$ , Boltzmann constant  $k$  and thermodynamic temperature  $T$ .

Using the `options` flag one can switch between regular (`VRNA_PBACKTRACK_DEFAULT`) backtracing mode, and non-redundant sampling (`VRNA_PBACKTRACK_NON_REDUNDANT`) along the lines of Michalik et al. 2017 [18].

In contrast to `vrna_pbacktrack5()` and `vrna_pbacktrack5_num()` this function yields the structure samples through a callback mechanism.

## Precondition

Unique multiloop decomposition has to be active upon creation of `fc` 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`.

**Warning**

In non-redundant sampling mode (`VRNA_PBACKTRACK_NON_REDUNDANT`), this function may not yield the full number of requested samples. This may happen if a) the number of requested structures is larger than the total number of structures in the ensemble, b) numeric instabilities prevent the backtracking function to enumerate structures with high free energies, or c) any other error occurs.

**See also**

`vrna_pbacktrack5()`, `vrna_pbacktrack5_num()`, `vrna_pbacktrack_cb()`, `VRNA_PBACKTRACK_DEFAULT`, `VRNA_PBACKTRACK_NON_REDUNDANT`

**Parameters**

<i>fc</i>	The fold compound data structure
<i>num_samples</i>	The size of the sample set, i.e. number of structures
<i>length</i>	The length of the subsequence to consider (starting with 5' end)
<i>cb</i>	The callback that receives the sampled structure
<i>data</i>	A data structure passed through to the callback <i>cb</i>
<i>options</i>	A bitwise OR-flag indicating the backtracing mode.

**Returns**

The number of structures actually backtraced

**SWIG Wrapper Notes** This function is attached as overloaded method `pbacktrack5()` to objects of type `fold_compound` where the last argument `options` is optional with default value `options = VRNA_PBACKTRACK_DEFAULT`. See also [Python Examples - Boltzmann Sampling](#)

**16.25.4.4 vrna\_pbacktrack5\_resume()**

```
char ** vrna_pbacktrack5_resume (
    vrna_fold_compound_t * fc,
    unsigned int num_samples,
    unsigned int length,
    vrna_pbacktrack_mem_t * nr_mem,
    unsigned int options )

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

Obtain a set of secondary structure samples for a subsequence from the Boltzmann ensemble according their probability.

Perform a probabilistic (stochastic) backtracing in the partition function DP arrays to obtain a set of `num_samples` secondary structures. The parameter `length` specifies the length of the substructure starting from the 5' end.

Any structure  $s$  with free energy  $E(s)$  is picked from the Boltzmann distributed ensemble according to its probability

$$p(s) = \frac{\exp(-E(s)/kT)}{Z}$$

with partition function  $Z = \sum_s \exp(-E(s)/kT)$ , Boltzmann constant  $k$  and thermodynamic temperature  $T$ .

Using the `options` flag one can switch between regular (`VRNA_PBACKTRACK_DEFAULT`) backtracing mode, and non-redundant sampling (`VRNA_PBACKTRACK_NON_REDUNDANT`) along the lines of Michalik et al. 2017 [18].

In contrast to `vrna_pbacktrack5_cb()` this function allows for resuming a previous sampling round in specialized Boltzmann sampling, such as non-redundant backtracking. For that purpose, the user passes the address of a Boltzmann sampling data structure (`vrna_pbacktrack_mem_t`) which will be re-used in each round of sampling, i.e. each successive call to `vrna_pbacktrack5_resume_cb()` or `vrna_pbacktrack5_resume()`.

A successive sample call to this function may look like:

```
vrna_pbacktrack_mem_t nonredundant_memory = NULL;
// sample the first 100 structures
vrna_pbacktrack5_resume(fc,
    100,
    fc->length,
    &nonredundant_memory,
    options);
// sample another 500 structures
vrna_pbacktrack5_resume(fc,
    500,
    fc->length,
    &nonredundant_memory,
    options);
// release memory occupied by the non-redundant memory data structure
vrna_pbacktrack_mem_free(nonredundant_memory);
```

### Precondition

Unique multiloop decomposition has to be active upon creation of `fc` 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`.

### Warning

In non-redundant sampling mode (`VRNA_PBACKTRACK_NON_REDUNDANT`), this function may not yield the full number of requested samples. This may happen if a) the number of requested structures is larger than the total number of structures in the ensemble, b) numeric instabilities prevent the backtracking function to enumerate structures with high free energies, or c) any other error occurs.

### See also

`vrna_pbacktrack5_resume_cb()`, `vrna_pbacktrack5_cb()`, `vrna_pbacktrack_resume()`, `vrna_pbacktrack_mem_t`, `VRNA_PBACKTRACK_DEFAULT`, `VRNA_PBACKTRACK_NON_REDUNDANT`, `vrna_pbacktrack_mem_free`

## Parameters

<i>fc</i>	The fold compound data structure
<i>num_samples</i>	The size of the sample set, i.e. number of structures
<i>length</i>	The length of the subsequence to consider (starting with 5' end)
<i>nr_mem</i>	The address of the Boltzmann sampling memory data structure
<i>options</i>	A bitwise OR-flag indicating the backtracing mode.

## Returns

A set of secondary structure samples in dot-bracket notation terminated by NULL (or NULL on error)

**SWIG Wrapper Notes** This function is attached as overloaded method `pbacktrack5()` to objects of type `fold_compound`. In addition to the list of structures, this function also returns the `nr_mem` data structure as first element. See also [Python Examples - Boltzmann Sampling](#)

16.25.4.5 `vrna_pbacktrack5_resume_cb()`

```
unsigned int vrna_pbacktrack5_resume_cb (
    vrna_fold_compound_t * fc,
    unsigned int num_samples,
    unsigned int length,
    vrna_boltzmann_sampling_callback * cb,
    void * data,
    vrna_pbacktrack_mem_t * nr_mem,
    unsigned int options )
```

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

Obtain a set of secondary structure samples for a subsequence from the Boltzmann ensemble according to their probability.

Perform a probabilistic (stochastic) backtracing in the partition function DP arrays to obtain a set of `num_samples` secondary structures. The parameter `length` specifies the length of the substructure starting from the 5' end.

Any structure  $s$  with free energy  $E(s)$  is picked from the Boltzmann distributed ensemble according to its probability

$$p(s) = \frac{\exp(-E(s)/kT)}{Z}$$

with partition function  $Z = \sum_s \exp(-E(s)/kT)$ , Boltzmann constant  $k$  and thermodynamic temperature  $T$ .

Using the `options` flag one can switch between regular (`VRNA_PBACKTRACK_DEFAULT`) backtracing mode, and non-redundant sampling (`VRNA_PBACKTRACK_NON_REDUNDANT`) along the lines of Michalik et al. 2017 [18].

In contrast to `vrna_pbacktrack5_resume()` this function yields the structure samples through a callback mechanism.

A successive sample call to this function may look like:

```
vrna_pbacktrack_mem_t nonredundant_memory = NULL;
// sample the first 100 structures
vrna_pbacktrack5_resume_cb(fc,
    100,
    fc->length,
```

```

        &callback_function,
        (void *)&callback_data,
        &nonredundant_memory,
        options);
// sample another 500 structures
vrna_pbacktrack5_resume_cb(fc,
    500,
    fc->length,
    &callback_function,
    (void *)&callback_data,
    &nonredundant_memory,
    options);
// release memory occupied by the non-redundant memory data structure
vrna_pbacktrack_mem_free(nonredundant_memory);

```

### Precondition

Unique multiloop decomposition has to be active upon creation of `fc` 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`.

### Warning

In non-redundant sampling mode (`VRNA_PBACKTRACK_NON_REDUNDANT`), this function may not yield the full number of requested samples. This may happen if a) the number of requested structures is larger than the total number of structures in the ensemble, b) numeric instabilities prevent the backtracking function to enumerate structures with high free energies, or c) any other error occurs.

### See also

`vrna_pbacktrack5_resume()`, `vrna_pbacktrack5_cb()`, `vrna_pbacktrack_resume_cb()`, `vrna_pbacktrack_mem_t`, `VRNA_PBACKTRACK_DEFAULT`, `VRNA_PBACKTRACK_NON_REDUNDANT`, `vrna_pbacktrack_mem_free`

### Parameters

<i>fc</i>	The fold compound data structure
<i>num_samples</i>	The size of the sample set, i.e. number of structures
<i>length</i>	The length of the subsequence to consider (starting with 5' end)
<i>cb</i>	The callback that receives the sampled structure
<i>data</i>	A data structure passed through to the callback <i>cb</i>
<i>nr_mem</i>	The address of the Boltzmann sampling memory data structure
<i>options</i>	A bitwise OR-flag indicating the backtracing mode.

### Returns

The number of structures actually backtraced

**SWIG Wrapper Notes** This function is attached as overloaded method `pbacktrack5()` to objects of type `fold_compound`. In addition to the number of structures backtraced, this function also returns the `nr_mem` data structure as first element. See also [Python Examples - Boltzmann Sampling](#)



**16.25.4.6 vrna\_pbacktrack()**

```
char * vrna_pbacktrack (
    vrna_fold_compound_t * fc )

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

Sample a secondary structure from the Boltzmann ensemble according to its probability.

Perform a probabilistic (stochastic) backtracing in the partition function DP arrays to obtain a secondary structure.

The structure  $s$  with free energy  $E(s)$  is picked from the Boltzmann distributed ensemble according to its probability

$$p(s) = \frac{\exp(-E(s)/kT)}{Z}$$

with partition function  $Z = \sum_s \exp(-E(s)/kT)$ , Boltzmann constant  $k$  and thermodynamic temperature  $T$ .

**Precondition**

Unique multiloop decomposition has to be active upon creation of `fc` 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`.

**See also**

`vrna_pbacktrack5()`, `vrna_pbacktrack_num`, `vrna_pbacktrack_cb()`

**Parameters**

<code>fc</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`. See also [Python Examples - Boltzmann Sampling](#)

### 16.25.4.7 `vrna_pbacktrack_num()`

```
char ** vrna_pbacktrack_num (
    vrna_fold_compound_t * fc,
    unsigned int num_samples,
    unsigned int options )

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

Obtain a set of secondary structure samples from the Boltzmann ensemble according their probability.

Perform a probabilistic (stochastic) backtracing in the partition function DP arrays to obtain a set of `num_samples` secondary structures.

Any structure  $s$  with free energy  $E(s)$  is picked from the Boltzmann distributed ensemble according to its probability

$$p(s) = \frac{\exp(-E(s)/kT)}{Z}$$

with partition function  $Z = \sum_s \exp(-E(s)/kT)$ , Boltzmann constant  $k$  and thermodynamic temperature  $T$ .

Using the `options` flag one can switch between regular (`VRNA_PBACKTRACK_DEFAULT`) backtracing mode, and non-redundant sampling (`VRNA_PBACKTRACK_NON_REDUNDANT`) along the lines of Michalik et al. 2017 [18].

#### Precondition

Unique multiloop decomposition has to be active upon creation of `fc` 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`.

#### Warning

In non-redundant sampling mode (`VRNA_PBACKTRACK_NON_REDUNDANT`), this function may not yield the full number of requested samples. This may happen if a) the number of requested structures is larger than the total number of structures in the ensemble, b) numeric instabilities prevent the backtracking function to enumerate structures with high free energies, or c) any other error occurs.

#### See also

`vrna_pbacktrack()`, `vrna_pbacktrack_cb()`, `vrna_pbacktrack5_num()`, `VRNA_PBACKTRACK_DEFAULT`, `VRNA_PBACKTRACK_NON_REDUNDANT`

#### Parameters

<code>fc</code>	The fold compound data structure
<code>num_samples</code>	The size of the sample set, i.e. number of structures
<code>options</code>	A bitwise OR-flag indicating the backtracing mode.

## Returns

A set of secondary structure samples in dot-bracket notation terminated by NULL (or NULL on error)

**SWIG Wrapper Notes** This function is attached as overloaded method `pbacktrack()` to objects of type `fold_↔compound` where the last argument `options` is optional with default value `options = VRNA_PBACKTRACK_DEFAULT`. See also [Python Examples - Boltzmann Sampling](#)

16.25.4.8 `vrna_pbacktrack_cb()`

```
unsigned int vrna_pbacktrack_cb (
    vrna_fold_compound_t * fc,
    unsigned int num_samples,
    vrna_boltzmann_sampling_callback * cb,
    void * data,
    unsigned int options )
```

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

Obtain a set of secondary structure samples from the Boltzmann ensemble according their probability.

Perform a probabilistic (stochastic) backtracing in the partition function DP arrays to obtain a set of `num_samples` secondary structures.

Any structure  $s$  with free energy  $E(s)$  is picked from the Boltzmann distributed ensemble according to its probability

$$p(s) = \frac{\exp(-E(s)/kT)}{Z}$$

with partition function  $Z = \sum_s \exp(-E(s)/kT)$ , Boltzmann constant  $k$  and thermodynamic temperature  $T$ .

Using the `options` flag one can switch between regular (`VRNA_PBACKTRACK_DEFAULT`) backtracing mode, and non-redundant sampling (`VRNA_PBACKTRACK_NON_REDUNDANT`) along the lines of Michalik et al. 2017 [18].

In contrast to `vrna_pbacktrack()` and `vrna_pbacktrack_num()` this function yields the structure samples through a callback mechanism.

## Precondition

Unique multiloop decomposition has to be active upon creation of `fc` 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`.

## Warning

In non-redundant sampling mode (`VRNA_PBACKTRACK_NON_REDUNDANT`), this function may not yield the full number of requested samples. This may happen if a) the number of requested structures is larger than the total number of structures in the ensemble, b) numeric instabilities prevent the backtracking function to enumerate structures with high free energies, or c) any other error occurs.

## See also

`vrna_pbacktrack()`, `vrna_pbacktrack_num()`, `vrna_pbacktrack5_cb()`, `VRNA_PBACKTRACK_DEFAULT`, `VRNA_PBACKTRACK_NON_REDUNDANT`

## Parameters

<i>fc</i>	The fold compound data structure
<i>num_samples</i>	The size of the sample set, i.e. number of structures
<i>cb</i>	The callback that receives the sampled structure
<i>data</i>	A data structure passed through to the callback <i>cb</i>
<i>options</i>	A bitwise OR-flag indicating the backtracing mode.

## Returns

The number of structures actually backtraced

**SWIG Wrapper Notes** This function is attached as overloaded method **pbacktrack()** to objects of type *fold\_compound* where the last argument *options* is optional with default value *options* = *VRNA\_PBACKTRACK\_DEFAULT*. See also [Python Examples - Boltzmann Sampling](#)

16.25.4.9 *vrna\_pbacktrack\_resume()*

```
char ** vrna_pbacktrack_resume (
    vrna_fold_compound_t * fc,
    unsigned int num_samples,
    vrna_pbacktrack_mem_t * nr_mem,
    unsigned int options )
```

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

Obtain a set of secondary structure samples from the Boltzmann ensemble according their probability.

Perform a probabilistic (stochastic) backtracing in the partition function DP arrays to obtain a set of *num\_samples* secondary structures.

Any structure *s* with free energy  $E(s)$  is picked from the Boltzmann distributed ensemble according to its probability

$$p(s) = \frac{\exp(-E(s)/kT)}{Z}$$

with partition function  $Z = \sum_s \exp(-E(s)/kT)$ , Boltzmann constant *k* and thermodynamic temperature *T*.

Using the *options* flag one can switch between regular (*VRNA\_PBACKTRACK\_DEFAULT*) backtracing mode, and non-redundant sampling (*VRNA\_PBACKTRACK\_NON\_REDUNDANT*) along the lines of Michalik et al. 2017 [18].

In contrast to *vrna\_pbacktrack\_cb()* this function allows for resuming a previous sampling round in specialized Boltzmann sampling, such as non-redundant backtracking. For that purpose, the user passes the address of a Boltzmann sampling data structure (*vrna\_pbacktrack\_mem\_t*) which will be re-used in each round of sampling, i.e. each successive call to *vrna\_pbacktrack\_resume\_cb()* or *vrna\_pbacktrack\_resume()*.

A successive sample call to this function may look like:

```
vrna_pbacktrack_mem_t nonredundant_memory = NULL;
// sample the first 100 structures
vrna_pbacktrack_resume(fc,
    100,
    &nonredundant_memory,
    options);
// sample another 500 structures
vrna_pbacktrack_resume(fc,
    500,
    &nonredundant_memory,
    options);
// release memory occupied by the non-redundant memory data structure
vrna_pbacktrack_mem_free(nonredundant_memory);
```

**Precondition**

Unique multiloop decomposition has to be active upon creation of `fc` 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`.

**Warning**

In non-redundant sampling mode (`VRNA_PBACKTRACK_NON_REDUNDANT`), this function may not yield the full number of requested samples. This may happen if a) the number of requested structures is larger than the total number of structures in the ensemble, b) numeric instabilities prevent the backtracking function to enumerate structures with high free energies, or c) any other error occurs.

**See also**

`vrna_pbacktrack_resume_cb()`, `vrna_pbacktrack_cb()`, `vrna_pbacktrack5_resume()`, `vrna_pbacktrack_mem_t`, `VRNA_PBACKTRACK_DEFAULT`, `VRNA_PBACKTRACK_NON_REDUNDANT`, `vrna_pbacktrack_mem_free`

**Parameters**

<i>fc</i>	The fold compound data structure
<i>num_samples</i>	The size of the sample set, i.e. number of structures
<i>nr_mem</i>	The address of the Boltzmann sampling memory data structure
<i>options</i>	A bitwise OR-flag indicating the backtracking mode.

**Returns**

A set of secondary structure samples in dot-bracket notation terminated by NULL (or NULL on error)

**SWIG Wrapper Notes** This function is attached as overloaded method `pbacktrack()` to objects of type `fold_compound`. In addition to the list of structures, this function also returns the `nr_mem` data structure as first element. See also [Python Examples - Boltzmann Sampling](#)

**16.25.4.10 vrna\_pbacktrack\_resume\_cb()**

```
unsigned int vrna_pbacktrack_resume_cb (
    vrna_fold_compound_t * fc,
    unsigned int num_samples,
    vrna_boltzmann_sampling_callback * cb,
    void * data,
    vrna_pbacktrack_mem_t * nr_mem,
    unsigned int options )
```

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

Obtain a set of secondary structure samples from the Boltzmann ensemble according their probability.

Perform a probabilistic (stochastic) backtracing in the partition function DP arrays to obtain a set of `num_samples` secondary structures.

Any structure  $s$  with free energy  $E(s)$  is picked from the Boltzmann distributed ensemble according to its probability

$$p(s) = \frac{\exp(-E(s)/kT)}{Z}$$

with partition function  $Z = \sum_s \exp(-E(s)/kT)$ , Boltzmann constant  $k$  and thermodynamic temperature  $T$ .

Using the `options` flag one can switch between regular (`VRNA_PBACKTRACK_DEFAULT`) backtracing mode, and non-redundant sampling (`VRNA_PBACKTRACK_NON_REDUNDANT`) along the lines of Michalik et al. 2017 [18].

In contrast to `vrna_pbacktrack5_resume()` this function yields the structure samples through a callback mechanism.

A successive sample call to this function may look like:

```
vrna_pbacktrack_mem_t nonredundant_memory = NULL;
// sample the first 100 structures
vrna_pbacktrack5_resume_cb(fc,
    100,
    &callback_function,
    (void *)&callback_data,
    &nonredundant_memory,
    options);
// sample another 500 structures
vrna_pbacktrack5_resume_cb(fc,
    500,
    &callback_function,
    (void *)&callback_data,
    &nonredundant_memory,
    options);
// release memory occupied by the non-redundant memory data structure
vrna_pbacktrack_mem_free(nonredundant_memory);
```

### Precondition

Unique multiloop decomposition has to be active upon creation of `fc` 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`.

### Warning

In non-redundant sampling mode (`VRNA_PBACKTRACK_NON_REDUNDANT`), this function may not yield the full number of requested samples. This may happen if a) the number of requested structures is larger than the total number of structures in the ensemble, b) numeric instabilities prevent the backtracking function to enumerate structures with high free energies, or c) any other error occurs.

### See also

`vrna_pbacktrack_resume()`, `vrna_pbacktrack_cb()`, `vrna_pbacktrack5_resume_cb()`, `vrna_pbacktrack_mem_t`, `VRNA_PBACKTRACK_DEFAULT`, `VRNA_PBACKTRACK_NON_REDUNDANT`, `vrna_pbacktrack_mem_free`

## Parameters

<i>fc</i>	The fold compound data structure
<i>num_samples</i>	The size of the sample set, i.e. number of structures
<i>cb</i>	The callback that receives the sampled structure
<i>data</i>	A data structure passed through to the callback <i>cb</i>
<i>nr_mem</i>	The address of the Boltzmann sampling memory data structure
<i>options</i>	A bitwise OR-flag indicating the backtracing mode.

## Returns

The number of structures actually backtraced

**SWIG Wrapper Notes** This function is attached as overloaded method [pbacktrack\(\)](#) to objects of type *fold\_↔compound*. In addition to the number of structures backtraced, this function also returns the *nr\_mem* data structure as first element. See also [Python Examples - Boltzmann Sampling](#)

**16.25.4.11 vrna\_pbacktrack\_mem\_free()**

```
void vrna_pbacktrack_mem_free (
    vrna_pbacktrack_mem_t s )
```

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

Release memory occupied by a Boltzmann sampling memory data structure.

## See also

[vrna\\_pbacktrack\\_mem\\_t](#), [vrna\\_pbacktrack5\\_resume\(\)](#), [vrna\\_pbacktrack5\\_resume\\_cb\(\)](#), [vrna\\_pbacktrack\\_resume\(\)](#), [vrna\\_pbacktrack\\_resume\\_cb\(\)](#)

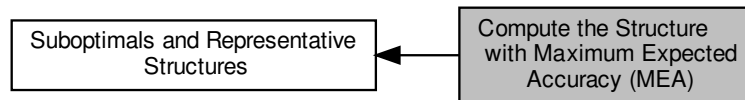
## Parameters

<i>s</i>	The non-redundancy memory data structure
----------	--

## 16.26 Compute the Structure with Maximum Expected Accuracy (MEA)

### 16.26.1 Detailed Description

Collaboration diagram for Compute the Structure with Maximum Expected Accuracy (MEA):



### Functions

- char \* [vrna\\_MEA](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, double gamma, float \*mea)  
*Compute a MEA (maximum expected accuracy) structure.*
- char \* [vrna\\_MEA\\_from\\_plist](#) ([vrna\\_ep\\_t](#) \*plist, const char \*sequence, double gamma, [vrna\\_md\\_t](#) \*md, float \*mea)  
*Compute a MEA (maximum expected accuracy) structure from a list of probabilities.*
- float [MEA](#) ([plist](#) \*p, char \*structure, double gamma)  
*Computes a MEA (maximum expected accuracy) structure.*

### 16.26.2 Function Documentation

#### 16.26.2.1 vrna\_MEA()

```
char * vrna_MEA (
    vrna\_fold\_compound\_t * fc,
    double gamma,
    float * mea )
```

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

Compute 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.

#### Precondition

[vrna\\_pf\(\)](#) must be executed on input parameter `fc`



## Parameters

<i>fc</i>	The fold compound data structure with pre-filled base pair probability matrix
<i>gamma</i>	The weighting factor for base pairs vs. unpaired nucleotides
<i>mea</i>	A pointer to a variable where the MEA value will be written to

## Returns

An MEA structure (or NULL on any error)

**SWIG Wrapper Notes** This function is attached as overloaded method **MEA**(gamma = 1.) to objects of type *fold\_compound*. Note, that it returns the MEA structure and MEA value as a tuple (MEA\_↵ structure, MEA)

## 16.26.2.2 vrna\_MEA\_from\_plist()

```
char * vrna_MEA_from_plist (
    vrna_ep_t * plist,
    const char * sequence,
    double gamma,
    vrna_md_t * md,
    float * mea )
```

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

Compute a MEA (maximum expected accuracy) structure from a list of probabilities.

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.

## Note

To include G-Quadruplex support, the corresponding field in *md* must be set.

## Parameters

<i>plist</i>	A list of base pair probabilities the MEA structure is computed from
<i>sequence</i>	The RNA sequence that corresponds to the list of probability values
<i>gamma</i>	The weighting factor for base pairs vs. unpaired nucleotides
<i>md</i>	A model details data structure (maybe NULL)
<i>mea</i>	A pointer to a variable where the MEA value will be written to

**Returns**

An MEA structure (or NULL on any error)

**SWIG Wrapper Notes** This function is available as overloaded function **MEA\_from\_plist**(gamma = 1., md = N↔ULL). Note, that it returns the MEA structure and MEA value as a tuple (MEA\_structure, MEA)

**16.26.2.3 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

$$A(S) = \sum_{(i,j) \in S} 2\gamma p_{ij} + \sum_{i \notin S} p_i^u$$

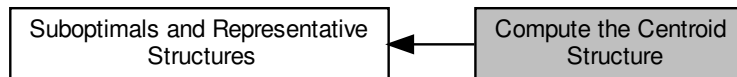
Higher values of  $\gamma$  result in more base pairs of lower probability and thus higher sensitivity. Low values of  $\gamma$  result in structures containing only highly likely pairs (high specificity). The code of the MEA function also demonstrates the use of sparse dynamic programming scheme to reduce the time and memory complexity of folding.

**Deprecated** Use `vrna_MEA()` or `vrna_MEA_from_plist()` instead!

## 16.27 Compute the Centroid Structure

### 16.27.1 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.*

### 16.27.2 Function Documentation

#### 16.27.2.1 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

$$\langle d(S) \rangle = \sum_{(i,j) \in S} (1 - p_{ij}) + \sum_{(i,j) \notin S} p_{ij}$$

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>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)

**16.27.2.2 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

$$< d(S) >= \sum_{(i,j) \in S} (1 - p_{ij}) + \sum_{(i,j) \notin S} p_{ij}$$

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>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)

**16.27.2.3 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

$$< d(S) >= \sum_{(i,j) \in S} (1 - p_{ij}) + \sum_{(i,j) \notin S} p_{ij}$$

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>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> )

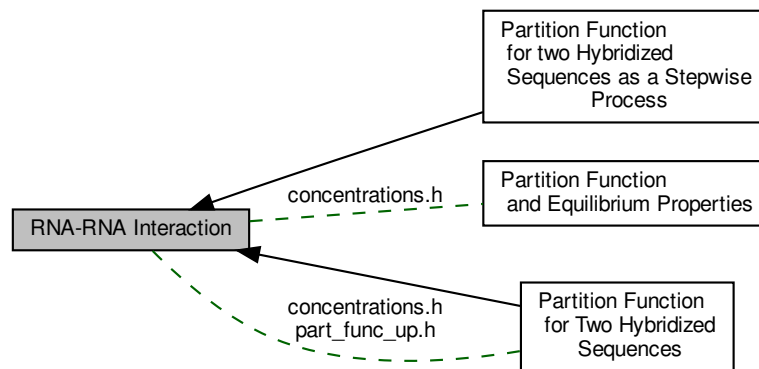
## Returns

The centroid structure of the ensemble in dot-bracket notation (NULL on error)

## 16.28 RNA-RNA Interaction

### 16.28.1 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.*

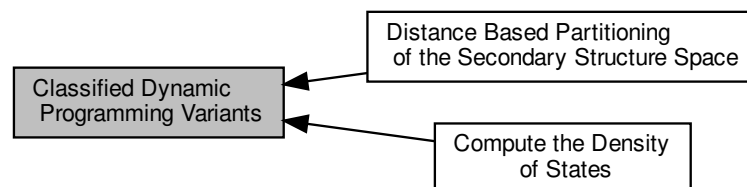
### Files

- 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.*

## 16.29 Classified Dynamic Programming Variants

### 16.29.1 Detailed Description

Collaboration diagram for Classified Dynamic Programming Variants:



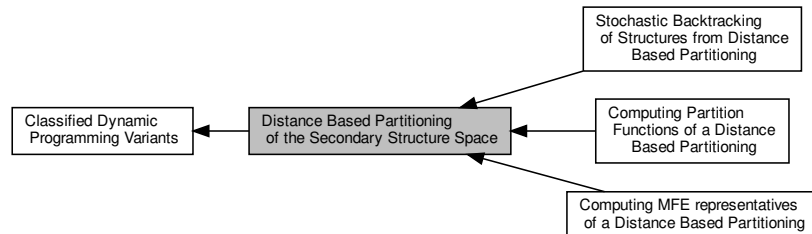
### Modules

- [Distance Based Partitioning of the Secondary Structure Space](#)
- [Compute the Density of States](#)

## 16.30 Distance Based Partitioning of the Secondary Structure Space

### 16.30.1 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.*



## 16.31 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.

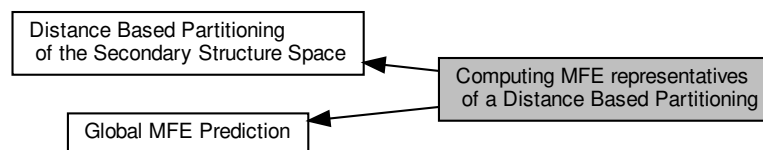
### 16.31.1 Detailed Description

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.*

## 16.31.2 Data Structure Documentation

### 16.31.2.1 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 datastructure 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

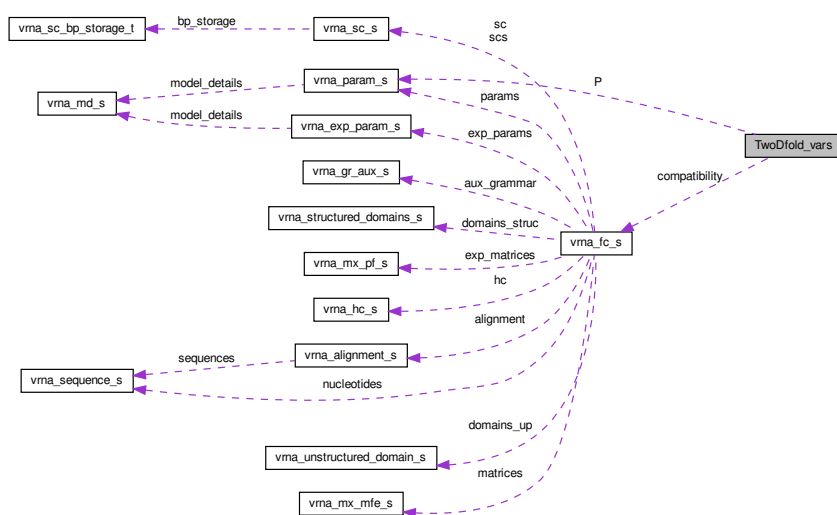
- int `k`  
*Distance to first reference.*
- int `l`  
*Distance to second reference.*
- float `en`  
*Free energy in kcal/mol.*
- char \* `s`  
*MFE representative structure in dot-bracket notation.*

### 16.31.2.2 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].*

### 16.31.3 Typedef Documentation

#### 16.31.3.1 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\(\)](#)

#### 16.31.3.2 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!

### 16.31.4 Function Documentation

**16.31.4.1 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

**16.31.4.2 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

**16.31.4.3 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

**16.31.4.4 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**

<code>our_variables</code>	A pointer to the datastructure to be destroyed
----------------------------	--

**16.31.4.5 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)

**16.31.4.6 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



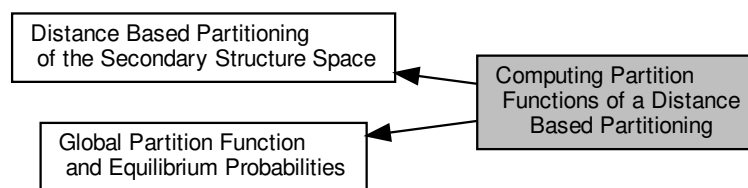
## 16.32 Computing Partition Functions of a Distance Based Partitioning

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.

### 16.32.1 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.

## 16.32.2 Data Structure Documentation

### 16.32.2.1 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*

## 16.32.3 Typedef Documentation

### 16.32.3.1 `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()`

## 16.32.4 Function Documentation

### 16.32.4.1 `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

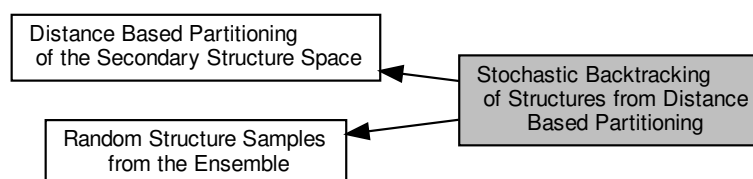
## 16.33 Stochastic Backtracking of Structures from Distance Based Partitioning

Contains functions related to stochastic backtracking from a specified distance class.

### 16.33.1 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.*

### 16.33.2 Function Documentation

#### 16.33.2.1 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

**16.33.2.2 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

## 16.34 Predicting various thermodynamic properties

Compute various thermodynamic properties using the partition function.

### 16.34.1 Detailed Description

Compute various thermodynamic properties using the partition function.

Many thermodynamic properties can be derived from the partition function

$$Q = \sum_{s \in \omega} e^{\frac{-E(s)}{kT}}.$$

In particular, for nucleic acids in equilibrium the probability  $p(F)$  of a particular structural feature  $F$  follows Boltzmann's law, i.e.

$$p(F) \propto \sum_{s|F \in s} e^{\frac{-E(s)}{kT}}.$$

The actual probabilities can then be obtained from the ratio of those structures containing  $F$  and *all* structures, i.e.

$$p(F) = \frac{1}{Q} \sum_{s|F \in s} e^{\frac{-E(s)}{kT}}.$$

Consequently, a particular secondary structure  $s$  has equilibrium probability

$$p(s) = \frac{1}{Q} e^{\frac{-E(s)}{kT}}$$

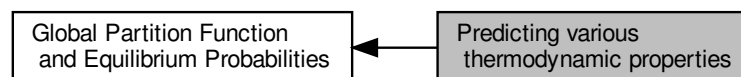
which can be easily computed once  $Q$  and  $E(s)$  are known.

On the other hand, efficient dynamic programming algorithms exist to compute the equilibrium probabilities

$$p_{ij} = \frac{1}{Q} \sum_{s|(i,j) \in s} e^{\frac{-E(s)}{kT}}$$

of base pairs  $(i, j)$  without the need for exhaustive enumeration of  $s$ .

This interface provides the functions for all thermodynamic property computations implemented in *RNAlib*. Collaboration diagram for Predicting various thermodynamic properties:



### Files

- file [equilibrium\\_probs.h](#)  
*Equilibrium Probability implementations.*
- file [heat\\_capacity.h](#)  
*Compute heat capacity for an RNA.*

## Data Structures

- struct [vrna\\_heat\\_capacity\\_s](#)  
A single result from heat capacity computations. [More...](#)

## Typedefs

- typedef void() [vrna\\_heat\\_capacity\\_callback](#)(float temp, float heat\_capacity, void \*data)  
The callback for heat capacity predictions.
- typedef struct [vrna\\_heat\\_capacity\\_s](#) [vrna\\_heat\\_capacity\\_t](#)  
A single result from heat capacity computations.

## Base pair probabilities and derived computations

- int [vrna\\_pairing\\_probs](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, char \*structure)
- 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.
- double \* [vrna\\_positional\\_entropy](#) ([vrna\\_fold\\_compound\\_t](#) \*fc)  
Compute a vector of positional entropies.
- [vrna\\_ep\\_t](#) \* [vrna\\_stack\\_prob](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, double cutoff)  
Compute stacking probabilities.

## Multimer probabilities computations

- 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.

## Structure probability computations

- double [vrna\\_pr\\_structure](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, const char \*structure)  
Compute the equilibrium probability of a particular secondary structure.
- double [vrna\\_pr\\_energy](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, double e)

## Basic heat capacity function interface

- [vrna\\_heat\\_capacity\\_t](#) \* [vrna\\_heat\\_capacity](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, float T\_min, float T\_max, float T\_increment, unsigned int mpoints)  
Compute the specific heat for an RNA.
- int [vrna\\_heat\\_capacity\\_cb](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, float T\_min, float T\_max, float T\_increment, unsigned int mpoints, [vrna\\_heat\\_capacity\\_callback](#) \*cb, void \*data)  
Compute the specific heat for an RNA (callback variant)



## Simplified heat capacity computation

- [vrna\\_heat\\_capacity\\_t](#) \* [vrna\\_heat\\_capacity\\_simple](#) (const char \*sequence, float T\_min, float T\_max, float T\_increment, unsigned int mpoints)

*Compute the specific heat for an RNA (simplified variant)*

## 16.34.2 Data Structure Documentation

### 16.34.2.1 struct vrna\_heat\_capacity\_s

A single result from heat capacity computations.

See also

[vrna\\_heat\\_capacity\(\)](#)

#### Data Fields

- float [temperature](#)  
*The temperature in °C.*
- float [heat\\_capacity](#)  
*The specific heat at this temperature in Kcal/(Mol \* K)*

## 16.34.3 Typedef Documentation

### 16.34.3.1 vrna\_heat\_capacity\_callback

```
typedef void() vrna_heat_capacity_callback(float temp, float heat_capacity, void *data)
```

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

The callback for heat capacity predictions.

**Notes on Callback Functions** This function will be called for each evaluated temperature in the heat capacity prediction.

See also

[vrna\\_heat\\_capacity\\_cb\(\)](#)

#### Parameters

<i>temp</i>	The current temperature this results corresponds to in °C
<i>heat_capacity</i>	The heat capacity in Kcal/(Mol * K)
<i>data</i>	Some arbitrary data pointer passed through by the function executing the callback

### 16.34.3.2 vrna\_heat\_capacity\_t

```
typedef struct vrna_heat_capacity_s vrna_heat_capacity_t

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

A single result from heat capacity computations.

This is a convenience typedef for `vrna_heat_capacity_s`, i.e. results as obtained from `vrna_heat_capacity()`

## 16.34.4 Function Documentation

### 16.34.4.1 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

### 16.34.4.2 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

vc	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`

#### 16.34.4.3 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

<i>fc</i>	A fold_compound with pre-computed base pair probabilities
<i>structure</i>	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

**SWIG Wrapper Notes** This function is attached as method **ensemble\_defect()** to objects of type *fold\_compound*

**16.34.4.4 vrna\_positional\_entropy()**

```
double * vrna_positional_entropy (
    vrna_fold_compound_t * fc )

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

Compute a vector of positional entropies.

This function computes the positional entropies from base pair probabilities as

$$S(i) = - \sum_j p_{ij} \log(p_{ij}) - q_i \log(q_i)$$

with unpaired probabilities  $q_i = 1 - \sum_j p_{ij}$ .

Low entropy regions have little structural flexibility and the reliability of the predicted structure is high. High entropy implies many structural alternatives. While these alternatives may be functionally important, they make structure prediction more difficult and thus less reliable.

## Precondition

This function requires pre-computed base pair probabilities! Thus, **vrna\_pf()** must be called beforehand.

## Parameters

<i>fc</i>	A fold_compound with pre-computed base pair probabilities
-----------	---

## Returns

A 1-based vector of positional entropies  $S(i)$ . (position 0 contains the sequence length)

**SWIG Wrapper Notes** This function is attached as method **positional\_entropy()** to objects of type *fold\_compound*

16.34.4.5 `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  $p$

16.34.4.6 `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

#### 16.34.4.7 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  $[0 : 1]$ )

**SWIG Wrapper Notes** This function is attached as method `pr_structure()` to objects of type *fold\_compound*

#### 16.34.4.8 vrna\_pr\_energy()

```
double vrna_pr_energy (
    vrna_fold_compound_t * fc,
    double e )

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

**SWIG Wrapper Notes** This function is attached as method `pr_energy()` to objects of type *fold\_compound*

16.34.4.9 `vrna_heat_capacity()`

```
vrna_heat_capacity_t * vrna_heat_capacity (
    vrna_fold_compound_t * fc,
    float T_min,
    float T_max,
    float T_increment,
    unsigned int mpoints )

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

Compute the specific heat for an RNA.

This function computes an RNAs specific heat in a given temperature range from the partition function by numeric differentiation. The result is returned as a list of pairs of temperature in °C and specific heat in Kcal/(Mol\*K).

Users can specify the temperature range for the computation from `T_min` to `T_max`, as well as the increment step size `T_increment`. The latter also determines how many times the partition function is computed. Finally, the parameter `mpoints` determines how smooth the curve should be. The algorithm itself fits a parabola to  $2 \cdot mpoints + 1$  data points to calculate 2nd derivatives. Increasing this parameter produces a smoother curve.

See also

[vrna\\_heat\\_capacity\\_cb\(\)](#), [vrna\\_heat\\_capacity\\_t](#), [vrna\\_heat\\_capacity\\_s](#)

## Parameters

<code>fc</code>	The <code>vrna_fold_compound_t</code> with the RNA sequence to analyze
<code>T_min</code>	Lowest temperature in °C
<code>T_max</code>	Highest temperature in °C
<code>T_increment</code>	Stepsize for temperature incrementation in °C (a reasonable choice might be 1 °C)
<code>mpoints</code>	The number of interpolation points to calculate 2nd derivative (a reasonable choice might be 2, min: 1, max: 100)

## Returns

A list of pairs of temperatures and corresponding heat capacity or *NULL* upon any failure. The last entry of the list is indicated by a **temperature** field set to a value smaller than `T_min`

**SWIG Wrapper Notes** This function is attached as overloaded method **heat\_capacity()** to objects of type `fold_compound`. If the optional function arguments `T_min`, `T_max`, `T_increment`, and `mpoints` are omitted, they default to 0.0, 100.0, 1.0 and 2, respectively.

16.34.4.10 `vrna_heat_capacity_cb()`

```
int vrna_heat_capacity_cb (
    vrna_fold_compound_t * fc,
    float T_min,
    float T_max,
```

```
float T_increment,
unsigned int mpoints,
vrna_heat_capacity_callback * cb,
void * data )
```

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

Compute the specific heat for an RNA (callback variant)

Similar to [vrna\\_heat\\_capacity\(\)](#), this function computes an RNAs specific heat in a given temperature range from the partition function by numeric differentiation. Instead of returning a list of temperature/specific heat pairs, however, this function returns the individual results through a callback mechanism. The provided function will be called for each result and passed the corresponding temperature and specific heat values along with the arbitrary data as provided through the `data` pointer argument.

Users can specify the temperature range for the computation from `T_min` to `T_max`, as well as the increment step size `T_increment`. The latter also determines how many times the partition function is computed. Finally, the parameter `mpoints` determines how smooth the curve should be. The algorithm itself fits a parabola to  $2 \cdot mpoints + 1$  data points to calculate 2nd derivatives. Increasing this parameter produces a smoother curve.

See also

[vrna\\_heat\\_capacity\(\)](#), [vrna\\_heat\\_capacity\\_callback](#)

#### Parameters

<i>fc</i>	The <a href="#">vrna_fold_compound_t</a> with the RNA sequence to analyze
<i>T_min</i>	Lowest temperature in °C
<i>T_max</i>	Highest temperature in °C
<i>T_increment</i>	Stepsize for temperature incrementation in °C (a reasonable choice might be 1 °C)
<i>mpoints</i>	The number of interpolation points to calculate 2nd derivative (a reasonable choice might be 2, min: 1, max: 100)
<i>cb</i>	The user-defined callback function that receives the individual results
<i>data</i>	An arbitrary data structure that will be passed to the callback in conjunction with the results

#### Returns

Returns 0 upon failure, and non-zero otherwise

**SWIG Wrapper Notes** This function is attached as method **heat\_capacity\_cb()** to objects of type *fold\_compound*

#### 16.34.4.11 vrna\_heat\_capacity\_simple()

```
vrna_heat_capacity_t * vrna_heat_capacity_simple (
    const char * sequence,
    float T_min,
    float T_max,
    float T_increment,
    unsigned int mpoints )
```



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

Compute the specific heat for an RNA (simplified variant)

Similar to [vrna\\_heat\\_capacity\(\)](#), this function computes an RNAs specific heat in a given temperature range from the partition function by numeric differentiation. This simplified version, however, only requires the RNA sequence as input instead of a `vrna_fold_compound_t` data structure. The result is returned as a list of pairs of temperature in °C and specific heat in Kcal/(Mol\*K).

Users can specify the temperature range for the computation from `T_min` to `T_max`, as well as the increment step size `T_increment`. The latter also determines how many times the partition function is computed. Finally, the parameter `mpoints` determines how smooth the curve should be. The algorithm itself fits a parabola to  $2 \cdot mpoints + 1$  data points to calculate 2nd derivatives. Increasing this parameter produces a smoother curve.

See also

[vrna\\_heat\\_capacity\(\)](#), [vrna\\_heat\\_capacity\\_cb\(\)](#), [vrna\\_heat\\_capacity\\_t](#), [vrna\\_heat\\_capacity\\_s](#)

#### Parameters

<i>sequence</i>	The RNA sequence input (must be uppercase)
<i>T_min</i>	Lowest temperature in °C
<i>T_max</i>	Highest temperature in °C
<i>T_increment</i>	Stepsize for temperature incrementation in °C (a reasonable choice might be 1 °C)
<i>mpoints</i>	The number of interpolation points to calculate 2nd derivative (a reasonable choice might be 2, min: 1, max: 100)

#### Returns

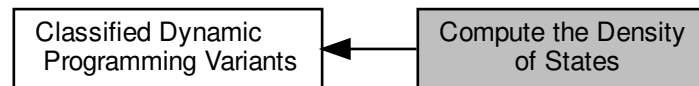
A list of pairs of temperatures and corresponding heat capacity or *NULL* upon any failure. The last entry of the list is indicated by a **temperature** field set to a value smaller than `T_min`

**SWIG Wrapper Notes** This function is available as overloaded function **heat\_capacity()**. If the optional function arguments `T_min`, `T_max`, `T_increment`, and `mpoints` are omitted, they default to 0.0, 100.0, 1.0 and 2, respectively.

## 16.35 Compute the Density of States

### 16.35.1 Detailed Description

Collaboration diagram for Compute the Density of States:



### Variables

- int `density_of_states` [MAXDOS+1]  
*The Density of States.*

### 16.35.2 Variable Documentation

#### 16.35.2.1 `density_of_states`

```
int density_of_states[MAXDOS+1] [extern]
```

```
#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()`

## 16.36 Inverse Folding (Design)

RNA sequence design.

### 16.36.1 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](#)

### 16.36.2 Function Documentation

#### 16.36.2.1 [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

**16.36.2.2 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

**16.36.3 Variable Documentation****16.36.3.1 final\_cost**

```
float final_cost [extern]

#include <ViennaRNA/inverse.h>

when to stop inverse\_pf\_fold\(\)
```

**16.36.3.2 give\_up**

```
int give_up [extern]
```

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

default 0: try to minimize structure distance even if no exact solution can be found

**16.36.3.3 inv\_verbose**

```
int inv_verbose [extern]
```

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

print out substructure on which `inverse_fold()` fails

## 16.37 Neighborhood Relation and Move Sets for Secondary Structures

Different functions to generate structural neighbors of a secondary structure according to a particular Move Set.

### 16.37.1 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:

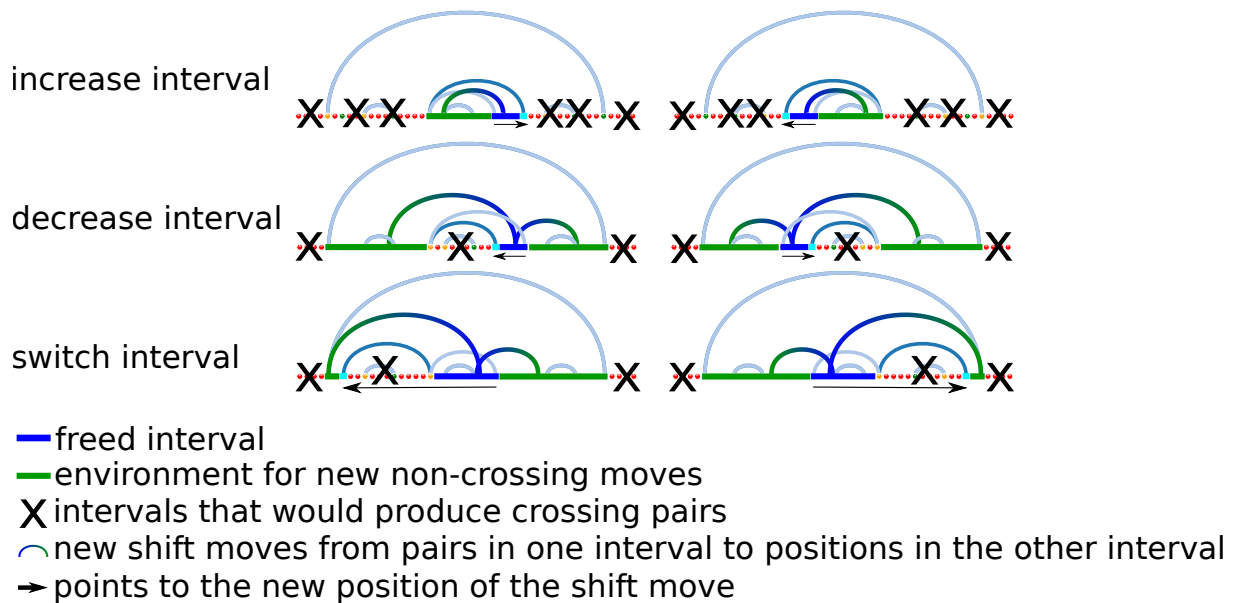
```
We have given a sequence and a structure.
Sequence  AAGGAAACC
Structure ..(.....)
Indices   123456789
The given base pair is (3,9) and the neighbors are the insertion (4, 8), the deletion (-3,-9), the
shift (3,-8)
and the shift (-4, 9).
This leads to the neighbored structures:
... (....)
.....
... (....)
.... (....)
```

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 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.

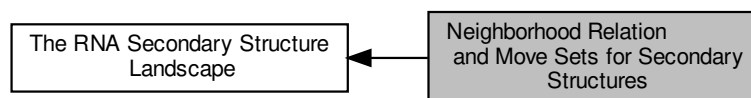
If we need to construct all neighbors several times for subsequent moves, we can speed up the task by using the move set of the previous structure. The previous move set has to be filtered, such that all moves that would cross the next selected move are non-crossing. Next, the selected move has to be 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 [move.h](#)  
*Methods to operate with structural neighbors of RNA secondary structures.*
- 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.*
- `#define VRNA_NEIGHBOR_CHANGE 1`  
*State indicator for a neighbor that has been changed.*
- `#define VRNA_NEIGHBOR_INVALID 2`  
*State indicator for a neighbor that has been invalidated.*
- `#define VRNA_NEIGHBOR_NEW 3`  
*State indicator for a neighbor that has become newly available.*

## Typedefs

- `typedef struct vrna_move_s vrna_move_t`  
*A single move that transforms a secondary structure into one of its neighbors.*
- `typedef void() vrna_callback_move_update(vrna_fold_compound_t *fc, vrna_move_t neighbor, unsigned int state, void *data)`  
*Prototype of the neighborhood update callback.*

## Functions

- `vrna_move_t vrna_move_init (int pos_5, int pos_3)`  
*Create an atomic move.*
- `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.*
- `int vrna_move_is_removal (const vrna_move_t *m)`  
*Test whether a move is a base pair removal.*
- `int vrna_move_is_insertion (const vrna_move_t *m)`  
*Test whether a move is a base pair insertion.*
- `int vrna_move_is_shift (const vrna_move_t *m)`  
*Test whether a move is a base pair shift.*
- `int vrna_move_compare (const vrna_move_t *a, const vrna_move_t *b, const short *pt)`  
*Compare two moves.*
- `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)*



- `int vrna_move_neighbor_diff_cb (vrna_fold_compound_t *fc, short *ptable, vrna_move_t move, vrna_callback_move_update *cb, void *data, unsigned int options)`

*Apply a move to a secondary structure and indicate which neighbors have changed consequentially.*

- `vrna_move_t * vrna_move_neighbor_diff (vrna_fold_compound_t *fc, short *ptable, vrna_move_t move, vrna_move_t **invalid_moves, unsigned int options)`

*Apply a move to a secondary structure and indicate which neighbors have changed consequentially.*

## 16.37.2 Data Structure Documentation

### 16.37.2.1 struct vrna\_move\_s

An atomic representation of the transition / move from one structure to its neighbor.

An atomic transition / move may be one of the following:

- a **base pair insertion**,
- a **base pair removal**, or
- a **base pair shift** where an existing base pair changes one of its pairing partner.

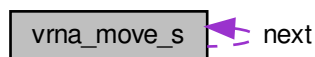
These moves are encoded by two integer values that represent the affected 5' and 3' nucleotide positions. Furthermore, we use the following convention on the signedness of these encodings:

- both values are positive for *insertion moves*
- both values are negative for *base pair removals*
- both values have different signedness for *shift moves*, where the positive value indicates the nucleotide that stays constant, and the others absolute value is the new pairing partner

#### Note

A value of 0 in either field is used as list-end indicator and doesn't represent any valid move.

Collaboration diagram for `vrna_move_s`:



#### Data Fields

- `int pos_5`  
*The (absolute value of the) 5' position of a base pair, or any position of a shifted pair.*
- `int pos_3`  
*The (absolute value of the) 3' position of a base pair, or any position of a shifted pair.*
- `vrna_move_t * next`  
*The next base pair (if an elementary move changes more than one base pair), or NULL. Has to be terminated with move 0,0.*

### 16.37.3 Macro Definition Documentation

#### 16.37.3.1 VRNA\_MOVESET\_INSERTION

```
#define VRNA_MOVESET_INSERTION 4
```

```
#include <ViennaRNA/landscape/move.h>
```

Option flag indicating insertion move.

See also

[vrna\\_neighbors\(\)](#), [vrna\\_neighbors\\_successive](#), [vrna\\_path\(\)](#)

#### 16.37.3.2 VRNA\_MOVESET\_DELETION

```
#define VRNA_MOVESET_DELETION 8
```

```
#include <ViennaRNA/landscape/move.h>
```

Option flag indicating deletion move.

See also

[vrna\\_neighbors\(\)](#), [vrna\\_neighbors\\_successive](#), [vrna\\_path\(\)](#)

#### 16.37.3.3 VRNA\_MOVESET\_SHIFT

```
#define VRNA_MOVESET_SHIFT 16
```

```
#include <ViennaRNA/landscape/move.h>
```

Option flag indicating shift move.

See also

[vrna\\_neighbors\(\)](#), [vrna\\_neighbors\\_successive](#), [vrna\\_path\(\)](#)

#### 16.37.3.4 VRNA\_MOVESET\_NO\_LP

```
#define VRNA_MOVESET_NO_LP 32
```

```
#include <ViennaRNA/landscape/move.h>
```

Option flag indicating moves without lonely base pairs.

See also

[vrna\\_neighbors\(\)](#), [vrna\\_neighbors\\_successive](#), [vrna\\_path\(\)](#)

#### 16.37.3.5 VRNA\_MOVESET\_DEFAULT

```
#define VRNA_MOVESET_DEFAULT (VRNA_MOVESET_INSERTION | VRNA_MOVESET_DELETION)
```

```
#include <ViennaRNA/landscape/move.h>
```

Option flag indicating default move set, i.e. insertions/deletion of a base pair.

See also

[vrna\\_neighbors\(\)](#), [vrna\\_neighbors\\_successive](#), [vrna\\_path\(\)](#)

#### 16.37.3.6 VRNA\_NEIGHBOR\_CHANGE

```
#define VRNA_NEIGHBOR_CHANGE 1
```

```
#include <ViennaRNA/landscape/neighbor.h>
```

State indicator for a neighbor that has been changed.

See also

[vrna\\_move\\_neighbor\\_diff\\_cb\(\)](#)

#### 16.37.3.7 VRNA\_NEIGHBOR\_INVALID

```
#define VRNA_NEIGHBOR_INVALID 2
```

```
#include <ViennaRNA/landscape/neighbor.h>
```

State indicator for a neighbor that has been invalidated.

See also

[vrna\\_move\\_neighbor\\_diff\\_cb\(\)](#)

### 16.37.3.8 VRNA\_NEIGHBOR\_NEW

```
#define VRNA_NEIGHBOR_NEW 3
```

```
#include <ViennaRNA/landscape/neighbor.h>
```

State indicator for a neighbor that has become newly available.

See also

[vrna\\_move\\_neighbor\\_diff\\_cb\(\)](#)

## 16.37.4 Typedef Documentation

### 16.37.4.1 vrna\_callback\_move\_update

```
typedef void() vrna_callback_move_update(vrna_fold_compound_t *fc, vrna_move_t neighbor, unsigned
int state, void *data)
```

```
#include <ViennaRNA/landscape/neighbor.h>
```

Prototype of the neighborhood update callback.

See also

[vrna\\_move\\_neighbor\\_diff\\_cb\(\)](#), [VRNA\\_NEIGHBOR\\_CHANGE](#), [VRNA\\_NEIGHBOR\\_INVALID](#), [VRNA\\_NEIGHBOR\\_NEW](#)

Parameters

<i>fc</i>	The fold compound the calling function is working on
<i>neighbor</i>	The move that generates the (changed or new) neighbor
<i>state</i>	The state of the neighbor (move) as supplied by argument <i>neighbor</i>
<i>data</i>	Some arbitrary data pointer as passed to <a href="#">vrna_move_neighbor_diff_cb()</a>

## 16.37.5 Function Documentation

### 16.37.5.1 vrna\_move\_init()

```
vrna_move_t vrna_move_init (
    int pos_5,
    int pos_3 )
```

```
#include <ViennaRNA/landscape/move.h>
```

Create an atomic move.

See also

[vrna\\_move\\_s](#)

#### Parameters

<i>pos</i> <sub>←</sub> _5	The 5' position of the move (positive for insertions, negative for removal, any value for shift moves)
<i>pos</i> <sub>←</sub> _3	The 3' position of the move (positive for insertions, negative for removal, any value for shift moves)

#### Returns

An atomic move as specified by `pos_5` and `pos_3`

#### 16.37.5.2 `vrna_move_list_free()`

```
void vrna_move_list_free (
    vrna_move_t * moves )
```

```
#include <ViennaRNA/landscape/move.h>
```

delete all moves in a zero terminated list.

#### 16.37.5.3 `vrna_move_apply()`

```
void vrna_move_apply (
    short * pt,
    const vrna_move_t * m )
```

```
#include <ViennaRNA/landscape/move.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

#### 16.37.5.4 `vrna_move_is_removal()`

```
int vrna_move_is_removal (
    const vrna_move_t * m )
```

```
#include <ViennaRNA/landscape/move.h>
```

Test whether a move is a base pair removal.

**Parameters**

<i>m</i>	The move to test against
----------	--------------------------

**Returns**

Non-zero if the move is a base pair removal, 0 otherwise

**16.37.5.5 vrna\_move\_is\_insertion()**

```
int vrna_move_is_insertion (
    const vrna_move_t * m )
```

```
#include <ViennaRNA/landscape/move.h>
```

Test whether a move is a base pair insertion.

**Parameters**

<i>m</i>	The move to test against
----------	--------------------------

**Returns**

Non-zero if the move is a base pair insertion, 0 otherwise

**16.37.5.6 vrna\_move\_is\_shift()**

```
int vrna_move_is_shift (
    const vrna_move_t * m )
```

```
#include <ViennaRNA/landscape/move.h>
```

Test whether a move is a base pair shift.

**Parameters**

<i>m</i>	The move to test against
----------	--------------------------

**Returns**

Non-zero if the move is a base pair shift, 0 otherwise

**16.37.5.7 vrna\_move\_compare()**

```
int vrna_move_compare (
    const vrna_move_t * a,
    const vrna_move_t * b,
    const short * pt )

#include <ViennaRNA/landscape/move.h>
```

Compare two moves.

The function compares two moves *a* and *b* and returns whether move *a* is lexicographically smaller (-1), larger (1) or equal to move *b*.

If any of the moves *a* or *b* is a shift move, this comparison only makes sense in a structure context. Thus, the third argument with the current structure must be provided.

**Note**

This function returns 0 (equality) upon any error, e.g. missing input

**Warning**

Currently, shift moves are not supported!

**Parameters**

<i>a</i>	The first move of the comparison
<i>b</i>	The second move of the comparison
<i>pt</i>	The pair table of the current structure that is compatible with both moves (maybe NULL if moves are guaranteed to be no shifts)

**Returns**

-1 if  $a < b$ , 1 if  $a > b$ , 0 otherwise

**16.37.5.8 vrna\_loopidx\_update()**

```
void vrna_loopidx_update (
    int * loopidx,
    const short * pt,
    int length,
    const vrna_move_t * m )

#include <ViennaRNA/landscape/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

in, out	<i>loopidx</i>	The loop index data structure that needs an update
in	<i>pt</i>	A pair table on which the move will be executed
	<i>length</i>	The length of the structure
in	<i>m</i>	The move that is applied to the current structure

16.37.5.9 `vrna_neighbors()`

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

#include <ViennaRNA/landscape/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.

16.37.5.10 `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/landscape/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 <code>prev_pt</code>
in	<i>prev_pt</i>	A pair table representation of the structure before <code>curr_move</code> is/was applied
in	<i>prev_neighbors</i>	The list of neighbors of <code>prev_pt</code>
	<i>size_prev_neighbors</i>	The size of <code>prev_neighbors</code> , 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)

#### 16.37.5.11 vrna\_move\_neighbor\_diff\_cb()

```

int vrna_move_neighbor_diff_cb (
    vrna_fold_compound_t * fc,
    short * ptable,
    vrna_move_t move,
    vrna_callback_move_update * cb,
    void * data,
    unsigned int options )

```

```
#include <ViennaRNA/landscape/neighbor.h>
```

Apply a move to a secondary structure and indicate which neighbors have changed consequentially.

This function applies a move to a secondary structure and explores the local neighborhood of the affected loop. Any changes to previously compatible neighbors that have been affected by this loop will be reported through a callback function. In particular, any of the three cases might appear:

- A previously available neighbor move has changed, usually the free energy change of the move ([VRNA\\_NEIGHBOR\\_CHANGE](#))
- A previously available neighbor move became invalid ([VRNA\\_NEIGHBOR\\_INVALID](#))
- A new neighbor move becomes available ([VRNA\\_NEIGHBOR\\_NEW](#))

See also

[vrna\\_move\\_neighbor\\_diff\(\)](#), [VRNA\\_NEIGHBOR\\_CHANGE](#), [VRNA\\_NEIGHBOR\\_INVALID](#), [VRNA\\_NEIGHBOR\\_NEW](#), [vrna\\_callback\\_move\\_update](#)

#### Parameters

<i>fc</i>	A fold compound for the RNA sequence(s) that this function operates on
<i>ptable</i>	The current structure as pair table
<i>move</i>	The move to apply
<i>cb</i>	The address of the callback function that is passed the neighborhood changes
<i>data</i>	An arbitrary data pointer that will be passed through to the callback function <i>cb</i>
<i>options</i>	Options to modify the behavior of this function, .e.g available move set

#### Returns

Non-zero on success, 0 otherwise

#### 16.37.5.12 vrna\_move\_neighbor\_diff()

```
vrna_move_t* vrna_move_neighbor_diff (
    vrna_fold_compound_t * fc,
    short * ptable,
    vrna_move_t move,
    vrna_move_t ** invalid_moves,
    unsigned int options )
```

```
#include <ViennaRNA/landscape/neighbor.h>
```

Apply a move to a secondary structure and indicate which neighbors have changed consequentially.

Similar to [vrna\\_move\\_neighbor\\_diff\\_cb\(\)](#), this function applies a move to a secondary structure and reports back the neighbors of the current structure become affected by this move. Instead of executing a callback for each of the affected neighbors, this function compiles two lists of neighbor moves, one that is returned and consists of all moves that are novel or may have changed in energy, and a second, *invalid\_moves*, that consists of all the neighbor moves that become invalid, respectively.

#### Parameters

<i>fc</i>	A fold compound for the RNA sequence(s) that this function operates on
<i>ptable</i>	The current structure as pair table
<i>move</i>	The move to apply
<i>invalid_moves</i>	The address of a move list where the function stores those moves that become invalid
<i>options</i>	Options to modify the behavior of this function, .e.g available move set

**Returns**

A list of moves that might have changed in energy or are novel compared to the structure before application of the move

## 16.38 (Re-)folding Paths, Saddle Points, Energy Barriers, and Local Minima

API for various RNA folding path algorithms.

### 16.38.1 Detailed Description

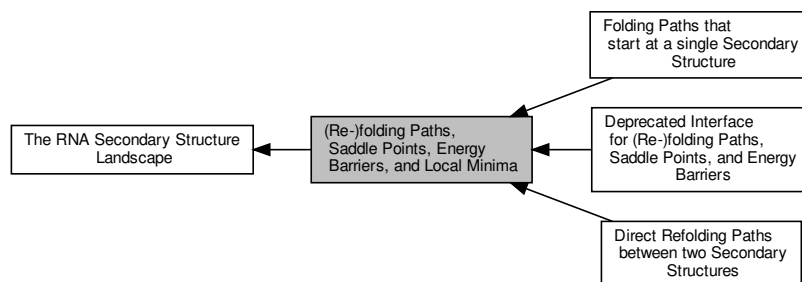
API for various RNA folding path algorithms.

This part of our API allows for generating RNA secondary structure (re-)folding paths between two secondary structures or simply starting from a single structure. This is most important if an estimate of the refolding energy barrier between two structures is required, or a structure's corresponding local minimum needs to be determined, e.g. through a gradient-descent walk.

This part of the interface is further split into the following sections:

- [Direct Refolding Paths between two Secondary Structures](#), and
- [Folding Paths that start at a single Secondary Structure](#)

Collaboration diagram for (Re-)folding Paths, Saddle Points, Energy Barriers, and Local Minima:



## Modules

- [Direct Refolding Paths between two Secondary Structures](#)  
*Heuristics to explore direct, optimal (re-)folding paths between two secondary structures.*
- [Folding Paths that start at a single Secondary Structure](#)  
*Implementation of gradient- and random walks starting from a single secondary structure.*
- [Deprecated Interface for \(Re-\)folding Paths, Saddle Points, and Energy Barriers](#)

## Files

- file [findpath.h](#)  
*A breadth-first search heuristic for optimal direct folding paths.*
- file [paths.h](#)  
*API for computing (optimal) (re-)folding paths between secondary structures.*
- file [walk.h](#)  
*Methods to generate particular paths such as gradient or random walks through the energy landscape of an RNA sequence.*

## Data Structures

- struct `vrna_path_s`  
An element of a refolding path list. [More...](#)

## Macros

- #define `VRNA_PATH_TYPE_DOT_BRACKET` 1U  
Flag to indicate producing a (re-)folding path as list of dot-bracket structures.
- #define `VRNA_PATH_TYPE_MOVES` 2U  
Flag to indicate producing a (re-)folding path as list of transition moves.

## Typedefs

- typedef struct `vrna_path_s` `vrna_path_t`  
Typename for the refolding path data structure `vrna_path_s`.
- typedef struct `vrna_path_options_s` \* `vrna_path_options_t`  
Options data structure for (re-)folding path implementations.

## Functions

- void `vrna_path_free` (`vrna_path_t` \*path)  
Release (free) memory occupied by a (re-)folding path.
- void `vrna_path_options_free` (`vrna_path_options_t` options)  
Release (free) memory occupied by an options data structure for (re-)folding path implementations.

## 16.38.2 Data Structure Documentation

### 16.38.2.1 struct `vrna_path_s`

An element of a refolding path list.

Usually, one has to deal with an array of `vrna_path_s`, e.g. returned from one of the refolding-path algorithms.

Since in most cases the length of the list is not known in advance, such lists have an *end-of-list* marker, which is either:

- a value of `NULL` for `vrna_path_s::s` if `vrna_path_s::type` = `VRNA_PATH_TYPE_DOT_BRACKET`, or
- a `vrna_path_s::move` with zero in both fields `vrna_move_t::pos_5` and `vrna_move_t::pos_3` if `vrna_path_s::type` = `VRNA_PATH_TYPE_MOVES`.

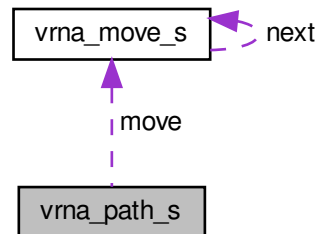
In the following we show an example for how to cover both cases of iteration:

```
vrna_path_t *ptr = path; // path was returned from one of the refolding path functions, e.g.
vrna_path_direct()
if (ptr) {
    if (ptr->type == VRNA_PATH_TYPE_DOT_BRACKET) {
        for (; ptr->s; ptr++)
            printf("%s [%6.2f]\n", ptr->s, ptr->en);
    } else if (ptr->type == VRNA_PATH_TYPE_MOVES) {
        for (; ptr->move.pos_5 != 0; ptr++)
            printf("move %d:%d, dG = %6.2f\n", ptr->move.pos_5, ptr->move.pos_3, ptr->en);
    }
}
```

See also

[vrna\\_path\\_free\(\)](#)

Collaboration diagram for `vrna_path_s`:



## Data Fields

- unsigned int [type](#)  
*The type of the path element.*
- double [en](#)  
*Free energy of current structure.*
- char \* [s](#)  
*Secondary structure in dot-bracket notation.*
- [vrna\\_move\\_t](#) [move](#)  
*Move that transforms the previous structure into it's next neighbor along the path.*

### 16.38.2.1.1 Field Documentation

#### 16.38.2.1.1.1 `type` unsigned int `vrna_path_s::type`

The type of the path element.

A value of `VRNA_PATH_TYPE_DOT_BRACKET` indicates that `vrna_path_s::s` consists of the secondary structure in dot-bracket notation, and `vrna_path_s::en` the corresponding free energy.

On the other hand, if the value is `VRNA_PATH_TYPE_MOVES`, `vrna_path_s::s` is `NULL` and `vrna_path_s::move` is set to the transition move that transforms a previous structure into it's neighbor along the path. In this case, the attribute `vrna_path_s::en` states the change in free energy with respect to the structure before application of `vrna_path_s::move`.

## 16.38.3 Macro Definition Documentation

### 16.38.3.1 VRNA\_PATH\_TYPE\_DOT\_BRACKET

```
#define VRNA_PATH_TYPE_DOT_BRACKET 1U  
  
#include <ViennaRNA/landscape/paths.h>
```

Flag to indicate producing a (re-)folding path as list of dot-bracket structures.

See also

[vrna\\_path\\_t](#), [vrna\\_path\\_options\\_findpath\(\)](#), [vrna\\_path\\_direct\(\)](#), [vrna\\_path\\_direct\\_ub\(\)](#)

### 16.38.3.2 VRNA\_PATH\_TYPE\_MOVES

```
#define VRNA_PATH_TYPE_MOVES 2U  
  
#include <ViennaRNA/landscape/paths.h>
```

Flag to indicate producing a (re-)folding path as list of transition moves.

See also

[vrna\\_path\\_t](#), [vrna\\_path\\_options\\_findpath\(\)](#), [vrna\\_path\\_direct\(\)](#), [vrna\\_path\\_direct\\_ub\(\)](#)

## 16.38.4 Function Documentation

### 16.38.4.1 vrna\_path\_free()

```
void vrna_path_free (  
    vrna_path_t * path )  
  
#include <ViennaRNA/landscape/paths.h>
```

Release (free) memory occupied by a (re-)folding path.

See also

[vrna\\_path\\_direct\(\)](#), [vrna\\_path\\_direct\\_ub\(\)](#), [vrna\\_path\\_findpath\(\)](#), [vrna\\_path\\_findpath\\_ub\(\)](#)

Parameters

<i>path</i>	The refolding path to be free'd
-------------	---------------------------------

#### 16.38.4.2 vrna\_path\_options\_free()

```
void vrna_path_options_free (
    vrna_path_options_t options )
```

```
#include <ViennaRNA/landscape/paths.h>
```

Release (free) memory occupied by an options data structure for (re-)folding path implementations.

See also

[vrna\\_path\\_options\\_findpath\(\)](#), [vrna\\_path\\_direct\(\)](#), [vrna\\_path\\_direct\\_ub\(\)](#)

##### Parameters

<i>options</i>	The options data structure to be free'd
----------------	---



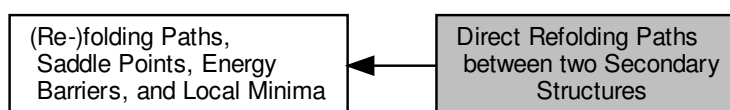
## 16.39 Direct Refolding Paths between two Secondary Structures

Heuristics to explore direct, optimal (re-)folding paths between two secondary structures.

### 16.39.1 Detailed Description

Heuristics to explore direct, optimal (re-)folding paths between two secondary structures.

Collaboration diagram for Direct Refolding Paths between two Secondary Structures:



### Functions

- `int vrna_path_findpath_saddle (vrna_fold_compound_t *fc, 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 *fc, 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 *fc, 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 *fc, const char *s1, const char *s2, int width, int maxE)`  
*Find refolding path between 2 structures (search only direct path)*
- `vrna_path_options_t vrna_path_options_findpath (int width, unsigned int type)`  
*Create options data structure for findpath direct (re-)folding path heuristic.*
- `vrna_path_t * vrna_path_direct (vrna_fold_compound_t *fc, const char *s1, const char *s2, vrna_path_options_t options)`  
*Determine an optimal direct (re-)folding path between two secondary structures.*
- `vrna_path_t * vrna_path_direct_ub (vrna_fold_compound_t *fc, const char *s1, const char *s2, int maxE, vrna_path_options_t options)`  
*Determine an optimal direct (re-)folding path between two secondary structures.*

### 16.39.2 Function Documentation

### 16.39.2.1 `vrna_path_findpath_saddle()`

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

```
#include <ViennaRNA/landscape/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 'fc'. 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:  
`fc = 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>fc</i>	The <code>vrna_fold_compound_t</code> 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_saddle()` to objects of type `fold_compound`. The optional parameter `width` defaults to 1 if it is omitted.

### 16.39.2.2 `vrna_path_findpath_saddle_ub()`

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

```
#include <ViennaRNA/landscape/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 'fc'. 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:  
`fc = 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>fc</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 structures 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  $E_{saddle} < E_{max}$  the function returns a `NULL` object, i.e. `undef` for Perl and `None` for Python.

**16.39.2.3 vrna\_path\_findpath()**

```
vrna_path_t * vrna_path_findpath (
    vrna_fold_compound_t * fc,
    const char * s1,
    const char * s2,
    int width )

#include <ViennaRNA/landscape/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 'fc'. 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:  
`fc = 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>fc</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()* to objects of type *fold\_compound*. The optional parameter *width* defaults to 1 if it is omitted.

## 16.39.2.4 vrna\_path\_findpath\_ub()

```
vrna_path_t * vrna_path_findpath_ub (
    vrna_fold_compound_t * fc,
    const char * s1,
    const char * s2,
    int width,
    int maxE )
```

```
#include <ViennaRNA/landscape/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 'fc'. 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:  
`fc = 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

<i>fc</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()` 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.

16.39.2.5 `vrna_path_options_findpath()`

```
vrna_path_options_t vrna_path_options_findpath (
    int width,
    unsigned int type )
```

```
#include <ViennaRNA/landscape/paths.h>
```

Create options data structure for findpath direct (re-)folding path heuristic.

This function returns an options data structure that switches the `vrna_path_direct()` and `vrna_path_direct_ub()` API functions to use the *findpath* [7] heuristic. The parameter `width` specifies the width of the breadth-first search while the second parameter `type` allows one to set the type of the returned (re-)folding path.

Currently, the following return types are available:

- A list of dot-bracket structures and corresponding free energy (flag: `VRNA_PATH_TYPE_DOT_BRACKET`)
- A list of transition moves and corresponding free energy changes (flag: `VRNA_PATH_TYPE_MOVES`)

## See also

`VRNA_PATH_TYPE_DOT_BRACKET`, `VRNA_PATH_TYPE_MOVES`, `vrna_path_options_free()`, `vrna_path_direct()`, `vrna_path_direct_ub()`

## Parameters

<i>width</i>	Width of the breath-first search strategy
<i>type</i>	Setting that specifies how the return (re-)folding path should be encoded

## Returns

An options data structure with settings for the findpath direct path heuristic

**SWIG Wrapper Notes** This function is available as overloaded function `path_options_findpath()`. The optional parameter `width` defaults to 10 if omitted, while the optional parameter `type` defaults to `VRNA_PATH_TYPE_DOT_BRACKET`.

### 16.39.2.6 vrna\_path\_direct()

```
vrna_path_t * vrna_path_direct (
    vrna_fold_compound_t * fc,
    const char * s1,
    const char * s2,
    vrna_path_options_t options )
```

```
#include <ViennaRNA/landscape/paths.h>
```

Determine an optimal direct (re-)folding path between two secondary structures.

This is the generic wrapper function to retrieve (an optimal) (re-)folding path between two secondary structures *s1* and *s2*. The actual algorithm that is used to generate the (re-)folding path is determined by the settings specified in the *options* data structure. This data structure also determines the return type, which might be either:

- a list of dot-bracket structures with corresponding free energy, or
- a list of transition moves with corresponding free energy change

If the *options* parameter is passed a *NULL* pointer, this function defaults to the *findpath heuristic* [7] with a breadth-first search width of 10, and the returned path consists of dot-bracket structures with corresponding free energies.

See also

[vrna\\_path\\_direct\\_ub\(\)](#), [vrna\\_path\\_options\\_findpath\(\)](#), [vrna\\_path\\_options\\_free\(\)](#), [vrna\\_path\\_free\(\)](#)

#### Parameters

<i>fc</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>options</i>	An options data structure that specifies the path heuristic and corresponding settings (maybe <i>NULL</i> )

#### Returns

An optimal (re-)folding path between the two input structures

**SWIG Wrapper Notes** This function is attached as an overloaded method *path\_direct()* to objects of type *fold\_compound*. The optional parameter *options* defaults to *NULL* if it is omitted.

### 16.39.2.7 vrna\_path\_direct\_ub()

```
vrna_path_t * vrna_path_direct_ub (
    vrna_fold_compound_t * fc,
    const char * s1,
    const char * s2,
```

```

    int maxE,
    vrna_path_options_t options )

#include <ViennaRNA/landscape/paths.h>

```

Determine an optimal direct (re-)folding path between two secondary structures.

This function is similar to [vrna\\_path\\_direct\(\)](#), but allows to specify an *upper-bound* for the saddle point energy. The underlying algorithms will stop determining an (optimal) (re-)folding path, if none can be found that has a saddle point below the specified upper-bound threshold `maxE`.

#### 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\\_direct\\_ub\(\)](#), [vrna\\_path\\_options\\_findpath\(\)](#), [vrna\\_path\\_options\\_free\(\)](#), [vrna\\_path\\_free\(\)](#)

#### Parameters

<i>fc</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>maxE</i>	Upper bound for the saddle point along the (re-)folding path
<i>options</i>	An options data structure that specifies the path heuristic and corresponding settings (maybe <code>NULL</code> )

#### Returns

An optimal (re-)folding path between the two input structures

**SWIG Wrapper Notes** This function is attached as an overloaded method `path_direct()` to objects of type `fold_compound`. The optional parameter `maxE` defaults to `#INT_MAX - 1` if it is omitted, while the optional parameter `options` defaults to `NULL`. In case the function did not find a path with  $E_{saddle} < E_{max}$  it returns an empty list.

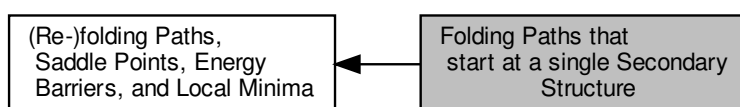
## 16.40 Folding Paths that start at a single Secondary Structure

Implementation of gradient- and random walks starting from a single secondary structure.

### 16.40.1 Detailed Description

Implementation of gradient- and random walks starting from a single secondary structure.

Collaboration diagram for Folding Paths that start at a single Secondary Structure:



### 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.*

### 16.40.2 Macro Definition Documentation



### 16.40.2.1 VRNA\_PATH\_STEEPEST\_DESCENT

```
#define VRNA_PATH_STEEPEST_DESCENT 128
```

```
#include <ViennaRNA/landscape/walk.h>
```

Option flag to request a steepest descent / gradient path.

See also

[vrna\\_path\(\)](#)

### 16.40.2.2 VRNA\_PATH\_RANDOM

```
#define VRNA_PATH_RANDOM 256
```

```
#include <ViennaRNA/landscape/walk.h>
```

Option flag to request a random walk path.

See also

[vrna\\_path\(\)](#)

### 16.40.2.3 VRNA\_PATH\_NO\_TRANSITION\_OUTPUT

```
#define VRNA_PATH_NO_TRANSITION_OUTPUT 512
```

```
#include <ViennaRNA/landscape/walk.h>
```

Option flag to omit returning the transition path.

See also

[vrna\\_path\(\)](#), [vrna\\_path\\_gradient\(\)](#), [vrna\\_path\\_random\(\)](#)

### 16.40.2.4 VRNA\_PATH\_DEFAULT

```
#define VRNA_PATH_DEFAULT (VRNA_PATH_STEEPEST_DESCENT | VRNA_MOVESET_DEFAULT)
```

```
#include <ViennaRNA/landscape/walk.h>
```

Option flag to request defaults (steepest descent / default move set)

See also

[vrna\\_path\(\)](#), [VRNA\\_PATH\\_STEEPEST\\_DESCENT](#), [VRNA\\_MOVESET\\_DEFAULT](#)

## 16.40.3 Function Documentation

### 16.40.3.1 vrna\_path()

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

```
#include <ViennaRNA/landscape/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

<code>in</code>	<code>vc</code>	A <code>vrna_fold_compound_t</code> containing the energy parameters and model details
<code>in, out</code>	<code>pt</code>	The pair table containing the start structure. Used to update to the final structure after execution of this function
<code>in</code>	<code>options</code>	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.

### 16.40.3.2 vrna\_path\_gradient()

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

```
#include <ViennaRNA/landscape/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.

### 16.40.3.3 vrna\_path\_random()

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

```
#include <ViennaRNA/landscape/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.

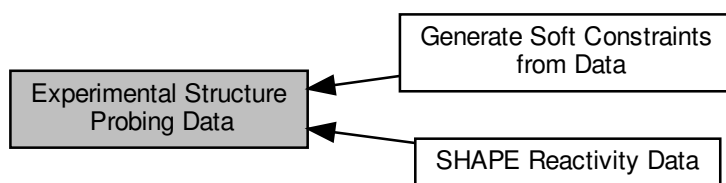
## 16.41 Experimental Structure Probing Data

Include Experimental Structure Probing Data to Guide Structure Predictions.

### 16.41.1 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.*

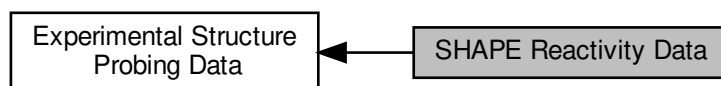
## 16.42 SHAPE Reactivity Data

Incorporate SHAPE reactivity structure probing data into the folding recursions by means of soft constraints.

### 16.42.1 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.*

### 16.42.2 Function Documentation

16.42.2.1 `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 <a href="#">vrna_fold_compound_t</a> 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`

16.42.2.2 `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_al()` to objects of type `fold↔_compound`

### 16.42.2.3 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*

**16.42.2.4 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

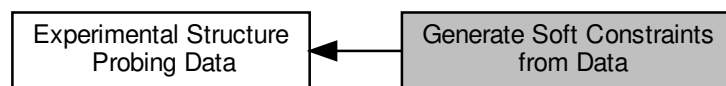
## 16.43 Generate Soft Constraints from Data

Find a vector of perturbation energies that minimizes the discrepancies between predicted and observed pairing probabilities and the amount of necessary adjustments.

### 16.43.1 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.*

## 16.43.2 Macro Definition Documentation

### 16.43.2.1 VRNA\_OBJECTIVE\_FUNCTION\_QUADRATIC

```
#define VRNA_OBJECTIVE_FUNCTION_QUADRATIC 0

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

Use the sum of squared aberrations as objective function.

$$F(\vec{\epsilon}) = \sum_{i=1}^n \frac{\epsilon_i^2}{\tau^2} + \sum_{i=1}^n \frac{(p_i(\vec{\epsilon}) - q_i)^2}{\sigma^2} \rightarrow \min$$

### 16.43.2.2 VRNA\_OBJECTIVE\_FUNCTION\_ABSOLUTE

```
#define VRNA_OBJECTIVE_FUNCTION_ABSOLUTE 1

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

Use the sum of absolute aberrations as objective function.

$$F(\vec{\epsilon}) = \sum_{i=1}^n \frac{|\epsilon_i|}{\tau^2} + \sum_{i=1}^n \frac{|p_i(\vec{\epsilon}) - q_i|}{\sigma^2} \rightarrow \min$$

### 16.43.2.3 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

### 16.43.2.4 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

### 16.43.2.5 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

### 16.43.2.6 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

### 16.43.2.7 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

## 16.43.3 Typedef Documentation

### 16.43.3.1 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

## 16.43.4 Function Documentation

## 16.43.4.1 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 [22].

## Parameters

<i>vc</i>	Pointer to a fold compound
<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

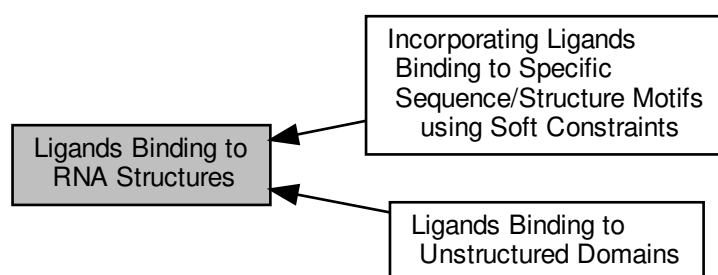
## 16.44 Ligands Binding to RNA Structures

Simple Extensions to Model Ligand Binding to RNA Structures.

### 16.44.1 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](#)

### Files

- file [ligand.h](#)  
*Functions for incorporation of ligands binding to hairpin and interior loop motifs using the soft constraints framework.*

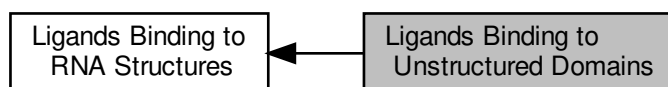
## 16.45 Ligands Binding to Unstructured Domains

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:

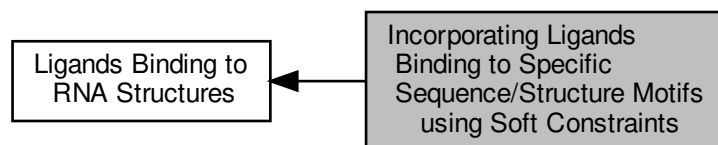




## 16.46 Incorporating Ligands Binding to Specific Sequence/Structure Motifs using Soft Constraints

### 16.46.1 Detailed Description

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.*

### 16.46.2 Function Documentation

#### 16.46.2.1 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

<code>vc</code>	The <code>vrna_fold_compound_t</code> the motif is applied to
<code>seq</code>	The sequence motif (may be interspaced by '&' character)
<code>structure</code>	The structure motif (may be interspaced by '&' character)
<code>energy</code>	The free energy of the motif (e.g. binding free energy)
<code>options</code>	Options

**Returns**

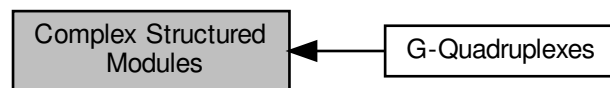
non-zero value if application of the motif using soft constraints was successful

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

## 16.47 Complex Structured Modules

### 16.47.1 Detailed Description

Collaboration diagram for Complex Structured Modules:



#### Modules

- [G-Quadruplexes](#)

*Various functions related to G-quadruplex computations.*

#### Files

- file [gquad.h](#)

*G-quadruplexes.*

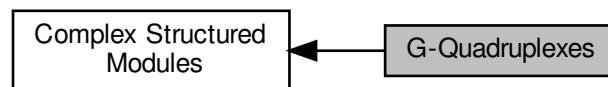
## 16.48 G-Quadruplexes

Various functions related to G-quadruplex computations.

### 16.48.1 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)

## 16.48.2 Function Documentation

### 16.48.2.1 [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

16.48.2.2 `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; ... ;`

16.48.2.3 `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

**16.48.2.4 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

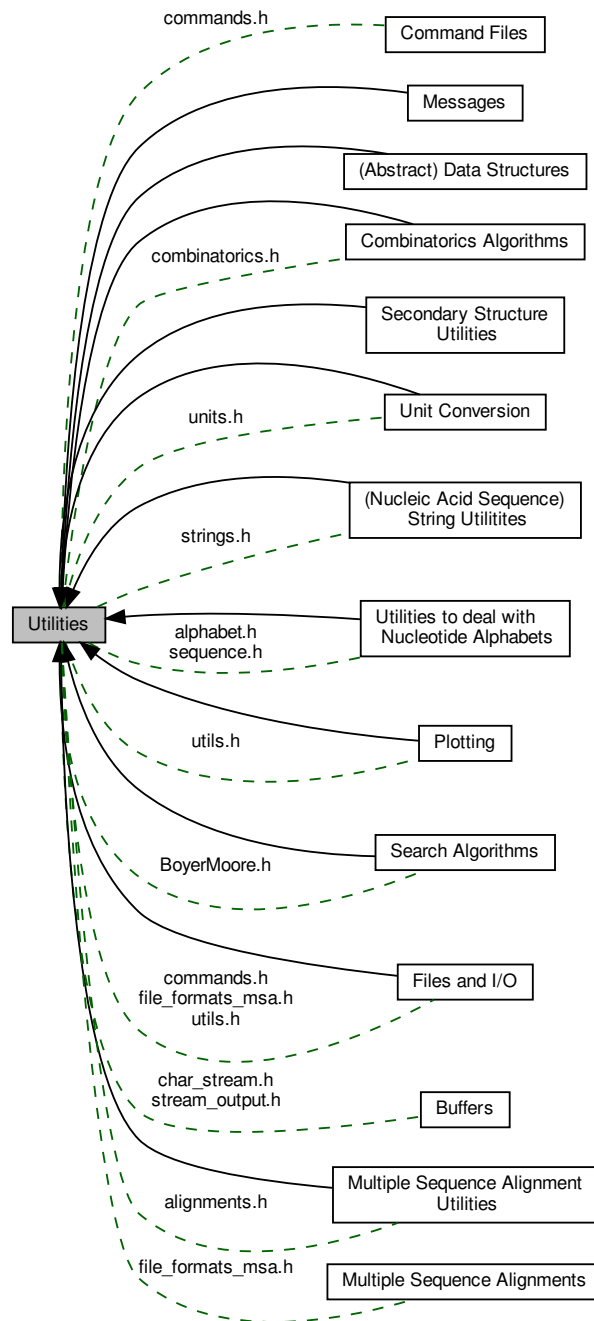
**Returns**

1 on success, 0 if no gquad found

## 16.49 Utilities

### 16.49.1 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 Utilities](#)

*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 [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 [units.h](#)

*Physical Units and Functions to convert them into each other.*
- 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.*

## 16.49.2 Macro Definition Documentation

### 16.49.2.1 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

## 16.49.3 Function Documentation

### 16.49.3.1 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

### 16.49.3.2 `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

### 16.49.3.3 `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]

### 16.49.3.4 `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]

**16.49.3.5 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

**16.49.3.6 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\\_COMMENT](#), [VRNA\\_INPUT\\_NOSKIP\\_COMMENTS](#), [VRNA\\_INPUT\\_NO\\_TRUNCATION](#)

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

**16.49.3.7 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

**16.49.3.8 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

## 16.49.4 Variable Documentation

### 16.49.4.1 xsubi

```
unsigned short xsubi[3] [extern]
```

```
#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\(\)](#)

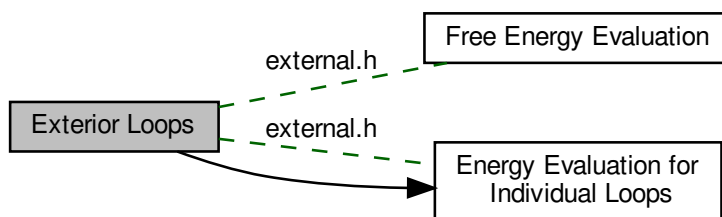
## 16.50 Exterior Loops

Functions to evaluate the free energy contributions for exterior loops.

### 16.50.1 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) ↵

## 16.50.2 Typedef Documentation

### 16.50.2.1 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()

## 16.50.3 Function Documentation

### 16.50.3.1 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

**16.50.3.2 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

**16.50.3.3 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

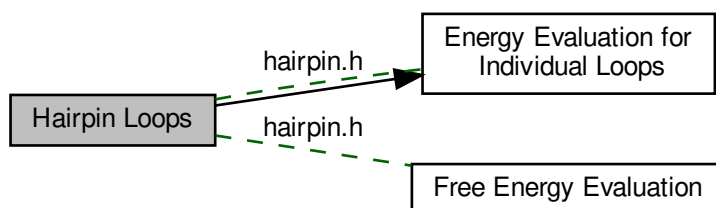
## 16.51 Hairpin Loops

Functions to evaluate the free energy contributions for hairpin loops.

### 16.51.1 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)*

## 16.51.2 Function Documentation

### 16.51.2.1 `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

### 16.51.2.2 `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`

16.51.2.3 `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`

16.51.2.4 `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 <code>NULL</code> , otherwise mst be at least $size + 2$ long)
<i>P</i>	The datastructure containing scaled energy parameters

#### Returns

The Free energy of the Hairpin-loop in dcal/mol

#### 16.51.2.5 exp\_E\_Hairpin()

```
PRIVATE FLT_OR_DBL exp_E_Hairpin (
    int u,
    int type,
    short si1,
    short sj1,
    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 <code>NULL</code> , 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

16.51.2.6 `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`

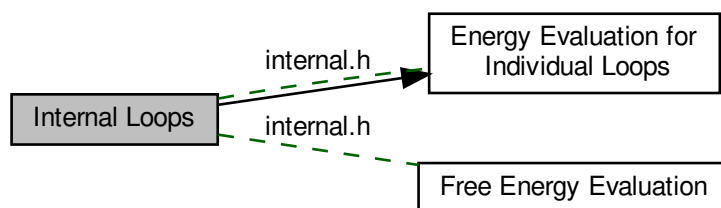
## 16.52 Internal Loops

Functions to evaluate the free energy contributions for internal loops.

### 16.52.1 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)`

### 16.52.2 Function Documentation



### 16.52.2.1 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`

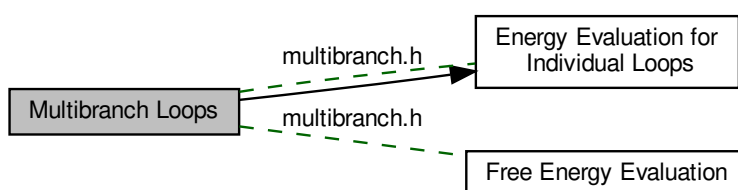
## 16.53 Multibranch Loops

Functions to evaluate the free energy contributions for multibranch loops.

### 16.53.1 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)

## 16.53.2 Typedef Documentation

### 16.53.2.1 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()

## 16.53.3 Function Documentation

### 16.53.3.1 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)

## 16.54 Partition Function for Two Hybridized Sequences

Partition Function Cofolding.

### 16.54.1 Detailed Description

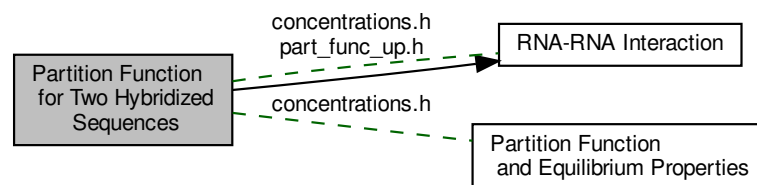
Partition Function Cofolding.

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.*

### 16.54.2 Function Documentation

#### 16.54.2.1 [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.

**16.54.2.2 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

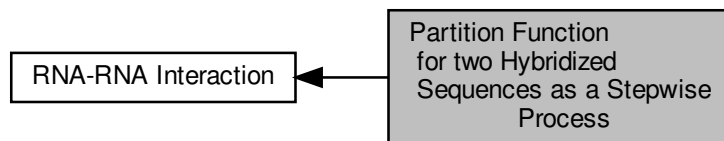
## 16.55 Partition Function for two Hybridized Sequences as a Stepwise Process

RNA-RNA interaction as a stepwise process.

### 16.55.1 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()`.*

### 16.55.2 Function Documentation

### 16.55.2.1 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

### 16.55.2.2 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  $G_{ijkl}$  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'.  $G_{ijkl\_wo}$  is  $G_{ijkl}$  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>←</sub></i>	
<i>_w</i>	
<i>cstruc</i>	
<i>incr3</i>	
<i>incr5</i>	

#### Returns

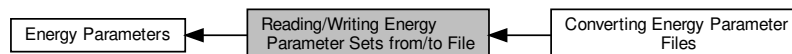
## 16.56 Reading/Writing Energy Parameter Sets from/to File

Read and Write energy parameter sets from and to files or strings.

### 16.56.1 Detailed Description

Read and Write energy parameter sets from and to files or strings.

Collaboration diagram for Reading/Writing Energy Parameter Sets from/to File:



## Modules

- [Converting Energy Parameter Files](#)  
*Convert energy parameter files into the latest format.*

## Macros

- `#define VRNA_PARAMETER_FORMAT_DEFAULT 0`  
*Default Energy Parameter File format.*

## Functions

- `int vrna_params_load` (const char fname[], unsigned int options)  
*Load energy parameters from a file.*
- `int vrna_params_save` (const char fname[], unsigned int options)  
*Save energy parameters to a file.*
- `int vrna_params_load_from_string` (const char \*string, const char \*name, unsigned int options)  
*Load energy parameters from string.*
- `int vrna_params_load_defaults` (void)  
*Load default RNA energy parameter set.*
- `int vrna_params_load_RNA_Turner2004` (void)  
*Load Turner 2004 RNA energy parameter set.*
- `int vrna_params_load_RNA_Turner1999` (void)  
*Load Turner 1999 RNA energy parameter set.*
- `int vrna_params_load_RNA_Andronescu2007` (void)  
*Load Andronescu 2007 RNA energy parameter set.*
- `int vrna_params_load_RNA_Langdon2018` (void)  
*Load Langdon 2018 RNA energy parameter set.*
- `int vrna_params_load_RNA_misc_special_hairpins` (void)  
*Load Misc Special Hairpin RNA energy parameter set.*

- int [vrna\\_params\\_load\\_DNA\\_Mathews2004](#) (void)  
*Load Mathews 2004 DNA energy parameter set.*
- int [vrna\\_params\\_load\\_DNA\\_Mathews1999](#) (void)  
*Load Mathews 1999 DNA energy parameter set.*
- 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.*

## 16.56.2 Macro Definition Documentation

### 16.56.2.1 VRNA\_PARAMETER\_FORMAT\_DEFAULT

```
#define VRNA_PARAMETER_FORMAT_DEFAULT 0

#include <ViennaRNA/params/io.h>
```

Default Energy Parameter File format.

See also

[vrna\\_params\\_load\(\)](#), [vrna\\_params\\_load\\_from\\_string\(\)](#), [vrna\\_params\\_save\(\)](#)

## 16.56.3 Function Documentation

### 16.56.3.1 vrna\_params\_load()

```
int vrna_params_load (
    const char fname[],
    unsigned int options )

#include <ViennaRNA/params/io.h>
```

Load energy parameters from a file.

See also

[vrna\\_params\\_load\\_from\\_string\(\)](#), [vrna\\_params\\_save\(\)](#), [vrna\\_params\\_load\\_defaults\(\)](#), [vrna\\_params\\_load\\_RNA\\_Turner2004\(\)](#), [vrna\\_params\\_load\\_RNA\\_Turner1999\(\)](#), [vrna\\_params\\_load\\_RNA\\_Andronescu2007\(\)](#), [vrna\\_params\\_load\\_RNA\\_Langdon2018\(\)](#), [vrna\\_params\\_load\\_RNA\\_misc\\_special\\_hairpins\(\)](#), [vrna\\_params\\_load\\_DNA\\_Mathews2004\(\)](#), [vrna\\_params\\_load\\_DNA\\_Mathev](#)

## Parameters

<i>fname</i>	The path to the file containing the energy parameters
<i>options</i>	File format bit-mask (usually <a href="#">VRNA_PARAMETER_FORMAT_DEFAULT</a> )

## Returns

Non-zero on success, 0 on failure

**SWIG Wrapper Notes** This function is available as overloaded function **params\_load**(fname="", options=[VRNA\\_PARAMETER\\_FORMAT\\_DEFAULT](#)). Here, the empty filename string indicates to load default RNA parameters, i.e. this is equivalent to calling [vrna\\_params\\_load\\_defaults\(\)](#).

**16.56.3.2 vrna\_params\_save()**

```
int vrna_params_save (
    const char fname[],
    unsigned int options )
```

```
#include <ViennaRNA/params/io.h>
```

Save energy parameters to a file.

## See also

[vrna\\_params\\_load\(\)](#)

## Parameters

<i>fname</i>	A filename (path) for the file where the current energy parameters will be written to
<i>options</i>	File format bit-mask (usually <a href="#">VRNA_PARAMETER_FORMAT_DEFAULT</a> )

## Returns

Non-zero on success, 0 on failure

**SWIG Wrapper Notes** This function is available as overloaded function **params\_save**(fname, options=[VRNA\\_PARAMETER\\_FORMAT\\_DEFAULT](#)).

**16.56.3.3 vrna\_params\_load\_from\_string()**

```
int vrna_params_load_from_string (
    const char * string,
```

```

    const char * name,
    unsigned int options )

#include <ViennaRNA/params/io.h>

```

Load energy paramters from string.

The string must follow the default energy parameter file convention! The optional `name` argument allows one to specify a name for the parameter set which is stored internally.

#### See also

[vrna\\_params\\_load\(\)](#), [vrna\\_params\\_save\(\)](#), [vrna\\_params\\_load\\_defaults\(\)](#), [vrna\\_params\\_load\\_RNA\\_Turner2004\(\)](#), [vrna\\_params\\_load\\_RNA\\_Turner1999\(\)](#), [vrna\\_params\\_load\\_RNA\\_Andronescu2007\(\)](#), [vrna\\_params\\_load\\_RNA\\_Langdon2018\(\)](#), [vrna\\_params\\_load\\_RNA\\_misc\\_special\\_hairpins\(\)](#), [vrna\\_params\\_load\\_DNA\\_Mathews2004\(\)](#), [vrna\\_params\\_load\\_DNA\\_Mathev](#)

#### Parameters

<i>string</i>	A 0-terminated string containing energy parameters
<i>name</i>	A name for the parameter set in <i>string</i> (Maybe NULL)
<i>options</i>	File format bit-mask (usually <a href="#">VRNA_PARAMETER_FORMAT_DEFAULT</a> )

#### Returns

Non-zero on success, 0 on failure

**SWIG Wrapper Notes** This function is available as overloaded function `params_load_from_string(string, name="", options=VRNA\_PARAMETER\_FORMAT\_DEFAULT)`.

#### 16.56.3.4 vrna\_params\_load\_defaults()

```

int vrna_params_load_defaults (
    void )

#include <ViennaRNA/params/io.h>

```

Load default RNA energy parameter set.

This is a convenience function to load the Turner 2004 RNA free energy parameters. It's the same as calling [vrna\\_params\\_load\\_RNA\\_Turner2004\(\)](#)

#### See also

[vrna\\_params\\_load\(\)](#), [vrna\\_params\\_load\\_from\\_string\(\)](#), [vrna\\_params\\_save\(\)](#), [vrna\\_params\\_load\\_RNA\\_Turner2004\(\)](#), [vrna\\_params\\_load\\_RNA\\_Turner1999\(\)](#), [vrna\\_params\\_load\\_RNA\\_Andronescu2007\(\)](#), [vrna\\_params\\_load\\_RNA\\_Langdon2018\(\)](#), [vrna\\_params\\_load\\_RNA\\_misc\\_special\\_hairpins\(\)](#), [vrna\\_params\\_load\\_DNA\\_Mathews2004\(\)](#), [vrna\\_params\\_load\\_DNA\\_Mathev](#)

#### Returns

Non-zero on success, 0 on failure

**SWIG Wrapper Notes** This function is available as overloaded function `params_load()`.

### 16.56.3.5 vrna\_params\_load\_RNA\_Turner2004()

```
int vrna_params_load_RNA_Turner2004 (
    void )

#include <ViennaRNA/params/io.h>
```

Load Turner 2004 RNA energy parameter set.

#### See also

[vrna\\_params\\_load\(\)](#), [vrna\\_params\\_load\\_from\\_string\(\)](#), [vrna\\_params\\_save\(\)](#), [vrna\\_params\\_load\\_defaults\(\)](#), [vrna\\_params\\_load\\_RNA\\_Turner1999\(\)](#), [vrna\\_params\\_load\\_RNA\\_Andronescu2007\(\)](#), [vrna\\_params\\_load\\_RNA\\_Langdon2018\(\)](#), [vrna\\_params\\_load\\_RNA\\_misc\\_special\\_hairpins\(\)](#), [vrna\\_params\\_load\\_DNA\\_Mathews2004\(\)](#), [vrna\\_params\\_load\\_DNA\\_Mathev](#)

#### Returns

Non-zero on success, 0 on failure

**SWIG Wrapper Notes** This function is available as function `params_load_RNA_Turner2004()`.

### 16.56.3.6 vrna\_params\_load\_RNA\_Turner1999()

```
int vrna_params_load_RNA_Turner1999 (
    void )

#include <ViennaRNA/params/io.h>
```

Load Turner 1999 RNA energy parameter set.

#### See also

[vrna\\_params\\_load\(\)](#), [vrna\\_params\\_load\\_from\\_string\(\)](#), [vrna\\_params\\_save\(\)](#), [vrna\\_params\\_load\\_RNA\\_Turner2004\(\)](#), [vrna\\_params\\_load\\_defaults\(\)](#), [vrna\\_params\\_load\\_RNA\\_Andronescu2007\(\)](#), [vrna\\_params\\_load\\_RNA\\_Langdon2018\(\)](#), [vrna\\_params\\_load\\_RNA\\_misc\\_special\\_hairpins\(\)](#), [vrna\\_params\\_load\\_DNA\\_Mathews2004\(\)](#), [vrna\\_params\\_load\\_DNA\\_Mathev](#)

#### Returns

Non-zero on success, 0 on failure

**SWIG Wrapper Notes** This function is available as function `params_load_RNA_Turner1999()`.

**16.56.3.7 vrna\_params\_load\_RNA\_Andronescu2007()**

```
int vrna_params_load_RNA_Andronescu2007 (
    void )
```

```
#include <ViennaRNA/params/io.h>
```

Load Andronescu 2007 RNA energy parameter set.

**See also**

[vrna\\_params\\_load\(\)](#), [vrna\\_params\\_load\\_from\\_string\(\)](#), [vrna\\_params\\_save\(\)](#), [vrna\\_params\\_load\\_RNA\\_Turner2004\(\)](#), [vrna\\_params\\_load\\_RNA\\_Turner1999\(\)](#), [vrna\\_params\\_load\\_defaults\(\)](#), [vrna\\_params\\_load\\_RNA\\_Langdon2018\(\)](#), [vrna\\_params\\_load\\_RNA\\_misc\\_special\\_hairpins\(\)](#), [vrna\\_params\\_load\\_DNA\\_Mathews2004\(\)](#), [vrna\\_params\\_load\\_DNA\\_Mathev](#)

**Returns**

Non-zero on success, 0 on failure

**SWIG Wrapper Notes** This function is available as function `params_load_RNA_Andronescu2007()`.

**16.56.3.8 vrna\_params\_load\_RNA\_Langdon2018()**

```
int vrna_params_load_RNA_Langdon2018 (
    void )
```

```
#include <ViennaRNA/params/io.h>
```

Load Langdon 2018 RNA energy parameter set.

**See also**

[vrna\\_params\\_load\(\)](#), [vrna\\_params\\_load\\_from\\_string\(\)](#), [vrna\\_params\\_save\(\)](#), [vrna\\_params\\_load\\_RNA\\_Turner2004\(\)](#), [vrna\\_params\\_load\\_RNA\\_Turner1999\(\)](#), [vrna\\_params\\_load\\_RNA\\_Andronescu2007\(\)](#), [vrna\\_params\\_load\\_defaults\(\)](#), [vrna\\_params\\_load\\_RNA\\_misc\\_special\\_hairpins\(\)](#), [vrna\\_params\\_load\\_DNA\\_Mathews2004\(\)](#), [vrna\\_params\\_load\\_DNA\\_Mathev](#)

**Returns**

Non-zero on success, 0 on failure

**SWIG Wrapper Notes** This function is available as function `params_load_RNA_Langdon2018()`.

### 16.56.3.9 vrna\_params\_load\_RNA\_misc\_special\_hairpins()

```
int vrna_params_load_RNA_misc_special_hairpins (  
    void )
```

```
#include <ViennaRNA/params/io.h>
```

Load Misc Special Hairpin RNA energy parameter set.

#### See also

[vrna\\_params\\_load\(\)](#), [vrna\\_params\\_load\\_from\\_string\(\)](#), [vrna\\_params\\_save\(\)](#), [vrna\\_params\\_load\\_RNA\\_Turner2004\(\)](#), [vrna\\_params\\_load\\_RNA\\_Turner1999\(\)](#), [vrna\\_params\\_load\\_RNA\\_Andronescu2007\(\)](#), [vrna\\_params\\_load\\_RNA\\_Langdon2018\(\)](#), [vrna\\_params\\_load\\_defaults\(\)](#), [vrna\\_params\\_load\\_DNA\\_Mathews2004\(\)](#), [vrna\\_params\\_load\\_DNA\\_Mathews1999\(\)](#)

#### Returns

Non-zero on success, 0 on failure

**SWIG Wrapper Notes** This function is available as function `params_load_RNA_misc_special_hairpins()`.

### 16.56.3.10 vrna\_params\_load\_DNA\_Mathews2004()

```
int vrna_params_load_DNA_Mathews2004 (  
    void )
```

```
#include <ViennaRNA/params/io.h>
```

Load Mathews 2004 DNA energy parameter set.

#### See also

[vrna\\_params\\_load\(\)](#), [vrna\\_params\\_load\\_from\\_string\(\)](#), [vrna\\_params\\_save\(\)](#), [vrna\\_params\\_load\\_RNA\\_Turner2004\(\)](#), [vrna\\_params\\_load\\_RNA\\_Turner1999\(\)](#), [vrna\\_params\\_load\\_RNA\\_Andronescu2007\(\)](#), [vrna\\_params\\_load\\_RNA\\_Langdon2018\(\)](#), [vrna\\_params\\_load\\_RNA\\_misc\\_special\\_hairpins\(\)](#), [vrna\\_params\\_load\\_defaults\(\)](#), [vrna\\_params\\_load\\_DNA\\_Mathews1999\(\)](#)

#### Returns

Non-zero on success, 0 on failure

**SWIG Wrapper Notes** This function is available as function `params_load_DNA_Mathews2004()`.



**16.56.3.11 vrna\_params\_load\_DNA\_Mathews1999()**

```
int vrna_params_load_DNA_Mathews1999 (
    void )

#include <ViennaRNA/params/io.h>
```

Load Mathews 1999 DNA energy parameter set.

**See also**

[vrna\\_params\\_load\(\)](#), [vrna\\_params\\_load\\_from\\_string\(\)](#), [vrna\\_params\\_save\(\)](#), [vrna\\_params\\_load\\_RNA\\_Turner2004\(\)](#), [vrna\\_params\\_load\\_RNA\\_Turner1999\(\)](#), [vrna\\_params\\_load\\_RNA\\_Andronescu2007\(\)](#), [vrna\\_params\\_load\\_RNA\\_Langdon2018\(\)](#), [vrna\\_params\\_load\\_RNA\\_misc\\_special\\_hairpins\(\)](#), [vrna\\_params\\_load\\_DNA\\_Mathews2004\(\)](#), [vrna\\_params\\_load\\_defaults\(\)](#)

**Returns**

Non-zero on success, 0 on failure

**SWIG Wrapper Notes** This function is available as function `params_load_DNA_Mathews1999()`.

**16.56.3.12 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

**16.56.3.13 read\_parameter\_file()**

```
void read_parameter_file (
    const char fname[] )

#include <ViennaRNA/params/io.h>
```

Read energy parameters from a file.

**Deprecated** Use [vrna\\_params\\_load\(\)](#) instead!

## Parameters

<i>fname</i>	The path to the file containing the energy parameters
--------------	---

**16.56.3.14 write\_parameter\_file()**

```
void write_parameter_file (  
    const char fname[] )
```

```
#include <ViennaRNA/params/io.h>
```

Write energy parameters to a file.

**Deprecated** Use [vrna\\_params\\_save\(\)](#) instead!

## Parameters

<i>fname</i>	A filename (path) for the file where the current energy parameters will be written to
--------------	---

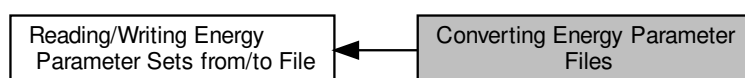
## 16.57 Converting Energy Parameter Files

Convert energy parameter files into the latest format.

### 16.57.1 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)

## 16.57.2 Macro Definition Documentation

### 16.57.2.1 VRNA\_CONVERT\_OUTPUT\_ALL

```
#define VRNA_CONVERT_OUTPUT_ALL 1U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of a complete parameter set

### 16.57.2.2 VRNA\_CONVERT\_OUTPUT\_HP

```
#define VRNA_CONVERT_OUTPUT_HP 2U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of hairpin contributions

### 16.57.2.3 VRNA\_CONVERT\_OUTPUT\_STACK

```
#define VRNA_CONVERT_OUTPUT_STACK 4U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of base pair stack contributions

### 16.57.2.4 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

### 16.57.2.5 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

**16.57.2.6 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

**16.57.2.7 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

**16.57.2.8 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

**16.57.2.9 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

**16.57.2.10 VRNA\_CONVERT\_OUTPUT\_DANGLE5**

```
#define VRNA_CONVERT_OUTPUT_DANGLE5 512U
```

```
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of 5' dangle contribution

**16.57.2.11 VRNA\_CONVERT\_OUTPUT\_DANGLE3**

```
#define VRNA_CONVERT_OUTPUT_DANGLE3 1024U
```

```
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of 3' dangle contribution

**16.57.2.12 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

**16.57.2.13 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

**16.57.2.14 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

**16.57.2.15 VRNA\_CONVERT\_OUTPUT\_BULGE**

```
#define VRNA_CONVERT_OUTPUT_BULGE 16384U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of bulge loop contribution

**16.57.2.16 VRNA\_CONVERT\_OUTPUT\_INT**

```
#define VRNA_CONVERT_OUTPUT_INT 32768U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of interior loop contribution

**16.57.2.17 VRNA\_CONVERT\_OUTPUT\_ML**

```
#define VRNA_CONVERT_OUTPUT_ML 65536U  
  
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of multi loop contribution

**16.57.2.18 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)

**16.57.2.19 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)

**16.57.2.20 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](#) !

**16.57.2.21 VRNA\_CONVERT\_OUTPUT\_NINIO**

```
#define VRNA_CONVERT_OUTPUT_NINIO 1048576U
```

```
#include <ViennaRNA/params/convert.h>
```

Flag to indicate printing of interior loop asymmetry contribution

**16.57.2.22 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

## 16.57.3 Function Documentation

### 16.57.3.1 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)



## 16.58 Utilities to deal with Nucleotide Alphabets

Functions to cope with various aspects related to the nucleotide sequence alphabet.

### 16.58.1 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...](#)*
- struct [vrna\\_alignment\\_s](#)

### 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.*

## 16.58.2 Data Structure Documentation

### 16.58.2.1 struct vrna\_sequence\_s

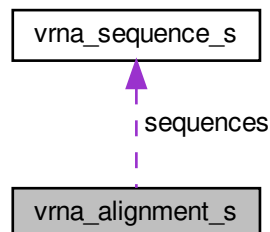
Data structure representing a nucleotide sequence.

#### 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.*

### 16.58.2.2 struct vrna\_alignment\_s

Collaboration diagram for vrna\_alignment\_s:



## 16.58.3 Enumeration Type Documentation

### 16.58.3.1 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.

## 16.58.4 Function Documentation

### 16.58.4.1 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](#)

### 16.58.4.2 vrna\_seq\_encode()

```
short * vrna_seq_encode (
    const char * sequence,
    vrna_md_t * md )
#include <ViennaRNA/alphabet.h>
```

Get a numerical representation of the nucleotide sequence.

## Parameters

<i>sequence</i>	The input sequence in upper-case letters
<i>md</i>	A pointer to a <a href="#">vrna_md_t</a> data structure that specifies the conversion type

## Returns

A list of integer encodings for each sequence letter (1-based). Position 0 denotes the length of the list

**SWIG Wrapper Notes** In the target scripting language, this function is wrapped as overloaded function `seq_↔ encode()` where the last parameter, the *model\_details* data structure, is optional. If it is omitted, default model settings are applied, i.e. default nucleotide letter conversion. The wrapped function returns a list/tuple of integer representations of the input sequence.

**16.58.4.3 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↔ NAlib.

## 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

**16.58.4.4 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

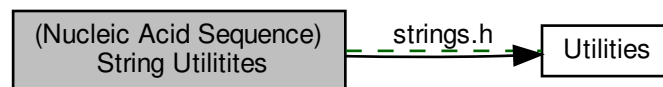
## 16.59 (Nucleic Acid Sequence) String Utilites

Functions to parse, convert, manipulate, create, and compare (nucleic acid sequence) strings.

### 16.59.1 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 \*sequence)  
*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.*

## 16.59.2 Macro Definition Documentation

### 16.59.2.1 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.

### 16.59.2.2 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

## 16.59.3 Function Documentation

### 16.59.3.1 `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 <i>asprintf</i> )
...	The list of variables used to fill the format string

#### Returns

The formatted, null-terminated string, or NULL if something has gone wrong

### 16.59.3.2 `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

**16.59.3.3 `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 allocates 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 <code>char *</code> 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

**16.59.3.4 `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 <a href="#">sprintf</a> )
<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

### 16.59.3.5 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&AAAAA", 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

**16.59.3.6 `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

**16.59.3.7 `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*

**16.59.3.8 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*

**16.59.3.9 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
-----------------	------------------------------

**16.59.3.10 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
-----------------	------------------------------

**16.59.3.11 vrna\_seq\_ungapped()**

```
char* vrna_seq_ungapped (
    const char * sequence )
```

```
#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

**16.59.3.12 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

**16.59.3.13 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

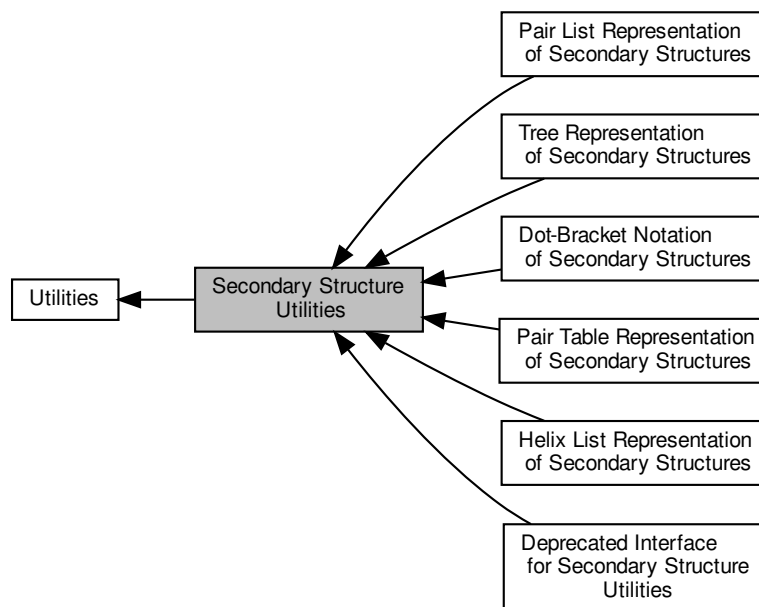
## 16.60 Secondary Structure Utilities

Functions to create, parse, convert, manipulate, and compare secondary structure representations.

### 16.60.1 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.*

## 16.60.2 Function Documentation

### 16.60.2.1 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

### 16.60.2.2 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!!!`

### 16.60.2.3 `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!!!`

### 16.60.2.4 `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

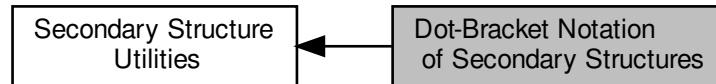
#### Returns

The secondary structure in dot-bracket notation as provided in the input

## 16.61 Dot-Bracket Notation of Secondary Structures

### 16.61.1 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.*
- `#define VRNA_BRACKETS_ANY`  
*Bitmask to indicate secondary structure notation using any pair of brackets or uppercase/lowercase alphabet letters.*

### 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.*
- `char * vrna_db_pk_remove (const char *structure, unsigned int options)`  
*Remove pseudo-knots from an input structure.*

## 16.61.2 Macro Definition Documentation

### 16.61.2.1 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\(\)](#)

### 16.61.2.2 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\(\)](#)

### 16.61.2.3 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\(\)](#)

#### 16.61.2.4 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\(\)](#)

#### 16.61.2.5 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\(\)](#)

#### 16.61.2.6 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.

This set of matching brackets/parenthesis is always nested, i.e. pseudo-knot free, in WUSS format. However, in general different kinds of brackets are mostly used for annotating pseudo-knots. Thus special care has to be taken to remove pseudo-knots if this bitmask is used in functions that return secondary structures without pseudo-knots!

See also

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

### 16.61.2.7 VRNA\_BRACKETS\_ANY

```
#define VRNA_BRACKETS_ANY
```

```
#include <ViennaRNA/utils/structures.h>
```

**Value:**

```
(VRNA_BRACKETS_RND | \
 VRNA_BRACKETS_CLY | \
 VRNA_BRACKETS_ANG | \
 VRNA_BRACKETS_SQR | \
 VRNA_BRACKETS_ALPHA)
```

Bitmask to indicate secondary structure notation using any pair of brackets or uppercase/lowercase alphabet letters.

**See also**

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

## 16.61.3 Function Documentation

### 16.61.3.1 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

### 16.61.3.2 `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

### 16.61.3.3 `vrna_db_flatten()`

```
void 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\(\)](#).

#### 16.61.3.4 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\(\)](#)

#### 16.61.3.5 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

**16.61.3.6 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

**16.61.3.7 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

**16.61.3.8 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

**16.61.3.9 vrna\_db\_pk\_remove()**

```
char * vrna_db_pk_remove (
    const char * structure,
    unsigned int options )
```

```
#include <ViennaRNA/utils/structures.h>
```

Remove pseudo-knots from an input structure.

This function removes pseudo-knots from an input structure by determining the minimum number of base pairs that need to be removed to make the structure pseudo-knot free.

To accomplish that, we use a dynamic programming algorithm similar to the Nussinov maximum matching approach.

The input structure must be in a dot-bracket string like form where crossing base pairs are denoted by the use of additional types of matching brackets, e.g. <>, {}, [], {}. Furthermore, crossing pairs may be annotated by matching uppercase/lowercase letters from the alphabet A–Z. For the latter, the uppercase letter must be the 5' and the lowercase letter the 3' nucleotide of the base pair. The actual type of brackets to be recognized by this function must be specified through the *options* parameter.

**Note**

Brackets in the input structure string that are not covered by the `options` bitmask will be silently ignored!

**See also**

[vrna\\_pt\\_pk\\_remove\(\)](#), [vrna\\_db\\_flatten\(\)](#), [VRNA\\_BRACKETS\\_RND](#), [VRNA\\_BRACKETS\\_ANG](#), [VRNA\\_BRACKETS\\_CLY](#), [VRNA\\_BRACKETS\\_SQR](#), [VRNA\\_BRACKETS\\_ALPHA](#), [VRNA\\_BRACKETS\\_DEFAULT](#), [VRNA\\_BRACKETS\\_ANY](#)

**Parameters**

<i>structure</i>	Input structure in dot-bracket format that may include pseudo-knots
<i>options</i>	A bitmask to specify which types of brackets should be processed

**Returns**

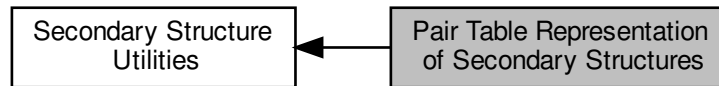
The input structure devoid of pseudo-knots in dot-bracket notation

**SWIG Wrapper Notes** This function is available as an overloaded function `db_pk_remove()` where the optional second parameter `options` defaults to [VRNA\\_BRACKETS\\_ANY](#).

## 16.62 Pair Table Representation of Secondary Structures

### 16.62.1 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\\_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)*
- short \* [vrna\\_pt\\_pk\\_remove](#) (const short \*ptable, unsigned int options)  
*Remove pseudo-knots from a pair table.*

### 16.62.2 Function Documentation

#### 16.62.2.1 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

**16.62.2.2 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.

**Note**

This function also extracts crossing base pairs, i.e. pseudo-knots if more than a single matching bracket type is allowed through the bitmask *options*.

**See also**

[vrna\\_ptable\(\)](#), [vrna\\_db\\_from\\_ptable\(\)](#), [vrna\\_db\\_flatten\\_to\(\)](#), [vrna\\_pt\\_pk\\_remove\(\)](#) [VRNA\\_BRACKETS\\_RND](#), [VRNA\\_BRACKETS\\_ANG](#), [VRNA\\_BRACKETS\\_CLY](#), [VRNA\\_BRACKETS\\_SQR](#), [VRNA\\_BRACKETS\\_ALPHA](#), [VRNA\\_BRACKETS\\_DEFAULT](#), [VRNA\\_BRACKETS\\_ANY](#)

**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

**16.62.2.3 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. Thus, this function behaves like

```
vrna_ptable_from_string(structure, #VRNA_BRACKET_RND | VRNA_BRACKETS_SQR)
```

See also

[vrna\\_ptable\\_from\\_string\(\)](#)

#### Parameters

<i>structure</i>	The secondary structure in (extended) dot-bracket notation
------------------	--

#### Returns

A pointer to the created `pair_table`

#### 16.62.2.4 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'

#### 16.62.2.5 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.

### 16.62.2.6 `vrna_pt_pk_remove()`

```
short* vrna_pt_pk_remove (
    const short * ptable,
    unsigned int options )

#include <ViennaRNA/utils/structures.h>
```

Remove pseudo-knots from a pair table.

This function removes pseudo-knots from an input structure by determining the minimum number of base pairs that need to be removed to make the structure pseudo-knot free.

To accomplish that, we use a dynamic programming algorithm similar to the Nussinov maximum matching approach.

See also

[vrna\\_db\\_pk\\_remove\(\)](#)

#### Parameters

<i>ptable</i>	Input structure that may include pseudo-knots
<i>options</i>	

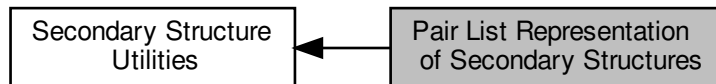
#### Returns

The input structure devoid of pseudo-knots

## 16.63 Pair List Representation of Secondary Structures

### 16.63.1 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.*

## 16.63.2 Data Structure Documentation

### 16.63.2.1 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.*

## 16.63.3 Function Documentation

### 16.63.3.1 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

<i>struc</i>	The secondary structure in dot-bracket notation
<i>pr</i>	The probability for each base pair used in the plist



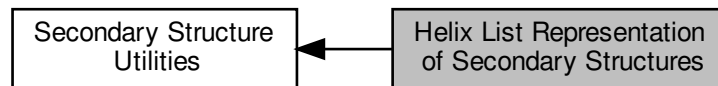
**Returns**

The plist array

## 16.64 Helix List Representation of Secondary Structures

### 16.64.1 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.

### 16.64.2 Data Structure Documentation

#### 16.64.2.1 struct vrna\_hx\_s

Data structure representing an entry of a helix list.

### 16.64.3 Function Documentation

#### 16.64.3.1 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**

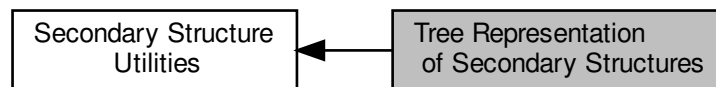
The secondary structure represented as a helix list

## 16.65 Tree Representation of Secondary Structures

### 16.65.1 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.*

### 16.65.2 Macro Definition Documentation

### 16.65.2.1 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\(\)](#)

### 16.65.2.2 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\(\)](#)

### 16.65.2.3 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\(\)](#)

### 16.65.2.4 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\(\)](#)

### 16.65.2.5 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\(\)](#)

### 16.65.2.6 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\(\)](#)

## 16.65.3 Function Documentation

### 16.65.3.1 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 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  $\mathbb{E}$ )
- [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`

### 16.65.3.2 `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

### 16.65.3.3 `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`



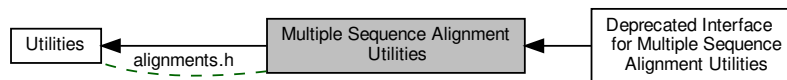
## 16.66 Multiple Sequence Alignment Utilities

Functions to extract features from and to manipulate multiple sequence alignments.

### 16.66.1 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.*

## 16.66.2 Data Structure Documentation

### 16.66.2.1 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*

**16.66.3 Macro Definition Documentation****16.66.3.1 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  $H = - \sum_c p_c \cdot \log_2 p_c$

**16.66.4 Function Documentation****16.66.4.1 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**

<code>alignment</code>	Aligned sequences
------------------------	-------------------

**Returns**

The mean pairwise identity

**16.66.4.2 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

**16.66.4.3 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*

**16.66.4.4 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
------------------	---------------------

**16.66.4.5 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

#### 16.66.4.6 `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)

#### 16.66.4.7 `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

**16.66.4.8 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*

**16.66.4.9 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.

**16.66.4.10 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

**16.66.4.11 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

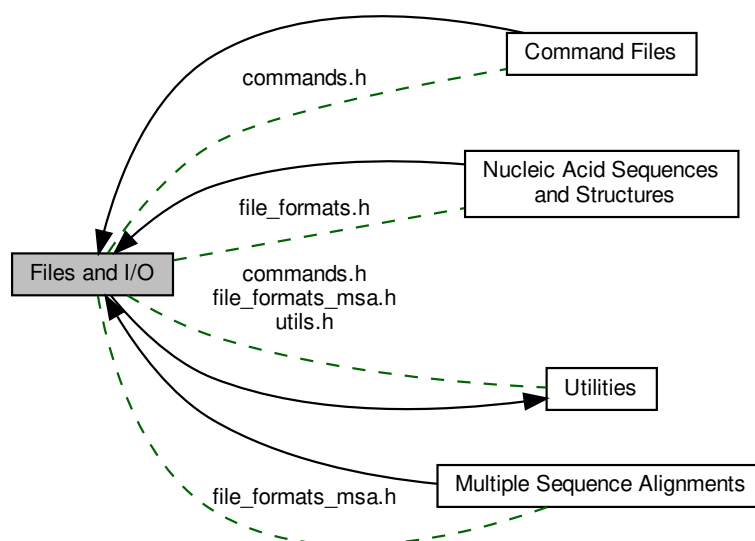
## 16.67 Files and I/O

Functions to parse, write, and convert various file formats and to deal with file system related issues.

### 16.67.1 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 \*\*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.*
- 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.*

## 16.67.2 Function Documentation

### 16.67.2.1 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

### 16.67.2.2 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

**16.67.2.3 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 exists, 1 otherwise

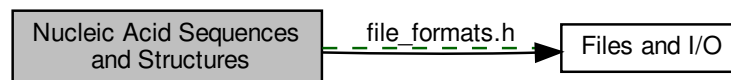
## 16.68 Nucleic Acid Sequences and Structures

Functions to read/write different file formats for nucleic acid sequences and secondary structures.

### 16.68.1 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)
- 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.*

## 16.68.2 Macro Definition Documentation

### 16.68.2.1 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\(\)](#)

### 16.68.2.2 VRNA\_CONSTRAINT\_MULTILINE

```
#define VRNA_CONSTRAINT_MULTILINE 32U
```

```
#include <ViennaRNA/io/file_formats.h>
```

parse multiline constraint

**Deprecated** see [vrna\\_extract\\_record\\_rest\\_structure\(\)](#)

## 16.68.3 Function Documentation

### 16.68.3.1 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) )

**16.68.3.2 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) )



### 16.68.3.3 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) )

### 16.68.3.4 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) )

### 16.68.3.5 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>
```

@brief 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  
 @verbatim >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.:

```
@endverbatim
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

**16.68.3.6 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

**16.68.3.7 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

### 16.68.3.8 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\\_VR](#), [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

### 16.68.3.9 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.

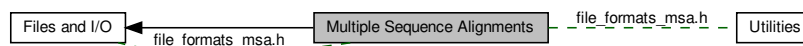
## 16.69 Multiple Sequence Alignments

Functions to read/write multiple sequence alignments (MSA) in various file formats.

### 16.69.1 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.*

## 16.69.2 Macro Definition Documentation

### 16.69.2.1 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\(\)](#)

### 16.69.2.2 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\(\)](#)

### 16.69.2.3 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\(\)](#)

### 16.69.2.4 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\(\)](#)

### 16.69.2.5 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\(\)](#)



### 16.69.2.6 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\(\)](#)

### 16.69.2.7 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\(\)](#)

### 16.69.2.8 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\(\)](#)

### 16.69.2.9 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\(\)](#)

### 16.69.2.10 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\(\)](#)

### 16.69.2.11 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\(\)](#)

## 16.69.3 Function Documentation

### 16.69.3.1 vrna\_file\_msa\_read()

```
int 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`.

### 16.69.3.2 vrna\_file\_msa\_read\_record()

```
int 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\\_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](#).

**16.69.3.3 vrna\_file\_msa\_detect\_format()**

```
unsigned int 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](#).

### 16.69.3.4 vrna\_file\_msa\_write()

```
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 )

#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](#)).

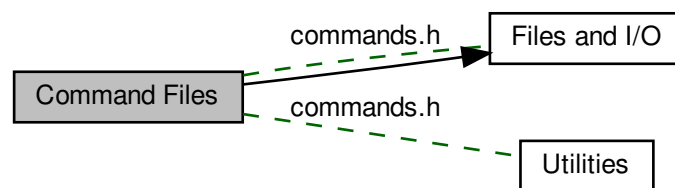
## 16.70 Command Files

Functions to parse and interpret the content of [Command Files](#).

### 16.70.1 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.*

## 16.70.2 Macro Definition Documentation

### 16.70.2.1 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\(\)](#)

### 16.70.2.2 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\(\)](#)

### 16.70.2.3 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\(\)](#)

### 16.70.2.4 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\(\)](#)

### 16.70.2.5 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\(\)](#)

## 16.70.3 Function Documentation

### 16.70.3.1 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

16.70.3.2 `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`

16.70.3.3 `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

**16.70.3.4 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
-----------------	---------------------------------

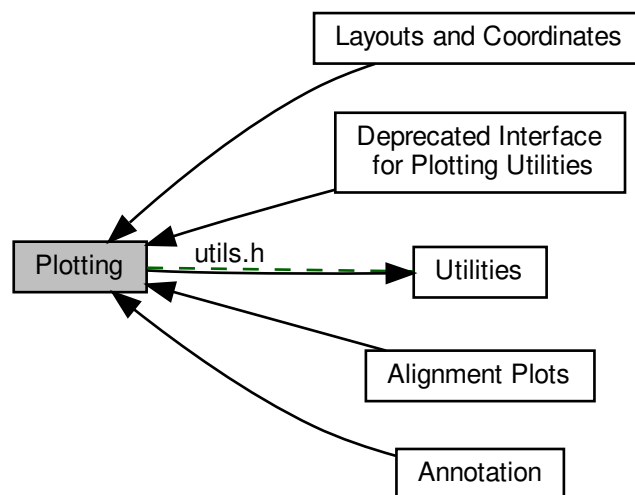
## 16.71 Plotting

Functions for Creating Secondary Structure Plots, Dot-Plots, and More.

### 16.71.1 Detailed Description

Functions for Creating Secondary Structure Plots, Dot-Plots, and More.

Collaboration diagram for Plotting:



### Modules

- [Layouts and Coordinates](#)  
*Functions to compute coordinate layouts for secondary structure plots.*
- [Annotation](#)  
*Functions to generate annotations for Secondary Structure Plots, Dot-Plots, and Others.*
- [Alignment Plots](#)  
*Functions to generate Alignment plots with annotated consensus structure.*
- [Deprecated Interface for Plotting Utilities](#)

### Files

- file [alignments.h](#)  
*Various functions for plotting Sequence / Structure Alignments.*
- file [layouts.h](#)  
*Secondary structure plot layout algorithms.*
- file [naview.h](#)

- Implementation of the Naviw RNA secondary structure layout algorithm [5].
- 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.
- file [RNApuzzler.h](#)  
 Implementation of the RNApuzzler RNA secondary structure layout algorithm [23].
- file [RNAturtle.h](#)  
 Implementation of the RNAturtle RNA secondary structure layout algorithm [23].

## Data Structures

- struct [vrna\\_dotplot\\_auxdata\\_t](#)

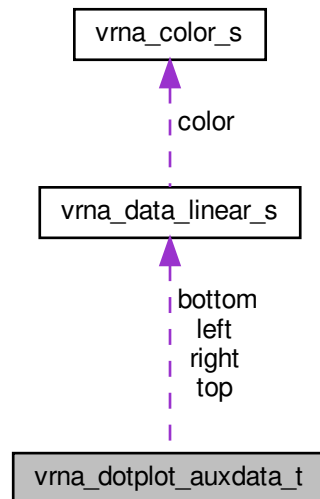
## Functions

- 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.
- 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)

## 16.71.2 Data Structure Documentation

### 16.71.2.1 struct vrna\_dotplot\_auxdata\_t

Collaboration diagram for vrna\_dotplot\_auxdata\_t:



## 16.71.3 Function Documentation

### 16.71.3.1 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

**16.71.3.2 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!

**16.71.3.3 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

16.71.3.4 `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

16.71.3.5 `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

### 16.71.3.6 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

### 16.71.3.7 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

### 16.71.3.8 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

### 16.71.3.9 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!

#### 16.71.3.10 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!

#### 16.71.3.11 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!

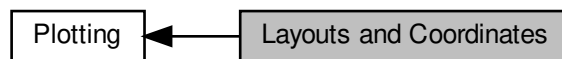
## 16.72 Layouts and Coordinates

Functions to compute coordinate layouts for secondary structure plots.

### 16.72.1 Detailed Description

Functions to compute coordinate layouts for secondary structure plots.

Collaboration diagram for Layouts and Coordinates:



### Data Structures

- struct [vrna\\_plot\\_layout\\_s](#)
- struct [vrna\\_plot\\_options\\_puzzler\\_t](#)

*Options data structure for RNApuzzler algorithm implementation. [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*
- `#define VRNA_PLOT_TYPE_TURTLE 3`  
*Definition of Plot type Turtle [23].*
- `#define VRNA_PLOT_TYPE_PUZZLER 4`  
*Definition of Plot type RNApuzzler [23].*

### Typedefs

- typedef struct [vrna\\_plot\\_layout\\_s](#) [vrna\\_plot\\_layout\\_t](#)  
*RNA secondary structure figure layout.*

## Functions

- [vrna\\_plot\\_layout\\_t \\* vrna\\_plot\\_layout](#) (const char \*structure, unsigned int plot\_type)  
*Create a layout (coordinates, etc.) for a secondary structure plot.*
- [vrna\\_plot\\_layout\\_t \\* vrna\\_plot\\_layout\\_simple](#) (const char \*structure)  
*Create a layout (coordinates, etc.) for a simple secondary structure plot.*
- [vrna\\_plot\\_layout\\_t \\* vrna\\_plot\\_layout\\_naview](#) (const char \*structure)  
*Create a layout (coordinates, etc.) for a secondary structure plot using the Naview Algorithm [5].*
- [vrna\\_plot\\_layout\\_t \\* vrna\\_plot\\_layout\\_circular](#) (const char \*structure)  
*Create a layout (coordinates, etc.) for a circular secondary structure plot.*
- [vrna\\_plot\\_layout\\_t \\* vrna\\_plot\\_layout\\_turtle](#) (const char \*structure)  
*Create a layout (coordinates, etc.) for a secondary structure plot using the Turtle Algorithm [23].*
- [vrna\\_plot\\_layout\\_t \\* vrna\\_plot\\_layout\\_puzzler](#) (const char \*structure, [vrna\\_plot\\_options\\_puzzler\\_t](#) \*options)  
*Create a layout (coordinates, etc.) for a secondary structure plot using the RNApuzzler Algorithm [23].*
- void [vrna\\_plot\\_layout\\_free](#) ([vrna\\_plot\\_layout\\_t](#) \*layout)  
*Free memory occupied by a figure layout data structure.*
- int [vrna\\_plot\\_coords](#) (const char \*structure, float \*\*x, float \*\*y, int plot\_type)  
*Compute nucleotide coordinates for secondary structure plot.*
- int [vrna\\_plot\\_coords\\_pt](#) (const short \*pt, float \*\*x, float \*\*y, int plot\_type)  
*Compute nucleotide coordinates for secondary structure plot.*
- int [vrna\\_plot\\_coords\\_simple](#) (const char \*structure, float \*\*x, float \*\*y)  
*Compute nucleotide coordinates for secondary structure plot the Simple way*
- int [vrna\\_plot\\_coords\\_simple\\_pt](#) (const short \*pt, float \*\*x, float \*\*y)  
*Compute nucleotide coordinates for secondary structure plot the Simple way*
- int [vrna\\_plot\\_coords\\_circular](#) (const char \*structure, float \*\*x, float \*\*y)  
*Compute coordinates of nucleotides mapped in equal distances onto a unit circle.*
- int [vrna\\_plot\\_coords\\_circular\\_pt](#) (const short \*pt, float \*\*x, float \*\*y)  
*Compute nucleotide coordinates for a Circular Plot*
- int [vrna\\_plot\\_coords\\_naview](#) (const char \*structure, float \*\*x, float \*\*y)  
*Compute nucleotide coordinates for secondary structure plot using the Naview algorithm [5].*
- int [vrna\\_plot\\_coords\\_naview\\_pt](#) (const short \*pt, float \*\*x, float \*\*y)  
*Compute nucleotide coordinates for secondary structure plot using the Naview algorithm [5].*
- int [vrna\\_plot\\_coords\\_puzzler](#) (const char \*structure, float \*\*x, float \*\*y, double \*\*arc\_coords, [vrna\\_plot\\_options\\_puzzler\\_t](#) \*options)  
*Compute nucleotide coordinates for secondary structure plot using the RNApuzzler algorithm [23].*
- int [vrna\\_plot\\_coords\\_puzzler\\_pt](#) (short const \*const pair\_table, float \*\*x, float \*\*y, double \*\*arc\_coords, [vrna\\_plot\\_options\\_puzzler\\_t](#) \*puzzler)  
*Compute nucleotide coordinates for secondary structure plot using the RNApuzzler algorithm [23].*
- [vrna\\_plot\\_options\\_puzzler\\_t \\* vrna\\_plot\\_options\\_puzzler](#) (void)  
*Create an RNApuzzler options data structure.*
- void [vrna\\_plot\\_options\\_puzzler\\_free](#) ([vrna\\_plot\\_options\\_puzzler\\_t](#) \*options)  
*Free memory occupied by an RNApuzzler options data structure.*
- int [vrna\\_plot\\_coords\\_turtle](#) (const char \*structure, float \*\*x, float \*\*y, double \*\*arc\_coords)  
*Compute nucleotide coordinates for secondary structure plot using the RNAturtle algorithm [23].*
- int [vrna\\_plot\\_coords\\_turtle\\_pt](#) (short const \*const pair\_table, float \*\*x, float \*\*y, double \*\*arc\_coords)  
*Compute nucleotide coordinates for secondary structure plot using the RNAturtle algorithm [23].*

## 16.72.2 Data Structure Documentation

### 16.72.2.1 struct vrna\_plot\_layout\_s

### 16.72.2.2 struct vrna\_plot\_options\_puzzler\_t

Options data structure for RNApuzzler algorithm implementation.

## 16.72.3 Macro Definition Documentation

### 16.72.3.1 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\(\)](#)

### 16.72.3.2 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 [5].

See also

[rna\\_plot\\_type](#), [vrna\\_file\\_PS\\_rnaplot\\_a\(\)](#), [vrna\\_file\\_PS\\_rnaplot\(\)](#), [svg\\_rna\\_plot\(\)](#), [gmlRNA\(\)](#), [ssv\\_rna\\_plot\(\)](#), [xrna\\_plot\(\)](#)

### 16.72.3.3 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\(\)](#)

## 16.72.4 Typedef Documentation

### 16.72.4.1 vrna\_plot\_layout\_t

```
typedef struct vrna_plot_layout_s vrna_plot_layout_t
```

```
#include <ViennaRNA/plotting/layouts.h>
```

RNA secondary structure figure layout.

See also

[vrna\\_plot\\_layout\(\)](#), [vrna\\_plot\\_layout\\_free\(\)](#), [vrna\\_plot\\_layout\\_simple\(\)](#), [vrna\\_plot\\_layout\\_circular\(\)](#), [vrna\\_plot\\_layout\\_navigate\(\)](#), [vrna\\_plot\\_layout\\_turtle\(\)](#), [vrna\\_plot\\_layout\\_puzzler\(\)](#)

## 16.72.5 Function Documentation



**16.72.5.1 vrna\_plot\_layout()**

```
vrna_plot_layout_t* vrna_plot_layout (
    const char * structure,
    unsigned int plot_type )

#include <ViennaRNA/plotting/layouts.h>
```

Create a layout (coordinates, etc.) for a secondary structure plot.

This function can be used to create a secondary structure nucleotide layout that is then further processed by an actual plotting function. The layout algorithm can be specified using the `plot_type` parameter, and the following algorithms are currently supported:

- [VRNA\\_PLOT\\_TYPE\\_SIMPLE](#)
- [VRNA\\_PLOT\\_TYPE\\_NAVIEW](#)
- [VRNA\\_PLOT\\_TYPE\\_CIRCULAR](#)
- [VRNA\\_PLOT\\_TYPE\\_TURTLE](#)
- [VRNA\\_PLOT\\_TYPE\\_PUZZLER](#)

Passing an unsupported selection leads to the default algorithm [VRNA\\_PLOT\\_TYPE\\_NAVIEW](#)

**Note**

If only X-Y coordinates of the corresponding structure layout are required, consider using [vrna\\_plot\\_coords\(\)](#) instead!

**See also**

[vrna\\_plot\\_layout\\_free\(\)](#), [vrna\\_plot\\_layout\\_simple\(\)](#), [vrna\\_plot\\_layout\\_naview\(\)](#), [vrna\\_plot\\_layout\\_circular\(\)](#), [vrna\\_plot\\_layout\\_turtle\(\)](#), [vrna\\_plot\\_layout\\_puzzler\(\)](#), [vrna\\_plot\\_coords\(\)](#), [vrna\\_file\\_PS\\_rnaplot\\_layout\(\)](#)

**Parameters**

<i>structure</i>	The secondary structure in dot-bracket notation
<i>plot_type</i>	The layout algorithm to be used

**Returns**

The layout data structure for the provided secondary structure

**16.72.5.2 vrna\_plot\_layout\_simple()**

```
vrna_plot_layout_t* vrna_plot_layout_simple (
    const char * structure )

#include <ViennaRNA/plotting/layouts.h>
```

Create a layout (coordinates, etc.) for a *simple* secondary structure plot.

This function basically is a wrapper to [vrna\\_plot\\_layout\(\)](#) that passes the `plot_type` [VRNA\\_PLOT\\_TYPE\\_SIMPLE](#).

**Note**

If only X-Y coordinates of the corresponding structure layout are required, consider using [vrna\\_plot\\_coords\\_simple\(\)](#) instead!

**See also**

[vrna\\_plot\\_layout\\_free\(\)](#), [vrna\\_plot\\_layout\(\)](#), [vrna\\_plot\\_layout\\_navigate\(\)](#), [vrna\\_plot\\_layout\\_circular\(\)](#), [vrna\\_plot\\_layout\\_turtle\(\)](#), [vrna\\_plot\\_layout\\_puzzler\(\)](#), [vrna\\_plot\\_coords\\_simple\(\)](#), [vrna\\_file\\_PS\\_rnaplot\\_layout\(\)](#)

**Parameters**

<i>structure</i>	The secondary structure in dot-bracket notation
------------------	---

**Returns**

The layout data structure for the provided secondary structure

**16.72.5.3 vrna\_plot\_layout\_navigate()**

```
vrna_plot_layout_t* vrna_plot_layout_navigate (
    const char * structure )
```

```
#include <ViennaRNA/plotting/layouts.h>
```

Create a layout (coordinates, etc.) for a secondary structure plot using the *Navigate Algorithm* [5].

This function basically is a wrapper to [vrna\\_plot\\_layout\(\)](#) that passes the `plot_type` [VRNA\\_PLOT\\_TYPE\\_NAVIGATE](#).

**Note**

If only X-Y coordinates of the corresponding structure layout are required, consider using [vrna\\_plot\\_coords\\_navigate\(\)](#) instead!

**See also**

[vrna\\_plot\\_layout\\_free\(\)](#), [vrna\\_plot\\_layout\(\)](#), [vrna\\_plot\\_layout\\_simple\(\)](#), [vrna\\_plot\\_layout\\_circular\(\)](#), [vrna\\_plot\\_layout\\_turtle\(\)](#), [vrna\\_plot\\_layout\\_puzzler\(\)](#), [vrna\\_plot\\_coords\\_navigate\(\)](#), [vrna\\_file\\_PS\\_rnaplot\\_layout\(\)](#)

**Parameters**

<i>structure</i>	The secondary structure in dot-bracket notation
------------------	---

**Returns**

The layout data structure for the provided secondary structure

**16.72.5.4 vrna\_plot\_layout\_circular()**

```
vrna_plot_layout_t* vrna_plot_layout_circular (
    const char * structure )

#include <ViennaRNA/plotting/layouts.h>
```

Create a layout (coordinates, etc.) for a *circular* secondary structure plot.

This function basically is a wrapper to [vrna\\_plot\\_layout\(\)](#) that passes the `plot_type` `VRNA_PLOT_TYPE_CIRCULAR`.

**Note**

If only X-Y coordinates of the corresponding structure layout are required, consider using [vrna\\_plot\\_coords\\_circular\(\)](#) instead!

**See also**

[vrna\\_plot\\_layout\\_free\(\)](#), [vrna\\_plot\\_layout\(\)](#), [vrna\\_plot\\_layout\\_naview\(\)](#), [vrna\\_plot\\_layout\\_simple\(\)](#), [vrna\\_plot\\_layout\\_turtle\(\)](#), [vrna\\_plot\\_layout\\_puzzler\(\)](#), [vrna\\_plot\\_coords\\_circular\(\)](#), [vrna\\_file\\_PS\\_rnaplot\\_layout\(\)](#)

**Parameters**

<i>structure</i>	The secondary structure in dot-bracket notation
------------------	---

**Returns**

The layout data structure for the provided secondary structure

**16.72.5.5 vrna\_plot\_layout\_turtle()**

```
vrna_plot_layout_t* vrna_plot_layout_turtle (
    const char * structure )

#include <ViennaRNA/plotting/layouts.h>
```

Create a layout (coordinates, etc.) for a secondary structure plot using the *Turtle Algorithm* [23].

This function basically is a wrapper to [vrna\\_plot\\_layout\(\)](#) that passes the `plot_type` `VRNA_PLOT_TYPE_TURTLE`.

**Note**

If only X-Y coordinates of the corresponding structure layout are required, consider using [vrna\\_plot\\_coords\\_turtle\(\)](#) instead!

**See also**

[vrna\\_plot\\_layout\\_free\(\)](#), [vrna\\_plot\\_layout\(\)](#), [vrna\\_plot\\_layout\\_simple\(\)](#), [vrna\\_plot\\_layout\\_circular\(\)](#), [vrna\\_plot\\_layout\\_naview\(\)](#), [vrna\\_plot\\_layout\\_puzzler\(\)](#), [vrna\\_plot\\_coords\\_turtle\(\)](#), [vrna\\_file\\_PS\\_rnaplot\\_layout\(\)](#)

**Parameters**

<i>structure</i>	The secondary structure in dot-bracket notation
------------------	---

**Returns**

The layout data structure for the provided secondary structure

**16.72.5.6 vrna\_plot\_layout\_puzzler()**

```
vrna_plot_layout_t* vrna_plot_layout_puzzler (
    const char * structure,
    vrna_plot_options_puzzler_t * options )
```

```
#include <ViennaRNA/plotting/layouts.h>
```

Create a layout (coordinates, etc.) for a secondary structure plot using the *RNApuzzler Algorithm* [23].

This function basically is a wrapper to [vrna\\_plot\\_layout\(\)](#) that passes the `plot_type` [VRNA\\_PLOT\\_TYPE\\_PUZZLER](#).

**Note**

If only X-Y coordinates of the corresponding structure layout are required, consider using [vrna\\_plot\\_coords\\_puzzler\(\)](#) instead!

**See also**

[vrna\\_plot\\_layout\\_free\(\)](#), [vrna\\_plot\\_layout\(\)](#), [vrna\\_plot\\_layout\\_simple\(\)](#), [vrna\\_plot\\_layout\\_circular\(\)](#), [vrna\\_plot\\_layout\\_navview\(\)](#), [vrna\\_plot\\_layout\\_turtle\(\)](#), [vrna\\_plot\\_coords\\_puzzler\(\)](#), [vrna\\_file\\_PS\\_rnaplot\\_layout\(\)](#)

**Parameters**

<i>structure</i>	The secondary structure in dot-bracket notation
------------------	---

**Returns**

The layout data structure for the provided secondary structure

**16.72.5.7 vrna\_plot\_layout\_free()**

```
void vrna_plot_layout_free (
    vrna_plot_layout_t * layout )

#include <ViennaRNA/plotting/layouts.h>
```

Free memory occupied by a figure layout data structure.

## See also

[vrna\\_plot\\_layout\\_t](#), [vrna\\_plot\\_layout\(\)](#), [vrna\\_plot\\_layout\\_simple\(\)](#), [vrna\\_plot\\_layout\\_circular\(\)](#), [vrna\\_plot\\_layout\\_navigate\(\)](#), [vrna\\_plot\\_layout\\_turtle\(\)](#), [vrna\\_plot\\_layout\\_puzzler\(\)](#), [vrna\\_file\\_PS\\_rnaplot\\_layout\(\)](#)

## Parameters

<i>layout</i>	The layout data structure to free
---------------	-----------------------------------

16.72.5.8 `vrna_plot_coords()`

```
int vrna_plot_coords (
    const char * structure,
    float ** x,
    float ** y,
    int plot_type )
```

```
#include <ViennaRNA/plotting/layouts.h>
```

Compute nucleotide coordinates for secondary structure plot.

This function takes a secondary structure and computes X-Y coordinates for each nucleotide that then can be used to create a structure plot. The parameter `plot_type` is used to select the underlying layout algorithm. Currently, the following selections are provided:

- [VRNA\\_PLOT\\_TYPE\\_SIMPLE](#)
- [VRNA\\_PLOT\\_TYPE\\_NAVIEW](#)
- [VRNA\\_PLOT\\_TYPE\\_CIRCULAR](#)
- [VRNA\\_PLOT\\_TYPE\\_TURTLE](#)
- [VRNA\\_PLOT\\_TYPE\\_PUZZLER](#)

Passing an unsupported selection leads to the default algorithm [VRNA\\_PLOT\\_TYPE\\_NAVIEW](#)

Here is a simple example how to use this function, assuming variable `structure` contains a valid dot-bracket string:

```
float *x, *y;
if (vrna_plot_coords(structure, &x, &y)) {
    printf("all fine");
} else {
    printf("some failure occurred!");
}
free(x);
free(y);
```

## Note

On success, this function allocates memory for X and Y coordinates and assigns the pointers at addresses `x` and `y` to the corresponding memory locations. It's the users responsibility to cleanup this memory after usage!

## See also

[vrna\\_plot\\_coords\\_pt\(\)](#), [vrna\\_plot\\_coords\\_simple\(\)](#), [vrna\\_plot\\_coords\\_navigate\(\)](#), [vrna\\_plot\\_coords\\_circular\(\)](#), [vrna\\_plot\\_coords\\_turtle\(\)](#), [vrna\\_plot\\_coords\\_puzzler\(\)](#)

**Parameters**

	<i>structure</i>	The secondary structure in dot-bracket notation
<i>in, out</i>	<i>x</i>	The address of a pointer of X coordinates (pointer will point to memory, or NULL on failure)
<i>in, out</i>	<i>y</i>	The address of a pointer of Y coordinates (pointer will point to memory, or NULL on failure)
	<i>plot_type</i>	The layout algorithm to be used

**Returns**

The length of the structure on success, 0 otherwise

**16.72.5.9 vrna\_plot\_coords\_pt()**

```
int vrna_plot_coords_pt (
    const short * pt,
    float ** x,
    float ** y,
    int plot_type )
```

```
#include <ViennaRNA/plotting/layouts.h>
```

Compute nucleotide coordinates for secondary structure plot.

Same as [vrna\\_plot\\_coords\(\)](#) but takes a pair table with the structure information as input.

**Note**

On success, this function allocates memory for X and Y coordinates and assigns the pointers at addresses *x* and *y* to the corresponding memory locations. It's the users responsibility to cleanup this memory after usage!

**See also**

[vrna\\_plot\\_coords\(\)](#), [vrna\\_plot\\_coords\\_simple\\_pt\(\)](#), [vrna\\_plot\\_coords\\_naview\\_pt\(\)](#) [vrna\\_plot\\_coords\\_circular\\_pt\(\)](#), [vrna\\_plot\\_coords\\_turtle\\_pt\(\)](#), [vrna\\_plot\\_coords\\_puzzler\\_pt\(\)](#)

**Parameters**

	<i>pt</i>	The pair table that holds the secondary structure
<i>in, out</i>	<i>x</i>	The address of a pointer of X coordinates (pointer will point to memory, or NULL on failure)
<i>in, out</i>	<i>y</i>	The address of a pointer of Y coordinates (pointer will point to memory, or NULL on failure)
	<i>plot_type</i>	The layout algorithm to be used

**Returns**

The length of the structure on success, 0 otherwise

**16.72.5.10 vrna\_plot\_coords\_simple()**

```
int vrna_plot_coords_simple (
    const char * structure,
    float ** x,
    float ** y )
```

```
#include <ViennaRNA/plotting/layouts.h>
```

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

This function basically is a wrapper to [vrna\\_plot\\_coords\(\)](#) that passes the `plot_type` [VRNA\\_PLOT\\_TYPE\\_SIMPLE](#).

Here is a simple example how to use this function, assuming variable `structure` contains a valid dot-bracket string:

```
float *x, *y;
if (vrna_plot_coords_simple(structure, &x, &y)) {
    printf("all fine");
} else {
    printf("some failure occurred!");
}
free(x);
free(y);
```

**Note**

On success, this function allocates memory for X and Y coordinates and assigns the pointers at addresses `x` and `y` to the corresponding memory locations. It's the users responsibility to cleanup this memory after usage!

**See also**

[vrna\\_plot\\_coords\(\)](#), [vrna\\_plot\\_coords\\_simple\\_pt\(\)](#), [vrna\\_plot\\_coords\\_circular\(\)](#), [vrna\\_plot\\_coords\\_naview\(\)](#), [vrna\\_plot\\_coords\\_turtle\(\)](#), [vrna\\_plot\\_coords\\_puzzler\(\)](#)

**Parameters**

	<i>structure</i>	The secondary structure in dot-bracket notation
in, out	<i>x</i>	The address of a pointer of X coordinates (pointer will point to memory, or NULL on failure)
in, out	<i>y</i>	The address of a pointer of Y coordinates (pointer will point to memory, or NULL on failure)

**Returns**

The length of the structure on success, 0 otherwise

### 16.72.5.11 vrna\_plot\_coords\_simple\_pt()

```
int vrna_plot_coords_simple_pt (
    const short * pt,
    float ** x,
    float ** y )
```

```
#include <ViennaRNA/plotting/layouts.h>
```

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

Same as [vrna\\_plot\\_coords\\_simple\(\)](#) but takes a pair table with the structure information as input.

#### Note

On success, this function allocates memory for X and Y coordinates and assigns the pointers at addressess `x` and `y` to the corresponding memory locations. It's the users responsibility to cleanup this memory after usage!

#### See also

[vrna\\_plot\\_coords\\_pt\(\)](#), [vrna\\_plot\\_coords\\_simple\(\)](#), [vrna\\_plot\\_coords\\_circular\\_pt\(\)](#), [vrna\\_plot\\_coords\\_naview\\_pt\(\)](#), [vrna\\_plot\\_coords\\_turtle\\_pt\(\)](#), [vrna\\_plot\\_coords\\_puzzler\\_pt\(\)](#)

#### Parameters

	<i>pt</i>	The pair table that holds the secondary structure
<i>in, out</i>	<i>x</i>	The address of a pointer of X coordinates (pointer will point to memory, or NULL on failure)
<i>in, out</i>	<i>y</i>	The address of a pointer of Y coordinates (pointer will point to memory, or NULL on failure)

#### Returns

The length of the structure on success, 0 otherwise

### 16.72.5.12 vrna\_plot\_coords\_circular()

```
int vrna_plot_coords_circular (
    const char * structure,
    float ** x,
    float ** y )
```

```
#include <ViennaRNA/plotting/layouts.h>
```

Compute coordinates of nucleotides mapped in equal distances onto a unit circle.

This function basically is a wrapper to [vrna\\_plot\\_coords\(\)](#) that passes the `plot_type` `VRNA_PLOT_TYPE_CIRCULAR`.

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

$$P_y^t[i] = Y[i] * rs$$

.



**Note**

On success, this function allocates memory for X and Y coordinates and assigns the pointers at addresses `x` and `y` to the corresponding memory locations. It's the users responsibility to cleanup this memory after usage!

**See also**

[vrna\\_plot\\_coords\(\)](#), [vrna\\_plot\\_coords\\_circular\\_pt\(\)](#), [vrna\\_plot\\_coords\\_simple\(\)](#), [vrna\\_plot\\_coords\\_naview\(\)](#), [vrna\\_plot\\_coords\\_turtle\(\)](#), [vrna\\_plot\\_coords\\_puzzler\(\)](#)

**Parameters**

	<i>structure</i>	The secondary structure in dot-bracket notation
<i>in, out</i>	<i>x</i>	The address of a pointer of X coordinates (pointer will point to memory, or NULL on failure)
<i>in, out</i>	<i>y</i>	The address of a pointer of Y coordinates (pointer will point to memory, or NULL on failure)

**Returns**

The length of the structure on success, 0 otherwise

**16.72.5.13 vrna\_plot\_coords\_circular\_pt()**

```
int vrna_plot_coords_circular_pt (
    const short * pt,
    float ** x,
    float ** y )
```

```
#include <ViennaRNA/plotting/layouts.h>
```

Compute nucleotide coordinates for a *Circular Plot*

Same as [vrna\\_plot\\_coords\\_circular\(\)](#) but takes a pair table with the structure information as input.

**Note**

On success, this function allocates memory for X and Y coordinates and assigns the pointers at addresses `x` and `y` to the corresponding memory locations. It's the users responsibility to cleanup this memory after usage!

**See also**

[vrna\\_plot\\_coords\\_pt\(\)](#), [vrna\\_plot\\_coords\\_circular\(\)](#), [vrna\\_plot\\_coords\\_simple\\_pt\(\)](#), [vrna\\_plot\\_coords\\_naview\\_pt\(\)](#), [vrna\\_plot\\_coords\\_turtle\\_pt\(\)](#), [vrna\\_plot\\_coords\\_puzzler\\_pt\(\)](#)

**Parameters**

	<i>pt</i>	The pair table that holds the secondary structure
<i>in, out</i>	<i>x</i>	The address of a pointer of X coordinates (pointer will point to memory, or NULL on failure)
<i>in, out</i>	<i>y</i>	The address of a pointer of Y coordinates (pointer will point to memory, or NULL on failure)

**Returns**

The length of the structure on success, 0 otherwise

**16.72.5.14 vrna\_plot\_coords\_navview()**

```
int vrna_plot_coords_navview (
    const char * structure,
    float ** x,
    float ** y )
```

```
#include <ViennaRNA/plotting/naview.h>
```

Compute nucleotide coordinates for secondary structure plot using the *Naview* algorithm [5].

This function basically is a wrapper to [vrna\\_plot\\_coords\(\)](#) that passes the `plot_type` [VRNA\\_PLOT\\_TYPE\\_NAVIEW](#).

Here is a simple example how to use this function, assuming variable `structure` contains a valid dot-bracket string:

```
float *x, *y;
if (vrna_plot_coords_navview(structure, &x, &y)) {
    printf("all fine");
} else {
    printf("some failure occured!");
}
free(x);
free(y);
```

**Note**

On success, this function allocates memory for X and Y coordinates and assigns the pointers at addresses `x` and `y` to the corresponding memory locations. It's the users responsibility to cleanup this memory after usage!

**See also**

[vrna\\_plot\\_coords\(\)](#), [vrna\\_plot\\_coords\\_simple\\_pt\(\)](#), [vrna\\_plot\\_coords\\_circular\(\)](#), [vrna\\_plot\\_coords\\_simple\(\)](#), [vrna\\_plot\\_coords\\_turtle\(\)](#), [vrna\\_plot\\_coords\\_puzzler\(\)](#)

**Parameters**

	<i>structure</i>	The secondary structure in dot-bracket notation
in, out	<i>x</i>	The address of a pointer of X coordinates (pointer will point to memory, or NULL on failure)
in, out	<i>y</i>	The address of a pointer of Y coordinates (pointer will point to memory, or NULL on failure)

**Returns**

The length of the structure on success, 0 otherwise

**16.72.5.15 vrna\_plot\_coords\_navview\_pt()**

```
int vrna_plot_coords_navview_pt (
    const short * pt,
    float ** x,
    float ** y )
```

```
#include <ViennaRNA/plotting/navview.h>
```

Compute nucleotide coordinates for secondary structure plot using the *Navview* algorithm [5].

Same as [vrna\\_plot\\_coords\\_navview\(\)](#) but takes a pair table with the structure information as input.

**Note**

On success, this function allocates memory for X and Y coordinates and assigns the pointers at addressess *x* and *y* to the corresponding memory locations. It's the users responsibility to cleanup this memory after usage!

**See also**

[vrna\\_plot\\_coords\\_pt\(\)](#), [vrna\\_plot\\_coords\\_navview\(\)](#), [vrna\\_plot\\_coords\\_circular\\_pt\(\)](#), [vrna\\_plot\\_coords\\_simple\\_pt\(\)](#), [vrna\\_plot\\_coords\\_turtle\\_pt\(\)](#), [vrna\\_plot\\_coords\\_puzzler\\_pt\(\)](#)

**Parameters**

	<i>pt</i>	The pair table that holds the secondary structure
<i>in, out</i>	<i>x</i>	The address of a pointer of X coordinates (pointer will point to memory, or NULL on failure)
<i>in, out</i>	<i>y</i>	The address of a pointer of Y coordinates (pointer will point to memory, or NULL on failure)

**Returns**

The length of the structure on success, 0 otherwise

**16.72.5.16 vrna\_plot\_coords\_puzzler()**

```
int vrna_plot_coords_puzzler (
    const char * structure,
    float ** x,
    float ** y,
    double ** arc_coords,
    vrna_plot_options_puzzler_t * options )
```

```
#include <ViennaRNA/plotting/RNApuzzler/RNApuzzler.h>
```

Compute nucleotide coordinates for secondary structure plot using the *RNApuzzler* algorithm [23].

This function basically is a wrapper to [vrna\\_plot\\_coords\(\)](#) that passes the `plot_type` `VRNA_PLOT_TYPE_PUZZLER`.

Here is a simple example how to use this function, assuming variable `structure` contains a valid dot-bracket string and using the default options (`options = NULL`):

```
float *x, *y;
double *arcs;
if (vrna_plot_coords_puzzler(structure, &x, &y, &arcs, NULL)) {
    printf("all fine");
} else {
    printf("some failure occurred!");
}
free(x);
free(y);
free(arcs);
```

#### Note

On success, this function allocates memory for X, Y and arc coordinates and assigns the pointers at addresses `x`, `y` and `arc_coords` to the corresponding memory locations. It's the users responsibility to cleanup this memory after usage!

#### See also

[vrna\\_plot\\_coords\(\)](#), [vrna\\_plot\\_coords\\_puzzler\\_pt\(\)](#), [vrna\\_plot\\_coords\\_circular\(\)](#), [vrna\\_plot\\_coords\\_simple\(\)](#), [vrna\\_plot\\_coords\\_turtle\(\)](#), [vrna\\_plot\\_coords\\_navview\(\)](#), [vrna\\_plot\\_options\\_puzzler\(\)](#)

#### Parameters

	<i>structure</i>	The secondary structure in dot-bracket notation
<i>in, out</i>	<i>x</i>	The address of a pointer of X coordinates (pointer will point to memory, or NULL on failure)
<i>in, out</i>	<i>y</i>	The address of a pointer of Y coordinates (pointer will point to memory, or NULL on failure)
<i>in, out</i>	<i>arc_coords</i>	The address of a pointer that will hold arc coordinates (pointer will point to memory, or NULL on failure)
	<i>options</i>	The options for the RNApuzzler algorithm (or NULL)

#### Returns

The length of the structure on success, 0 otherwise

#### 16.72.5.17 vrna\_plot\_coords\_puzzler\_pt()

```
int vrna_plot_coords_puzzler_pt (
    short const *const pair_table,
    float ** x,
    float ** y,
    double ** arc_coords,
    vrna_plot_options_puzzler_t * puzzler )
```

```
#include <ViennaRNA/plotting/RNApuzzler/RNApuzzler.h>
```

Compute nucleotide coordinates for secondary structure plot using the *RNApuzzler* algorithm [23].

Same as [vrna\\_plot\\_coords\\_puzzler\(\)](#) but takes a pair table with the structure information as input.

**Note**

On success, this function allocates memory for X, Y and arc coordinates and assigns the pointers at addresses `x`, `y` and `arc_coords` to the corresponding memory locations. It's the users responsibility to cleanup this memory after usage!

**See also**

[vrna\\_plot\\_coords\\_pt\(\)](#), [vrna\\_plot\\_coords\\_puzzler\(\)](#), [vrna\\_plot\\_coords\\_circular\\_pt\(\)](#), [vrna\\_plot\\_coords\\_simple\\_pt\(\)](#), [vrna\\_plot\\_coords\\_turtle\\_pt\(\)](#), [vrna\\_plot\\_coords\\_naview\\_pt\(\)](#)

**Parameters**

	<i>pt</i>	The pair table that holds the secondary structure
<i>in, out</i>	<i>x</i>	The address of a pointer of X coordinates (pointer will point to memory, or NULL on failure)
<i>in, out</i>	<i>y</i>	The address of a pointer of Y coordinates (pointer will point to memory, or NULL on failure)
<i>in, out</i>	<i>arc_coords</i>	The address of a pointer that will hold arc coordinates (pointer will point to memory, or NULL on failure)
	<i>options</i>	The options for the RNApuzzler algorithm (or NULL)

**Returns**

The length of the structure on success, 0 otherwise

**16.72.5.18 vrna\_plot\_options\_puzzler()**

```
vrna_plot_options_puzzler_t* vrna_plot_options_puzzler (
    void )
#include <ViennaRNA/plotting/RNApuzzler/RNApuzzler.h>
```

Create an RNApuzzler options data structure.

**See also**

[vrna\\_plot\\_options\\_puzzler\\_free\(\)](#), [vrna\\_plot\\_coords\\_puzzler\(\)](#), [vrna\\_plot\\_coords\\_puzzler\\_pt\(\)](#), [vrna\\_plot\\_layout\\_puzzler\(\)](#)

**Returns**

An RNApuzzler options data structure with default settings

**16.72.5.19 vrna\_plot\_options\_puzzler\_free()**

```
void vrna_plot_options_puzzler_free (
    vrna_plot_options_puzzler_t * options )
#include <ViennaRNA/plotting/RNApuzzler/RNApuzzler.h>
```

Free memory occupied by an RNApuzzler options data structure.

**See also**

[vrna\\_plot\\_options\\_puzzler\(\)](#), [vrna\\_plot\\_coords\\_puzzler\(\)](#), [vrna\\_plot\\_coords\\_puzzler\\_pt\(\)](#), [vrna\\_plot\\_layout\\_puzzler\(\)](#)

## Parameters

<i>options</i>	A pointer to the options data structure to free
----------------	---

**16.72.5.20 vrna\_plot\_coords\_turtle()**

```
int vrna_plot_coords_turtle (
    const char * structure,
    float ** x,
    float ** y,
    double ** arc_coords )
```

```
#include <ViennaRNA/plotting/RNApuzzler/RNAturtle.h>
```

Compute nucleotide coordinates for secondary structure plot using the *RNAturtle* algorithm [23].

This function basically is a wrapper to [vrna\\_plot\\_coords\(\)](#) that passes the `plot_type` `VRNA_PLOT_TYPE_TURTLE`.

Here is a simple example how to use this function, assuming variable `structure` contains a valid dot-bracket string:

```
float *x, *y;
double *arcs;
if (vrna_plot_coords_turtle(structure, &x, &y, &arcs)) {
    printf("all fine");
} else {
    printf("some failure occurred!");
}
free(x);
free(y);
free(arcs);
```

**Note**

On success, this function allocates memory for X, Y and arc coordinates and assigns the pointers at addressess `x`, `y` and `arc_coords` to the corresponding memory locations. It's the users responsibility to cleanup this memory after usage!

**See also**

[vrna\\_plot\\_coords\(\)](#), [vrna\\_plot\\_coords\\_turtle\\_pt\(\)](#), [vrna\\_plot\\_coords\\_circular\(\)](#), [vrna\\_plot\\_coords\\_simple\(\)](#), [vrna\\_plot\\_coords\\_navigate\(\)](#), [vrna\\_plot\\_coords\\_puzzler\(\)](#)

**Parameters**

	<i>structure</i>	The secondary structure in dot-bracket notation
in, out	<i>x</i>	The address of a pointer of X coordinates (pointer will point to memory, or NULL on failure)
in, out	<i>y</i>	The address of a pointer of Y coordinates (pointer will point to memory, or NULL on failure)
in, out	<i>arc_coords</i>	The address of a pointer that will hold arc coordinates (pointer will point to memory, or NULL on failure)

**Returns**

The length of the structure on success, 0 otherwise

**16.72.5.21 vrna\_plot\_coords\_turtle\_pt()**

```
int vrna_plot_coords_turtle_pt (
    short const *const pair_table,
    float ** x,
    float ** y,
    double ** arc_coords )
```

```
#include <ViennaRNA/plotting/RNApuzzler/RNAturtle.h>
```

Compute nucleotide coordinates for secondary structure plot using the *RNAturtle* algorithm [23].

Same as [vrna\\_plot\\_coords\\_turtle\(\)](#) but takes a pair table with the structure information as input.

**Note**

On success, this function allocates memory for X, Y and arc coordinates and assigns the pointers at addresses `x`, `y` and `arc_coords` to the corresponding memory locations. It's the users responsibility to cleanup this memory after usage!

**See also**

[vrna\\_plot\\_coords\\_pt\(\)](#), [vrna\\_plot\\_coords\\_turtle\(\)](#), [vrna\\_plot\\_coords\\_circular\\_pt\(\)](#), [vrna\\_plot\\_coords\\_simple\\_pt\(\)](#), [vrna\\_plot\\_coords\\_puzzler\\_pt\(\)](#), [vrna\\_plot\\_coords\\_navigate\\_pt\(\)](#)

**Parameters**

	<i>pt</i>	The pair table that holds the secondary structure
in, out	<i>x</i>	The address of a pointer of X coordinates (pointer will point to memory, or NULL on failure)
in, out	<i>y</i>	The address of a pointer of Y coordinates (pointer will point to memory, or NULL on failure)
in, out	<i>arc_coords</i>	The address of a pointer that will hold arc coordinates (pointer will point to memory, or NULL on failure)

**Returns**

The length of the structure on success, 0 otherwise

## 16.73 Annotation

Functions to generate annotations for Secondary Structure Plots, Dot-Plots, and Others.

### 16.73.1 Detailed Description

Functions to generate annotations for Secondary Structure Plots, Dot-Plots, and Others.

Collaboration diagram for Annotation:



## Functions

- char \*\* [vrna\\_annotate\\_covar\\_db](#) (const char \*\*alignment, const char \*structure, [vrna\\_md\\_t](#) \*md\_p)  
*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.*



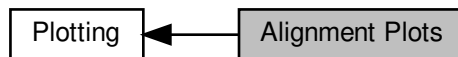
## 16.74 Alignment Plots

Functions to generate Alignment plots with annotated consensus structure.

### 16.74.1 Detailed Description

Functions to generate Alignment plots with annotated consensus structure.

Collaboration diagram for Alignment Plots:



### Functions

- `int vrna_file_PS_aln` (`const char *filename`, `const char **seqs`, `const char **names`, `const char *structure`, `unsigned int columns`)  
*Create an annotated PostScript alignment plot.*
- `int vrna_file_PS_aln_slice` (`const char *filename`, `const char **seqs`, `const char **names`, `const char *structure`, `unsigned int start`, `unsigned int end`, `int offset`, `unsigned int columns`)  
*Create an annotated PostScript alignment plot.*

### 16.74.2 Function Documentation

#### 16.74.2.1 `vrna_file_PS_aln()`

```
int vrna_file_PS_aln (  
    const char * filename,  
    const char ** seqs,  
    const char ** names,  
    const char * structure,  
    unsigned int columns )
```

```
#include <ViennaRNA/plotting/alignments.h>
```

Create an annotated PostScript alignment plot.

See also

[vrna\\_file\\_PS\\_aln\\_slice\(\)](#)

## Parameters

<i>filename</i>	The output file name
<i>seqs</i>	The aligned sequences
<i>names</i>	The names of the sequences
<i>structure</i>	The consensus structure in dot-bracket notation
<i>columns</i>	The number of columns before the alignment is wrapped as a new block (a value of 0 indicates no wrapping)

**SWIG Wrapper Notes** This function is available as overloaded function `file_PS_aln()` with three additional parameters `start`, `end`, and `offset` before the `columns` argument. Thus, it resembles the `vrna_file_PS_aln_slice()` function. The last four arguments may be omitted, indicating the default of `start = 0`, `end = 0`, `offset = 0`, and `columns = 60`.

### 16.74.2.2 vrna\_file\_PS\_aln\_slice()

```
int vrna_file_PS_aln_slice (
    const char * filename,
    const char ** seqs,
    const char ** names,
    const char * structure,
    unsigned int start,
    unsigned int end,
    int offset,
    unsigned int columns )

#include <ViennaRNA/plotting/alignments.h>
```

Create an annotated PostScript alignment plot.

Similar to `vrna_file_PS_aln()` but allows the user to print a particular slice of the alignment by specifying a `start` and `end` position. The additional `offset` parameter allows for adjusting the alignment position ruler value.

See also

[vrna\\_file\\_PS\\_aln\\_slice\(\)](#)

## Parameters

<i>filename</i>	The output file name
<i>seqs</i>	The aligned sequences
<i>names</i>	The names of the sequences
<i>structure</i>	The consensus structure in dot-bracket notation
<i>start</i>	The start of the alignment slice (a value of 0 indicates the first position of the alignment, i.e. no slicing at 5' side)
<i>end</i>	The end of the alignment slice (a value of 0 indicates the last position of the alignment, i.e. no slicing at 3' side)
<i>offset</i>	The alignment coordinate offset for the position ruler.
<i>columns</i>	The number of columns before the alignment is wrapped as a new block (a value of 0 indicates no wrapping)

**SWIG Wrapper Notes** This function is available as overloaded function `file_PS_aln()` where the last four parameter may be omitted, indicating `start = 0`, `end = 0`, `offset = 0`, and `columns = 60`.

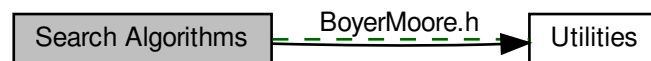
## 16.75 Search Algorithms

Implementations of various search algorithms to detect strings of objects within other strings of objects.

### 16.75.1 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.*

### 16.75.2 Function Documentation

### 16.75.2.1 `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*

### 16.75.2.2 `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*

### 16.75.2.3 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 <i>pattern</i>
<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)

**16.75.2.4 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)

## 16.76 Combinatorics Algorithms

Implementations to solve various combinatorial aspects for strings of objects.

### 16.76.1 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.*

### 16.76.2 Function Documentation



**16.76.2.1 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**

<code>type_counts</code>	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.

**16.76.2.2 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.

### 16.76.2.3 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.

**16.76.2.4 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.

**16.76.2.5 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

**SWIG Wrapper Notes** This function is available as overloaded global function `rotational_symmetry()`. It merges the functionalities of [vrna\\_rotational\\_symmetry\(\)](#), [vrna\\_rotational\\_symmetry\\_pos\(\)](#), [vrna\\_rotational\\_symmetry\\_num\(\)](#), and [vrna\\_rotational\\_symmetry\\_pos\\_num\(\)](#). In contrast to our C-implementation, this function doesn't return the order of rotational symmetry as a single value, but returns a list of cyclic permutation shifts that result in a rotationally symmetric string. The length of the list then determines the order of rotational symmetry.

### 16.76.2.6 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.

## 16.76.2.7 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 NULL)

### 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.

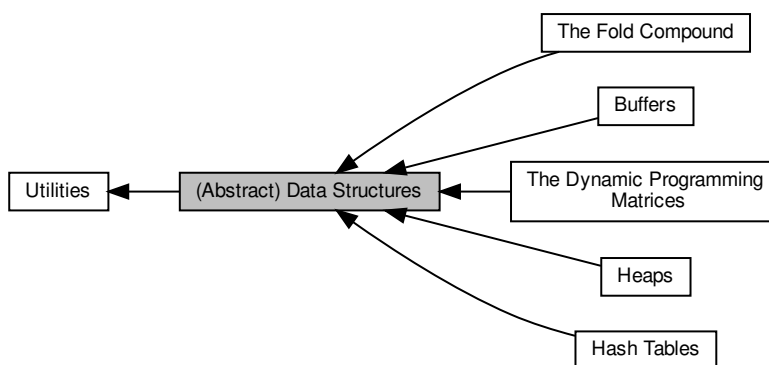
## 16.77 (Abstract) Data Structures

All datastructures and typedefs shared among the ViennaRNA Package can be found here.

### 16.77.1 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.*
- [Heaps](#)  
*Interface for an abstract implementation of a heap data structure.*
- [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 `RNAshoop` (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.*

## 16.77.2 Data Structure Documentation

### 16.77.2.1 struct `vrna_basepair_s`

Base pair data structure used in `subopt.c`.

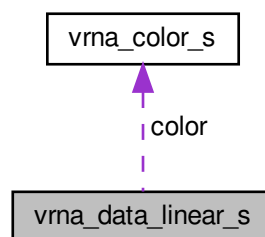
### 16.77.2.2 struct `vrna_cpair_s`

this datastructure is used as input parameter in functions of `PS_dot.c`

### 16.77.2.3 struct `vrna_color_s`

### 16.77.2.4 struct `vrna_data_linear_s`

Collaboration diagram for `vrna_data_linear_s`:



**16.77.2.5 struct vrna\_sect\_s**

Stack of partial structures for backtracking.

**16.77.2.6 struct vrna\_bp\_stack\_s**

Base pair stack element.

**16.77.2.7 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*

**16.77.2.8 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*

### 16.77.2.9 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*

### 16.77.2.10 struct constrain

constraints for cofolding

### 16.77.2.11 struct duplexT

Data structure for RNAduplex.

### 16.77.2.12 struct node

Data structure for RNAsnoop (fold energy list)

Collaboration diagram for node:



### 16.77.2.13 struct snoopT

Data structure for RNAsnoop.

#### 16.77.2.14 struct dupVar

Data structure used in RNAppkplex.

### 16.77.3 Typedef Documentation

#### 16.77.3.1 PAIR

```
typedef struct vrna_basepair_s PAIR
```

```
#include <ViennaRNA/datastructures/basic.h>
```

Old typename of `vrna_basepair_s`.

**Deprecated** Use `vrna_basepair_t` instead!

#### 16.77.3.2 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!

#### 16.77.3.3 cpair

```
typedef struct vrna_cpair_s cpair
```

```
#include <ViennaRNA/datastructures/basic.h>
```

Old typename of `vrna_cpair_s`.

**Deprecated** Use `vrna_cpair_t` instead!

### 16.77.3.4 sect

```
typedef struct vrna_sect_s sect
#include <ViennaRNA/datastructures/basic.h>
```

Old typename of `vrna_sect_s`.

**Deprecated** Use `vrna_sect_t` instead!

### 16.77.3.5 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!

## 16.77.4 Function Documentation

### 16.77.4.1 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

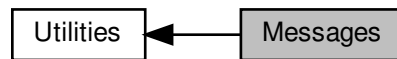
## 16.78 Messages

Functions to print various kind of messages.

### 16.78.1 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.*

### 16.78.2 Function Documentation

### 16.78.2.1 `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

### 16.78.2.2 `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

### 16.78.2.3 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

### 16.78.2.4 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

### 16.78.2.5 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

#### 16.78.2.6 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

#### 16.78.2.7 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

#### 16.78.2.8 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

<i>s</i>	A user defined string that will be printed to stdout
----------	--

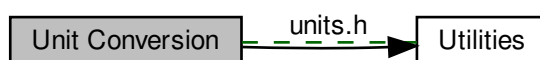
## 16.79 Unit Conversion

Functions to convert between various physical units.

### 16.79.1 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.*
- int [vrna\\_convert\\_kcal\\_to\\_dcal](#) (double energy)  
*Convert floating point energy value into integer representation.*
- double [vrna\\_convert\\_dcal\\_to\\_kcal](#) (int energy)  
*Convert an integer representation of free energy in deka-cal/mol to kcal/mol.*

## 16.79.2 Enumeration Type Documentation

### 16.79.2.1 vrna\_unit\_energy\_e

```
enum vrna_unit_energy_e
```

```
#include <ViennaRNA/utils/units.h>
```

Energy / Work Units.

See also

[vrna\\_convert\\_energy\(\)](#)

Enumerator

VRNA_UNIT_J	Joule ( $1\text{ J} = 1\text{ kg} \cdot \text{m}^2\text{s}^{-2}$ )
VRNA_UNIT_KJ	Kilojoule ( $1\text{ kJ} = 1,000\text{ J}$ )
VRNA_UNIT_CAL_IT	Calorie (International (Steam) Table, $1\text{ cal}_{IT} = 4.1868\text{ J}$ )
VRNA_UNIT_DACAL_IT	Decacalorie (International (Steam) Table, $1\text{ dcal}_{IT} = 10\text{ cal}_{IT} = 41.868\text{ J}$ )
VRNA_UNIT_KCAL_IT	Kilocalorie (International (Steam) Table, $1\text{ kcal}_{IT} = 4.1868\text{ kJ}$ )
VRNA_UNIT_CAL	Calorie (Thermochemical, $1\text{ cal}_{th} = 4.184\text{ J}$ )
VRNA_UNIT_DACAL	Decacalorie (Thermochemical, $1\text{ dcal}_{th} = 10\text{ cal}_{th} = 41.84\text{ J}$ )
VRNA_UNIT_KCAL	Kilocalorie (Thermochemical, $1\text{ kcal}_{th} = 4.184\text{ kJ}$ )
VRNA_UNIT_G_TNT	g TNT ( $1\text{ g TNT} = 1,000\text{ cal}_{th} = 4,184\text{ J}$ )
VRNA_UNIT_KG_TNT	kg TNT ( $1\text{ kg TNT} = 1,000\text{ kcal}_{th} = 4,184\text{ kJ}$ )
VRNA_UNIT_T_TNT	ton TNT ( $1\text{ t TNT} = 1,000,000\text{ kcal}_{th} = 4,184\text{ MJ}$ )
VRNA_UNIT_EV	Electronvolt ( $1\text{ eV} = 1.602176565 \times 10^{-19}\text{ J}$ )
VRNA_UNIT_WH	Watt hour ( $1\text{ W} \cdot \text{h} = 1\text{ W} \cdot 3,600\text{ s} = 3,600\text{ J} = 3.6\text{ kJ}$ )
VRNA_UNIT_KWH	Kilowatt hour ( $1\text{ kW} \cdot \text{h} = 1\text{ kW} \cdot 3,600\text{ s} = 3,600\text{ kJ} = 3.6\text{ MJ}$ )

### 16.79.2.2 vrna\_unit\_temperature\_e

```
enum vrna_unit_temperature_e
```

```
#include <ViennaRNA/utils/units.h>
```

Temperature Units.

See also

[vrna\\_convert\\_temperature\(\)](#)

## Enumerator

VRNA_UNIT_K	Kelvin (K)
VRNA_UNIT_DEG_C	Degree Celcius ( $^{\circ}\text{C}$ ) ( $[^{\circ}\text{C}] = [K] - 273.15$ )
VRNA_UNIT_DEG_F	Degree Fahrenheit ( $^{\circ}\text{F}$ ) ( $[^{\circ}\text{F}] = [K] \times \frac{9}{5} - 459.67$ )
VRNA_UNIT_DEG_R	Degree Rankine ( $^{\circ}\text{R}$ ) ( $[^{\circ}\text{R}] = [K] \times \frac{9}{5}$ )
VRNA_UNIT_DEG_N	Degree Newton ( $^{\circ}\text{N}$ ) ( $[^{\circ}\text{N}] = ([K] - 273.15) \times \frac{33}{100}$ )
VRNA_UNIT_DEG_DE	Degree Delisle ( $^{\circ}\text{De}$ ) ( $[^{\circ}\text{De}] = (373.15 - [K]) \times \frac{3}{2}$ )
VRNA_UNIT_DEG_RE	Degree Réaumur ( $^{\circ}\text{Ré}$ ) ( $[^{\circ}\text{Ré}] = ([K] - 273.15) \times \frac{4}{5}$ )
VRNA_UNIT_DEG_RO	Degree Rømer ( $^{\circ}\text{Rø}$ ) ( $[^{\circ}\text{Rø}] = ([K] - 273.15) \times \frac{21}{40} + 7.5$ )

## 16.79.3 Function Documentation

## 16.79.3.1 vrna\_convert\_energy()

```
double vrna_convert_energy (
    double energy,
    vrna_unit_energy_e from,
    vrna_unit_energy_e to )
```

```
#include <ViennaRNA/utils/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

## 16.79.3.2 vrna\_convert\_temperature()

```
double vrna_convert_temperature (
    double temp,
```

```
vrna_unit_temperature_e from,  
vrna_unit_temperature_e to )
```

```
#include <ViennaRNA/utils/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

### 16.79.3.3 vrna\_convert\_kcal\_to\_dcal()

```
int vrna_convert_kcal_to_dcal (  
    double energy )
```

```
#include <ViennaRNA/utils/units.h>
```

Convert floating point energy value into integer representation.

This function converts a floating point value in kcal/mol into its corresponding deka-cal/mol integer representation as used throughout RNAlib.

See also

[vrna\\_convert\\_dcal\\_to\\_kcal\(\)](#)

#### Parameters

<i>energy</i>	The energy value in kcal/mol
---------------	------------------------------

#### Returns

The energy value in deka-cal/mol

### 16.79.3.4 vrna\_convert\_dcal\_to\_kcal()

```
double vrna_convert_dcal_to_kcal (  
    int energy )
```

```
#include <ViennaRNA/utils/units.h>
```

Convert an integer representation of free energy in deka-cal/mol to kcal/mol.

This function converts a free energy value given as integer in deka-cal/mol into the corresponding floating point number in kcal/mol

See also

[vrna\\_convert\\_kcal\\_to\\_dcal\(\)](#)

#### Parameters

<i>energy</i>	The energy in deka-cal/mol
---------------	----------------------------

#### Returns

The energy in kcal/mol

## 16.80 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.

### 16.80.1 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, const 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`.*

## 16.80.2 Data Structure Documentation

### 16.80.2.1 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.

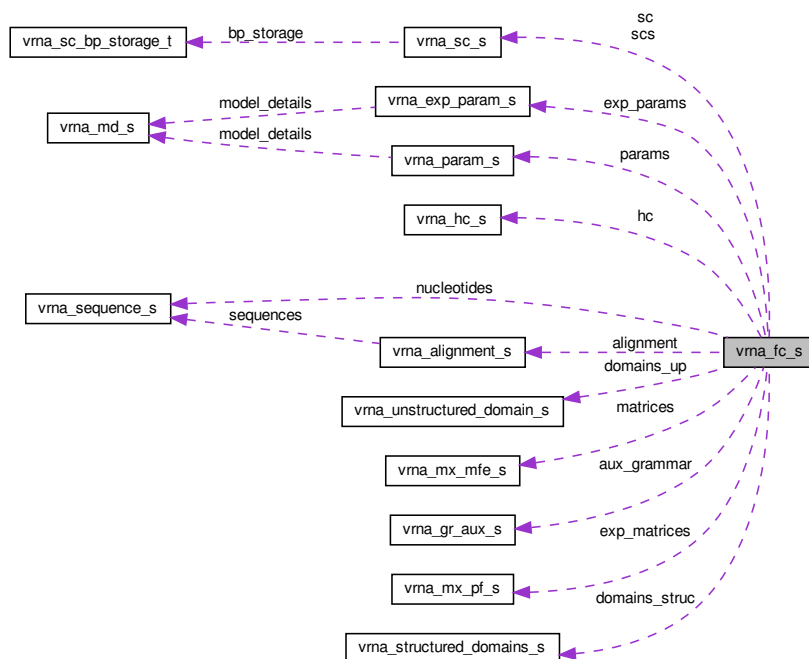
A new **fold\_compound** can be obtained by calling one of its constructors:

- *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](#)  
*The strand order, i.e. permutation of current concatenated sequence.*
- unsigned int \* [strand\\_start](#)  
*The start position of a particular strand within the current concatenated sequence.*
- unsigned int \* [strand\\_end](#)  
*The end (last) position of a particular strand within the current concatenated sequence.*
- unsigned int **strands**
- [vrna\\_seq\\_t](#) \* **nucleotides**
- [vrna\\_msa\\_t](#) \* **alignment**
- [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\_struc**  
*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 \*\* `pctype_local`  
*Pair type array (for local folding)*
- `vrna_zsc_dat_t` `zscore_data`

#### 16.80.2.1.1 Field Documentation

**16.80.2.1.1.1 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.

**16.80.2.1.1.2 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\(\)](#)

**16.80.2.1.1.3 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](#)

**16.80.2.1.1.4 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\(\)](#)

**16.80.2.1.1.5 sequence** `char* vrna_fc_s::sequence`

The input sequence string.

Warning

Only available if

`type==VRNA_FC_TYPE_SINGLE`

**16.80.2.1.1.6 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`

**16.80.2.1.1.7 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\(\)](#)

**16.80.2.1.1.8 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
```

**16.80.2.1.1.9 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
```

**16.80.2.1.1.10 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
```

**16.80.2.1.1.11 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
```

**16.80.2.1.1.12 cons\_seq** `char* vrna_fc_s::cons_seq`

The consensus sequence of the aligned sequences.

**Warning**

Only available if

```
type==VRNA_FC_TYPE_COMPARATIVE
```

**16.80.2.1.1.13 S\_cons** `short* vrna_fc_s::S_cons`

Numerical encoding of the consensus sequence.

**Warning**

Only available if

```
type==VRNA_FC_TYPE_COMPARATIVE
```

**16.80.2.1.1.14 S** `short** vrna_fc_s::S`

Numerical encoding of the sequences in the alignment.

**Warning**

Only available if

`type==VRNA_FC_TYPE_COMPARATIVE`

**16.80.2.1.1.15 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`

**16.80.2.1.1.16 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`

**16.80.2.1.1.17 pscore** `int* vrna_fc_s::pscore`

Precomputed array of pair types expressed as pairing scores.

**Warning**

Only available if

`type==VRNA_FC_TYPE_COMPARATIVE`



**16.80.2.1.1.18 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
```

**16.80.2.1.1.19 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
```

**16.80.2.1.1.20 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
```

## 16.80.3 Macro Definition Documentation

### 16.80.3.1 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\(\)](#)

### 16.80.3.2 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\(\)](#)

### 16.80.3.3 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\(\)](#)

### 16.80.3.4 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\(\)](#)

### 16.80.3.5 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](#)

### 16.80.3.6 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](#)

### 16.80.3.7 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\(\)](#)

## 16.80.4 Typedef Documentation

### 16.80.4.1 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
-------------	----------------------------------

**16.80.4.2 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

**16.80.5 Enumeration Type Documentation****16.80.5.1 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)

## 16.80.6 Function Documentation

### 16.80.6.1 `vrna_fold_compound()`

```
vrna_fold_compound_t* vrna_fold_compound (
    const char * sequence,
    const 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)

**16.80.6.2 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)

**16.80.6.3 `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
-----------	--

**16.80.6.4 `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)

**16.80.6.5 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



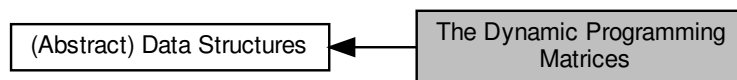
## 16.81 The Dynamic Programming Matrices

This module provides interfaces that deal with creation and destruction of dynamic programming matrices used within the RNAlib.

### 16.81.1 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.

## 16.81.2 Data Structure Documentation

### 16.81.2.1 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`

### 16.81.2.2 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`

### 16.81.3 Enumeration Type Documentation

#### 16.81.3.1 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>

### 16.81.4 Function Documentation

#### 16.81.4.1 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**

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> that holds pointers to the DP matrices
<i>type</i>	The type of DP matrices requested
<i>options</i>	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

**16.81.4.2 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**

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> storing the MFE DP matrices that are to be erased from memory
-----------	--

**16.81.4.3 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
----	---

## 16.82 Hash Tables

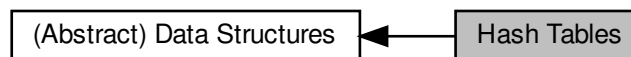
Various implementations of hash table functions.

### 16.82.1 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.*

## 16.82.2 Data Structure Documentation

### 16.82.2.1 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](#)

#### 16.82.2.1.1 Field Documentation

##### 16.82.2.1.1.1 **structure** `char* vrna_ht_entry_db_t::structure`

A secondary structure in dot-bracket notation

##### 16.82.2.1.1.2 **energy** `float vrna_ht_entry_db_t::energy`

The free energy of [structure](#)

## 16.82.3 Typedef Documentation

### 16.82.3.1 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\(\)](#)

### 16.82.3.2 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  $x == y$

### 16.82.3.3 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*

**16.82.3.4 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

**16.82.4 Function Documentation****16.82.4.1 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

**16.82.4.2 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

**16.82.4.3 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

#### 16.82.4.4 `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

#### 16.82.4.5 `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.

**16.82.4.6 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

**16.82.4.7 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 [vrna\\_callback\\_ht\\_free\\_entry\(\)](#) function.

See also

[vrna\\_ht\\_free\(\)](#), [vrna\\_ht\\_init\(\)](#)

Parameters

<i>ht</i>	The hash table
-----------	----------------

#### 16.82.4.8 [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
-----------	----------------

#### 16.82.4.9 [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.

**16.82.4.10 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*

**16.82.4.11 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

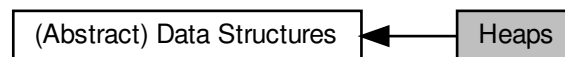
## 16.83 Heaps

Interface for an abstract implementation of a heap data structure.

### 16.83.1 Detailed Description

Interface for an abstract implementation of a heap data structure.

Collaboration diagram for Heaps:



### Files

- file [heap.h](#)  
*Implementation of an abstract heap data structure.*

### Typedefs

- typedef struct vrna\_heap\_s \* [vrna\\_heap\\_t](#)  
*An abstract heap data structure.*
- typedef int() [vrna\\_callback\\_heap\\_cmp](#)(const void \*a, const void \*b, void \*data)  
*Heap compare function prototype.*
- typedef size\_t() [vrna\\_callback\\_heap\\_get\\_pos](#)(const void \*a, void \*data)  
*Retrieve the position of a particular heap entry within the heap.*
- typedef void() [vrna\\_callback\\_heap\\_set\\_pos](#)(const void \*a, size\_t pos, void \*data)  
*Store the position of a particular heap entry within the heap.*

### Functions

- [vrna\\_heap\\_t vrna\\_heap\\_init](#) (size\_t n, [vrna\\_callback\\_heap\\_cmp](#) \*cmp, [vrna\\_callback\\_heap\\_get\\_pos](#) \*get↵\_entry\_pos, [vrna\\_callback\\_heap\\_set\\_pos](#) \*set\_entry\_pos, void \*data)  
*Initialize a heap data structure.*
- void [vrna\\_heap\\_free](#) ([vrna\\_heap\\_t](#) h)  
*Free memory occupied by a heap data structure.*
- size\_t [vrna\\_heap\\_size](#) (struct vrna\_heap\_s \*h)  
*Get the size of a heap data structure, i.e. the number of stored elements.*
- void [vrna\\_heap\\_insert](#) ([vrna\\_heap\\_t](#) h, void \*v)  
*Insert an element into the heap.*
- void \* [vrna\\_heap\\_pop](#) ([vrna\\_heap\\_t](#) h)  
*Pop (remove and return) the object at the root of the heap.*
- const void \* [vrna\\_heap\\_top](#) ([vrna\\_heap\\_t](#) h)  
*Get the object at the root of the heap.*
- void \* [vrna\\_heap\\_remove](#) ([vrna\\_heap\\_t](#) h, const void \*v)  
*Remove an arbitrary element within the heap.*
- void \* [vrna\\_heap\\_update](#) ([vrna\\_heap\\_t](#) h, void \*v)  
*Update an arbitrary element within the heap.*



## 16.83.2 Typedef Documentation

### 16.83.2.1 vrna\_heap\_t

```
typedef struct vrna_heap_s* vrna_heap_t  
  
#include <ViennaRNA/datastructures/heap.h>
```

An abstract heap data structure.

See also

[vrna\\_heap\\_init\(\)](#), [vrna\\_heap\\_free\(\)](#), [vrna\\_heap\\_insert\(\)](#), [vrna\\_heap\\_pop\(\)](#), [vrna\\_heap\\_top\(\)](#), [vrna\\_heap\\_remove\(\)](#), [vrna\\_heap\\_update\(\)](#)

### 16.83.2.2 vrna\_callback\_heap\_cmp

```
typedef int() vrna_callback_heap_cmp(const void *a, const void *b, void *data)  
  
#include <ViennaRNA/datastructures/heap.h>
```

Heap compare function prototype.

Use this prototype to design the compare function for the heap implementation. The arbitrary data pointer `data` may be used to get access to further information required to actually compare the two values `a` and `b`.

Note

The heap implementation acts as a *min-heap*, therefore, the minimum element will be present at the heap's root. In case a *max-heap* is required, simply reverse the logic of this compare function.

Parameters

<i>a</i>	The first object to compare
<i>b</i>	The second object to compare
<i>data</i>	An arbitrary data pointer passed through from the heap implementation

Returns

A value less than zero if  $a < b$ , a value greater than zero if  $a > b$ , and 0 otherwise

### 16.83.2.3 vrna\_callback\_heap\_get\_pos

```
typedef size_t() vrna_callback_heap_get_pos(const void *a, void *data)
```

```
#include <ViennaRNA/datastructures/heap.h>
```

Retrieve the position of a particular heap entry within the heap.

#### Parameters

<i>a</i>	The object to look-up within the heap
<i>data</i>	An arbitrary data pointer passed through from the heap implementation

#### Returns

The position of the element *a* within the heap, or 0 if it is not in the heap

### 16.83.2.4 vrna\_callback\_heap\_set\_pos

```
typedef void() vrna_callback_heap_set_pos(const void *a, size_t pos, void *data)
```

```
#include <ViennaRNA/datastructures/heap.h>
```

Store the position of a particular heap entry within the heap.

#### Parameters

<i>a</i>	The object whose position shall be stored
<i>pos</i>	The current position of <i>a</i> within the heap, or 0 if <i>a</i> was deleted
<i>data</i>	An arbitrary data pointer passed through from the heap implementation

## 16.83.3 Function Documentation

### 16.83.3.1 vrna\_heap\_init()

```
vrna_heap_t vrna_heap_init (
    size_t n,
    vrna_callback_heap_cmp * cmp,
    vrna_callback_heap_get_pos * get_entry_pos,
    vrna_callback_heap_set_pos * set_entry_pos,
    void * data )
```

```
#include <ViennaRNA/datastructures/heap.h>
```

Initialize a heap data structure.

This function initializes a heap data structure. The implementation is based on a *min-heap*, i.e. the minimal element is located at the root of the heap. However, by reversing the logic of the compare function, one can easily transform this into a *max-heap* implementation.

Beside the regular operations on a heap data structure, we implement removal and update of arbitrary elements within the heap. For that purpose, however, one requires a reverse-index lookup system that, (i) for a given element stores the current position in the heap, and (ii) allows for fast lookup of an elements current position within the heap. The corresponding getter- and setter- functions may be provided through the arguments `get_entry_pos` and `set_entry_pos`, respectively.

Sometimes, it is difficult to simply compare two data structures without any context. Therefore, the compare function is provided with a user-defined data pointer that can hold any context required.

#### Warning

If any of the arguments `get_entry_pos` or `set_entry_pos` is NULL, the operations `vrna_heap_update()` and `vrna_heap_remove()` won't work.

#### See also

`vrna_heap_free()`, `vrna_heap_insert()`, `vrna_heap_pop()`, `vrna_heap_top()`, `vrna_heap_remove()`, `vrna_heap_update()`, `vrna_heap_t`, `vrna_callback_heap_cmp`, `vrna_callback_heap_get_pos`, `vrna_callback_heap_set_pos`

#### Parameters

<i>n</i>	The initial size of the heap, i.e. the number of elements to store
<i>cmp</i>	The address of a compare function that will be used to fulfill the partial order requirement
<i>get_entry_pos</i>	The address of a function that retrieves the position of an element within the heap (or NULL)
<i>set_entry_pos</i>	The address of a function that stores the position of an element within the heap (or NULL)
<i>data</i>	An arbitrary data pointer passed through to the compare function <code>cmp</code> , and the set/get functions <code>get_entry_pos</code> / <code>set_entry_pos</code>

#### Returns

An initialized heap data structure, or NULL on error

#### 16.83.3.2 vrna\_heap\_free()

```
void vrna_heap_free (
    vrna_heap_t h )

#include <ViennaRNA/datastructures/heap.h>
```

Free memory occupied by a heap data structure.

#### See also

`vrna_heap_init()`

**Parameters**

<i>h</i>	The heap that should be free'd
----------	--------------------------------

**16.83.3.3 vrna\_heap\_size()**

```
size_t vrna_heap_size (  
    struct vrna_heap_s * h )
```

```
#include <ViennaRNA/datastructures/heap.h>
```

Get the size of a heap data structure, i.e. the number of stored elements.

**Parameters**

<i>h</i>	The heap data structure
----------	-------------------------

**Returns**

The number of elements currently stored in the heap, or 0 upon any error

**16.83.3.4 vrna\_heap\_insert()**

```
void vrna_heap_insert (  
    vrna_heap_t h,  
    void * v )
```

```
#include <ViennaRNA/datastructures/heap.h>
```

Insert an element into the heap.

**See also**

[vrna\\_heap\\_init\(\)](#), [vrna\\_heap\\_pop\(\)](#), [vrna\\_heap\\_top\(\)](#), [vrna\\_heap\\_free\(\)](#), [vrna\\_heap\\_remove\(\)](#), [vrna\\_heap\\_update\(\)](#)

**Parameters**

<i>h</i>	The heap data structure
<i>v</i>	A pointer to the object that is about to be inserted into the heap

### 16.83.3.5 vrna\_heap\_pop()

```
void* vrna_heap_pop (
    vrna_heap_t h )
```

```
#include <ViennaRNA/datastructures/heap.h>
```

Pop (remove and return) the object at the root of the heap.

This function removes the root from the heap and returns it to the caller.

#### See also

[vrna\\_heap\\_init\(\)](#), [vrna\\_heap\\_top\(\)](#), [vrna\\_heap\\_insert\(\)](#), [vrna\\_heap\\_free\(\)](#) [vrna\\_heap\\_remove\(\)](#), [vrna\\_heap\\_update\(\)](#)

#### Parameters

<i>h</i>	The heap data structure
----------	-------------------------

#### Returns

The object at the root of the heap, i.e. the minimal element (or NULL if (a) the heap is empty or (b) any error occurred)

### 16.83.3.6 vrna\_heap\_top()

```
const void* vrna_heap_top (
    vrna_heap_t h )
```

```
#include <ViennaRNA/datastructures/heap.h>
```

Get the object at the root of the heap.

#### See also

[vrna\\_heap\\_init\(\)](#), [vrna\\_heap\\_pop\(\)](#), [vrna\\_heap\\_insert\(\)](#), [vrna\\_heap\\_free\(\)](#) [vrna\\_heap\\_remove\(\)](#), [vrna\\_heap\\_update\(\)](#)

#### Parameters

<i>h</i>	The heap data structure
----------	-------------------------

#### Returns

The object at the root of the heap, i.e. the minimal element (or NULL if (a) the heap is empty or (b) any error occurred)

### 16.83.3.7 vrna\_heap\_remove()

```
void* vrna_heap_remove (
    vrna_heap_t h,
    const void * v )
```

```
#include <ViennaRNA/datastructures/heap.h>
```

Remove an arbitrary element within the heap.

#### See also

[vrna\\_heap\\_init\(\)](#), [vrna\\_callback\\_heap\\_get\\_pos](#), [vrna\\_callback\\_heap\\_set\\_pos](#), [vrna\\_heap\\_pop\(\)](#), [vrna\\_heap\\_free\(\)](#)

#### Warning

This function won't work if the heap was not properly initialized with callback functions for fast reverse-index mapping!

#### Parameters

<i>h</i>	The heap data structure
<i>v</i>	The object to remove from the heap

#### Returns

The object that was removed from the heap (or NULL if (a) it wasn't found or (b) any error occurred)

### 16.83.3.8 vrna\_heap\_update()

```
void* vrna_heap_update (
    vrna_heap_t h,
    void * v )
```

```
#include <ViennaRNA/datastructures/heap.h>
```

Update an arbitrary element within the heap.

#### Note

If the object that is to be updated is not currently stored in the heap, it will be inserted. In this case, the function returns NULL.

#### Warning

This function won't work if the heap was not properly initialized with callback functions for fast reverse-index mapping!

#### See also

[vrna\\_heap\\_init\(\)](#), [vrna\\_callback\\_heap\\_get\\_pos](#), [vrna\\_callback\\_heap\\_set\\_pos](#), [vrna\\_heap\\_pop\(\)](#), [vrna\\_heap\\_remove\(\)](#), [vrna\\_heap\\_free\(\)](#)

**Parameters**

$h$	The heap data structure
$v$	The object to update

**Returns**

The 'previous' object within the heap that now got replaced by  $v$  (or NULL if (a) it wasn't found or (b) any error occurred)

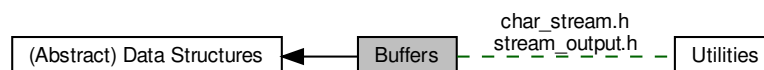
## 16.84 Buffers

Functions that provide dynamically buffered stream-like data structures.

### 16.84.1 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.*



## 16.84.2 Typedef Documentation

### 16.84.2.1 `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

## 16.84.3 Function Documentation

### 16.84.3.1 `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> )

### 16.84.3.2 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
------------	--

### 16.84.3.3 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
------------	--

### 16.84.3.4 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
------------	---

### 16.84.3.5 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

### 16.84.3.6 `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
------------	--

### 16.84.3.7 `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

### 16.84.3.8 `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\(\)](#)

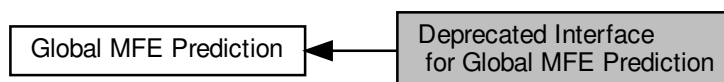
#### Parameters

<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

## 16.85 Deprecated Interface for Global MFE Prediction

### 16.85.1 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 gquadriplex 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 \*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)
  - void [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)
  - 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 [circularfold](#) (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.*

## 16.85.2 Function Documentation

### 16.85.2.1 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

**16.85.2.2 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

**16.85.2.3 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



#### 16.85.2.4 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\(\)](#)

#### 16.85.2.5 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

#### 16.85.2.6 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

### 16.85.2.7 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)

### 16.85.2.8 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)

#### 16.85.2.9 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

**16.85.2.10 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!}

**16.85.2.11 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

## 16.85.2.12 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

### 16.85.2.13 `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

### 16.85.2.14 `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!

**16.85.2.15 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!

**16.85.2.16 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!

**16.85.2.17 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!

### 16.85.2.18 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!

### 16.85.2.19 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!

### 16.85.2.20 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!



### 16.85.2.21 LoopEnergy()

```
int LoopEnergy (
    int n1,
    int n2,
    int type,
    int type_2,
    int sil,
    int sj1,
    int sp1,
    int sq1 )

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

**Deprecated** {This function is deprecated and will be removed soon. Use [E\\_IntLoop\(\)](#) instead!}

### 16.85.2.22 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!}

### 16.85.2.23 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!

#### 16.85.2.24 `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

#### 16.85.2.25 `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()`

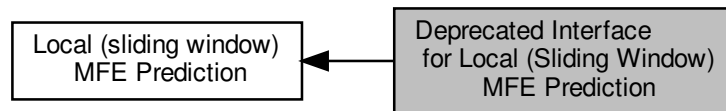
See also

`vrna_fold_compound_free()`

## 16.86 Deprecated Interface for Local (Sliding Window) MFE Prediction

### 16.86.1 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)

## 16.86.2 Function Documentation

### 16.86.2.1 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!

### 16.86.2.2 Lfoldz()

```
float Lfoldz (
    const char * string,
    const char * structure,
    int maxdist,
    int zsc,
    double min_z )
```

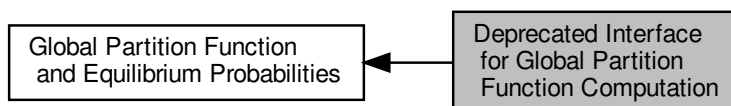
```
#include <ViennaRNA/Lfold.h>
```

**Deprecated** Use `vrna_mfe_window_zscore()` instead!

## 16.87 Deprecated Interface for Global Partition Function Computation

### 16.87.1 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\_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.*
- 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.*

## 16.87.2 Function Documentation

16.87.2.1 `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

16.87.2.2 `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 ensemble free energy  $G = -RT \cdot \log(Q)$  in kcal/mol

**16.87.2.3 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 ensemble free energy  $G = -RT \cdot \log(Q)$  in kcal/mol

**16.87.2.4 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 ensemble free energy  $G = -RT \cdot \log(Q)$  in kcal/mol

**16.87.2.5 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\(\)](#)

### 16.87.2.6 `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

### 16.87.2.7 `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

### 16.87.2.8 `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

### 16.87.2.9 `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 ** q1k_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>ptype</i> <sub>← _p</sub>	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

### 16.87.2.10 `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

**16.87.2.11 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

$$\langle d \rangle = \sum_{ij} p_{ij} (1 - p_{ij})$$

**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

**16.87.2.12 stackProb()**

```
vrna_ep_t* stackProb (
    double cutoff )
```

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

Get the probability of stacks.

**Deprecated** Use [vrna\\_stack\\_prob\(\)](#) instead!

**16.87.2.13 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!

**16.87.2.14 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.

**16.87.2.15 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.

#### 16.87.2.16 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

**16.87.2.17 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!}

**16.87.2.18 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



### 16.87.2.19 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

### 16.87.2.20 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
---------------	------------------------------------

### 16.87.2.21 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. Its 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

**16.87.2.22 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 <code>vrna_ep_t</code> that is to be created
<i>struc</i>	The secondary structure in dot-bracket notation
<i>pr</i>	The probability for each base pair

**16.87.2.23 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

**16.87.2.24 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****16.87.2.25 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

### 16.87.2.26 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

**16.87.2.27 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()`

**16.87.2.28 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

**16.87.2.29 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

<code>S_p</code>	A pointer to the 'S' array (integer representation of nucleotides)
<code>S5_p</code>	A pointer to the 'S5' array
<code>S3_p</code>	A pointer to the 'S3' array
<code>a2s↔ _p</code>	A pointer to the alignment-column to sequence position mapping array
<code>Ss_p</code>	A pointer to the 'Ss' array
<code>qb_p</code>	A pointer to the $Q^B$ matrix
<code>qm_p</code>	A pointer to the $Q^M$ matrix
<code>q1k↔ _p</code>	A pointer to the 5' slice of the Q matrix ( $q1k(k) = Q(1, k)$ )
<code>qln_p</code>	A pointer to the 3' slice of the Q matrix ( $qln(l) = Q(l, n)$ )
<code>pscore</code>	A pointer to the start of a pscore list

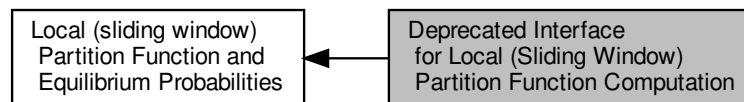
#### Returns

Non Zero if everything went fine, 0 otherwise

## 16.88 Deprecated Interface for Local (Sliding Window) Partition Function Computation

### 16.88.1 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.*

### 16.88.2 Function Documentation

#### 16.88.2.1 [update\\_pf\\_paramsLP\(\)](#)

```

void update_pf_paramsLP (
    int length )

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

## Parameters

<i>length</i>	
---------------	--

## 16.88.2.2 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



**16.88.2.3 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

**16.88.2.4 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

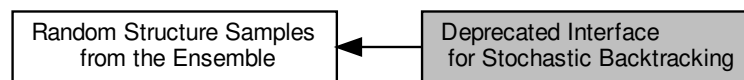
**Parameters**

<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

## 16.89 Deprecated Interface for Stochastic Backtracking

### 16.89.1 Detailed Description

Collaboration diagram for Deprecated Interface for Stochastic Backtracking:



### Functions

- char \* [pbacktrack](#) (char \*sequence)  
*Sample a secondary structure from the Boltzmann ensemble according its probability.*
- char \* [pbacktrack5](#) (char \*sequence, int length)  
*Sample a sub-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.*

### 16.89.2 Function Documentation

#### 16.89.2.1 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

**16.89.2.2 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

**16.89.3 Variable Documentation****16.89.3.1 st\_back**

```
int st_back [extern]
```

```
#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()`.

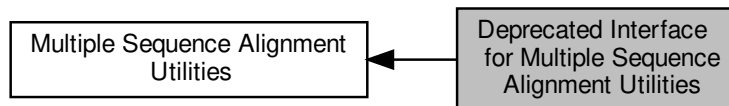
See also

`pbacktrack()`, `pbacktrack_circ`

## 16.90 Deprecated Interface for Multiple Sequence Alignment Utilities

### 16.90.1 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.*

### 16.90.2 Typedef Documentation

#### 16.90.2.1 [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!

## 16.90.3 Function Documentation

### 16.90.3.1 `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

### 16.90.3.2 `encode_ali_sequence()`

```
void encode_ali_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)

**16.90.3.3 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)

**16.90.3.4 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

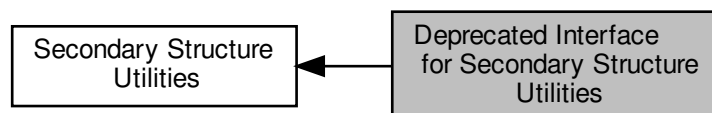
<i>n_seq</i>	The number of aligned sequences
<i>S</i>	A pointer to the array of encoded sequences
<i>S5</i>	A pointer to the array that contains the next 5' nucleotide of a sequence position
<i>S3</i>	A pointer to the array that contains the next 3' nucleotide of a sequence position
<i>a2s</i>	A pointer to the array that contains the alignment to sequence position mapping
<i>Ss</i>	A pointer to the array that contains the ungapped sequence



## 16.91 Deprecated Interface for Secondary Structure Utilities

### 16.91.1 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.*

## 16.91.2 Function Documentation

### 16.91.2.1 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

<i>structure</i>	
------------------	--

## Returns

**16.91.2.2 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

**16.91.2.3 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****16.91.2.4 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****16.91.2.5 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**

### 16.91.2.6 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

### 16.91.2.7 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

### 16.91.2.8 unweight()

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

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

Strip weights from any weighted tree.

#### Parameters

<i>wcoarse</i>	
----------------	--

## Returns

### 16.91.2.9 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>	
--------------	--

### 16.91.2.10 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

### 16.91.2.11 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>struc</i>	The secondary structure in dot-bracket notation
--------------	---

#### Returns

The binary encoded structure

### 16.91.2.12 `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

### 16.91.2.13 `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

### 16.91.2.14 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'

### 16.91.2.15 alimake\_pair\_table()

```
short* alimake_pair_table (
    const char * structure )
```

```
#include <ViennaRNA/utils/structures.h>
```

Pair table for snoop align

**Deprecated** Use `vrna_pt.ali_get()` instead!



**16.91.2.16 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!

**16.91.2.17 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*

**16.91.2.18 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 `iindx!!!`

**Deprecated** Use `vrna_refBPcnt_matrix()` instead

#### 16.91.2.19 `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

#### 16.91.2.20 `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

**16.91.2.21 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

**16.91.2.22 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!

**16.91.2.23 bppm\_symbol()**

```
char bppm_symbol (
    const float * x )
```

```
#include <ViennaRNA/utils/structures.h>
```

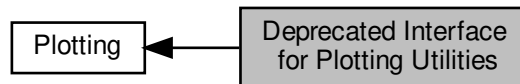
Get a pseudo dot bracket notation for a given probability information.

**Deprecated** Use `vrna_bpp_symbol()` instead!

## 16.92 Deprecated Interface for Plotting Utilities

### 16.92.1 Detailed Description

Collaboration diagram for Deprecated Interface for Plotting Utilities:



### Data Structures

- struct [COORDINATE](#)

*this is a workaround for the SWIG Perl Wrapper RNA plot function that returns an array of type [COORDINATE](#) [More...](#)*

### 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 [aliPS\\_color\\_aln](#) (const char \*structure, const char \*filename, const char \*seqs[], const char \*names[])  
*PS\_color\_aln for duplexes.*
- 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 [naview\\_xy\\_coordinates](#) (short \*pair\_table, float \*X, float \*Y)

### Variables

- int [rna\\_plot\\_type](#)  
*Switch for changing the secondary structure layout algorithm.*

## 16.92.2 Data Structure Documentation

### 16.92.2.1 struct COORDINATE

this is a workaround for the SWIG Perl Wrapper RNA plot function that returns an array of type [COORDINATE](#)

## 16.92.3 Function Documentation

### 16.92.3.1 PS\_color\_aln()

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

```
#include <ViennaRNA/plotting/alignments.h>
```

Produce PostScript sequence alignment color-annotated by consensus structure.

**Deprecated** Use [vrna\\_file\\_PS\\_aln\(\)](#) instead!

### 16.92.3.2 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.

**Deprecated** Use [vrna\\_file\\_PS\\_aln\(\)](#) instead!

### 16.92.3.3 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\(\)](#)

**Deprecated** Consider switching to [vrna\\_plot\\_coords\\_simple\\_pt\(\)](#) instead!

## 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

**16.92.3.4 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\(\)](#)

**Deprecated** Consider switching to [vrna\\_plot\\_coords\\_circular\\_pt\(\)](#) instead!

## 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

### 16.92.3.5 `naview_xy_coordinates()`

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

#include <ViennaRNA/plotting/naview.h>
```

**Deprecated** Consider using `vrna_plot_coords_naview_pt()` instead!

## 16.92.4 Variable Documentation

### 16.92.4.1 `rna_plot_type`

```
int rna_plot_type [extern]

#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!

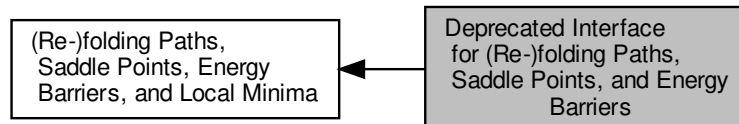
#### See also

[VRNA\\_PLOT\\_TYPE\\_SIMPLE](#), [VRNA\\_PLOT\\_TYPE\\_NAVIEW](#), [VRNA\\_PLOT\\_TYPE\\_CIRCULAR](#)

## 16.93 Deprecated Interface for (Re-)folding Paths, Saddle Points, and Energy Barriers

### 16.93.1 Detailed Description

Collaboration diagram for Deprecated Interface for (Re-)folding Paths, Saddle Points, and Energy Barriers:



### Typedefs

- typedef struct [vrna\\_path\\_s](#) path\_t  
*Old typename of [vrna\\_path\\_s](#).*

### Functions

- 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)*

### 16.93.2 Typedef Documentation

#### 16.93.2.1 path\_t

```
typedef struct vrna\_path\_s path_t
```

```
#include <ViennaRNA/landscape/paths.h>
```

Old typename of [vrna\\_path\\_s](#).

**Deprecated** Use [vrna\\_path\\_t](#) instead!



### 16.93.3 Function Documentation

#### 16.93.3.1 find\_saddle()

```
int find_saddle (
    const char * seq,
    const char * s1,
    const char * s2,
    int width )
```

```
#include <ViennaRNA/landscape/findpath.h>
```

Find energy of a saddle point between 2 structures (search only direct path)

**Deprecated** Use `vrna_path_findpath_saddle()` instead!

##### 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

#### 16.93.3.2 free\_path()

```
void free_path (
    vrna_path_t * path )
```

```
#include <ViennaRNA/landscape/findpath.h>
```

Free memory allocated by `get_path()` function.

**Deprecated** Use `vrna_path_free()` instead!

##### Parameters

<i>path</i>	pointer to memory to be freed
-------------	-------------------------------

### 16.93.3.3 `get_path()`

```
vrna_path_t* get_path (
    const char * seq,
    const char * s1,
    const char * s2,
    int width )
```

```
#include <ViennaRNA/landscape/findpath.h>
```

Find refolding path between 2 structures (search only direct path)

**Deprecated** Use `vrna_path_findpath()` instead!

#### 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

## Chapter 17

# Data Structure Documentation

### 17.1 `_struct_en` Struct Reference

Data structure for [energy\\_of\\_move\(\)](#)

#### 17.1.1 Detailed Description

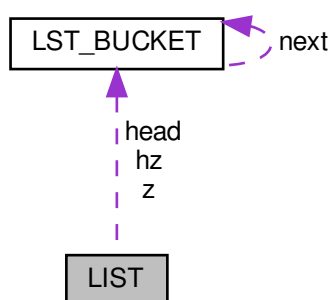
Data structure for [energy\\_of\\_move\(\)](#)

The documentation for this struct was generated from the following file:

- ViennaRNA/move\_set.h

### 17.2 LIST Struct Reference

Collaboration diagram for LIST:



The documentation for this struct was generated from the following file:

- ViennaRNA/datastructures/lists.h

## 17.3 LST\_BUCKET Struct Reference

Collaboration diagram for LST\_BUCKET:



The documentation for this struct was generated from the following file:

- ViennaRNA/datastructures/lists.h

## 17.4 Postorder\_list Struct Reference

Postorder data structure.

### 17.4.1 Detailed Description

Postorder data structure.

The documentation for this struct was generated from the following file:

- ViennaRNA/[dist\\_vars.h](#)

## 17.5 swString Struct Reference

Some other data structure.

### 17.5.1 Detailed Description

Some other data structure.

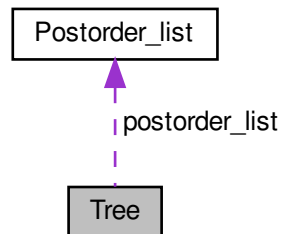
The documentation for this struct was generated from the following file:

- ViennaRNA/[dist\\_vars.h](#)

## 17.6 Tree Struct Reference

[Tree](#) data structure.

Collaboration diagram for Tree:



### 17.6.1 Detailed Description

[Tree](#) data structure.

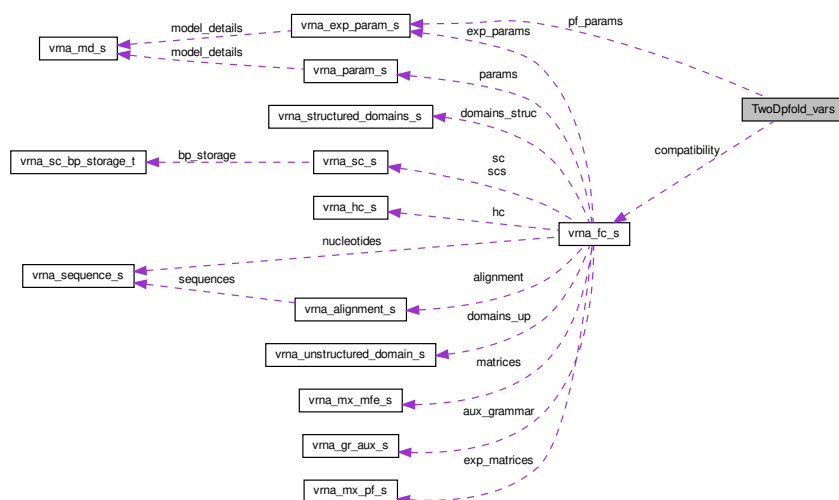
The documentation for this struct was generated from the following file:

- [ViennaRNA/dist\\_vars.h](#)

## 17.7 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.*

### 17.7.1 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](#)

## 17.8 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.*

### 17.8.1 Detailed Description

Data structure for concentration dependency computations.

The documentation for this struct was generated from the following file:

- ViennaRNA/[concentrations.h](#)

## 17.9 vrna\_sc\_bp\_storage\_t Struct Reference

A base pair constraint.

### 17.9.1 Detailed Description

A base pair constraint.

The documentation for this struct was generated from the following file:

- ViennaRNA/constraints/[soft.h](#)

## 17.10 vrna\_sc\_motif\_s Struct Reference

The documentation for this struct was generated from the following file:

- ViennaRNA/constraints/[ligand.h](#)

## 17.11 vrna\_structured\_domains\_s Struct Reference

The documentation for this struct was generated from the following file:

- ViennaRNA/[structured\\_domains.h](#)

## 17.12 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.*

### 17.12.1 Detailed Description

Solution element from subopt.c.

The documentation for this struct was generated from the following file:

- ViennaRNA/[subopt.h](#)

## 17.13 vrna\_unstructured\_domain\_motif\_s Struct Reference

The documentation for this struct was generated from the following file:

- ViennaRNA/[unstructured\\_domains.h](#)



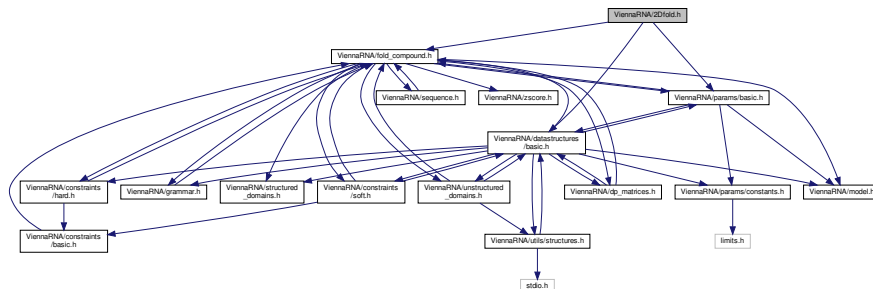
## Chapter 18

# File Documentation

### 18.1 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_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.*

### 18.2.1 Detailed Description

Partition function implementations for base pair distance classes.

### 18.2.2 Function Documentation

#### 18.2.2.1 `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

**18.2.2.2 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
-------------	--------------------------------

**18.2.2.3 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

#### 18.2.2.4 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

**18.2.2.5 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

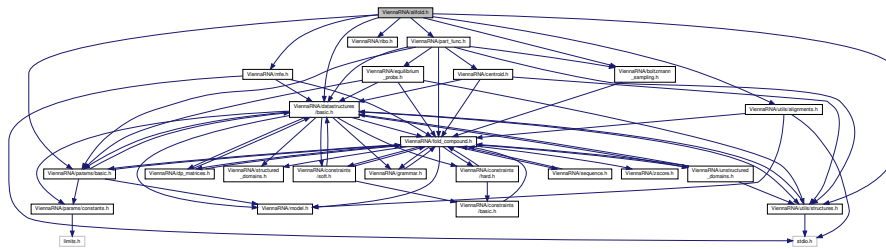
## Returns

A sampled secondary structure in dot-bracket notation

### 18.3 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 [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.*
- 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.*

### 18.3.1 Detailed Description

Functions for comparative structure prediction using RNA sequence alignments.

### 18.3.2 Function Documentation

#### 18.3.2.1 `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

#### 18.3.2.2 `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](#).

### 18.3.3 Variable Documentation

#### 18.3.3.1 cv\_fact

```
double cv_fact [extern]
```

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.

#### 18.3.3.2 nc\_fact

```
double nc_fact [extern]
```

This variable controls the magnitude of the penalty for non-compatible sequences in the covariance term of alignment folding algorithms.

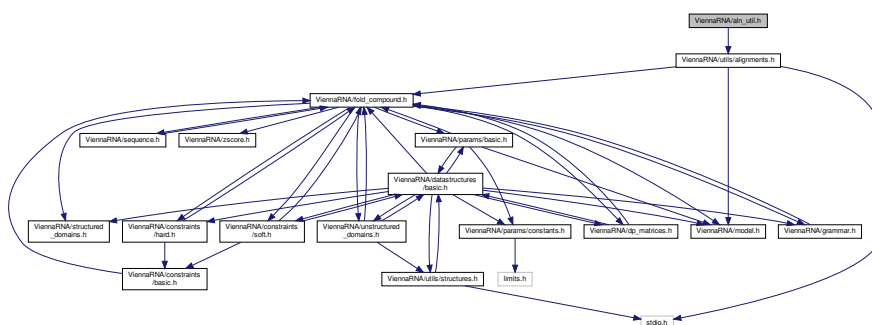
**Deprecated** See [vrna\\_md\\_t.nc\\_fact](#), and [vrna\\_mfe\(\)](#) to avoid using global variables

Default is 1.

## 18.4 ViennaRNA/aln\_util.h File Reference

Use [ViennaRNA/utis/alignments.h](#) instead.

Include dependency graph for aln\_util.h:







## Typedefs

- typedef void() [vrna\\_boltzmann\\_sampling\\_callback](#)(const char \*stucture, void \*data)  
*Callback for Boltzmann sampling.*
- typedef struct vrna\_pbacktrack\_memory\_s \* [vrna\\_pbacktrack\\_mem\\_t](#)  
*Boltzmann sampling memory data structure.*

## Functions

- char \* [vrna\\_pbacktrack5](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, unsigned int length)  
*Sample a secondary structure of a subsequence from the Boltzmann ensemble according its probability.*
- char \*\* [vrna\\_pbacktrack5\\_num](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, unsigned int num\_samples, unsigned int length, unsigned int options)  
*Obtain a set of secondary structure samples for a subsequence from the Boltzmann ensemble according their probability.*
- unsigned int [vrna\\_pbacktrack5\\_cb](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, unsigned int num\_samples, unsigned int length, [vrna\\_boltzmann\\_sampling\\_callback](#) \*cb, void \*data, unsigned int options)  
*Obtain a set of secondary structure samples for a subsequence from the Boltzmann ensemble according their probability.*
- char \*\* [vrna\\_pbacktrack5\\_resume](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, unsigned int num\_samples, unsigned int length, [vrna\\_pbacktrack\\_mem\\_t](#) \*nr\_mem, unsigned int options)  
*Obtain a set of secondary structure samples for a subsequence from the Boltzmann ensemble according their probability.*
- unsigned int [vrna\\_pbacktrack5\\_resume\\_cb](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, unsigned int num\_samples, unsigned int length, [vrna\\_boltzmann\\_sampling\\_callback](#) \*cb, void \*data, [vrna\\_pbacktrack\\_mem\\_t](#) \*nr\_mem, unsigned int options)  
*Obtain a set of secondary structure samples for a subsequence from the Boltzmann ensemble according their probability.*
- char \* [vrna\\_pbacktrack](#) ([vrna\\_fold\\_compound\\_t](#) \*fc)  
*Sample a secondary structure from the Boltzmann ensemble according its probability.*
- char \*\* [vrna\\_pbacktrack\\_num](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, unsigned int num\_samples, unsigned int options)  
*Obtain a set of secondary structure samples from the Boltzmann ensemble according their probability.*
- unsigned int [vrna\\_pbacktrack\\_cb](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, unsigned int num\_samples, [vrna\\_boltzmann\\_sampling\\_callback](#) \*cb, void \*data, unsigned int options)  
*Obtain a set of secondary structure samples from the Boltzmann ensemble according their probability.*
- char \*\* [vrna\\_pbacktrack\\_resume](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, unsigned int num\_samples, [vrna\\_pbacktrack\\_mem\\_t](#) \*nr\_mem, unsigned int options)  
*Obtain a set of secondary structure samples from the Boltzmann ensemble according their probability.*
- unsigned int [vrna\\_pbacktrack\\_resume\\_cb](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, unsigned int num\_samples, [vrna\\_boltzmann\\_sampling\\_callback](#) \*cb, void \*data, [vrna\\_pbacktrack\\_mem\\_t](#) \*nr\_mem, unsigned int options)  
*Obtain a set of secondary structure samples from the Boltzmann ensemble according their probability.*
- void [vrna\\_pbacktrack\\_mem\\_free](#) ([vrna\\_pbacktrack\\_mem\\_t](#) s)  
*Release memory occupied by a Boltzmann sampling memory data structure.*

### 18.6.1 Detailed Description

Boltzmann Sampling of secondary structures from the ensemble.

A.k.a. Stochastic backtracking



## 18.7.2 Function Documentation

### 18.7.2.1 `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()`

### 18.7.2.2 `get_centroid_struct_pr()`

```
char* get_centroid_struct_pr (
    int length,
    double * dist,
    FLT_OR_DBL * pr )
```

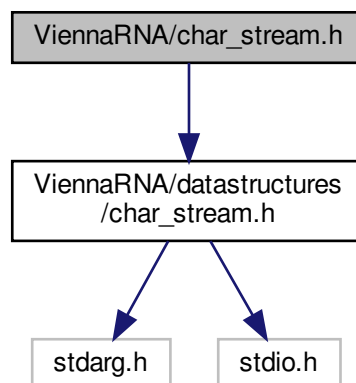
Get the centroid structure of the ensemble.

**Deprecated** This function was renamed to `vrna_centroid_from_probs()`

## 18.8 ViennaRNA/char\_stream.h File Reference

Use [ViennaRNA/datastructures/char\\_stream.h](#) instead.

Include dependency graph for `char_stream.h`:



### 18.8.1 Detailed Description

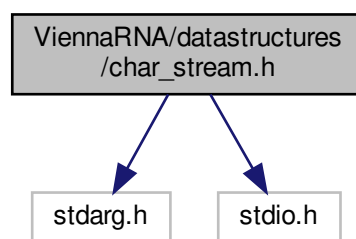
Use [ViennaRNA/datastructures/char\\_stream.h](#) instead.

**Deprecated** Use [ViennaRNA/datastructures/char\\_stream.h](#) instead

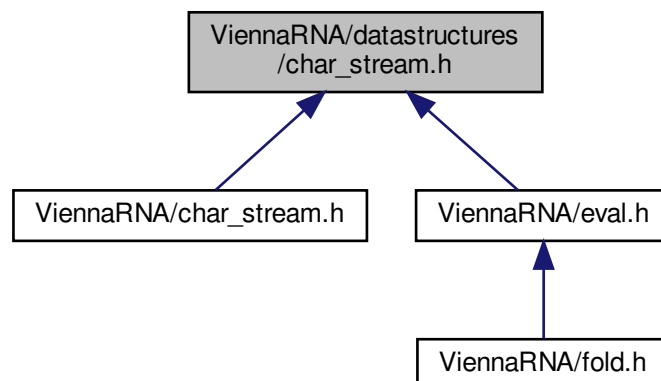
## 18.9 ViennaRNA/datastructures/char\_stream.h File Reference

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:





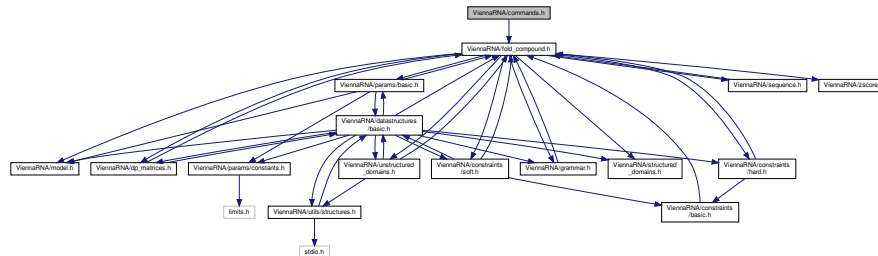




## 18.12 ViennaRNA/commands.h File Reference

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.

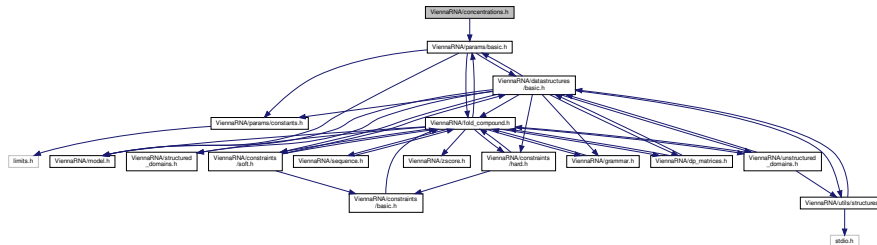
#### 18.12.1 Detailed Description

Parse and apply different commands that alter the behavior of secondary structure prediction and evaluation.

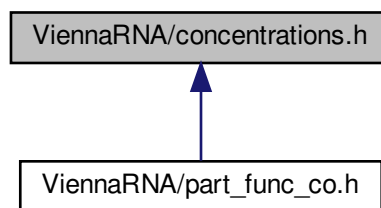
### 18.13 ViennaRNA/concentrations.h File Reference

### 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.*

### 18.13.1 Detailed Description

Concentration computations for RNA-RNA interactions.

### 18.13.2 Function Documentation

#### 18.13.2.1 `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]

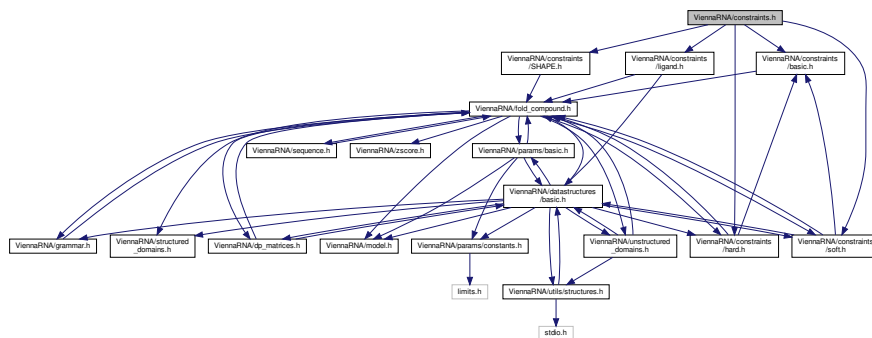
#### Returns

`vrna_dimer_conc_t` array containing the equilibrium energies and start concentrations

## 18.14 ViennaRNA/constraints.h File Reference

Use [ViennaRNA/constraints/basic.h](#) instead.

Include dependency graph for constraints.h:



### 18.14.1 Detailed Description

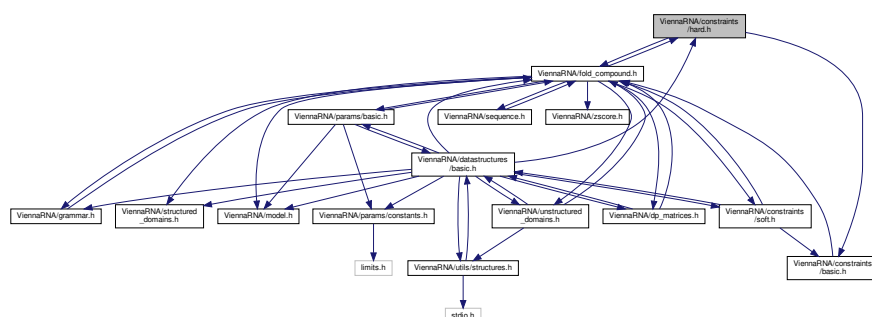
Use [ViennaRNA/constraints/basic.h](#) instead.

**Deprecated** Use [ViennaRNA/constraints/basic.h](#) instead

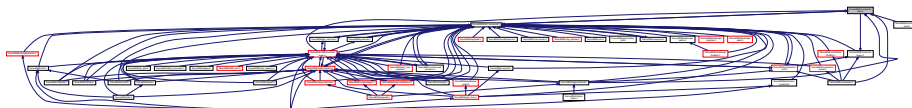
## 18.15 ViennaRNA/constraints/hard.h File Reference

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\\_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 quad)
- `#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.*
- int `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.*

### 18.15.1 Detailed Description

Functions and data structures for handling of secondary structure hard constraints.

### 18.15.2 Macro Definition Documentation

#### 18.15.2.1 VRNA\_CONSTRAINT\_NO\_HEADER

```
#define VRNA_CONSTRAINT_NO_HEADER 0
```

do not print the header information line

**Deprecated** This mode is not supported anymore!

#### 18.15.2.2 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\(\)](#)

### 18.15.3 Enumeration Type Documentation

#### 18.15.3.1 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>

## 18.15.4 Function Documentation

## 18.15.4.1 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 <code>NULL</code> )

## 18.15.4.2 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
---------------	--

### 18.15.4.3 `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!

### 18.15.4.4 `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)

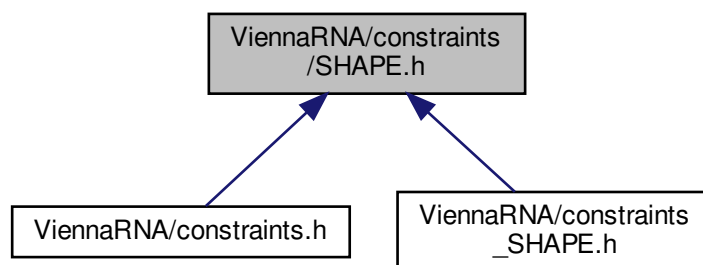
## 18.16 ViennaRNA/constraints/ligand.h File Reference

Functions for incorporation of ligands binding to hairpin and interior loop motifs using the soft constraints framework.





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\\_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\\_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.*

### 18.17.1 Detailed Description

This module provides function to incorporate SHAPE reactivity data into the folding recursions by means of soft constraints.

### 18.17.2 Function Documentation

#### 18.17.2.1 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.



## 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)*

### 18.18.1 Detailed Description

Functions and data structures for secondary structure soft constraints.

### 18.18.2 Enumeration Type Documentation

#### 18.18.2.1 [vrna\\_sc\\_type\\_e](#)

enum [vrna\\_sc\\_type\\_e](#)

The type of a soft constraint.

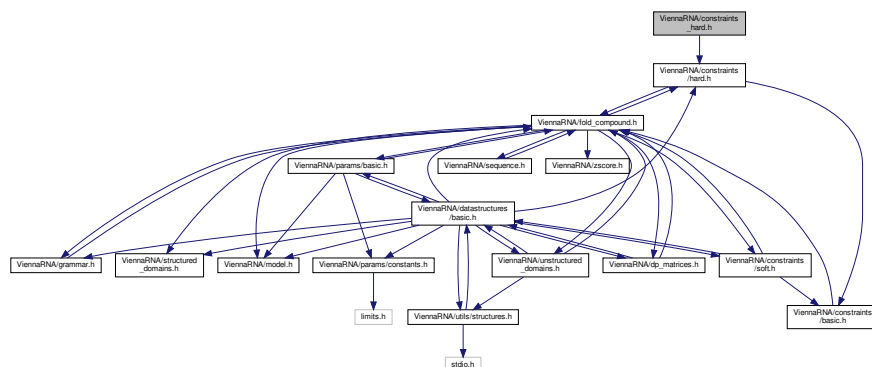
## Enumerator

VRNA_SC_DEFAULT	Default Soft Constraints.
VRNA_SC_WINDOW	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>

## 18.19 ViennaRNA/constraints\_hard.h File Reference

Use [ViennaRNA/constraints/hard.h](#) instead.

Include dependency graph for constraints\_hard.h:



### 18.19.1 Detailed Description

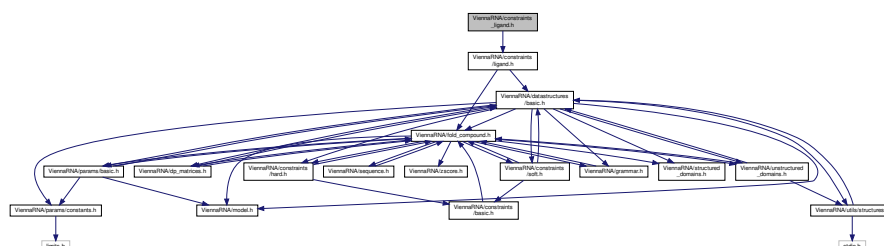
Use [ViennaRNA/constraints/hard.h](#) instead.

**Deprecated** Use [ViennaRNA/constraints/hard.h](#) instead

## 18.20 ViennaRNA/constraints\_ligand.h File Reference

Use [ViennaRNA/constraints/ligand.h](#) instead.

Include dependency graph for constraints\_ligand.h:







### 18.22.1 Detailed Description

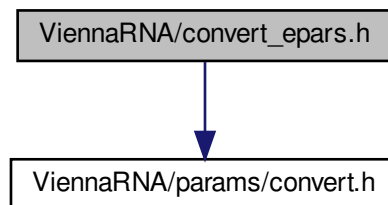
Use [ViennaRNA/constraints/soft.h](#) instead.

**Deprecated** Use [ViennaRNA/constraints/soft.h](#) instead

## 18.23 ViennaRNA/convert\_epars.h File Reference

Use [ViennaRNA/params/convert.h](#) instead.

Include dependency graph for convert\_epars.h:



### 18.23.1 Detailed Description

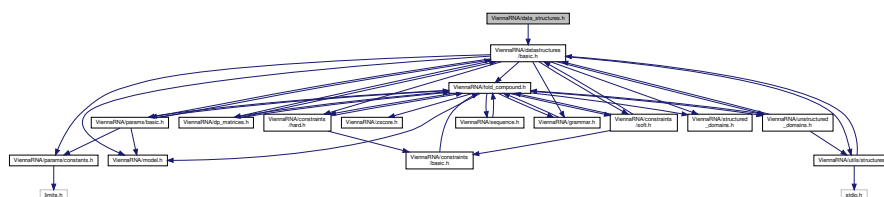
Use [ViennaRNA/params/convert.h](#) instead.

**Deprecated** Use [ViennaRNA/params/convert.h](#) instead

## 18.24 ViennaRNA/data\_structures.h File Reference

Use [ViennaRNA/datastructures/basic.h](#) instead.

Include dependency graph for data\_structures.h:



### 18.24.1 Detailed Description

Use [ViennaRNA/datastructures/basic.h](#) instead.

**Deprecated** Use [ViennaRNA/datastructures/basic.h](#) instead

## 18.25 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.

### 18.25.1 Detailed Description

Implementations of hash table functions.

## 18.26 ViennaRNA/datastructures/heap.h File Reference

Implementation of an abstract heap data structure.

### Typedefs

- typedef struct vrna\_heap\_s \* [vrna\\_heap\\_t](#)  
*An abstract heap data structure.*
- typedef int() [vrna\\_callback\\_heap\\_cmp](#)(const void \*a, const void \*b, void \*data)  
*Heap compare function prototype.*
- typedef size\_t() [vrna\\_callback\\_heap\\_get\\_pos](#)(const void \*a, void \*data)  
*Retrieve the position of a particular heap entry within the heap.*
- typedef void() [vrna\\_callback\\_heap\\_set\\_pos](#)(const void \*a, size\_t pos, void \*data)  
*Store the position of a particular heap entry within the heap.*

### Functions

- [vrna\\_heap\\_t](#) [vrna\\_heap\\_init](#) (size\_t n, [vrna\\_callback\\_heap\\_cmp](#) \*cmp, [vrna\\_callback\\_heap\\_get\\_pos](#) \*get↵  
\_entry\_pos, [vrna\\_callback\\_heap\\_set\\_pos](#) \*set\_entry\_pos, void \*data)  
*Initialize a heap data structure.*
- void [vrna\\_heap\\_free](#) ([vrna\\_heap\\_t](#) h)  
*Free memory occupied by a heap data structure.*
- size\_t [vrna\\_heap\\_size](#) (struct vrna\_heap\_s \*h)  
*Get the size of a heap data structure, i.e. the number of stored elements.*
- void [vrna\\_heap\\_insert](#) ([vrna\\_heap\\_t](#) h, void \*v)  
*Insert an element into the heap.*
- void \* [vrna\\_heap\\_pop](#) ([vrna\\_heap\\_t](#) h)  
*Pop (remove and return) the object at the root of the heap.*
- const void \* [vrna\\_heap\\_top](#) ([vrna\\_heap\\_t](#) h)  
*Get the object at the root of the heap.*
- void \* [vrna\\_heap\\_remove](#) ([vrna\\_heap\\_t](#) h, const void \*v)  
*Remove an arbitrary element within the heap.*
- void \* [vrna\\_heap\\_update](#) ([vrna\\_heap\\_t](#) h, void \*v)  
*Update an arbitrary element within the heap.*

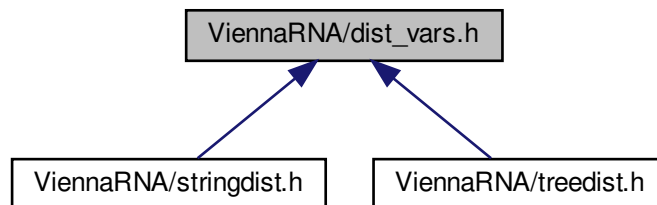
### 18.26.1 Detailed Description

Implementation of an abstract heap data structure.

## 18.27 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.*

### 18.27.1 Detailed Description

Global variables for Distance-Package.

### 18.27.2 Variable Documentation



## 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.

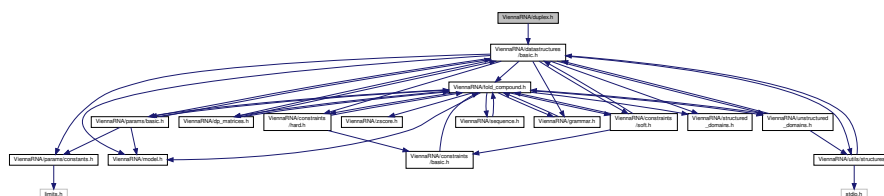
### 18.28.1 Detailed Description

Functions to deal with standard dynamic programming (DP) matrices.

## 18.29 ViennaRNA/duplex.h File Reference

Functions for simple RNA-RNA duplex interactions.

Include dependency graph for duplex.h:



### 18.29.1 Detailed Description

Functions for simple RNA-RNA duplex interactions.

## 18.30 ViennaRNA/edit\_cost.h File Reference

global variables for Edit Costs included by treedist.c and stringdist.c

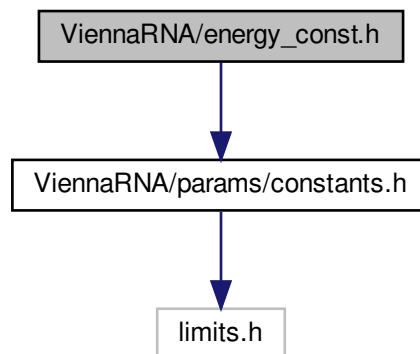
### 18.30.1 Detailed Description

global variables for Edit Costs included by treedist.c and stringdist.c

## 18.31 ViennaRNA/energy\_const.h File Reference

Use [ViennaRNA/params/constants.h](#) instead.

Include dependency graph for energy\_const.h:



### 18.31.1 Detailed Description

Use [ViennaRNA/params/constants.h](#) instead.

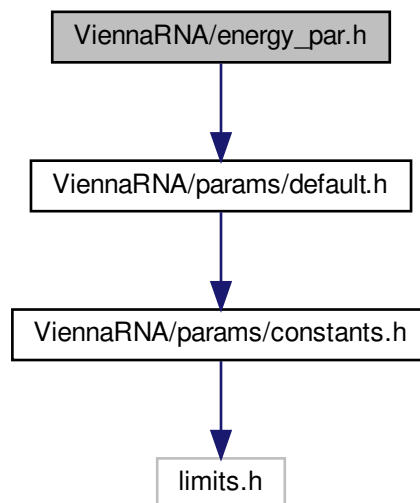
**Deprecated** Use [ViennaRNA/params/constants.h](#) instead



## 18.32 ViennaRNA/energy\_par.h File Reference

Use [ViennaRNA/params/default.h](#) instead.

Include dependency graph for energy\_par.h:



### 18.32.1 Detailed Description

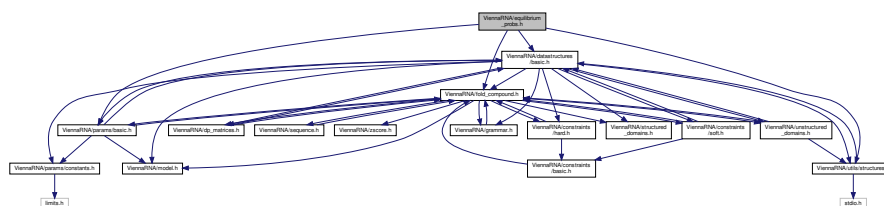
Use [ViennaRNA/params/default.h](#) instead.

**Deprecated** Use [ViennaRNA/params/default.h](#) instead

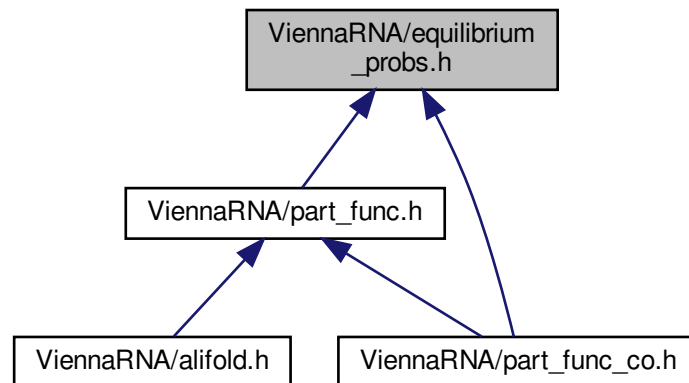
## 18.33 ViennaRNA/equilibrium\_probs.h File Reference

Equilibrium Probability implementations.

Include dependency graph for equilibrium\_probs.h:



This graph shows which files directly or indirectly include this file:



## Functions

### Base pair probabilities and derived computations

- int **vrna\_pairing\_probs** (**vrna\_fold\_compound\_t** \*vc, char \*structure)
- 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.*
- double \* **vrna\_positional\_entropy** (**vrna\_fold\_compound\_t** \*fc)  
*Compute a vector of positional entropies.*
- **vrna\_ep\_t** \* **vrna\_stack\_prob** (**vrna\_fold\_compound\_t** \*vc, double cutoff)  
*Compute stacking probabilities.*

### Multimer probabilities computations

- 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.*

### Structure probability computations

- double **vrna\_pr\_structure** (**vrna\_fold\_compound\_t** \*fc, const char \*structure)  
*Compute the equilibrium probability of a particular secondary structure.*
- double **vrna\_pr\_energy** (**vrna\_fold\_compound\_t** \*vc, double e)

## 18.33.1 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.



- 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.*





## 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)
- 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.*

### 18.37.1 Detailed Description

Read and write different file formats for RNA sequences, structures.

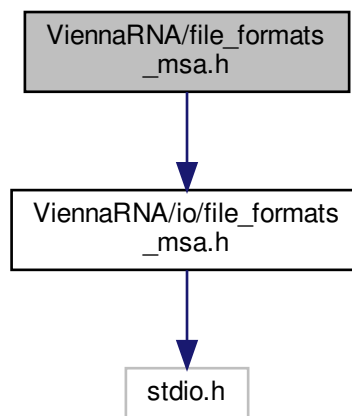
,

## 18.38 ViennaRNA/file\_formats\_msa.h File Reference

Use [ViennaRNA/io/file\\_formats\\_msa.h](#) instead.



Include dependency graph for file\_formats\_msa.h:



### 18.38.1 Detailed Description

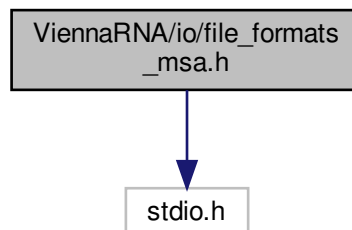
Use [ViennaRNA/io/file\\_formats\\_msa.h](#) instead.

**Deprecated** Use [ViennaRNA/io/file\\_formats\\_msa.h](#) instead

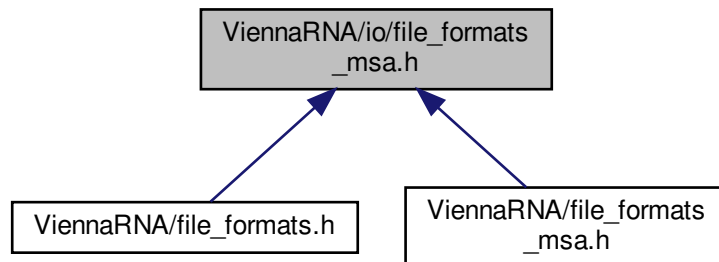
## 18.39 ViennaRNA/io/file\_formats\_msa.h File Reference

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.*

### 18.39.1 Detailed Description

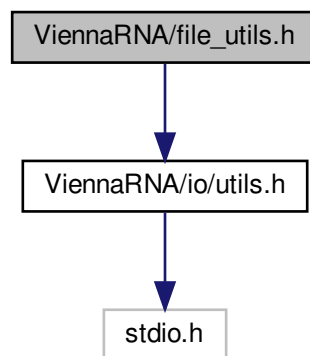
Functions dealing with file formats for Multiple Sequence Alignments (MSA)

, ,

## 18.40 ViennaRNA/file\_utils.h File Reference

Use [ViennaRNA/io/utils.h](#) instead.

Include dependency graph for file\_utils.h:



### 18.40.1 Detailed Description

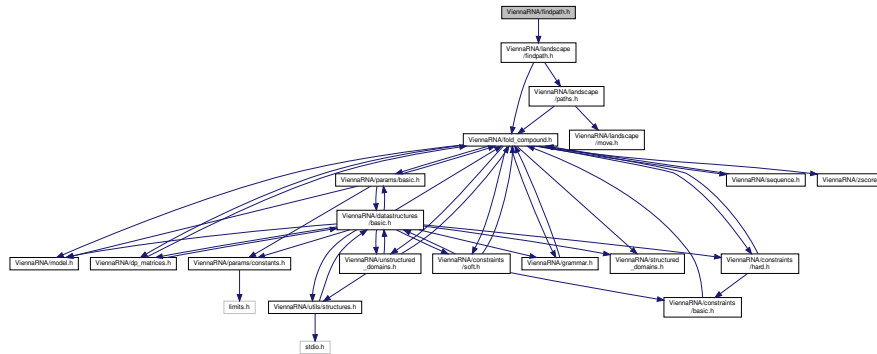
Use [ViennaRNA/io/utils.h](#) instead.

**Deprecated** Use [ViennaRNA/io/utils.h](#) instead

## 18.41 ViennaRNA/findpath.h File Reference

Use [ViennaRNA/landscape/findpath.h](#) instead.

Include dependency graph for findpath.h:



### 18.41.1 Detailed Description

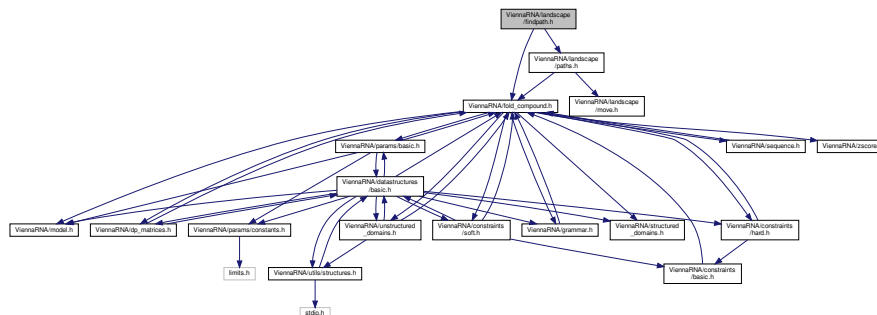
Use [ViennaRNA/landscape/findpath.h](#) instead.

**Deprecated** Use [ViennaRNA/landscape/findpath.h](#) instead

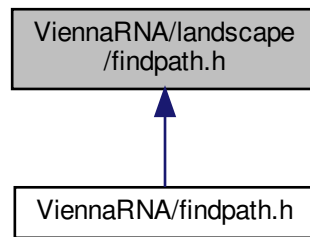
## 18.42 ViennaRNA/landscape/findpath.h File Reference

A breadth-first search heuristic for optimal direct folding paths.

Include dependency graph for findpath.h:



This graph shows which files directly or indirectly include this file:



## Functions

- int [vrna\\_path\\_findpath\\_saddle](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, 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](#) \*fc, 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](#) \*fc, 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](#) \*fc, 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)*

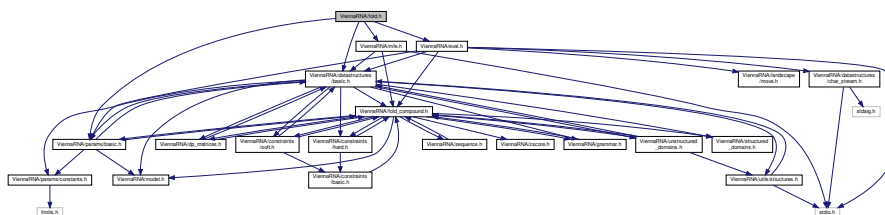
### 18.42.1 Detailed Description

A breadth-first search heuristic for optimal direct folding paths.

### 18.43 ViennaRNA/fold.h File Reference

MFE calculations for single RNA sequences.

Include dependency graph for fold.h:





## 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](#).*





## 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.*

### 18.45.1 Detailed Description

Here all all declarations of the global variables used throughout RNAlib.

### 18.45.2 Variable Documentation

#### 18.45.2.1 RibosumFile

```
char* RibosumFile [extern]
```

warning this variable will vanish in the future ribosums will be compiled in instead

#### 18.45.2.2 james\_rule

```
int james_rule [extern]
```

interior loops of size 2 get energy 0.8Kcal and no mismatches, default 1

#### 18.45.2.3 logML

```
int logML [extern]
```

use logarithmic multiloop energy function

#### 18.45.2.4 cut\_point

```
int cut_point [extern]
```

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.

#### 18.45.2.5 base\_pair

```
bondT* base_pair [extern]
```

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!

#### 18.45.2.6 pr

```
FLT_OR_DBL* pr [extern]
```

A pointer to the base pair probability matrix.

**Deprecated** Do not use this variable anymore!

#### 18.45.2.7 iindx

```
int* iindx [extern]
```

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!





## Data Structures

- struct `vrna_heat_capacity_s`  
*A single result from heat capacity computations. [More...](#)*

## Typedefs

- typedef void([vrna\\_heat\\_capacity\\_callback](#)(float temp, float heat\_capacity, void \*data)  
*The callback for heat capacity predictions.*
- typedef struct [vrna\\_heat\\_capacity\\_s](#) [vrna\\_heat\\_capacity\\_t](#)  
*A single result from heat capacity computations.*

## Functions

### Basic heat capacity function interface

- `vrna_heat_capacity_t * vrna_heat_capacity (vrna_fold_compound_t *fc, float T_min, float T_max, float T_increment, unsigned int mpoints)`  
*Compute the specific heat for an RNA.*
- `int vrna_heat_capacity_cb (vrna_fold_compound_t *fc, float T_min, float T_max, float T_increment, unsigned int mpoints, vrna_heat_capacity_callback *cb, void *data)`  
*Compute the specific heat for an RNA (callback variant)*

### Simplified heat capacity computation

- `vrna_heat_capacity_t * vrna_heat_capacity_simple` (const char \*sequence, float T\_min, float T\_max, float T\_increment, unsigned int mpoints)  
*Compute the specific heat for an RNA (simplified variant)*

### 18.49.1 Detailed Description

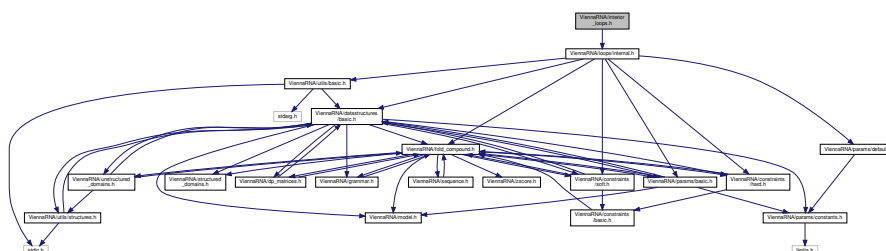
Compute heat capacity for an RNA.

This file includes the interface to all functions related to predicting the heat capacity for an RNA.

## 18.50 ViennaRNA/interior\_loops.h File Reference

Use [ViennaRNA/loops/internal.h](#) instead.

Include dependency graph for interior loops.h:



### 18.50.1 Detailed Description

Use [ViennaRNA/loops/internal.h](#) instead.

**Deprecated** Use [ViennaRNA/loops/internal.h](#) instead

## 18.51 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](#)

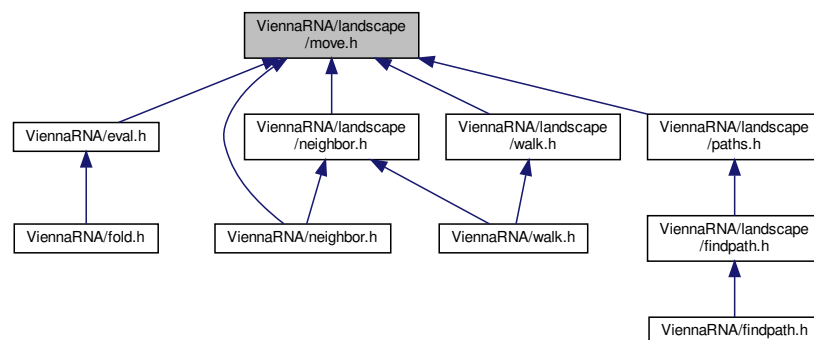
### 18.51.1 Detailed Description

Inverse folding routines.

## 18.52 ViennaRNA/landscape/move.h File Reference

Methods to operate with structural neighbors of RNA secondary structures.

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.*

## Typedefs

- typedef struct [vrna\\_move\\_s](#) [vrna\\_move\\_t](#)  
*A single move that transforms a secondary structure into one of its neighbors.*

## Functions

- [vrna\\_move\\_t](#) [vrna\\_move\\_init](#) (int pos\_5, int pos\_3)  
*Create an atomic move.*
- 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.*
- int [vrna\\_move\\_is\\_removal](#) (const [vrna\\_move\\_t](#) \*m)  
*Test whether a move is a base pair removal.*
- int [vrna\\_move\\_is\\_insertion](#) (const [vrna\\_move\\_t](#) \*m)  
*Test whether a move is a base pair insertion.*
- int [vrna\\_move\\_is\\_shift](#) (const [vrna\\_move\\_t](#) \*m)  
*Test whether a move is a base pair shift.*
- int [vrna\\_move\\_compare](#) (const [vrna\\_move\\_t](#) \*a, const [vrna\\_move\\_t](#) \*b, const short \*pt)  
*Compare two moves.*

### 18.52.1 Detailed Description

Methods to operate with structural neighbors of RNA secondary structures.





## Typedefs

- typedef struct [vrna\\_path\\_s](#) [vrna\\_path\\_t](#)  
*Typename for the refolding path data structure [vrna\\_path\\_s](#).*
- typedef struct [vrna\\_path\\_options\\_s](#) \* [vrna\\_path\\_options\\_t](#)  
*Options data structure for (re-)folding path implementations.*
- typedef struct [vrna\\_path\\_s](#) [path\\_t](#)  
*Old typename of [vrna\\_path\\_s](#).*

## Functions

- void [vrna\\_path\\_free](#) ([vrna\\_path\\_t](#) \*path)  
*Release (free) memory occupied by a (re-)folding path.*
- void [vrna\\_path\\_options\\_free](#) ([vrna\\_path\\_options\\_t](#) options)  
*Release (free) memory occupied by an options data structure for (re-)folding path implementations.*
- [vrna\\_path\\_options\\_t](#) [vrna\\_path\\_options\\_findpath](#) (int width, unsigned int type)  
*Create options data structure for findpath direct (re-)folding path heuristic.*
- [vrna\\_path\\_t](#) \* [vrna\\_path\\_direct](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, const char \*s1, const char \*s2, [vrna\\_path\\_options\\_t](#) options)  
*Determine an optimal direct (re-)folding path between two secondary structures.*
- [vrna\\_path\\_t](#) \* [vrna\\_path\\_direct\\_ub](#) ([vrna\\_fold\\_compound\\_t](#) \*fc, const char \*s1, const char \*s2, int maxE, [vrna\\_path\\_options\\_t](#) options)  
*Determine an optimal direct (re-)folding path between two secondary structures.*

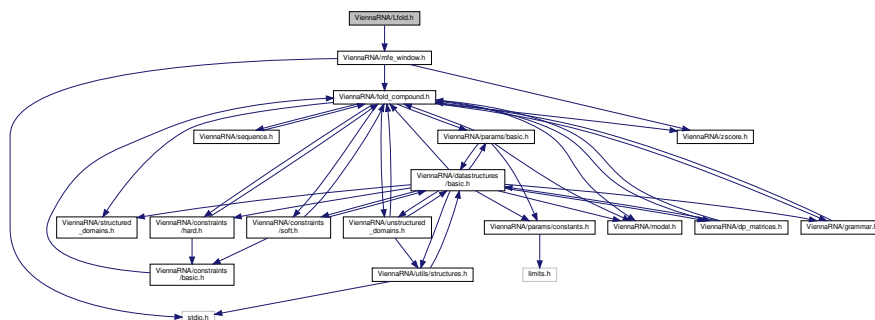
### 18.53.1 Detailed Description

API for computing (optional) (re-)folding paths between secondary structures.

## 18.54 ViennaRNA/Lfold.h File Reference

Functions for locally optimal MFE structure prediction.

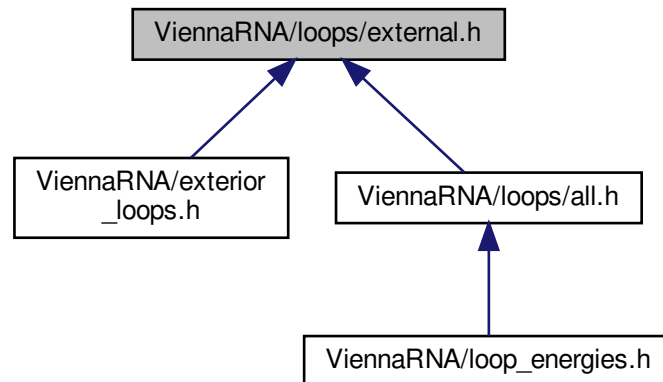
Include dependency graph for Lfold.h:







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)

### 18.57.1 Detailed Description

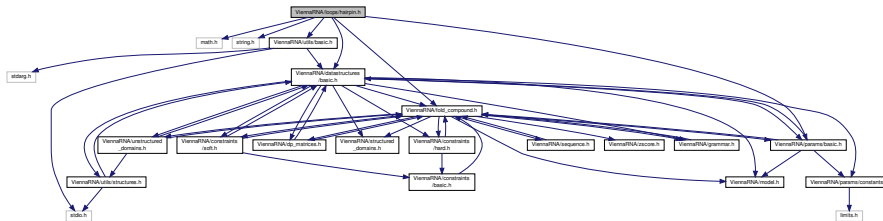
### Energy evaluation of exterior loops for MFE and partition function calculations.

, ,

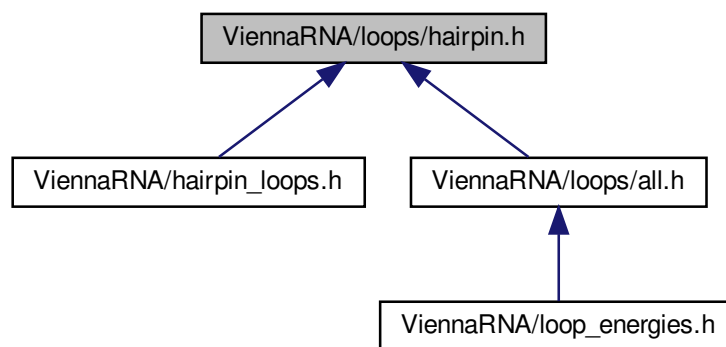
## 18.58 ViennaRNA/loops/hairpin.h File Reference

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

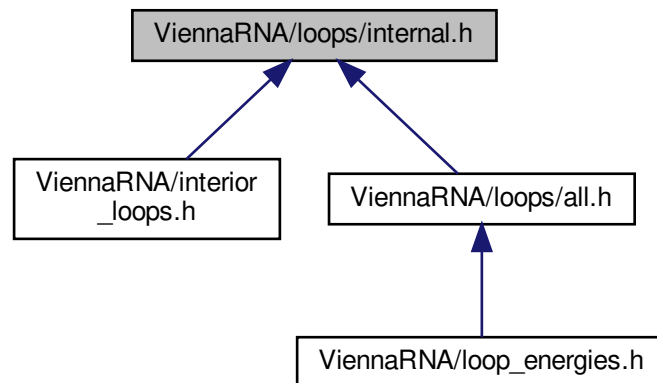
- int `vrna_BT_hp_loop` (`vrna_fold_compound_t` \*fc, int i, int j, int en, `vrna_bp_stack_t` \*bp\_stack, int \*stack\_← count)
- Backtrack a hairpin loop closed by  $(i, j)$ .*

### Basic free energy interface

- `int vrna_E_hp_loop (vrna_fold_compound_t *fc, int i, int j)`



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)

## 18.59.1 Detailed Description

Energy evaluation of interior loops for MFE and partition function calculations.

‘ ‘





## 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)

### 18.60.1 Detailed Description

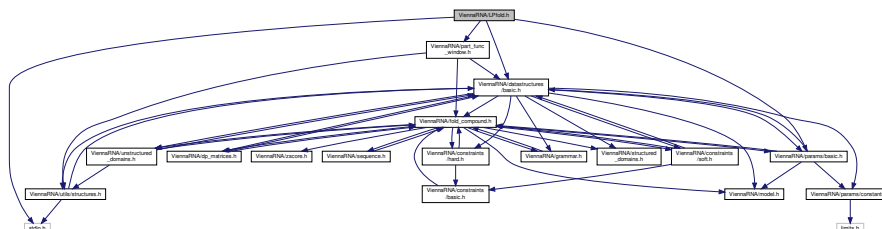
Energy evaluation of multibranch loops for MFE and partition function calculations.

2 2

### 18.61 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)









### 18.65.1 Detailed Description

Several Maximum Matching implementations.

This file contains the declarations for several maximum matching implementations

### 18.65.2 Function Documentation

#### 18.65.2.1 `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`).

#### 18.65.2.2 `vrna_maximum_matching_simple()`

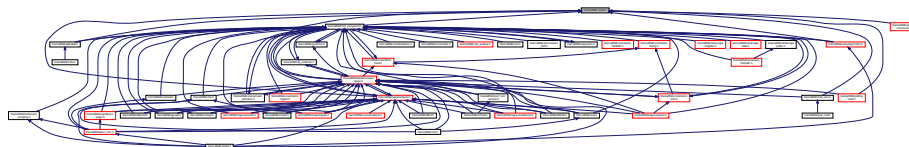
```
int vrna_maximum_matching_simple (
    const char * sequence )
```

**SWIG Wrapper Notes** This function is available as global function **maximum\_matching()**.

## 18.66 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 cmdline 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)  
*Set the default scaling factor used to avoid under-/overflows in partition function computation.*
- double [vrna\\_md\\_defaults\\_nc\\_fact\\_get](#) (void)  
*Get the default scaling factor used to avoid under-/overflows in partition function computation.*
- 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](#)*

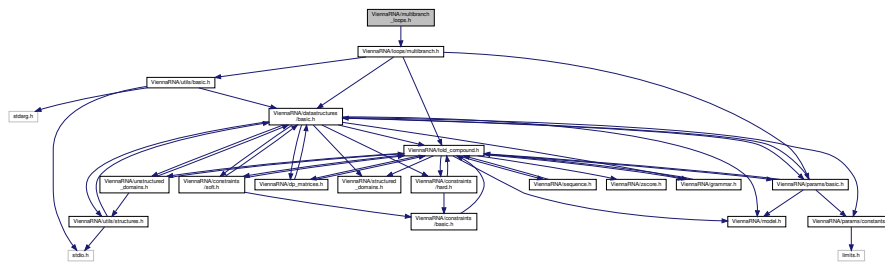
### 18.66.1 Detailed Description

The model details data structure and its corresponding modifiers.

## 18.67 ViennaRNA/multibranch\_loops.h File Reference

Use [ViennaRNA/loops/multibranch.h](#) instead.

Include dependency graph for multibranch\_loops.h:



### 18.67.1 Detailed Description

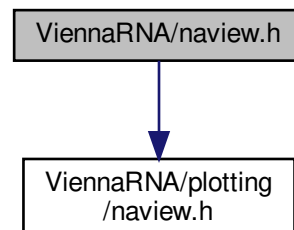
Use [ViennaRNA/loops/multibranch.h](#) instead.

**Deprecated** Use [ViennaRNA/loops/multibranch.h](#) instead

## 18.68 ViennaRNA/naview.h File Reference

Use [ViennaRNA/plotting/naview.h](http://ViennaRNA/plotting/naview.h) instead.

Include dependency graph for naview.h:



### 18.68.1 Detailed Description

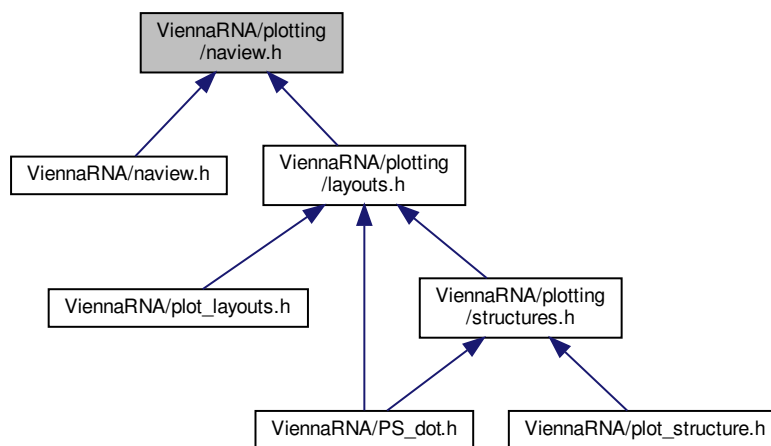
Use [ViennaRNA/plotting/naview.h](#) instead.

**Deprecated** Use [ViennaRNA/plotting/naview.h](#) instead

## 18.69 ViennaRNA/plotting/naview.h File Reference

Implementation of the Naview RNA secondary structure layout algorithm [5].

This graph shows which files directly or indirectly include this file:



### Functions

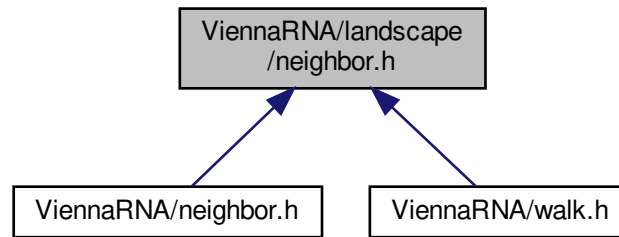
- `int vrna_plot_coords_naview` (const char \*structure, float \*\*x, float \*\*y)  
Compute nucleotide coordinates for secondary structure plot using the Naview algorithm [5].
- `int vrna_plot_coords_naview_pt` (const short \*pt, float \*\*x, float \*\*y)  
Compute nucleotide coordinates for secondary structure plot using the Naview algorithm [5].
- `int naview_xy_coordinates` (short \*pair\_table, float \*X, float \*Y)

### 18.69.1 Detailed Description

Implementation of the Naview RNA secondary structure layout algorithm [5].



This graph shows which files directly or indirectly include this file:



## Macros

- `#define VRNA_NEIGHBOR_CHANGE 1`  
*State indicator for a neighbor that has been changed.*
- `#define VRNA_NEIGHBOR_INVALID 2`  
*State indicator for a neighbor that has been invalidated.*
- `#define VRNA_NEIGHBOR_NEW 3`  
*State indicator for a neighbor that has become newly available.*

## Typedefs

- `typedef void() vrna_callback_move_update(vrna_fold_compound_t *fc, vrna_move_t neighbor, unsigned int state, void *data)`  
*Prototype of the neighborhood update callback.*

## Functions

- `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)*
- `int vrna_move_neighbor_diff_cb (vrna_fold_compound_t *fc, short *ptable, vrna_move_t move, vrna_callback_move_update *cb, void *data, unsigned int options)`  
*Apply a move to a secondary structure and indicate which neighbors have changed consequentially.*
- `vrna_move_t * vrna_move_neighbor_diff (vrna_fold_compound_t *fc, short *ptable, vrna_move_t move, vrna_move_t **invalid_moves, unsigned int options)`  
*Apply a move to a secondary structure and indicate which neighbors have changed consequentially.*

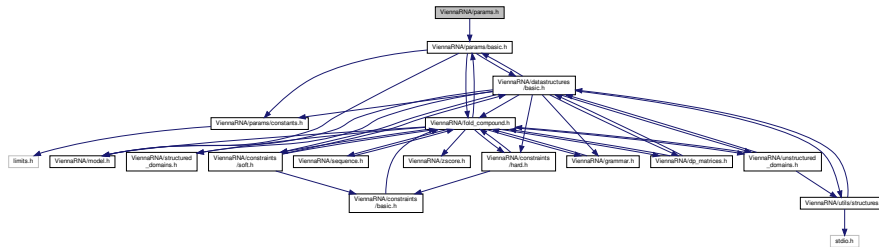
### 18.71.1 Detailed Description

## Methods to compute the neighbors of an RNA secondary structure.

## 18.72 ViennaRNA/params.h File Reference

Use [ViennaRNA/params/basic.h](#) instead.

Include dependency graph for params.h:



### 18.72.1 Detailed Description

Use [ViennaRNA/params/basic.h](#) instead.

**Deprecated** Use [ViennaRNA/params/basic.h](#) instead

### 18.73 ViennaRNA/params/1.8.4\_epars.h File Reference

Free energy parameters for parameter file conversion.

### 18.73.1 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 Lyttle, 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-Neighbor 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





## 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.*

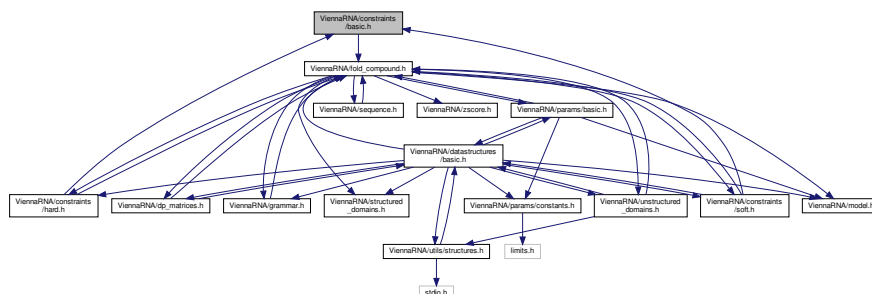
### 18.75.1 Detailed Description

Functions to deal with sets of energy parameters.

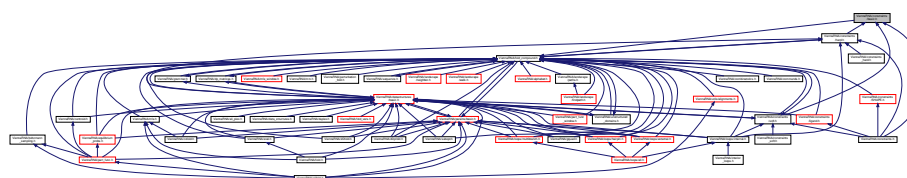
## 18.76 ViennaRNA/constraints/basic.h File Reference

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`

- #define VRNA\_DECOMP\_ML\_COAXIAL (unsigned char)10

- #define VRNA\_DECOMP\_ML\_COAXIAL\_ENC (unsigned char)11

- `#define VRNA_DECOMP_EXT_EXT (unsigned char)12`

- `#define VRNA_DECOMP_EXT_UP` (unsigned char)13

- #define VRNA\_DECOMP\_EXT\_STEM (unsigned char)14

- #define VRNA\_DECOMP\_EXT\_EXT\_EXT (unsigned char)15

- #define VRNA\_DECOMP\_EXT\_STEM\_EXT (unsigned char)16

- `#define VRNA_DECOMP_EXT_STEM_OUTSIDE (unsigned char)17`

- `#define VRNA_DECOMP_EXT_EXT_STEM (unsigned char)18`

- #define VRNA\_DECOMP\_EXT\_EXT\_STEM1 (unsigned char)19

*Indicator for decomposition of exterior loop part.*

- `void vrna_constraints_add (vrna_fold_compound_t *vc, const char *constraint, unsigned int options)`

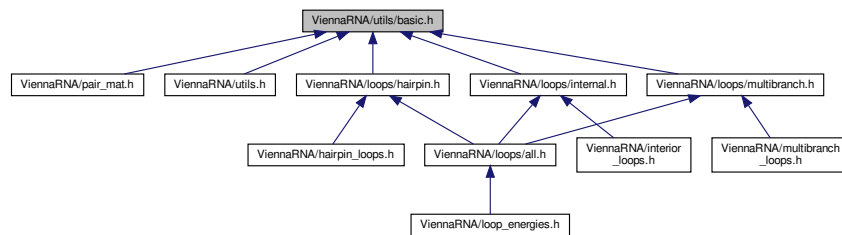
Add constraints to a `vrna_fold_compound_t` data structure.

## Functions and data structures for constraining secondary structure predictions and evaluation.

General utility- and helper-functions used throughout the *ViennaRNA Package*.

[illegible]

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.*

### 18.77.1 Detailed Description

General utility- and helper-functions used throughout the *ViennaRNA Package*.

### 18.77.2 Function Documentation

#### 18.77.2.1 `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

<code>fp</code>	A file pointer to the stream where the function should read from
-----------------	--

#### Returns

A pointer to the resulting string

### 18.77.2.2 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!

### 18.77.2.3 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!

### 18.77.2.4 warn\_user()

```
void warn_user (
    const char message[] )
```

Print a warning message.

Print a warning message to *stderr*

**Deprecated** Use [vrna\\_message\\_warning\(\)](#) instead!

### 18.77.2.5 nerror()

```
void nerror (
    const char message[] )
```

Die with an error message.

**Deprecated** Use [vrna\\_message\\_error\(\)](#) instead!

### 18.77.2.6 space()

```
void* space (
    unsigned size )
```

Allocate space safely.

**Deprecated** Use [vrna\\_alloc\(\)](#) instead!

### 18.77.2.7 xrealloc()

```
void* xrealloc (
    void * p,
    unsigned size )
```

Reallocate space safely.

**Deprecated** Use [vrna\\_realloc\(\)](#) instead!

### 18.77.2.8 init\_rand()

```
void init_rand (
    void )
```

Make random number seeds.

**Deprecated** Use [vrna\\_init\\_rand\(\)](#) instead!

### 18.77.2.9 urn()

```
double urn (
    void )
```

get a random number from [0..1]

**Deprecated** Use [vrna\\_urn\(\)](#) instead!



```
int int_urn (
                int from,
                int to )
```

**Deprecated** Use `vrna_int_urn()` instead!

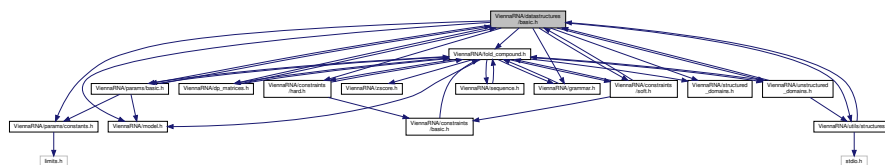
```
void filecopy (
    FILE * from,
    FILE * to )
```

**Deprecated** Use `vrna_file_copy()` instead!

```
char* time_stamp (
                void )
```

**Deprecated** Use `vrna_time_stamp()` instead!

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 RNAp [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 `RNAshoop` (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.*

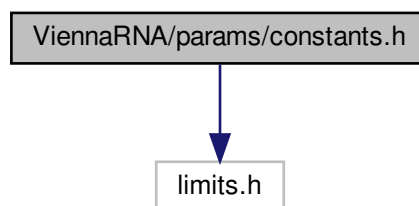
### 18.78.1 Detailed Description

Various data structures and pre-processor macros.

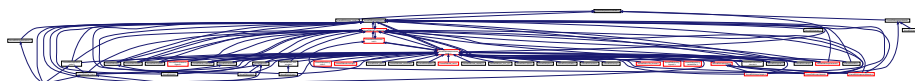
## 18.79 ViennaRNA/params/constants.h File Reference

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`

### 18.79.1 Detailed Description

Energy parameter constants.

### 18.79.2 Macro Definition Documentation

#### 18.79.2.1 GASCONST

```
#define GASCONST 1.98717 /* in [cal/K] */
```

The gas constant

#### 18.79.2.2 K0

```
#define K0 273.15
```

0 deg Celsius in Kelvin

#### 18.79.2.3 INF

```
#define INF 10000000 /* (INT_MAX/10) */
```

Infinity as used in minimization routines

#### 18.79.2.4 FORBIDDEN

```
#define FORBIDDEN 9999
```

forbidden

### 18.79.2.5 BONUS

```
#define BONUS 10000
```

bonus contribution

### 18.79.2.6 NBPAIRS

```
#define NBPAIRS 7
```

The number of distinguishable base pairs

### 18.79.2.7 TURN

```
#define TURN 3
```

The minimum loop length

### 18.79.2.8 MAXLOOP

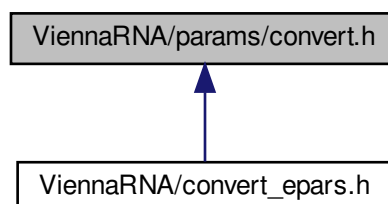
```
#define MAXLOOP 30
```

The maximum loop length

## 18.80 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)

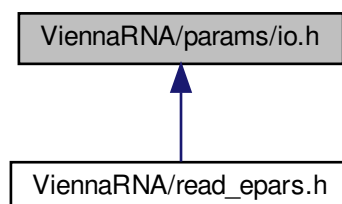
### 18.80.1 Detailed Description

Functions and definitions for energy parameter file format conversion.

## 18.81 ViennaRNA/params/io.h File Reference

Read and write energy parameter files.

This graph shows which files directly or indirectly include this file:



## Macros

- `#define VRNA_PARAMETER_FORMAT_DEFAULT 0`  
*Default Energy Parameter File format.*

## Functions

- `int vrna_params_load` (const char fname[], unsigned int options)  
*Load energy parameters from a file.*
- `int vrna_params_save` (const char fname[], unsigned int options)  
*Save energy parameters to a file.*
- `int vrna_params_load_from_string` (const char \*string, const char \*name, unsigned int options)  
*Load energy parameters from string.*
- `int vrna_params_load_defaults` (void)  
*Load default RNA energy parameter set.*
- `int vrna_params_load_RNA_Turner2004` (void)  
*Load Turner 2004 RNA energy parameter set.*
- `int vrna_params_load_RNA_Turner1999` (void)  
*Load Turner 1999 RNA energy parameter set.*
- `int vrna_params_load_RNA_Andronescu2007` (void)  
*Load Andronescu 2007 RNA energy parameter set.*
- `int vrna_params_load_RNA_Langdon2018` (void)  
*Load Langdon 2018 RNA energy parameter set.*
- `int vrna_params_load_RNA_misc_special_hairpins` (void)  
*Load Misc Special Hairpin RNA energy parameter set.*
- `int vrna_params_load_DNA_Mathews2004` (void)  
*Load Mathews 2004 DNA energy parameter set.*
- `int vrna_params_load_DNA_Mathews1999` (void)  
*Load Mathews 1999 DNA energy parameter set.*
- `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.*

### 18.81.1 Detailed Description

Read and write energy parameter files.





## 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\_↔  
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.*
- char \* [pbacktrack](#) (char \*sequence)  
*Sample a secondary structure from the Boltzmann ensemble according its probability.*
- char \* [pbacktrack5](#) (char \*sequence, int length)  
*Sample a sub-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\\_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\(\)](#)*
- 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.*

## 18.82.1 Detailed Description

Partition function implementations.

,

This file includes (almost) all function declarations within the **RNAlib** that are related to Partition function computations

## 18.82.2 Function Documentation

### 18.82.2.1 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\(\)](#)

### 18.82.2.2 `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\(\)](#)

### 18.82.2.3 `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!

### 18.82.2.4 `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

### 18.82.2.5 `expHairpinEnergy()`

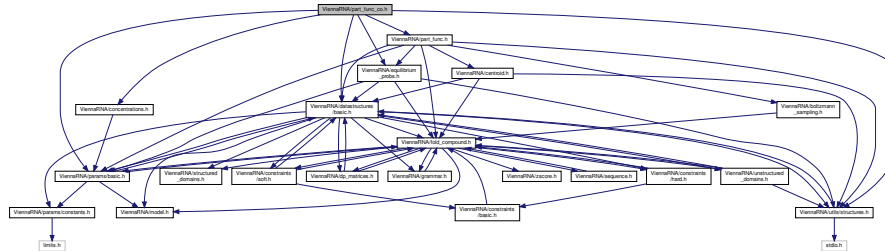
```
double expHairpinEnergy (
    int u,
    int type,
    short sil,
    short sj1,
    const char * string )
```

**Deprecated** Use [exp\\_E\\_Hairpin\(\)](#) from [loop\\_energies.h](#) instead

## 18.83 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.

#### 18.83.1 Detailed Description

Partition function for two RNA sequences.

### 18.83.2 Function Documentation

### 18.83.2.1 get\_plist()

```
vrna_ep_t* get_plist (
    vrna_ep_t * pl,
    int length,
    double cut_off )
```

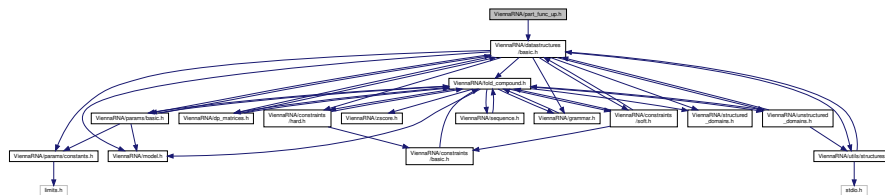
DO NOT USE THIS FUNCTION ANYMORE

**Deprecated** { This function is deprecated and will be removed soon!} use [assign\\_plist\\_from\\_pr\(\)](#) instead!

## 18.84 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()`.*

### 18.84.1 Detailed Description

Implementations for accessibility and RNA-RNA interaction as a stepwise process.



- 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

- ### Simplified global partition function computation using sequence(s) or multiple sequence alignment(s)

- ### 18.85.1 Detailed Description

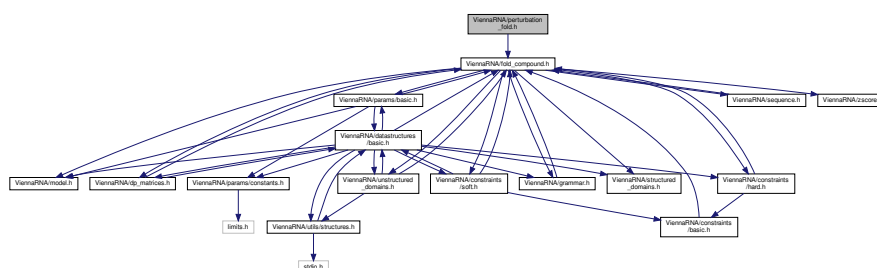
### Partition function and equilibrium probability implementation for the sliding window algorithm.

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]

## 18.86 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_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.*

### 18.86.1 Detailed Description

Find a vector of perturbation energies that minimizes the discrepancies between predicted and observed pairing probabilities and the amount of necessary adjustments.



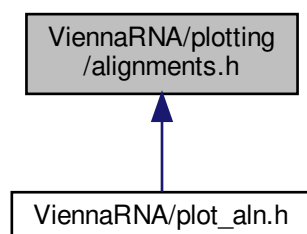




## 18.91 ViennaRNA/plotting/alignments.h File Reference

Various functions for plotting Sequence / Structure Alignments.

This graph shows which files directly or indirectly include this file:



### Functions

- int [vrna\\_file\\_PS\\_aln](#) (const char \*filename, const char \*\*seqs, const char \*\*names, const char \*structure, unsigned int columns)  
*Create an annotated PostScript alignment plot.*
- int [vrna\\_file\\_PS\\_aln\\_slice](#) (const char \*filename, const char \*\*seqs, const char \*\*names, const char \*structure, unsigned int start, unsigned int end, int offset, unsigned int columns)  
*Create an annotated PostScript alignment plot.*
- 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 [aliPS\\_color\\_aln](#) (const char \*structure, const char \*filename, const char \*seqs[], const char \*names[])  
*PS\_color\_aln for duplexes.*

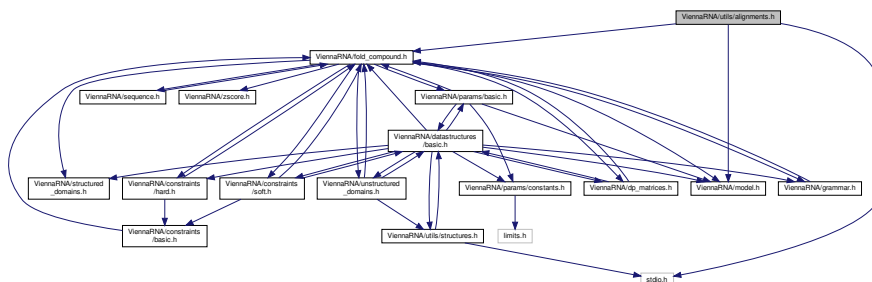
### 18.91.1 Detailed Description

Various functions for plotting Sequence / Structure Alignments.

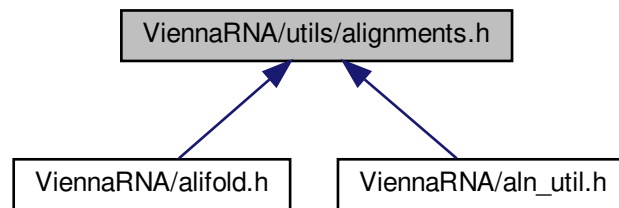
## 18.92 ViennaRNA/utis/alignments.h File Reference

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.*

### 18.92.1 Detailed Description

Various utility- and helper-functions for sequence alignments and comparative structure prediction.



## Typedefs

- typedef struct [vrna\\_plot\\_layout\\_s](#) [vrna\\_plot\\_layout\\_t](#)  
*RNA secondary structure figure layout.*

## Functions

- [vrna\\_plot\\_layout\\_t](#) \* [vrna\\_plot\\_layout](#) (const char \*structure, unsigned int plot\_type)  
*Create a layout (coordinates, etc.) for a secondary structure plot.*
- [vrna\\_plot\\_layout\\_t](#) \* [vrna\\_plot\\_layout\\_simple](#) (const char \*structure)  
*Create a layout (coordinates, etc.) for a simple secondary structure plot.*
- [vrna\\_plot\\_layout\\_t](#) \* [vrna\\_plot\\_layout\\_navview](#) (const char \*structure)  
*Create a layout (coordinates, etc.) for a secondary structure plot using the Naview Algorithm [5].*
- [vrna\\_plot\\_layout\\_t](#) \* [vrna\\_plot\\_layout\\_circular](#) (const char \*structure)  
*Create a layout (coordinates, etc.) for a circular secondary structure plot.*
- [vrna\\_plot\\_layout\\_t](#) \* [vrna\\_plot\\_layout\\_turtle](#) (const char \*structure)  
*Create a layout (coordinates, etc.) for a secondary structure plot using the Turtle Algorithm [23].*
- [vrna\\_plot\\_layout\\_t](#) \* [vrna\\_plot\\_layout\\_puzzler](#) (const char \*structure, [vrna\\_plot\\_options\\_puzzler\\_t](#) \*options)  
*Create a layout (coordinates, etc.) for a secondary structure plot using the RNApuzzler Algorithm [23].*
- void [vrna\\_plot\\_layout\\_free](#) ([vrna\\_plot\\_layout\\_t](#) \*layout)  
*Free memory occupied by a figure layout data structure.*
- int [vrna\\_plot\\_coords](#) (const char \*structure, float \*\*x, float \*\*y, int plot\_type)  
*Compute nucleotide coordinates for secondary structure plot.*
- int [vrna\\_plot\\_coords\\_pt](#) (const short \*pt, float \*\*x, float \*\*y, int plot\_type)  
*Compute nucleotide coordinates for secondary structure plot.*
- int [vrna\\_plot\\_coords\\_simple](#) (const char \*structure, float \*\*x, float \*\*y)  
*Compute nucleotide coordinates for secondary structure plot the Simple way*
- int [vrna\\_plot\\_coords\\_simple\\_pt](#) (const short \*pt, float \*\*x, float \*\*y)  
*Compute nucleotide coordinates for secondary structure plot the Simple way*
- int [vrna\\_plot\\_coords\\_circular](#) (const char \*structure, float \*\*x, float \*\*y)  
*Compute coordinates of nucleotides mapped in equal distances onto a unit circle.*
- int [vrna\\_plot\\_coords\\_circular\\_pt](#) (const short \*pt, float \*\*x, float \*\*y)  
*Compute nucleotide coordinates for a Circular Plot*
- 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.*

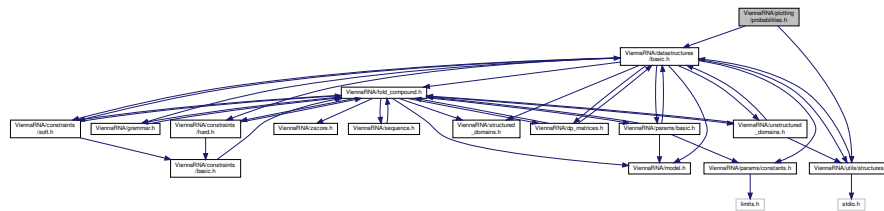
### 18.93.1 Detailed Description

Secondary structure plot layout algorithms.

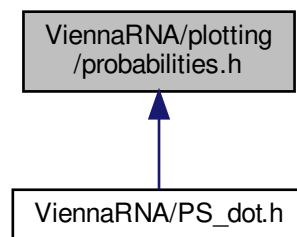
## 18.94 ViennaRNA/plotting/probabilities.h File Reference

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.*

### 18.94.1 Detailed Description

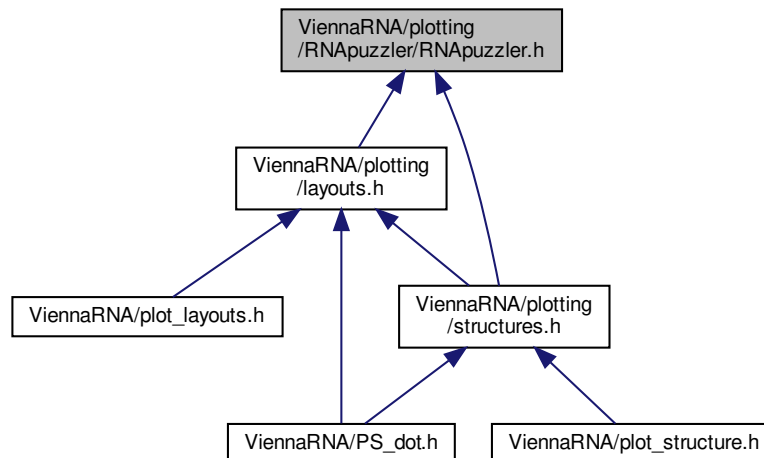
Various functions for plotting RNA secondary structures, dot-plots and other visualizations.



## 18.95 ViennaRNA/plotting/RNApuzzler/RNApuzzler.h File Reference

Implementation of the RNApuzzler RNA secondary structure layout algorithm [23].

This graph shows which files directly or indirectly include this file:



### Data Structures

- struct [vrna\\_plot\\_options\\_puzzler\\_t](#)

Options data structure for RNApuzzler algorithm implementation. [More...](#)

### Functions

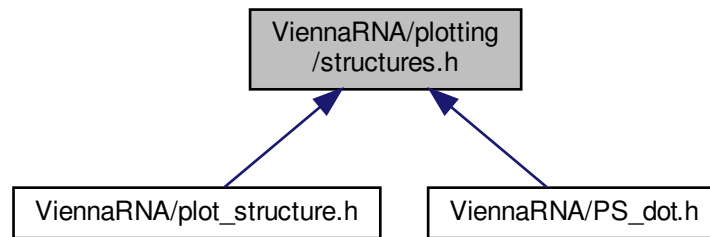
- int [vrna\\_plot\\_coords\\_puzzler](#) (const char \*structure, float \*\*x, float \*\*y, double \*\*arc\_coords, [vrna\\_plot\\_options\\_puzzler\\_t](#) \*options)  
Compute nucleotide coordinates for secondary structure plot using the RNApuzzler algorithm [23].
- int [vrna\\_plot\\_coords\\_puzzler\\_pt](#) (short const \*const pair\_table, float \*\*x, float \*\*y, double \*\*arc\_coords, [vrna\\_plot\\_options\\_puzzler\\_t](#) \*puzzler)  
Compute nucleotide coordinates for secondary structure plot using the RNApuzzler algorithm [23].
- [vrna\\_plot\\_options\\_puzzler\\_t](#) \* [vrna\\_plot\\_options\\_puzzler](#) (void)  
Create an RNApuzzler options data structure.
- void [vrna\\_plot\\_options\\_puzzler\\_free](#) ([vrna\\_plot\\_options\\_puzzler\\_t](#) \*options)  
Free memory occupied by an RNApuzzler options data structure.

#### 18.95.1 Detailed Description

Implementation of the RNApuzzler RNA secondary structure layout algorithm [23].



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)*

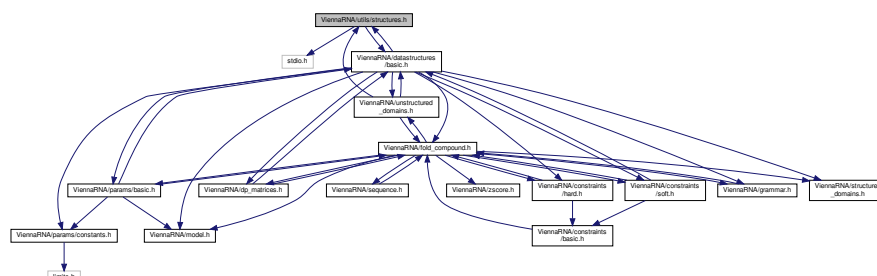
### 18.97.1 Detailed Description

Various functions for plotting RNA secondary structures.

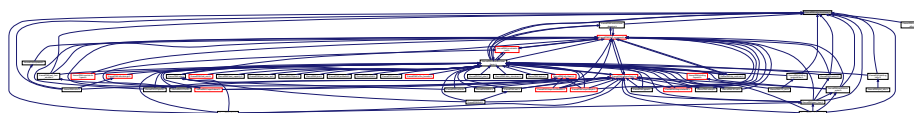
## 18.98 ViennaRNA/utils/structures.h File Reference

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\_BRACKETS\_ANY`  
Bitmask to indicate secondary structure notation using any pair of brackets or uppercase/lowercase alphabet letters.
- `#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.*
- `char * vrna_db_pk_remove (const char *structure, unsigned int options)`

- Remove pseudo-knots from an input structure.*

  - 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)*

  - short \* [vrna\\_pt\\_pk\\_remove](#) (const short \*ptable, unsigned int options)
- Remove pseudo-knots from a pair table.*

  - [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.*

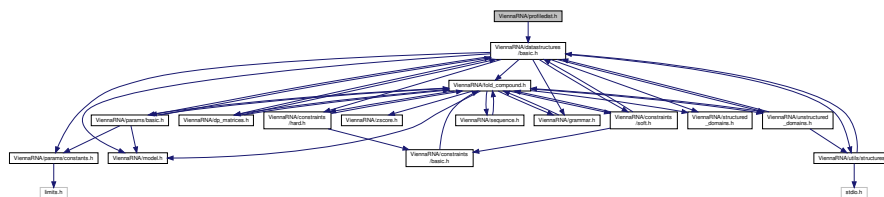
- short \* [copy\\_pair\\_table](#) (const short \*pt)  
*Get an exact copy of a pair table.*
- short \* [alimake\\_pair\\_table](#) (const char \*structure)
- short \* [make\\_pair\\_table\\_snoop](#) (const char \*structure)
- int [bp\\_distance](#) (const char \*str1, const char \*str2)  
*Compute 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.*

### 18.98.1 Detailed Description

Various utility- and helper-functions for secondary structure parsing, converting, etc.

## 18.99 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)

## 18.99.1 Detailed Description

## 18.99.2 Function Documentation

### 18.99.2.1 `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

### 18.99.2.2 `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

### 18.99.2.3 `free_profile()`

```
void free_profile (
    float * T )
```

free space allocated in `Make_bp_profile`

Backward compatibility only. You can just use plain `free()`



#### 18.99.2.4 Make\_bp\_profile()

```
float* Make_bp_profile (
    int length )
```

### Note

This function is NOT threadsafe

## See also

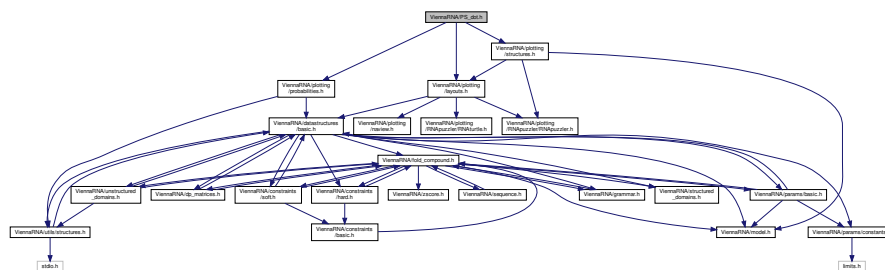
Make\_bp\_profile\_bpppm()

**Deprecated** This function is deprecated and will be removed soon! See [Make\\_bp\\_profile\\_bppm\(\)](#) for a replacement

## 18.100 ViennaRNA/PS\_dot.h File Reference

Use [ViennaRNA/plotting/probabilities.h](#) instead.

Include dependency graph for PS\_dot.h:



### 18.100.1 Detailed Description

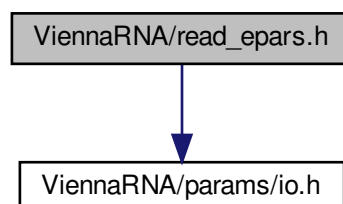
Use [ViennaRNA/plotting/probabilities.h](#) instead.

**Deprecated** Use [ViennaRNA/plotting/probabilities.h](#) instead

## 18.101 ViennaRNA/read\_epars.h File Reference

Use [ViennaRNA/params/io.h](#) instead.

Include dependency graph for read\_epars.h:



### 18.101.1 Detailed Description

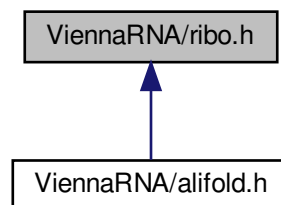
Use [ViennaRNA/params/io.h](#) instead.

**Deprecated** Use [ViennaRNA/params/io.h](#) instead

## 18.102 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.*

### 18.102.1 Detailed Description

Parse RiboSum Scoring Matrices for Covariance Scoring of Alignments.

## 18.103 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.*

### 18.103.1 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)
*
```

## 18.104 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.*

### 18.104.1 Detailed Description

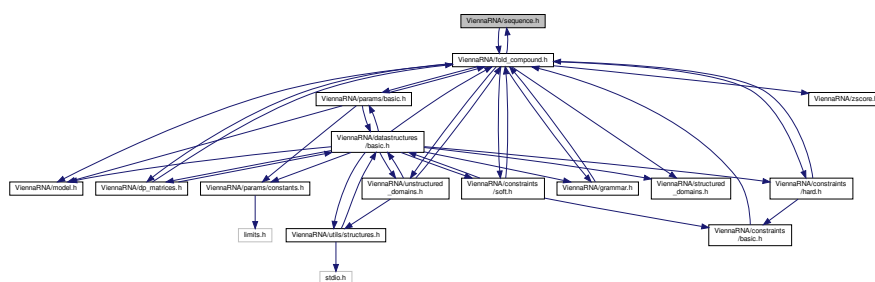
Variants of the Boyer-Moore string search algorithm.

,

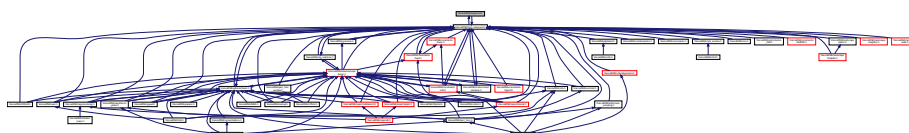
## 18.105 ViennaRNA/sequence.h File Reference

Functions and data structures related to sequence representations ,.

Include dependency graph for sequence.h:



This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [vrna\\_sequence\\_s](#)  
*Data structure representing a nucleotide sequence. [More...](#)*
- struct [vrna\\_alignment\\_s](#)

## 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.*

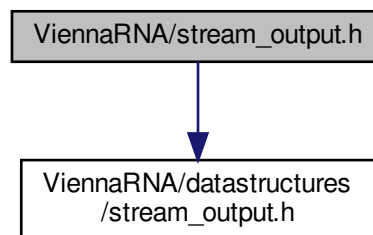
### 18.105.1 Detailed Description

Functions and data structures related to sequence representations ,.

## 18.106 ViennaRNA/stream\_output.h File Reference

Use [ViennaRNA/datastructures/stream\\_output.h](#) instead.

Include dependency graph for stream\_output.h:



### 18.106.1 Detailed Description

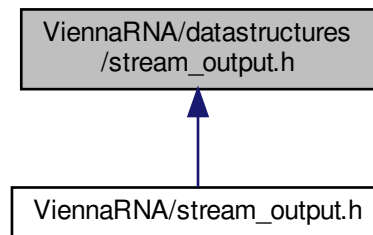
Use [ViennaRNA/datastructures/stream\\_output.h](#) instead.

**Deprecated** Use [ViennaRNA/datastructures/stream\\_output.h](#) instead

## 18.107 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.*

### 18.107.1 Detailed Description

An implementation of a buffered, ordered stream output data structure.

,



### 18.109.1 Detailed Description

Functions for String Alignment.

### 18.109.2 Function Documentation

#### 18.109.2.1 Make\_swString()

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

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

##### Parameters

<i>string</i>	
---------------	--

##### Returns

#### 18.109.2.2 string\_edit\_distance()

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

Calculate the string edit distance of T1 and T2.

##### Parameters

<i>T1</i>	
<i>T2</i>	

##### Returns

## 18.110 ViennaRNA/structure\_utils.h File Reference

Use [ViennaRNA/utis/structures.h](#) instead.



[illegible]

**Deprecated** Use [ViennaRNA/utils/structures.h](#) instead

This module provides interfaces that deal with additional structured domains in the folding grammar.



## 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.*

### 18.112.1 Detailed Description

RNAsubopt and density of states declarations.

### 18.112.2 Typedef Documentation

#### 18.112.2.1 SOLUTION

```
typedef struct vrna_subopt_sol_s SOLUTION
```

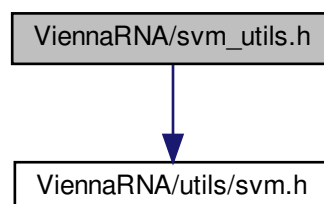
Backward compatibility typedef for [vrna\\_subopt\\_sol\\_s](#).

**Deprecated** Use [vrna\\_subopt\\_solution\\_t](#) instead!

## 18.113 ViennaRNA/svm\_utils.h File Reference

Use [ViennaRNA/utis/svm.h](#) instead.

Include dependency graph for svm\_utils.h:



### 18.113.1 Detailed Description

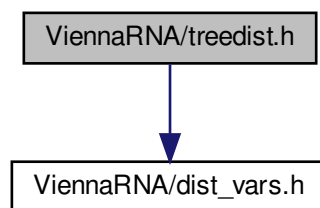
Use [ViennaRNA/utis/svm.h](#) instead.

**Deprecated** Use [ViennaRNA/utis/svm.h](#) instead

## 18.114 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.

### 18.114.1 Detailed Description

Functions for [Tree](#) Edit Distances.

### 18.114.2 Function Documentation

#### 18.114.2.1 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

**18.114.2.2 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

**18.114.2.3 free\_tree()**

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

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

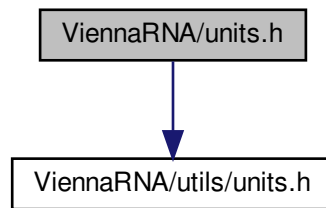
## Parameters

<i>t</i>	
----------	--

**18.115 ViennaRNA/units.h File Reference**

Use [ViennaRNA/utlis/units.h](#) instead.

Include dependency graph for `units.h`:



### 18.115.1 Detailed Description

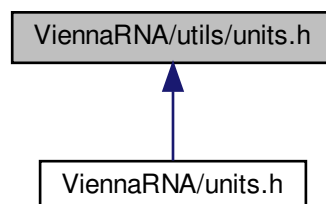
Use [ViennaRNA/utls/units.h](#) instead.

**Deprecated** Use [ViennaRNA/utls/units.h](#) instead

## 18.116 ViennaRNA/utls/units.h File Reference

Physical Units and Functions to convert them into each other.

This graph shows which files directly or indirectly include this file:



### 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.*

- 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.*
- int `vrna_convert_kcal_to_dcal` (double energy)  
*Convert floating point energy value into integer representation.*
- double `vrna_convert_dcal_to_kcal` (int energy)  
*Convert an integer representation of free energy in deka-cal/mol to kcal/mol.*

Physical Units and Functions to convert them into each other.

Functions to modify unstructured domains, e.g. to incorporate ligands binding to unpaired stretches.

- 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)`





### 18.118.1 Detailed Description

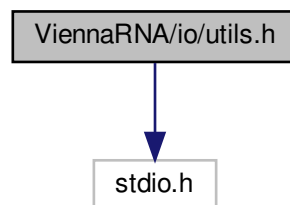
Use [ViennaRNA/utils/basic.h](#) instead.

**Deprecated** Use [ViennaRNA/utils/basic.h](#) instead

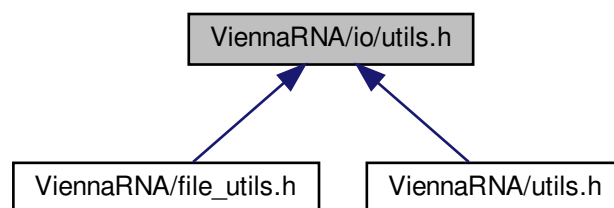
### 18.119 ViennaRNA/io/utils.h File Reference

Several utilities for file handling.

Include dependency graph for utils.h:



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.*

### 18.119.1 Detailed Description

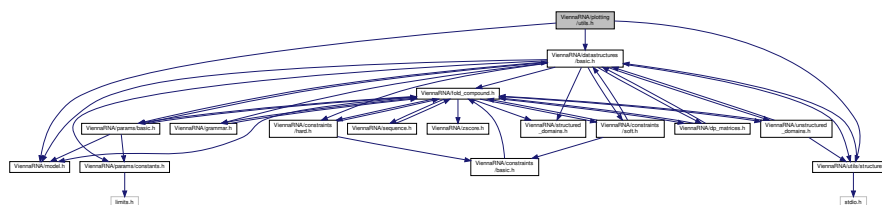
Several utilities for file handling.

,

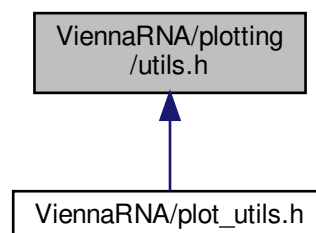
## 18.120 ViennaRNA/plotting/utils.h File Reference

Various utilities to assist in plotting secondary structures and consensus structures.

Include dependency graph for utils.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 *sequence)`  
*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.*

### 18.121.1 Detailed Description

General utility- and helper-functions for RNA sequence and structure strings used throughout the ViennaRNA Package.

,

### 18.121.2 Function Documentation

#### 18.121.2.1 `str_uppercase()`

```
void str_uppercase (
    char * sequence )
```

Convert an input sequence to uppercase.

**Deprecated** Use `vrna_seq_toupper()` instead!

#### 18.121.2.2 `str_DNA2RNA()`

```
void str_DNA2RNA (
    char * sequence )
```

Convert a DNA input sequence to RNA alphabet.

**Deprecated** Use `vrna_seq_toRNA()` instead!

#### 18.121.2.3 `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!

#### 18.121.2.4 hamming()

```
int hamming (
    const char * s1,
    const char * s2 )
```

Calculate hamming distance between two sequences.

**Deprecated** Use `vrna_hamming_distance()` instead!

### 18.121.2.5 hamming\_bound()

```
int hamming_bound (
    const char * s1,
    const char * s2,
    int n )
```

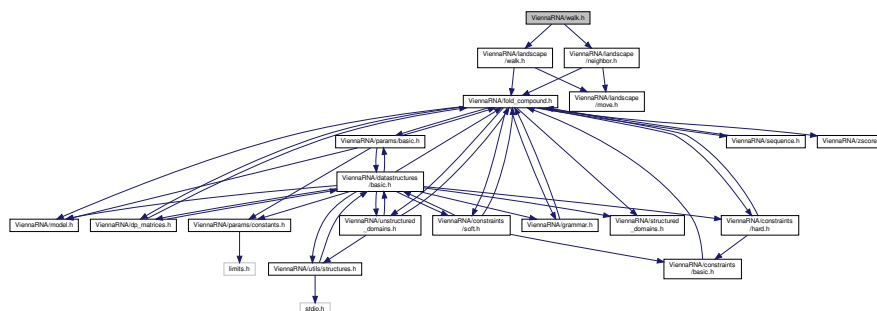
Calculate hamming distance between two sequences up to a specified length.

**Deprecated** Use `vrna_hamming_distance_bound()` instead!

## 18.122 ViennaRNA/walk.h File Reference

Use [ViennaRNA/landscape/walk.h](#) instead.

Include dependency graph for walk.h:



### 18.122.1 Detailed Description

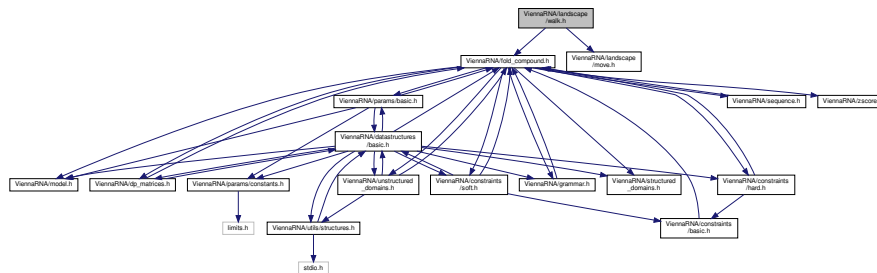
Use [ViennaRNA/landscape/walk.h](#) instead.

**Deprecated** Use [ViennaRNA/landscape/walk.h](#) instead

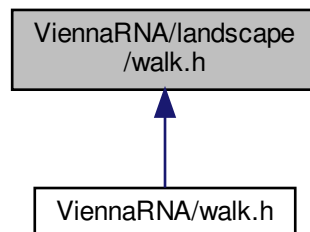
## 18.123 ViennaRNA/landscape/walk.h File Reference

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:



This graph shows which files directly or indirectly include this file:



## 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.*

### 18.123.1 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. [25](#)
- [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. [456](#)
- [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. [314](#)
- [4] Stephan H Bernhart, Ullrike Mückstein, and Ivo L Hofacker. RNA accessibility in cubic time. *Algorithms for Molecular Biology*, 6(1):3, 2011. [315](#), [798](#), [835](#)
- [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. [129](#), [554](#), [562](#), [563](#), [566](#), [574](#), [575](#), [715](#), [808](#), [843](#)
- [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. [419](#)
- [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. [406](#), [407](#), [408](#), [409](#), [410](#)
- [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](#), [514](#)
- [9] Eva Freyhult, Vincent Moulton, and Paul Gardner. Predicting RNA structure using mutual information. *Applied bioinformatics*, 4(1):53–59, 2005. [524](#)
- [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. [25](#)
- [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. [22](#), [293](#), [295](#), [309](#), [311](#)
- [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. [25](#), [357](#)
- [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. [20](#)
- [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. [418](#)

- [17] J.S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29(6-7):1105–1119, 1990. [23](#)
- [18] Juraj Michálik, Hélène Touzet, and Yann Ponty. Efficient approximations of RNA kinetics landscape using non-redundant sampling. *Bioinformatics*, 33(14):i283–i292, 2017. [333](#), [335](#), [336](#), [338](#), [339](#), [342](#), [343](#), [344](#), [346](#)
- [19] Joe Sawada. A fast algorithm to generate necklaces with fixed content. *Theoretical Computer Science*, 301(1):477–489, 2003. [589](#)
- [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](#), [514](#), [515](#)
- [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. [15](#)
- [22] 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. [425](#)
- [23] Daniel Wiegrefe, Daniel Alexander, Peter F. Stadler, and Dirk Zeckzer. RNApuzzler: efficient outerplanar drawing of RNA-secondary structures. *Bioinformatics*, 2018. [129](#), [554](#), [561](#), [562](#), [567](#), [568](#), [575](#), [576](#), [578](#), [579](#), [842](#), [843](#), [845](#), [846](#)
- [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. [327](#), [328](#)
- [25] Kourosh Zarrinhalam, 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. [420](#)
- [26] M. Zuker. On finding all suboptimal foldings of an RNA molecule. *Science*, 244(4900):48–52, April 1989. [324](#)
- [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. [22](#)

# Index

(Abstract) Data Structures, [595](#)

    bondT, [601](#)

    cpair, [600](#)

    PAIR, [600](#)

    plist, [600](#)

    sect, [600](#)

    vrna\_C11\_features, [601](#)

(Nucleic Acid Sequence) String Utilities, [482](#)

    FILENAME\_ID\_LENGTH, [483](#)

    FILENAME\_MAX\_LENGTH, [483](#)

    vrna\_cut\_point\_insert, [489](#)

    vrna\_cut\_point\_remove, [490](#)

    vrna\_hamming\_distance, [487](#)

    vrna\_hamming\_distance\_bound, [488](#)

    vrna\_random\_string, [487](#)

    vrna\_seq\_toRNA, [488](#)

    vrna\_seq\_toupper, [488](#)

    vrna\_seq\_ungapped, [489](#)

    vrna\_strcat\_printf, [485](#)

    vrna\_strcat\_vprintf, [485](#)

    vrna\_strdup\_printf, [483](#)

    vrna\_strdup\_vprintf, [484](#)

    vrna\_strsplit, [486](#)

(Re-)folding Paths, Saddle Points, Energy Barriers, and  
    Local Minima, [400](#)

    vrna\_path\_free, [403](#)

    vrna\_path\_options\_free, [403](#)

    VRNA\_PATH\_TYPE\_DOT\_BRACKET, [402](#)

    VRNA\_PATH\_TYPE\_MOVES, [403](#)

\_struct\_en, [719](#)

2Dpfold.h

    destroy\_TwoDpfold\_variables, [728](#)

    get\_TwoDpfold\_variables, [727](#)

    TwoDpfold\_pbacktrack, [729](#)

    TwoDpfold\_pbacktrack5, [730](#)

    TwoDpfoldList, [728](#)

add\_root

    Deprecated Interface for Secondary Structure Utilities, [704](#)

alifold

    Deprecated Interface for Global MFE Prediction, [659](#)

alifold.h

    cv\_fact, [733](#)

    energy\_of\_alistruct, [732](#)

    nc\_fact, [733](#)

    update\_alifold\_params, [732](#)

Alignment Plots, [581](#)

    vrna\_file\_PS\_aln, [581](#)

    vrna\_file\_PS\_aln\_slice, [582](#)

alimake\_pair\_table

    Deprecated Interface for Secondary Structure Utilities, [708](#)

alipbacktrack

    Deprecated Interface for Global Partition Function Computation, [689](#)

alipf\_circ\_fold

    Deprecated Interface for Global Partition Function Computation, [687](#)

alipf\_fold

    Deprecated Interface for Global Partition Function Computation, [687](#)

alipf\_fold\_par

    Deprecated Interface for Global Partition Function Computation, [674](#)

aliPS\_color\_aln

    Deprecated Interface for Plotting Utilities, [713](#)

alloc\_sequence\_arrays

    Deprecated Interface for Multiple Sequence Alignment Utilities, [699](#)

alpha

    vrna\_exp\_param\_s, [219](#)

Annotation, [580](#)

assign\_plist\_from\_db

    Deprecated Interface for Global Partition Function Computation, [686](#)

assign\_plist\_from\_pr

    Deprecated Interface for Global Partition Function Computation, [686](#)

auxdata

    vrna\_fc\_s, [617](#)

b2C

    Deprecated Interface for Secondary Structure Utilities, [703](#)

b2HIT

    Deprecated Interface for Secondary Structure Utilities, [702](#)

b2Shapiro

    Deprecated Interface for Secondary Structure Utilities, [703](#)

backtrack\_GQuad\_IntLoop

    G-Quadruplexes, [433](#)

backtrack\_GQuad\_IntLoop\_L

    G-Quadruplexes, [434](#)

backtrack\_type

    Fine-tuning of the Implemented Models, [215](#)

Backtracking MFE structures, [302](#)

    vrna\_backtrack5, [302](#)

- vrna\_BT\_hp\_loop, [303](#)
- vrna\_BT\_mb\_loop, [304](#)
- base\_pair
  - fold\_vars.h, [782](#)
- basic.h
  - filecopy, [821](#)
  - get\_line, [818](#)
  - init\_rand, [820](#)
  - int\_urn, [820](#)
  - nerror, [819](#)
  - print\_tty\_input\_seq, [818](#)
  - print\_tty\_input\_seq\_str, [819](#)
  - space, [819](#)
  - time\_stamp, [821](#)
  - urn, [820](#)
  - warn\_user, [819](#)
  - xrealloc, [820](#)
- bondT
  - (Abstract) Data Structures, [601](#)
- BONUS
  - constants.h, [824](#)
- bp\_distance
  - Deprecated Interface for Secondary Structure Utilities, [709](#)
- bppm\_symbol
  - Deprecated Interface for Secondary Structure Utilities, [711](#)
- bppm\_to\_structure
  - Deprecated Interface for Secondary Structure Utilities, [711](#)
- bt
  - vrna\_sc\_s, [276](#)
- Buffers, [652](#)
  - vrna\_callback\_stream\_output, [653](#)
  - vrna\_cstr, [653](#)
  - vrna\_cstr\_close, [654](#)
  - vrna\_cstr\_fflush, [654](#)
  - vrna\_cstr\_free, [654](#)
  - vrna\_ostream\_free, [655](#)
  - vrna\_ostream\_init, [655](#)
  - vrna\_ostream\_provide, [656](#)
  - vrna\_ostream\_request, [656](#)
- centroid
  - part\_func.h, [830](#)
- centroid.h
  - get\_centroid\_struct\_pl, [738](#)
  - get\_centroid\_struct\_pr, [738](#)
- circularfold
  - Deprecated Interface for Global MFE Prediction, [669](#)
- circfold
  - Deprecated Interface for Global MFE Prediction, [665](#)
- Classified Dynamic Programming Variants, [355](#)
- co\_pf\_fold
  - Deprecated Interface for Global Partition Function Computation, [682](#)
- co\_pf\_fold\_par
  - Deprecated Interface for Global Partition Function Computation, [682](#)
- cofold
  - Deprecated Interface for Global MFE Prediction, [660](#)
- cofold\_par
  - Deprecated Interface for Global MFE Prediction, [660](#)
- Combinatorics Algorithms, [588](#)
  - vrna\_enumerate\_necklaces, [588](#)
  - vrna\_rotational\_symmetry, [591](#)
  - vrna\_rotational\_symmetry\_db, [592](#)
  - vrna\_rotational\_symmetry\_db\_pos, [593](#)
  - vrna\_rotational\_symmetry\_num, [589](#)
  - vrna\_rotational\_symmetry\_pos, [591](#)
  - vrna\_rotational\_symmetry\_pos\_num, [590](#)
- Command Files, [548](#)
  - VRNA\_CMD\_PARSE\_DEFAULTS, [550](#)
  - VRNA\_CMD\_PARSE\_HC, [549](#)
  - VRNA\_CMD\_PARSE\_SC, [549](#)
  - VRNA\_CMD\_PARSE\_SD, [550](#)
  - VRNA\_CMD\_PARSE\_UD, [549](#)
  - vrna\_commands\_apply, [551](#)
  - vrna\_commands\_free, [552](#)
  - vrna\_file\_commands\_apply, [551](#)
  - vrna\_file\_commands\_read, [550](#)
- Complex Structured Modules, [431](#)
- Compute the Centroid Structure, [351](#)
  - vrna\_centroid, [351](#)
  - vrna\_centroid\_from\_plist, [352](#)
  - vrna\_centroid\_from\_probs, [352](#)
- Compute the Density of States, [382](#)
  - density\_of\_states, [382](#)
- Compute the Structure with Maximum Expected Accuracy (MEA), [348](#)
  - MEA, [350](#)
  - vrna\_MEA, [348](#)
  - vrna\_MEA\_from\_plist, [349](#)
- compute\_BPdifferences
  - Deprecated Interface for Secondary Structure Utilities, [710](#)
- compute\_probabilities
  - Deprecated Interface for Global Partition Function Computation, [683](#)
- Computing MFE representatives of a Distance Based Partitioning, [357](#)
  - destroy\_TwoDfold\_variables, [363](#)
  - get\_TwoDfold\_variables, [362](#)
  - TwoDfold\_backtrack\_f5, [364](#)
  - TwoDfold\_vars, [360](#)
  - TwoDfoldList, [363](#)
  - vrna\_backtrack5\_TwoD, [361](#)
  - vrna\_mfe\_TwoD, [360](#)
  - vrna\_sol\_TwoD\_t, [360](#)
- Computing Partition Functions of a Distance Based Partitioning, [365](#)
  - vrna\_pf\_TwoD, [366](#)
  - vrna\_sol\_TwoD\_pf\_t, [366](#)

- concentrations.h
  - get\_concentrations, [744](#)
- cons\_seq
  - vrna\_fc\_s, [619](#)
- constants.h
  - BONUS, [824](#)
  - FORBIDDEN, [824](#)
  - GASCONST, [824](#)
  - INF, [824](#)
  - K0, [824](#)
  - MAXLOOP, [825](#)
  - NBPAIRS, [825](#)
  - TURN, [825](#)
- constrain, [599](#)
- constrain\_ptypes
  - hard.h, [750](#)
- Constraining the RNA Folding Grammar, [245](#)
  - VRNA\_CONSTRAINT\_FILE, [248](#)
  - VRNA\_CONSTRAINT\_SOFT\_MFE, [248](#)
  - VRNA\_CONSTRAINT\_SOFT\_PF, [249](#)
  - vrna\_constraints\_add, [258](#)
  - VRNA\_DECOMP\_EXT\_EXT, [254](#)
  - VRNA\_DECOMP\_EXT\_EXT\_EXT, [256](#)
  - VRNA\_DECOMP\_EXT\_EXT\_STEM, [257](#)
  - VRNA\_DECOMP\_EXT\_EXT\_STEM1, [257](#)
  - VRNA\_DECOMP\_EXT\_STEM, [255](#)
  - VRNA\_DECOMP\_EXT\_STEM\_EXT, [256](#)
  - VRNA\_DECOMP\_EXT\_UP, [255](#)
  - VRNA\_DECOMP\_ML\_COAXIAL, [253](#)
  - VRNA\_DECOMP\_ML\_COAXIAL\_ENC, [254](#)
  - VRNA\_DECOMP\_ML\_ML, [252](#)
  - VRNA\_DECOMP\_ML\_ML\_ML, [251](#)
  - VRNA\_DECOMP\_ML\_ML\_STEM, [253](#)
  - VRNA\_DECOMP\_ML\_STEM, [251](#)
  - VRNA\_DECOMP\_ML\_UP, [252](#)
  - VRNA\_DECOMP\_PAIR\_HP, [249](#)
  - VRNA\_DECOMP\_PAIR\_IL, [249](#)
  - VRNA\_DECOMP\_PAIR\_ML, [250](#)
  - vrna\_message\_constraint\_options, [259](#)
  - vrna\_message\_constraint\_options\_all, [261](#)
- convert\_parameter\_file
  - Converting Energy Parameter Files, [476](#)
- Converting Energy Parameter Files, [471](#)
  - convert\_parameter\_file, [476](#)
  - VRNA\_CONVERT\_OUTPUT\_ALL, [472](#)
  - VRNA\_CONVERT\_OUTPUT\_BULGE, [474](#)
  - VRNA\_CONVERT\_OUTPUT\_DANGLE3, [473](#)
  - VRNA\_CONVERT\_OUTPUT\_DANGLE5, [473](#)
  - VRNA\_CONVERT\_OUTPUT\_DUMP, [475](#)
  - VRNA\_CONVERT\_OUTPUT\_HP, [472](#)
  - VRNA\_CONVERT\_OUTPUT\_INT, [474](#)
  - VRNA\_CONVERT\_OUTPUT\_INT\_11, [473](#)
  - VRNA\_CONVERT\_OUTPUT\_INT\_21, [474](#)
  - VRNA\_CONVERT\_OUTPUT\_INT\_22, [474](#)
  - VRNA\_CONVERT\_OUTPUT\_MISC, [474](#)
  - VRNA\_CONVERT\_OUTPUT\_ML, [474](#)
  - VRNA\_CONVERT\_OUTPUT\_MM\_EXT, [473](#)
  - VRNA\_CONVERT\_OUTPUT\_MM\_HP, [472](#)
  - VRNA\_CONVERT\_OUTPUT\_MM\_INT, [472](#)
  - VRNA\_CONVERT\_OUTPUT\_MM\_INT\_1N, [472](#)
  - VRNA\_CONVERT\_OUTPUT\_MM\_INT\_23, [473](#)
  - VRNA\_CONVERT\_OUTPUT\_MM\_MULTI, [473](#)
  - VRNA\_CONVERT\_OUTPUT\_NINIO, [475](#)
  - VRNA\_CONVERT\_OUTPUT\_SPECIAL\_HP, [475](#)
  - VRNA\_CONVERT\_OUTPUT\_STACK, [472](#)
  - VRNA\_CONVERT\_OUTPUT\_VANILLA, [475](#)
- COORDINATE, [712](#)
- copy\_pair\_table
  - Deprecated Interface for Secondary Structure Utilities, [708](#)
- cost\_matrix
  - dist\_vars.h, [762](#)
- cpair
  - (Abstract) Data Structures, [600](#)
- cut\_point
  - fold\_vars.h, [781](#)
- cv\_fact
  - alifold.h, [733](#)
- dangles
  - Fine-tuning of the Implemented Models, [213](#)
  - vrna\_md\_s, [187](#)
- density\_of\_states
  - Compute the Density of States, [382](#)
- Deprecated Interface for (Re-)folding Paths, Saddle Points, and Energy Barriers, [716](#)
  - find\_saddle, [717](#)
  - free\_path, [717](#)
  - get\_path, [718](#)
  - path\_t, [716](#)
- Deprecated Interface for Free Energy Evaluation, [163](#)
  - E\_IntLoop, [174](#)
  - E\_Stem, [172](#)
  - energy\_of\_circ\_struct, [171](#)
  - energy\_of\_circ\_struct\_par, [166](#)
  - energy\_of\_circ\_structure, [165](#)
  - energy\_of\_move, [168](#)
  - energy\_of\_move\_pt, [169](#)
  - energy\_of\_struct, [170](#)
  - energy\_of\_struct\_par, [164](#)
  - energy\_of\_struct\_pt, [171](#)
  - energy\_of\_struct\_pt\_par, [167](#)
  - energy\_of\_structure, [164](#)
  - energy\_of\_structure\_pt, [166](#)
  - exp\_E\_ExtLoop, [173](#)
  - exp\_E\_IntLoop, [176](#)
  - exp\_E\_Stem, [173](#)
  - loop\_energy, [169](#)
- Deprecated Interface for Global MFE Prediction, [658](#)
  - alifold, [659](#)
  - circalifold, [669](#)
  - circfold, [665](#)
  - cofold, [660](#)
  - cofold\_par, [660](#)
  - export\_circfold\_arrays, [668](#)
  - export\_circfold\_arrays\_par, [668](#)
  - export\_cofold\_arrays, [662](#)

- export\_cofold\_arrays\_gq, [661](#)
- export\_fold\_arrays, [667](#)
- export\_fold\_arrays\_par, [667](#)
- fold, [665](#)
- fold\_par, [664](#)
- free\_alifold\_arrays, [670](#)
- free\_arrays, [666](#)
- free\_co\_arrays, [660](#)
- get\_monomere\_mfes, [663](#)
- HairpinE, [669](#)
- initialize\_cofold, [664](#)
- initialize\_fold, [669](#)
- LoopEnergy, [668](#)
- update\_cofold\_params, [661](#)
- update\_cofold\_params\_par, [661](#)
- update\_fold\_params, [666](#)
- update\_fold\_params\_par, [667](#)
- Deprecated Interface for Global Partition Function Computation, [673](#)
- alipbacktrack, [689](#)
- alipf\_circ\_fold, [687](#)
- alipf\_fold, [687](#)
- alipf\_fold\_par, [674](#)
- assign\_plist\_from\_db, [686](#)
- assign\_plist\_from\_pr, [686](#)
- co\_pf\_fold, [682](#)
- co\_pf\_fold\_par, [682](#)
- compute\_probabilities, [683](#)
- export\_ali\_bppm, [688](#)
- export\_bppm, [679](#)
- export\_co\_bppm, [684](#)
- free\_alipf\_arrays, [688](#)
- free\_co\_pf\_arrays, [684](#)
- free\_pf\_arrays, [678](#)
- get\_alipf\_arrays, [689](#)
- get\_pf\_arrays, [679](#)
- init\_co\_pf\_fold, [684](#)
- init\_pf\_fold, [681](#)
- mean\_bp\_distance, [680](#)
- mean\_bp\_distance\_pr, [681](#)
- pf\_circ\_fold, [677](#)
- pf\_fold, [676](#)
- pf\_fold\_par, [675](#)
- stackProb, [681](#)
- update\_co\_pf\_params, [685](#)
- update\_co\_pf\_params\_par, [685](#)
- update\_pf\_params, [678](#)
- update\_pf\_params\_par, [679](#)
- Deprecated Interface for Local (Sliding Window) MFE Prediction, [671](#)
- Lfold, [671](#)
- Lfoldz, [671](#)
- Deprecated Interface for Local (Sliding Window) Partition Function Computation, [691](#)
- pfl\_fold, [692](#)
- putoutpU\_prob, [692](#)
- putoutpU\_prob\_bin, [693](#)
- update\_pf\_paramsLP, [691](#)
- Deprecated Interface for Multiple Sequence Alignment Utilities, [697](#)
- alloc\_sequence\_arrays, [699](#)
- encode\_ali\_sequence, [698](#)
- free\_sequence\_arrays, [699](#)
- get\_mpi, [698](#)
- pair\_info, [697](#)
- Deprecated Interface for Plotting Utilities, [712](#)
- aliPS\_color\_aln, [713](#)
- naview\_xy\_coordinates, [715](#)
- PS\_color\_aln, [712](#)
- rna\_plot\_type, [715](#)
- simple\_circplot\_coordinates, [714](#)
- simple\_xy\_coordinates, [713](#)
- Deprecated Interface for Secondary Structure Utilities, [701](#)
- add\_root, [704](#)
- alimake\_pair\_table, [708](#)
- b2C, [703](#)
- b2HIT, [702](#)
- b2Shapiro, [703](#)
- bp\_distance, [709](#)
- bppm\_symbol, [711](#)
- bppm\_to\_structure, [711](#)
- compute\_BPdifferences, [710](#)
- copy\_pair\_table, [708](#)
- expand\_Full, [704](#)
- expand\_Shapiro, [704](#)
- make\_pair\_table, [707](#)
- make\_pair\_table\_snoop, [708](#)
- make\_referenceBP\_array, [709](#)
- pack\_structure, [706](#)
- parenthesis\_structure, [710](#)
- parenthesis\_zucker, [710](#)
- parse\_structure, [706](#)
- unexpand\_aligned\_F, [706](#)
- unexpand\_Full, [705](#)
- unpack\_structure, [707](#)
- unweight, [705](#)
- Deprecated Interface for Stochastic Backtracking, [694](#)
- pbacktrack, [694](#)
- pbacktrack\_circ, [695](#)
- st\_back, [695](#)
- destroy\_TwoDfold\_variables
- Computing MFE representatives of a Distance Based Partitioning, [363](#)
- destroy\_TwoDpfold\_variables
- 2Dpfold.h, [728](#)
- Direct Refolding Paths between two Secondary Structures, [405](#)
- vrna\_path\_direct, [409](#)
- vrna\_path\_direct\_ub, [410](#)
- vrna\_path\_findpath, [407](#)
- vrna\_path\_findpath\_saddle, [405](#)
- vrna\_path\_findpath\_saddle\_ub, [406](#)
- vrna\_path\_findpath\_ub, [408](#)
- vrna\_path\_options\_findpath, [409](#)
- dist\_vars.h



- cost\_matrix, 762
- edit\_backtrack, 761
- Distance Based Partitioning of the Secondary Structure Space, 356
- do\_backtrack
  - Fine-tuning of the Implemented Models, 214
- Dot-Bracket Notation of Secondary Structures, 494
  - VRNA\_BRACKETS\_ALPHA, 495
  - VRNA\_BRACKETS\_ANG, 495
  - VRNA\_BRACKETS\_ANY, 496
  - VRNA\_BRACKETS\_CLY, 495
  - VRNA\_BRACKETS\_DEFAULT, 496
  - VRNA\_BRACKETS\_RND, 495
  - VRNA\_BRACKETS\_SQR, 496
  - vrna\_db\_flatten, 498
  - vrna\_db\_flatten\_to, 499
  - vrna\_db\_from\_plist, 500
  - vrna\_db\_from\_ptable, 499
  - vrna\_db\_from\_WUSS, 500
  - vrna\_db\_pack, 497
  - vrna\_db\_pk\_remove, 501
  - vrna\_db\_to\_element\_string, 501
  - vrna\_db\_unpack, 497
- duplexT, 599
- dupVar, 599
- E\_Hairpin
  - Hairpin Loops, 449
- E\_IntLoop
  - Deprecated Interface for Free Energy Evaluation, 174
- E\_Stem
  - Deprecated Interface for Free Energy Evaluation, 172
- edit\_backtrack
  - dist\_vars.h, 761
- encode\_ali\_sequence
  - Deprecated Interface for Multiple Sequence Alignment Utilities, 698
- energy
  - vrna\_ht\_entry\_db\_t, 635
- Energy Evaluation for Atomic Moves, 160
  - vrna\_eval\_move, 160
  - vrna\_eval\_move\_pt, 161
- Energy Evaluation for Individual Loops, 157
  - vrna\_eval\_loop\_pt, 158
  - vrna\_eval\_loop\_pt\_v, 158
- Energy Parameters, 216
  - get\_boltzmann\_factor\_copy, 227
  - get\_boltzmann\_factors, 226
  - get\_boltzmann\_factors\_ali, 228
  - get\_scaled\_alipf\_parameters, 228
  - get\_scaled\_parameters, 229
  - get\_scaled\_pf\_parameters, 226
  - paramT, 219
  - pf\_paramT, 219
  - scale\_parameters, 228
  - vrna\_exp\_params, 221
  - vrna\_exp\_params\_comparative, 221
  - vrna\_exp\_params\_copy, 222
  - vrna\_exp\_params\_rescale, 224
  - vrna\_exp\_params\_reset, 225
  - vrna\_exp\_params\_subst, 223
  - vrna\_params, 220
  - vrna\_params\_copy, 220
  - vrna\_params\_reset, 225
  - vrna\_params\_subst, 223
- energy\_of\_alistruct
  - alifold.h, 732
- energy\_of\_circ\_struct
  - Deprecated Interface for Free Energy Evaluation, 171
- energy\_of\_circ\_struct\_par
  - Deprecated Interface for Free Energy Evaluation, 166
- energy\_of\_circ\_structure
  - Deprecated Interface for Free Energy Evaluation, 165
- energy\_of\_move
  - Deprecated Interface for Free Energy Evaluation, 168
- energy\_of\_move\_pt
  - Deprecated Interface for Free Energy Evaluation, 169
- energy\_of\_struct
  - Deprecated Interface for Free Energy Evaluation, 170
- energy\_of\_struct\_par
  - Deprecated Interface for Free Energy Evaluation, 164
- energy\_of\_struct\_pt
  - Deprecated Interface for Free Energy Evaluation, 171
- energy\_of\_struct\_pt\_par
  - Deprecated Interface for Free Energy Evaluation, 167
- energy\_of\_structure
  - Deprecated Interface for Free Energy Evaluation, 164
- energy\_of\_structure\_pt
  - Deprecated Interface for Free Energy Evaluation, 166
- energy\_set
  - Fine-tuning of the Implemented Models, 214
- exp\_E\_ExtLoop
  - Deprecated Interface for Free Energy Evaluation, 173
- exp\_E\_Hairpin
  - Hairpin Loops, 450
- exp\_E\_IntLoop
  - Deprecated Interface for Free Energy Evaluation, 176
- exp\_E\_Stem
  - Deprecated Interface for Free Energy Evaluation, 173
- exp\_f
  - vrna\_sc\_s, 277

- expand\_Full
  - Deprecated Interface for Secondary Structure Utilities, [704](#)
- expand\_Shapiro
  - Deprecated Interface for Secondary Structure Utilities, [704](#)
- Experimental Structure Probing Data, [417](#)
- expHairpinEnergy
  - part\_func.h, [831](#)
- expLoopEnergy
  - part\_func.h, [831](#)
- export\_ali\_bppm
  - Deprecated Interface for Global Partition Function Computation, [688](#)
- export\_bppm
  - Deprecated Interface for Global Partition Function Computation, [679](#)
- export\_circfold\_arrays
  - Deprecated Interface for Global MFE Prediction, [668](#)
- export\_circfold\_arrays\_par
  - Deprecated Interface for Global MFE Prediction, [668](#)
- export\_co\_bppm
  - Deprecated Interface for Global Partition Function Computation, [684](#)
- export\_cofold\_arrays
  - Deprecated Interface for Global MFE Prediction, [662](#)
- export\_cofold\_arrays\_gq
  - Deprecated Interface for Global MFE Prediction, [661](#)
- export\_fold\_arrays
  - Deprecated Interface for Global MFE Prediction, [667](#)
- export\_fold\_arrays\_par
  - Deprecated Interface for Global MFE Prediction, [667](#)
- Extending the Folding Grammar with Additional Domains, [231](#)
- Exterior Loops, [443](#)
  - vrna\_E\_ext\_loop, [445](#)
  - vrna\_E\_ext\_stem, [444](#)
  - vrna\_exp\_E\_ext\_stem, [445](#)
  - vrna\_mx\_pf\_aux\_el\_t, [444](#)
- f
  - vrna\_sc\_s, [276](#)
- filecopy
  - basic.h, [821](#)
- FILENAME\_ID\_LENGTH
  - (Nucleic Acid Sequence) String Utilities, [483](#)
- FILENAME\_MAX\_LENGTH
  - (Nucleic Acid Sequence) String Utilities, [483](#)
- Files and I/O, [526](#)
  - vrna\_file\_exists, [529](#)
  - vrna\_filename\_sanitize, [527](#)
  - vrna\_read\_line, [527](#)
- final\_cost
  - Inverse Folding (Design), [384](#)
- find\_saddle
  - Deprecated Interface for (Re-)folding Paths, Saddle Points, and Energy Barriers, [717](#)
- Fine-tuning of the Implemented Models, [180](#)
  - backtrack\_type, [215](#)
  - dangles, [213](#)
  - do\_backtrack, [214](#)
  - energy\_set, [214](#)
  - max\_bp\_span, [215](#)
  - noLonelyPairs, [214](#)
  - nonstandards, [215](#)
  - pf\_scale, [213](#)
  - set\_model\_details, [212](#)
  - temperature, [213](#)
  - tetra\_loop, [214](#)
  - vrna\_md\_copy, [193](#)
  - vrna\_md\_defaults\_backtrack, [204](#)
  - vrna\_md\_defaults\_backtrack\_get, [204](#)
  - vrna\_md\_defaults\_backtrack\_type, [205](#)
  - vrna\_md\_defaults\_backtrack\_type\_get, [205](#)
  - vrna\_md\_defaults\_betaScale, [196](#)
  - vrna\_md\_defaults\_betaScale\_get, [196](#)
  - vrna\_md\_defaults\_circ, [201](#)
  - vrna\_md\_defaults\_circ\_get, [201](#)
  - vrna\_md\_defaults\_compute\_bpp, [205](#)
  - vrna\_md\_defaults\_compute\_bpp\_get, [206](#)
  - vrna\_md\_defaults\_cv\_fact, [210](#)
  - vrna\_md\_defaults\_cv\_fact\_get, [210](#)
  - vrna\_md\_defaults\_dangles, [196](#)
  - vrna\_md\_defaults\_dangles\_get, [197](#)
  - vrna\_md\_defaults\_energy\_set, [203](#)
  - vrna\_md\_defaults\_energy\_set\_get, [204](#)
  - vrna\_md\_defaults\_gquad, [202](#)
  - vrna\_md\_defaults\_gquad\_get, [202](#)
  - vrna\_md\_defaults\_logML, [200](#)
  - vrna\_md\_defaults\_logML\_get, [201](#)
  - vrna\_md\_defaults\_max\_bp\_span, [206](#)
  - vrna\_md\_defaults\_max\_bp\_span\_get, [207](#)
  - vrna\_md\_defaults\_min\_loop\_size, [207](#)
  - vrna\_md\_defaults\_min\_loop\_size\_get, [207](#)
  - vrna\_md\_defaults\_nc\_fact, [211](#)
  - vrna\_md\_defaults\_nc\_fact\_get, [211](#)
  - vrna\_md\_defaults\_noGU, [199](#)
  - vrna\_md\_defaults\_noGU\_get, [199](#)
  - vrna\_md\_defaults\_noGUclosure, [199](#)
  - vrna\_md\_defaults\_noGUclosure\_get, [200](#)
  - vrna\_md\_defaults\_noLP, [198](#)
  - vrna\_md\_defaults\_noLP\_get, [198](#)
  - vrna\_md\_defaults\_oldAliEn, [208](#)
  - vrna\_md\_defaults\_oldAliEn\_get, [209](#)
  - vrna\_md\_defaults\_reset, [194](#)
  - vrna\_md\_defaults\_ribo, [209](#)
  - vrna\_md\_defaults\_ribo\_get, [210](#)
  - vrna\_md\_defaults\_sfact, [211](#)
  - vrna\_md\_defaults\_sfact\_get, [212](#)
  - vrna\_md\_defaults\_special\_hp, [197](#)
  - vrna\_md\_defaults\_special\_hp\_get, [198](#)

- vrna\_md\_defaults\_temperature, 195
- vrna\_md\_defaults\_temperature\_get, 195
- vrna\_md\_defaults\_uniq\_ML, 202
- vrna\_md\_defaults\_uniq\_ML\_get, 203
- vrna\_md\_defaults\_window\_size, 208
- vrna\_md\_defaults\_window\_size\_get, 208
- vrna\_md\_option\_string, 194
- vrna\_md\_set\_default, 193
- vrna\_md\_update, 193
- VRNA\_MODEL\_DEFAULT\_ALI\_CV\_FACT, 192
- VRNA\_MODEL\_DEFAULT\_ALI\_NC\_FACT, 192
- VRNA\_MODEL\_DEFAULT\_ALI\_OLD\_EN, 192
- VRNA\_MODEL\_DEFAULT\_ALI\_RIBO, 192
- VRNA\_MODEL\_DEFAULT\_BACKTRACK, 190
- VRNA\_MODEL\_DEFAULT\_BACKTRACK\_TYPE, 190
- VRNA\_MODEL\_DEFAULT\_BETA\_SCALE, 188
- VRNA\_MODEL\_DEFAULT\_CIRC, 189
- VRNA\_MODEL\_DEFAULT\_COMPUTE\_BPP, 191
- VRNA\_MODEL\_DEFAULT\_DANGLES, 188
- VRNA\_MODEL\_DEFAULT\_ENERGY\_SET, 190
- VRNA\_MODEL\_DEFAULT\_GQUAD, 189
- VRNA\_MODEL\_DEFAULT\_LOG\_ML, 191
- VRNA\_MODEL\_DEFAULT\_MAX\_BP\_SPAN, 191
- VRNA\_MODEL\_DEFAULT\_NO\_GU, 189
- VRNA\_MODEL\_DEFAULT\_NO\_GU\_CLOSURE, 189
- VRNA\_MODEL\_DEFAULT\_NO\_LP, 188
- VRNA\_MODEL\_DEFAULT\_PF\_SCALE, 187
- VRNA\_MODEL\_DEFAULT\_SPECIAL\_HP, 188
- VRNA\_MODEL\_DEFAULT\_TEMPERATURE, 187
- VRNA\_MODEL\_DEFAULT\_UNIQ\_ML, 190
- VRNA\_MODEL\_DEFAULT\_WINDOW\_SIZE, 191
- fold
  - Deprecated Interface for Global MFE Prediction, 665
- fold\_par
  - Deprecated Interface for Global MFE Prediction, 664
- fold\_vars.h
  - base\_pair, 782
  - cut\_point, 781
  - iindx, 782
  - james\_rule, 781
  - logML, 781
  - pr, 782
  - RibosumFile, 781
- Folding Paths that start at a single Secondary Structure, 412
  - vrna\_path, 414
  - VRNA\_PATH\_DEFAULT, 413
  - vrna\_path\_gradient, 415
  - VRNA\_PATH\_NO\_TRANSITION\_OUTPUT, 413
  - VRNA\_PATH\_RANDOM, 413
  - vrna\_path\_random, 415
  - VRNA\_PATH\_STEEPEST\_DESCENT, 412
- FORBIDDEN
  - constants.h, 824
- Free Energy Evaluation, 131
  - vrna\_eval\_circ\_consensus\_structure, 147
  - vrna\_eval\_circ\_consensus\_structure\_v, 150
  - vrna\_eval\_circ\_gquad\_consensus\_structure, 148
  - vrna\_eval\_circ\_gquad\_consensus\_structure\_v, 152
  - vrna\_eval\_circ\_gquad\_structure, 141
  - vrna\_eval\_circ\_gquad\_structure\_v, 145
  - vrna\_eval\_circ\_structure, 140
  - vrna\_eval\_circ\_structure\_v, 144
  - vrna\_eval\_consensus\_structure\_pt\_simple, 155
  - vrna\_eval\_consensus\_structure\_simple, 146
  - vrna\_eval\_consensus\_structure\_simple\_v, 150
  - vrna\_eval\_consensus\_structure\_simple\_verbose, 149
  - vrna\_eval\_covar\_structure, 135
  - vrna\_eval\_gquad\_consensus\_structure, 147
  - vrna\_eval\_gquad\_consensus\_structure\_v, 151
  - vrna\_eval\_gquad\_structure, 141
  - vrna\_eval\_gquad\_structure\_v, 144
  - vrna\_eval\_structure, 135
  - vrna\_eval\_structure\_pt, 137
  - vrna\_eval\_structure\_pt\_simple, 153
  - vrna\_eval\_structure\_pt\_simple\_v, 154
  - vrna\_eval\_structure\_pt\_simple\_verbose, 154
  - vrna\_eval\_structure\_pt\_v, 139
  - vrna\_eval\_structure\_pt\_verbose, 138
  - vrna\_eval\_structure\_simple, 140
  - vrna\_eval\_structure\_simple\_v, 143
  - vrna\_eval\_structure\_simple\_verbose, 142
  - vrna\_eval\_structure\_v, 137
  - vrna\_eval\_structure\_verbose, 136
- free\_alifold\_arrays
  - Deprecated Interface for Global MFE Prediction, 670
- free\_alipf\_arrays
  - Deprecated Interface for Global Partition Function Computation, 688
- free\_arrays
  - Deprecated Interface for Global MFE Prediction, 666
- free\_auxdata
  - vrna\_fc\_s, 617
- free\_co\_arrays
  - Deprecated Interface for Global MFE Prediction, 660
- free\_co\_pf\_arrays
  - Deprecated Interface for Global Partition Function Computation, 684
- free\_data
  - vrna\_hc\_s, 264
- free\_path
  - Deprecated Interface for (Re-)folding Paths, Saddle Points, and Energy Barriers, 717
- free\_pf\_arrays
  - Deprecated Interface for Global Partition Function Computation, 678
- free\_profile

- profiledist.h, [852](#)
- free\_sequence\_arrays
  - Deprecated Interface for Multiple Sequence Alignment Utilities, [699](#)
- free\_tree
  - treedist.h, [865](#)
- G-Quadruplexes, [432](#)
  - backtrack\_GQuad\_IntLoop, [433](#)
  - backtrack\_GQuad\_IntLoop\_L, [434](#)
  - get\_gquad\_matrix, [432](#)
  - parse\_gquad, [433](#)
- GASCONST
  - constants.h, [824](#)
- Generate Soft Constraints from Data, [422](#)
  - progress\_callback, [424](#)
  - VRNA\_MINIMIZER\_CONJUGATE\_FR, [423](#)
  - VRNA\_MINIMIZER\_CONJUGATE\_PR, [423](#)
  - VRNA\_MINIMIZER\_STEEPEST\_DESCENT, [424](#)
  - VRNA\_MINIMIZER\_VECTOR\_BFGS, [424](#)
  - VRNA\_MINIMIZER\_VECTOR\_BFGS2, [424](#)
  - VRNA\_OBJECTIVE\_FUNCTION\_ABSOLUTE, [423](#)
  - VRNA\_OBJECTIVE\_FUNCTION\_QUADRATIC, [423](#)
  - vrna\_sc\_minimize\_perturbation, [425](#)
- get\_alipf\_arrays
  - Deprecated Interface for Global Partition Function Computation, [689](#)
- get\_boltzmann\_factor\_copy
  - Energy Parameters, [227](#)
- get\_boltzmann\_factors
  - Energy Parameters, [226](#)
- get\_boltzmann\_factors\_al
  - Energy Parameters, [228](#)
- get\_centroid\_struct\_gquad\_pr
  - part\_func.h, [830](#)
- get\_centroid\_struct\_pl
  - centroid.h, [738](#)
- get\_centroid\_struct\_pr
  - centroid.h, [738](#)
- get\_concentrations
  - concentrations.h, [744](#)
- get\_gquad\_matrix
  - G-Quadruplexes, [432](#)
- get\_input\_line
  - Utilities, [440](#)
- get\_line
  - basic.h, [818](#)
- get\_monomere\_mfes
  - Deprecated Interface for Global MFE Prediction, [663](#)
- get\_mpi
  - Deprecated Interface for Multiple Sequence Alignment Utilities, [698](#)
- get\_path
  - Deprecated Interface for (Re-)folding Paths, Saddle Points, and Energy Barriers, [718](#)
- get\_pf\_arrays
  - Deprecated Interface for Global Partition Function Computation, [679](#)
- get\_plist
  - part\_func\_co.h, [833](#)
- get\_scaled\_alipf\_parameters
  - Energy Parameters, [228](#)
- get\_scaled\_parameters
  - Energy Parameters, [229](#)
- get\_scaled\_pf\_parameters
  - Energy Parameters, [226](#)
- get\_TwoDfold\_variables
  - Computing MFE representatives of a Distance Based Partitioning, [362](#)
- get\_TwoDpfold\_variables
  - 2Dpfold.h, [727](#)
- give\_up
  - Inverse Folding (Design), [384](#)
- Global MFE Prediction, [290](#)
  - vrna\_alifold, [294](#)
  - vrna\_circalifold, [294](#)
  - vrna\_circfold, [293](#)
  - vrna\_cofold, [295](#)
  - vrna\_fold, [292](#)
  - vrna\_mfe, [291](#)
  - vrna\_mfe\_dimer, [292](#)
- Global Partition Function and Equilibrium Probabilities, [305](#)
  - vrna\_pf, [307](#)
  - vrna\_pf\_alifold, [310](#)
  - vrna\_pf\_circalifold, [310](#)
  - vrna\_pf\_circfold, [309](#)
  - vrna\_pf\_co\_fold, [312](#)
  - vrna\_pf\_dimer, [307](#)
  - vrna\_pf\_fold, [308](#)
  - vrna\_plist\_from\_probs, [311](#)
- gmIRNA
  - Plotting, [557](#)
- Hairpin Loops, [447](#)
  - E\_Hairpin, [449](#)
  - exp\_E\_Hairpin, [450](#)
  - vrna\_E\_ext\_hp\_loop, [448](#)
  - vrna\_E\_hp\_loop, [448](#)
  - vrna\_eval\_hp\_loop, [448](#)
  - vrna\_exp\_E\_hp\_loop, [451](#)
- HairpinE
  - Deprecated Interface for Global MFE Prediction, [669](#)
- hamming
  - strings.h, [874](#)
- hamming\_bound
  - strings.h, [875](#)
- Hard Constraints, [262](#)
  - vrna\_callback\_hc\_evaluate, [269](#)
  - VRNA\_CONSTRAINT\_DB, [265](#)
  - VRNA\_CONSTRAINT\_DB\_DEFAULT, [268](#)
  - VRNA\_CONSTRAINT\_DB\_DOT, [266](#)
  - VRNA\_CONSTRAINT\_DB\_ENFORCE\_BP, [265](#)
  - VRNA\_CONSTRAINT\_DB\_GQUAD, [267](#)

- VRNA\_CONSTRAINT\_DB\_INTERMOL, [267](#)
- VRNA\_CONSTRAINT\_DB\_INTRAMOL, [267](#)
- VRNA\_CONSTRAINT\_DB\_PIPE, [265](#)
- VRNA\_CONSTRAINT\_DB\_RND\_BRACK, [266](#)
- VRNA\_CONSTRAINT\_DB\_WUSS, [268](#)
- VRNA\_CONSTRAINT\_DB\_X, [266](#)
- vrna\_hc\_add\_bp, [271](#)
- vrna\_hc\_add\_bp\_nonspecific, [271](#)
- vrna\_hc\_add\_from\_db, [272](#)
- vrna\_hc\_add\_up, [270](#)
- vrna\_hc\_add\_up\_batch, [270](#)
- vrna\_hc\_free, [272](#)
- vrna\_hc\_init, [269](#)
- hard.h
  - constrain\_ptypes, [750](#)
  - print\_tty\_constraint, [749](#)
  - print\_tty\_constraint\_full, [750](#)
  - VRNA\_CONSTRAINT\_DB\_ANG\_BRACK, [748](#)
  - VRNA\_CONSTRAINT\_NO\_HEADER, [748](#)
  - vrna\_hc\_add\_data, [749](#)
  - VRNA\_HC\_DEFAULT, [749](#)
  - vrna\_hc\_type\_e, [748](#)
  - VRNA\_HC\_WINDOW, [749](#)
- Hash Tables, [634](#)
  - vrna\_callback\_ht\_compare\_entries, [636](#)
  - vrna\_callback\_ht\_free\_entry, [637](#)
  - vrna\_callback\_ht\_hash\_function, [636](#)
  - vrna\_hash\_table\_t, [636](#)
  - vrna\_ht\_clear, [640](#)
  - vrna\_ht\_collisions, [638](#)
  - vrna\_ht\_db\_comp, [641](#)
  - vrna\_ht\_db\_free\_entry, [642](#)
  - vrna\_ht\_db\_hash\_func, [642](#)
  - vrna\_ht\_free, [641](#)
  - vrna\_ht\_get, [639](#)
  - vrna\_ht\_init, [637](#)
  - vrna\_ht\_insert, [639](#)
  - vrna\_ht\_remove, [640](#)
  - vrna\_ht\_size, [638](#)
- Heaps, [644](#)
  - vrna\_callback\_heap\_cmp, [645](#)
  - vrna\_callback\_heap\_get\_pos, [645](#)
  - vrna\_callback\_heap\_set\_pos, [646](#)
  - vrna\_heap\_free, [647](#)
  - vrna\_heap\_init, [646](#)
  - vrna\_heap\_insert, [648](#)
  - vrna\_heap\_pop, [648](#)
  - vrna\_heap\_remove, [649](#)
  - vrna\_heap\_size, [648](#)
  - vrna\_heap\_t, [645](#)
  - vrna\_heap\_top, [649](#)
  - vrna\_heap\_update, [650](#)
- Helix List Representation of Secondary Structures, [510](#)
  - vrna\_hx\_from\_ptable, [510](#)
- id
  - vrna\_exp\_param\_s, [219](#)
- iindx
  - fold\_vars.h, [782](#)
- Incorporating Ligands Binding to Specific Sequence/Structure Motifs using Soft Constraints, [429](#)
  - vrna\_sc\_add\_hi\_motif, [429](#)
- INF
  - constants.h, [824](#)
- init\_co\_pf\_fold
  - Deprecated Interface for Global Partition Function Computation, [684](#)
- init\_pf\_fold
  - Deprecated Interface for Global Partition Function Computation, [681](#)
- init\_pf\_foldLP
  - LPfold.h, [798](#)
- init\_rand
  - basic.h, [820](#)
- initialize\_cofold
  - Deprecated Interface for Global MFE Prediction, [664](#)
- initialize\_fold
  - Deprecated Interface for Global MFE Prediction, [669](#)
- int\_urn
  - basic.h, [820](#)
- interact, [598](#)
- Internal Loops, [452](#)
  - vrna\_eval\_int\_loop, [452](#)
- inv\_verbose
  - Inverse Folding (Design), [385](#)
- Inverse Folding (Design), [383](#)
  - final\_cost, [384](#)
  - give\_up, [384](#)
  - inv\_verbose, [385](#)
  - inverse\_fold, [383](#)
  - inverse\_pf\_fold, [384](#)
- inverse\_fold
  - Inverse Folding (Design), [383](#)
- inverse\_pf\_fold
  - Inverse Folding (Design), [384](#)
- james\_rule
  - fold\_vars.h, [781](#)
- K0
  - constants.h, [824](#)
- last\_parameter\_file
  - Reading/Writing Energy Parameter Sets from/to File, [469](#)
- Layouts and Coordinates, [561](#)
  - vrna\_plot\_coords, [569](#)
  - vrna\_plot\_coords\_circular, [572](#)
  - vrna\_plot\_coords\_circular\_pt, [573](#)
  - vrna\_plot\_coords\_navigate, [574](#)
  - vrna\_plot\_coords\_navigate\_pt, [574](#)
  - vrna\_plot\_coords\_pt, [570](#)
  - vrna\_plot\_coords\_puzzler, [575](#)
  - vrna\_plot\_coords\_puzzler\_pt, [576](#)
  - vrna\_plot\_coords\_simple, [571](#)

- vrna\_plot\_coords\_simple\_pt, 571
- vrna\_plot\_coords\_turtle, 578
- vrna\_plot\_coords\_turtle\_pt, 579
- vrna\_plot\_layout, 564
- vrna\_plot\_layout\_circular, 566
- vrna\_plot\_layout\_free, 568
- vrna\_plot\_layout\_naview, 566
- vrna\_plot\_layout\_puzzler, 568
- vrna\_plot\_layout\_simple, 565
- vrna\_plot\_layout\_t, 564
- vrna\_plot\_layout\_turtle, 567
- vrna\_plot\_options\_puzzler, 577
- vrna\_plot\_options\_puzzler\_free, 577
- VRNA\_PLOT\_TYPE\_CIRCULAR, 563
- VRNA\_PLOT\_TYPE\_NAVIEW, 563
- VRNA\_PLOT\_TYPE\_SIMPLE, 563
- Lfold
  - Deprecated Interface for Local (Sliding Window) MFE Prediction, 671
- Lfoldz
  - Deprecated Interface for Local (Sliding Window) MFE Prediction, 671
- ligand.h
  - vrna\_sc\_motif\_t, 752
- Ligands Binding to RNA Structures, 427
- Ligands Binding to Unstructured Domains, 428
- LIST, 719
- Local (sliding window) MFE Prediction, 297
  - vrna\_Lfold, 300
  - vrna\_Lfoldz, 301
  - vrna\_mfe\_window, 299
  - vrna\_mfe\_window\_callback, 298
  - vrna\_mfe\_window\_zscore, 299
- Local (sliding window) Partition Function and Equilibrium Probabilities, 313
  - vrna\_pfl\_fold, 318
  - vrna\_pfl\_fold\_cb, 319
  - vrna\_pfl\_fold\_up, 319
  - vrna\_pfl\_fold\_up\_cb, 320
  - vrna\_probs\_window, 317
  - VRNA\_PROBS\_WINDOW\_BPP, 314
  - vrna\_probs\_window\_callback, 316
  - VRNA\_PROBS\_WINDOW\_PF, 315
  - VRNA\_PROBS\_WINDOW\_STACKP, 315
  - VRNA\_PROBS\_WINDOW\_UP, 314
  - VRNA\_PROBS\_WINDOW\_UP\_SPLIT, 315
- logML
  - fold\_vars.h, 781
- loop\_energy
  - Deprecated Interface for Free Energy Evaluation, 169
- LoopEnergy
  - Deprecated Interface for Global MFE Prediction, 668
- LPfold.h
  - init\_pf\_foldLP, 798
- LST\_BUCKET, 720
- Make\_bp\_profile
  - profiledist.h, 852
- Make\_bp\_profile\_bppm
  - profiledist.h, 852
- make\_pair\_table
  - Deprecated Interface for Secondary Structure Utilities, 707
- make\_pair\_table\_snoop
  - Deprecated Interface for Secondary Structure Utilities, 708
- make\_referenceBP\_array
  - Deprecated Interface for Secondary Structure Utilities, 709
- Make\_swString
  - stringdist.h, 860
- make\_tree
  - treedist.h, 864
- max\_bp\_span
  - Fine-tuning of the Implemented Models, 215
- MAXLOOP
  - constants.h, 825
- MEA
  - Compute the Structure with Maximum Expected Accuracy (MEA), 350
- mean\_bp\_dist
  - part\_func.h, 831
- mean\_bp\_distance
  - Deprecated Interface for Global Partition Function Computation, 680
- mean\_bp\_distance\_pr
  - Deprecated Interface for Global Partition Function Computation, 681
- Messages, 602
  - vrna\_message\_error, 602
  - vrna\_message\_info, 604
  - vrna\_message\_input\_seq, 606
  - vrna\_message\_input\_seq\_simple, 605
  - vrna\_message\_verror, 603
  - vrna\_message\_vinfo, 605
  - vrna\_message\_vwarning, 604
  - vrna\_message\_warning, 603
- min\_loop\_size
  - vrna\_md\_s, 187
- Minimum Free Energy (MFE) Algorithms, 287
- mm.h
  - vrna\_maximum\_matching, 802
  - vrna\_maximum\_matching\_simple, 802
- Multibranch Loops, 454
  - vrna\_E\_mb\_loop\_stack, 455
  - vrna\_mx\_pf\_aux\_ml\_t, 455
- Multiple Sequence Alignment Utilities, 517
  - vrna\_aln\_consensus\_mis, 524
  - vrna\_aln\_consensus\_sequence, 524
  - vrna\_aln\_conservation\_col, 523
  - vrna\_aln\_conservation\_struct, 522
  - vrna\_aln\_copy, 522
  - vrna\_aln\_free, 521
  - vrna\_aln\_mpi, 519
  - vrna\_aln\_pinfo, 520



- vrna\_aln\_slice, [520](#)
- vrna\_aln\_toRNA, [521](#)
- vrna\_aln\_uppercase, [521](#)
- VRNA\_MEASURE\_SHANNON\_ENTROPY, [519](#)
- Multiple Sequence Alignments, [538](#)
  - VRNA\_FILE\_FORMAT\_MSA\_APPEND, [541](#)
  - VRNA\_FILE\_FORMAT\_MSA\_CLUSTAL, [539](#)
  - VRNA\_FILE\_FORMAT\_MSA\_DEFAULT, [540](#)
  - VRNA\_FILE\_FORMAT\_MSA\_FASTA, [539](#)
  - VRNA\_FILE\_FORMAT\_MSA\_MAF, [540](#)
  - VRNA\_FILE\_FORMAT\_MSA\_MIS, [540](#)
  - VRNA\_FILE\_FORMAT\_MSA\_NOCHECK, [541](#)
  - VRNA\_FILE\_FORMAT\_MSA\_QUIET, [542](#)
  - VRNA\_FILE\_FORMAT\_MSA\_SILENT, [542](#)
  - VRNA\_FILE\_FORMAT\_MSA\_STOCKHOLM, [539](#)
  - VRNA\_FILE\_FORMAT\_MSA\_UNKNOWN, [541](#)
  - vrna\_file\_msa\_detect\_format, [545](#)
  - vrna\_file\_msa\_read, [542](#)
  - vrna\_file\_msa\_read\_record, [544](#)
  - vrna\_file\_msa\_write, [546](#)
- n\_seq
  - vrna\_fc\_s, [619](#)
- naview\_xy\_coordinates
  - Deprecated Interface for Plotting Utilities, [715](#)
- NBPAIRS
  - constants.h, [825](#)
- nc\_fact
  - alifold.h, [733](#)
- Neighborhood Relation and Move Sets for Secondary Structures, [386](#)
  - vrna\_callback\_move\_update, [392](#)
  - vrna\_loopidx\_update, [395](#)
  - vrna\_move\_apply, [393](#)
  - vrna\_move\_compare, [394](#)
  - vrna\_move\_init, [392](#)
  - vrna\_move\_is\_insertion, [394](#)
  - vrna\_move\_is\_removal, [393](#)
  - vrna\_move\_is\_shift, [394](#)
  - vrna\_move\_list\_free, [393](#)
  - vrna\_move\_neighbor\_diff, [398](#)
  - vrna\_move\_neighbor\_diff\_cb, [397](#)
  - VRNA\_MOVESET\_DEFAULT, [391](#)
  - VRNA\_MOVESET\_DELETION, [390](#)
  - VRNA\_MOVESET\_INSERTION, [390](#)
  - VRNA\_MOVESET\_NO\_LP, [390](#)
  - VRNA\_MOVESET\_SHIFT, [390](#)
  - VRNA\_NEIGHBOR\_CHANGE, [391](#)
  - VRNA\_NEIGHBOR\_INVALID, [391](#)
  - VRNA\_NEIGHBOR\_NEW, [391](#)
  - vrna\_neighbors, [396](#)
  - vrna\_neighbors\_successive, [396](#)
- node, [599](#)
- noLonelyPairs
  - Fine-tuning of the Implemented Models, [214](#)
- nonstandards
  - Fine-tuning of the Implemented Models, [215](#)
- nrrror
  - basic.h, [819](#)
- Nucleic Acid Sequences and Structures, [530](#)
  - read\_record, [536](#)
  - VRNA\_CONSTRAINT\_MULTILINE, [531](#)
  - vrna\_extract\_record\_rest\_constraint, [536](#)
  - vrna\_extract\_record\_rest\_structure, [535](#)
  - vrna\_file\_bpseq, [532](#)
  - vrna\_file\_connect, [532](#)
  - vrna\_file\_fasta\_read\_record, [533](#)
  - vrna\_file\_helixlist, [531](#)
  - vrna\_file\_json, [533](#)
  - vrna\_file\_SHAPE\_read, [535](#)
  - VRNA\_OPTION\_MULTILINE, [531](#)
- pack\_structure
  - Deprecated Interface for Secondary Structure Utilities, [706](#)
- PAIR
  - (Abstract) Data Structures, [600](#)
- Pair List Representation of Secondary Structures, [507](#)
  - vrna\_plist, [508](#)
- Pair Table Representation of Secondary Structures, [503](#)
  - vrna\_pt\_pk\_get, [504](#)
  - vrna\_pt\_pk\_remove, [505](#)
  - vrna\_pt\_snoop\_get, [505](#)
  - vrna\_ptable, [503](#)
  - vrna\_ptable\_copy, [505](#)
  - vrna\_ptable\_from\_string, [504](#)
- pair\_info
  - Deprecated Interface for Multiple Sequence Alignment Utilities, [697](#)
- paramT
  - Energy Parameters, [219](#)
- parenthesis\_structure
  - Deprecated Interface for Secondary Structure Utilities, [710](#)
- parenthesis\_zuker
  - Deprecated Interface for Secondary Structure Utilities, [710](#)
- parse\_gquad
  - G-Quadruplexes, [433](#)
- parse\_structure
  - Deprecated Interface for Secondary Structure Utilities, [706](#)
- part\_func.h
  - centroid, [830](#)
  - expHairpinEnergy, [831](#)
  - expLoopEnergy, [831](#)
  - get\_centroid\_struct\_gquad\_pr, [830](#)
  - mean\_bp\_dist, [831](#)
- part\_func\_co.h
  - get\_plist, [833](#)
- Partition Function and Equilibrium Properties, [288](#)
  - vrna\_pf\_float\_precision, [289](#)
- Partition Function for Two Hybridized Sequences, [456](#)
  - vrna\_pf\_co\_fold, [457](#)
  - vrna\_pf\_dimer\_concentrations, [458](#)
- Partition Function for two Hybridized Sequences as a Stepwise Process, [459](#)
  - pf\_interact, [460](#)

- pf\_unstru, [459](#)
- path\_t
  - Deprecated Interface for (Re-)folding Paths, Saddle Points, and Energy Barriers, [716](#)
- pbacktrack
  - Deprecated Interface for Stochastic Backtracking, [694](#)
- pbacktrack\_circ
  - Deprecated Interface for Stochastic Backtracking, [695](#)
- pf\_circ\_fold
  - Deprecated Interface for Global Partition Function Computation, [677](#)
- pf\_fold
  - Deprecated Interface for Global Partition Function Computation, [676](#)
- pf\_fold\_par
  - Deprecated Interface for Global Partition Function Computation, [675](#)
- pf\_interact
  - Partition Function for two Hybridized Sequences as a Stepwise Process, [460](#)
- pf\_paramT
  - Energy Parameters, [219](#)
- pf\_scale
  - Fine-tuning of the Implemented Models, [213](#)
- pf\_unstru
  - Partition Function for two Hybridized Sequences as a Stepwise Process, [459](#)
- pfl\_fold
  - Deprecated Interface for Local (Sliding Window) Partition Function Computation, [692](#)
- plist
  - (Abstract) Data Structures, [600](#)
- Plotting, [553](#)
  - gmlRNA, [557](#)
  - PS\_dot\_plot, [556](#)
  - PS\_dot\_plot\_list, [555](#)
  - PS\_rna\_plot, [559](#)
  - PS\_rna\_plot\_a, [560](#)
  - PS\_rna\_plot\_a\_gquad, [560](#)
  - ssv\_rna\_plot, [558](#)
  - svg\_rna\_plot, [558](#)
  - vrna\_file\_PS\_rnaplot, [556](#)
  - vrna\_file\_PS\_rnaplot\_a, [557](#)
  - xrna\_plot, [559](#)
- Postorder\_list, [720](#)
- pr
  - fold\_vars.h, [782](#)
- Predicting various thermodynamic properties, [371](#)
  - vrna\_ensemble\_defect, [375](#)
  - vrna\_heat\_capacity, [378](#)
  - vrna\_heat\_capacity\_callback, [373](#)
  - vrna\_heat\_capacity\_cb, [379](#)
  - vrna\_heat\_capacity\_simple, [380](#)
  - vrna\_heat\_capacity\_t, [374](#)
  - vrna\_mean\_bp\_distance, [374](#)
  - vrna\_mean\_bp\_distance\_pr, [374](#)
  - vrna\_pf\_dimer\_probs, [377](#)
  - vrna\_positional\_entropy, [376](#)
  - vrna\_pr\_energy, [378](#)
  - vrna\_pr\_structure, [378](#)
  - vrna\_stack\_prob, [376](#)
- print\_tty\_constraint
  - hard.h, [749](#)
- print\_tty\_constraint\_full
  - hard.h, [750](#)
- print\_tty\_input\_seq
  - basic.h, [818](#)
- print\_tty\_input\_seq\_str
  - basic.h, [819](#)
- prod\_cb
  - vrna\_unstructured\_domain\_s, [235](#)
- profile\_edit\_distance
  - profiledist.h, [852](#)
- profiledist.h
  - free\_profile, [852](#)
  - Make\_bp\_profile, [852](#)
  - Make\_bp\_profile\_bppm, [852](#)
  - profile\_edit\_distance, [852](#)
- progress\_callback
  - Generate Soft Constraints from Data, [424](#)
- PS\_color\_aln
  - Deprecated Interface for Plotting Utilities, [712](#)
- PS\_dot\_plot
  - Plotting, [556](#)
- PS\_dot\_plot\_list
  - Plotting, [555](#)
- PS\_rna\_plot
  - Plotting, [559](#)
- PS\_rna\_plot\_a
  - Plotting, [560](#)
- PS\_rna\_plot\_a\_gquad
  - Plotting, [560](#)
- pscore
  - vrna\_fc\_s, [620](#)
- pscore\_local
  - vrna\_fc\_s, [620](#)
- pscore\_pf\_compat
  - vrna\_fc\_s, [621](#)
- pctype
  - vrna\_fc\_s, [617](#)
- pctype\_pf\_compat
  - vrna\_fc\_s, [618](#)
- pu\_contrib, [598](#)
- pu\_out, [598](#)
- putoutpU\_prob
  - Deprecated Interface for Local (Sliding Window) Partition Function Computation, [692](#)
- putoutpU\_prob\_bin
  - Deprecated Interface for Local (Sliding Window) Partition Function Computation, [693](#)
- Random Structure Samples from the Ensemble, [331](#)
  - vrna\_boltzmann\_sampling\_callback, [333](#)
  - vrna\_pbacktrack, [341](#)
  - vrna\_pbacktrack5, [334](#)



- vrna\_pbacktrack5\_cb, 336
- vrna\_pbacktrack5\_num, 335
- vrna\_pbacktrack5\_resume, 337
- vrna\_pbacktrack5\_resume\_cb, 339
- vrna\_pbacktrack\_cb, 343
- VRNA\_PBACKTRACK\_DEFAULT, 332
- vrna\_pbacktrack\_mem\_free, 347
- vrna\_pbacktrack\_mem\_t, 333
- VRNA\_PBACKTRACK\_NON\_REDUNDANT, 332
- vrna\_pbacktrack\_num, 341
- vrna\_pbacktrack\_resume, 344
- vrna\_pbacktrack\_resume\_cb, 345
- random\_string
  - strings.h, 874
- read\_parameter\_file
  - Reading/Writing Energy Parameter Sets from/to File, 469
- read\_record
  - Nucleic Acid Sequences and Structures, 536
- Reading/Writing Energy Parameter Sets from/to File, 462
  - last\_parameter\_file, 469
  - read\_parameter\_file, 469
  - VRNA\_PARAMETER\_FORMAT\_DEFAULT, 463
  - vrna\_params\_load, 463
  - vrna\_params\_load\_defaults, 465
  - vrna\_params\_load\_DNA\_Mathews1999, 468
  - vrna\_params\_load\_DNA\_Mathews2004, 468
  - vrna\_params\_load\_from\_string, 464
  - vrna\_params\_load\_RNA\_Andronescu2007, 466
  - vrna\_params\_load\_RNA\_Langdon2018, 467
  - vrna\_params\_load\_RNA\_misc\_special\_hairpins, 467
  - vrna\_params\_load\_RNA\_Turner1999, 466
  - vrna\_params\_load\_RNA\_Turner2004, 465
  - vrna\_params\_save, 464
  - write\_parameter\_file, 470
- RibosumFile
  - fold\_vars.h, 781
- RNA-RNA Interaction, 354
- rna\_plot\_type
  - Deprecated Interface for Plotting Utilities, 715
- S
  - vrna\_fc\_s, 619
- S3
  - vrna\_fc\_s, 620
- S5
  - vrna\_fc\_s, 620
- S\_cons
  - vrna\_fc\_s, 619
- sc
  - vrna\_fc\_s, 618
- scale\_parameters
  - Energy Parameters, 228
- scs
  - vrna\_fc\_s, 621
- Search Algorithms, 584
  - vrna\_search\_BM\_BCT, 587
  - vrna\_search\_BM\_BCT\_num, 586
  - vrna\_search\_BMH, 585
  - vrna\_search\_BMH\_num, 584
- Secondary Structure Utilities, 491
  - vrna\_bp\_distance, 492
  - vrna\_db\_from\_bp\_stack, 493
  - vrna\_refBPcnt\_matrix, 492
  - vrna\_refBPdist\_matrix, 493
- sect
  - (Abstract) Data Structures, 600
- sequence
  - vrna\_fc\_s, 617
- sequence\_encoding
  - vrna\_fc\_s, 617
- sequences
  - vrna\_fc\_s, 618
- set\_model\_details
  - Fine-tuning of the Implemented Models, 212
- SHAPE Reactivity Data, 418
  - vrna\_sc\_add\_SHAPE\_deigan, 418
  - vrna\_sc\_add\_SHAPE\_deigan\_ali, 419
  - vrna\_sc\_add\_SHAPE\_zarringhalam, 420
  - vrna\_sc\_SHAPE\_to\_pr, 421
- SHAPE.h
  - vrna\_sc\_SHAPE\_parse\_method, 753
- simple\_circplot\_coordinates
  - Deprecated Interface for Plotting Utilities, 714
- simple\_xy\_coordinates
  - Deprecated Interface for Plotting Utilities, 713
- snoopT, 599
- Soft Constraints, 274
  - vrna\_callback\_sc\_backtrack, 279
  - vrna\_callback\_sc\_energy, 277
  - vrna\_callback\_sc\_exp\_energy, 278
  - vrna\_sc\_add\_bp, 281
  - vrna\_sc\_add\_bt, 284
  - vrna\_sc\_add\_data, 283
  - vrna\_sc\_add\_exp\_f, 285
  - vrna\_sc\_add\_f, 284
  - vrna\_sc\_add\_up, 282
  - vrna\_sc\_free, 283
  - vrna\_sc\_init, 279
  - vrna\_sc\_remove, 282
  - vrna\_sc\_set\_bp, 280
  - vrna\_sc\_set\_up, 281
- soft.h
  - VRNA\_SC\_DEFAULT, 756
  - vrna\_sc\_type\_e, 755
  - VRNA\_SC\_WINDOW, 756
- SOLUTION
  - subopt.h, 863
- space
  - basic.h, 819
- ssv\_rna\_plot
  - Plotting, 558
- st\_back
  - Deprecated Interface for Stochastic Backtracking, 695

- stackProb
  - Deprecated Interface for Global Partition Function Computation, [681](#)
- stat\_cb
  - vrna\_fc\_s, [616](#)
- Stochastic Backtracking of Structures from Distance Based Partitioning, [368](#)
  - vrna\_pbacktrack5\_TwoD, [369](#)
  - vrna\_pbacktrack\_TwoD, [368](#)
- str\_DNA2RNA
  - strings.h, [874](#)
- str\_uppercase
  - strings.h, [874](#)
- string\_edit\_distance
  - stringdist.h, [860](#)
- stringdist.h
  - Make\_swString, [860](#)
  - string\_edit\_distance, [860](#)
- strings.h
  - hamming, [874](#)
  - hamming\_bound, [875](#)
  - random\_string, [874](#)
  - str\_DNA2RNA, [874](#)
  - str\_uppercase, [874](#)
- structure
  - vrna\_ht\_entry\_db\_t, [635](#)
- Structured Domains, [244](#)
- subopt
  - Suboptimal Structures within an Energy Band around the MFE, [329](#)
- subopt.h
  - SOLUTION, [863](#)
- subopt\_circ
  - Suboptimal Structures within an Energy Band around the MFE, [329](#)
- Suboptimal Structures sensu Stiegler et al. 1984 / Zuker et al. 1989, [324](#)
  - vrna\_subopt\_zuker, [324](#)
  - zukersubopt, [325](#)
  - zukersubopt\_par, [325](#)
- Suboptimal Structures within an Energy Band around the MFE, [326](#)
  - subopt, [329](#)
  - subopt\_circ, [329](#)
  - vrna\_subopt, [327](#)
  - vrna\_subopt\_callback, [326](#)
  - vrna\_subopt\_cb, [328](#)
- Suboptimals and Representative Structures, [323](#)
- svg\_rna\_plot
  - Plotting, [558](#)
- swString, [720](#)
- temperature
  - Fine-tuning of the Implemented Models, [213](#)
- tetra\_loop
  - Fine-tuning of the Implemented Models, [214](#)
- The Dynamic Programming Matrices, [629](#)
  - VRNA\_MX\_2DFOLD, [631](#)
  - vrna\_mx\_add, [631](#)
  - VRNA\_MX\_DEFAULT, [631](#)
  - vrna\_mx\_mfe\_free, [632](#)
  - vrna\_mx\_pf\_free, [632](#)
  - vrna\_mx\_type\_e, [631](#)
  - VRNA\_MX\_WINDOW, [631](#)
- The Fold Compound, [612](#)
  - vrna\_callback\_free\_auxdata, [623](#)
  - vrna\_callback\_recursion\_status, [624](#)
  - VRNA\_FC\_TYPE\_COMPARATIVE, [624](#)
  - vrna\_fc\_type\_e, [624](#)
  - VRNA\_FC\_TYPE\_SINGLE, [624](#)
  - vrna\_fold\_compound, [625](#)
  - vrna\_fold\_compound\_add\_auxdata, [627](#)
  - vrna\_fold\_compound\_add\_callback, [628](#)
  - vrna\_fold\_compound\_comparative, [626](#)
  - vrna\_fold\_compound\_free, [627](#)
  - VRNA\_OPTION\_EVAL\_ONLY, [623](#)
  - VRNA\_OPTION\_MFE, [622](#)
  - VRNA\_OPTION\_PF, [622](#)
  - VRNA\_STATUS\_MFE\_POST, [621](#)
  - VRNA\_STATUS\_MFE\_PRE, [621](#)
  - VRNA\_STATUS\_PF\_POST, [622](#)
  - VRNA\_STATUS\_PF\_PRE, [622](#)
- The RNA Folding Grammar, [178](#)
- The RNA Secondary Structure Landscape, [286](#)
- time\_stamp
  - basic.h, [821](#)
- Tree, [721](#)
- Tree Representation of Secondary Structures, [512](#)
  - vrna\_db\_to\_tree\_string, [514](#)
  - VRNA\_STRUCTURE\_TREE\_EXPANDED, [514](#)
  - VRNA\_STRUCTURE\_TREE\_HIT, [512](#)
  - VRNA\_STRUCTURE\_TREE\_SHAPIRO, [513](#)
  - VRNA\_STRUCTURE\_TREE\_SHAPIRO\_EXT, [513](#)
  - VRNA\_STRUCTURE\_TREE\_SHAPIRO\_SHORT, [513](#)
  - VRNA\_STRUCTURE\_TREE\_SHAPIRO\_WEIGHT, [513](#)
  - vrna\_tree\_string\_to\_db, [515](#)
  - vrna\_tree\_string\_unweight, [515](#)
- tree\_edit\_distance
  - treedist.h, [865](#)
- treedist.h
  - free\_tree, [865](#)
  - make\_tree, [864](#)
  - tree\_edit\_distance, [865](#)
- TURN
  - constants.h, [825](#)
- TwoDfold\_backtrack\_f5
  - Computing MFE representatives of a Distance Based Partitioning, [364](#)
- TwoDfold\_vars, [358](#)
  - Computing MFE representatives of a Distance Based Partitioning, [360](#)
- TwoDfoldList
  - Computing MFE representatives of a Distance Based Partitioning, [363](#)
- TwoDpfold\_pbacktrack

- 2Dpfold.h, [729](#)
- TwoDpfold\_pbacktrack5
  - 2Dpfold.h, [730](#)
- TwoDpfold\_vars, [721](#)
- TwoDpfoldList
  - 2Dpfold.h, [728](#)
- type
  - vrna\_fc\_s, [616](#)
  - vrna\_path\_s, [402](#)
- unexpand\_aligned\_F
  - Deprecated Interface for Secondary Structure Utilities, [706](#)
- unexpand\_Full
  - Deprecated Interface for Secondary Structure Utilities, [705](#)
- Unit Conversion, [607](#)
  - vrna\_convert\_dcal\_to\_kcal, [610](#)
  - vrna\_convert\_energy, [609](#)
  - vrna\_convert\_kcal\_to\_dcal, [610](#)
  - vrna\_convert\_temperature, [609](#)
  - VRNA\_UNIT\_CAL, [608](#)
  - VRNA\_UNIT\_CAL\_IT, [608](#)
  - VRNA\_UNIT\_DACAL, [608](#)
  - VRNA\_UNIT\_DACAL\_IT, [608](#)
  - VRNA\_UNIT\_DEG\_C, [609](#)
  - VRNA\_UNIT\_DEG\_DE, [609](#)
  - VRNA\_UNIT\_DEG\_F, [609](#)
  - VRNA\_UNIT\_DEG\_N, [609](#)
  - VRNA\_UNIT\_DEG\_R, [609](#)
  - VRNA\_UNIT\_DEG\_RE, [609](#)
  - VRNA\_UNIT\_DEG\_RO, [609](#)
  - vrna\_unit\_energy\_e, [608](#)
  - VRNA\_UNIT\_EV, [608](#)
  - VRNA\_UNIT\_G\_TNT, [608](#)
  - VRNA\_UNIT\_J, [608](#)
  - VRNA\_UNIT\_K, [609](#)
  - VRNA\_UNIT\_KCAL, [608](#)
  - VRNA\_UNIT\_KCAL\_IT, [608](#)
  - VRNA\_UNIT\_KG\_TNT, [608](#)
  - VRNA\_UNIT\_KJ, [608](#)
  - VRNA\_UNIT\_KWH, [608](#)
  - VRNA\_UNIT\_T\_TNT, [608](#)
  - vrna\_unit\_temperature\_e, [608](#)
  - VRNA\_UNIT\_WH, [608](#)
- unpack\_structure
  - Deprecated Interface for Secondary Structure Utilities, [707](#)
- Unstructured Domains, [232](#)
  - vrna\_callback\_ud\_energy, [235](#)
  - vrna\_callback\_ud\_exp\_energy, [236](#)
  - vrna\_callback\_ud\_exp\_production, [236](#)
  - vrna\_callback\_ud\_probs\_add, [237](#)
  - vrna\_callback\_ud\_probs\_get, [237](#)
  - vrna\_callback\_ud\_production, [236](#)
  - vrna\_ud\_add\_motif, [239](#)
  - vrna\_ud\_motifs\_centroid, [237](#)
  - vrna\_ud\_motifs\_MEA, [238](#)
  - vrna\_ud\_motifs\_MFE, [238](#)
  - vrna\_ud\_remove, [240](#)
  - vrna\_ud\_set\_data, [240](#)
  - vrna\_ud\_set\_exp\_prod\_rule\_cb, [242](#)
  - vrna\_ud\_set\_prod\_rule\_cb, [241](#)
- unstructured\_domains.h
  - vrna\_ud\_set\_prob\_cb, [869](#)
- unweight
  - Deprecated Interface for Secondary Structure Utilities, [705](#)
- update\_alifold\_params
  - alifold.h, [732](#)
- update\_co\_pf\_params
  - Deprecated Interface for Global Partition Function Computation, [685](#)
- update\_co\_pf\_params\_par
  - Deprecated Interface for Global Partition Function Computation, [685](#)
- update\_cofold\_params
  - Deprecated Interface for Global MFE Prediction, [661](#)
- update\_cofold\_params\_par
  - Deprecated Interface for Global MFE Prediction, [661](#)
- update\_fold\_params
  - Deprecated Interface for Global MFE Prediction, [666](#)
- update\_fold\_params\_par
  - Deprecated Interface for Global MFE Prediction, [667](#)
- update\_pf\_params
  - Deprecated Interface for Global Partition Function Computation, [678](#)
- update\_pf\_params\_par
  - Deprecated Interface for Global Partition Function Computation, [679](#)
- update\_pf\_paramsLP
  - Deprecated Interface for Local (Sliding Window) Partition Function Computation, [691](#)
- urn
  - basic.h, [820](#)
- Utilities, [435](#)
  - get\_input\_line, [440](#)
  - vrna\_alloc, [438](#)
  - vrna\_idx\_col\_wise, [441](#)
  - vrna\_idx\_row\_wise, [441](#)
  - VRNA\_INPUT\_FASTA\_HEADER, [438](#)
  - vrna\_int\_urn, [439](#)
  - vrna\_realloc, [439](#)
  - vrna\_time\_stamp, [440](#)
  - vrna\_urn, [439](#)
  - xsubi, [442](#)
- Utilities to deal with Nucleotide Alphabets, [477](#)
  - vrna\_nucleotide\_decode, [480](#)
  - vrna\_nucleotide\_encode, [480](#)
  - vrna\_ptypes, [479](#)
  - VRNA\_SEQ\_DNA, [479](#)
  - vrna\_seq\_encode, [479](#)
  - VRNA\_SEQ\_RNA, [479](#)

- vrna\_seq\_type\_e, [479](#)
- VRNA\_SEQ\_UNKNOWN, [479](#)
- ViennaRNA/2Dfold.h, [725](#)
- ViennaRNA/2Dpfold.h, [726](#)
- ViennaRNA/alifold.h, [731](#)
- ViennaRNA/aln\_util.h, [733](#)
- ViennaRNA/alphabet.h, [734](#)
- ViennaRNA/boltzmann\_sampling.h, [735](#)
- ViennaRNA/centroid.h, [737](#)
- ViennaRNA/char\_stream.h, [738](#)
- ViennaRNA/cofold.h, [740](#)
- ViennaRNA/combinatorics.h, [741](#)
- ViennaRNA/commands.h, [742](#)
- ViennaRNA/concentrations.h, [743](#)
- ViennaRNA/constraints.h, [744](#)
- ViennaRNA/constraints/basic.h, [814](#)
- ViennaRNA/constraints/hard.h, [745](#)
- ViennaRNA/constraints/ligand.h, [750](#)
- ViennaRNA/constraints/SHAPE.h, [752](#)
- ViennaRNA/constraints/soft.h, [754](#)
- ViennaRNA/constraints\_hard.h, [756](#)
- ViennaRNA/constraints\_ligand.h, [756](#)
- ViennaRNA/constraints\_SHAPE.h, [757](#)
- ViennaRNA/constraints\_soft.h, [757](#)
- ViennaRNA/convert\_epars.h, [758](#)
- ViennaRNA/data\_structures.h, [758](#)
- ViennaRNA/datastructures/basic.h, [821](#)
- ViennaRNA/datastructures/char\_stream.h, [739](#)
- ViennaRNA/datastructures/hash\_tables.h, [759](#)
- ViennaRNA/datastructures/heap.h, [760](#)
- ViennaRNA/datastructures/stream\_output.h, [858](#)
- ViennaRNA/dist\_vars.h, [761](#)
- ViennaRNA/dp\_matrices.h, [762](#)
- ViennaRNA/duplex.h, [763](#)
- ViennaRNA/edit\_cost.h, [764](#)
- ViennaRNA/energy\_const.h, [764](#)
- ViennaRNA/energy\_par.h, [765](#)
- ViennaRNA/equilibrium\_probs.h, [765](#)
- ViennaRNA/eval.h, [767](#)
- ViennaRNA/exterior\_loops.h, [770](#)
- ViennaRNA/file\_formats.h, [771](#)
- ViennaRNA/file\_formats\_msa.h, [772](#)
- ViennaRNA/file\_utils.h, [775](#)
- ViennaRNA/findpath.h, [776](#)
- ViennaRNA/fold.h, [777](#)
- ViennaRNA/fold\_compound.h, [778](#)
- ViennaRNA/fold\_vars.h, [780](#)
- ViennaRNA/gquad.h, [783](#)
- ViennaRNA/grammar.h, [783](#)
- ViennaRNA/hairpin\_loops.h, [784](#)
- ViennaRNA/heat\_capacity.h, [784](#)
- ViennaRNA/interior\_loops.h, [785](#)
- ViennaRNA/inverse.h, [786](#)
- ViennaRNA/io/file\_formats.h, [771](#)
- ViennaRNA/io/file\_formats\_msa.h, [773](#)
- ViennaRNA/io/utils.h, [870](#)
- ViennaRNA/landscape/findpath.h, [776](#)
- ViennaRNA/landscape/move.h, [786](#)
- ViennaRNA/landscape/neighbor.h, [809](#)
- ViennaRNA/landscape/paths.h, [788](#)
- ViennaRNA/landscape/walk.h, [876](#)
- ViennaRNA/Lfold.h, [789](#)
- ViennaRNA/loop\_energies.h, [790](#)
- ViennaRNA/loops/all.h, [790](#)
- ViennaRNA/loops/external.h, [791](#)
- ViennaRNA/loops/hairpin.h, [793](#)
- ViennaRNA/loops/internal.h, [794](#)
- ViennaRNA/loops/multibranch.h, [796](#)
- ViennaRNA/LPfold.h, [797](#)
- ViennaRNA/MEA.h, [798](#)
- ViennaRNA/mfe.h, [799](#)
- ViennaRNA/mfe\_window.h, [800](#)
- ViennaRNA/mm.h, [801](#)
- ViennaRNA/model.h, [802](#)
- ViennaRNA/multibranch\_loops.h, [807](#)
- ViennaRNA/naview.h, [807](#)
- ViennaRNA/neighbor.h, [809](#)
- ViennaRNA/params.h, [811](#)
- ViennaRNA/params/1.8.4\_epars.h, [811](#)
- ViennaRNA/params/1.8.4\_intloops.h, [812](#)
- ViennaRNA/params/basic.h, [812](#)
- ViennaRNA/params/constants.h, [823](#)
- ViennaRNA/params/convert.h, [825](#)
- ViennaRNA/params/io.h, [826](#)
- ViennaRNA/part\_func.h, [828](#)
- ViennaRNA/part\_func\_co.h, [832](#)
- ViennaRNA/part\_func\_up.h, [833](#)
- ViennaRNA/part\_func\_window.h, [834](#)
- ViennaRNA/perturbation\_fold.h, [835](#)
- ViennaRNA/plot\_aln.h, [837](#)
- ViennaRNA/plot\_layouts.h, [837](#)
- ViennaRNA/plot\_structure.h, [838](#)
- ViennaRNA/plot\_utils.h, [838](#)
- ViennaRNA/plotting/alignments.h, [839](#)
- ViennaRNA/plotting/layouts.h, [842](#)
- ViennaRNA/plotting/naview.h, [808](#)
- ViennaRNA/plotting/probabilities.h, [844](#)
- ViennaRNA/plotting/RNApuzzler/RNApuzzler.h, [845](#)
- ViennaRNA/plotting/RNApuzzler/RNAturtle.h, [846](#)
- ViennaRNA/plotting/structures.h, [846](#)
- ViennaRNA/plotting/utils.h, [871](#)
- ViennaRNA/profiledist.h, [851](#)
- ViennaRNA/PS\_dot.h, [853](#)
- ViennaRNA/read\_epars.h, [853](#)
- ViennaRNA/ribo.h, [854](#)
- ViennaRNA/RNAstruct.h, [854](#)
- ViennaRNA/search/BoyerMoore.h, [856](#)
- ViennaRNA/sequence.h, [856](#)
- ViennaRNA/stream\_output.h, [857](#)
- ViennaRNA/string\_utils.h, [859](#)
- ViennaRNA/stringdist.h, [859](#)
- ViennaRNA/structure\_utils.h, [860](#)
- ViennaRNA/structured\_domains.h, [861](#)
- ViennaRNA/subopt.h, [862](#)
- ViennaRNA/svm\_utils.h, [863](#)
- ViennaRNA/treedist.h, [864](#)

- ViennaRNA/units.h, [865](#)
- ViennaRNA/unstructured\_domains.h, [867](#)
- ViennaRNA/utls.h, [869](#)
- ViennaRNA/utls/alignments.h, [839](#)
- ViennaRNA/utls/basic.h, [815](#)
- ViennaRNA/utls/strings.h, [872](#)
- ViennaRNA/utls/structures.h, [848](#)
- ViennaRNA/utls/units.h, [866](#)
- ViennaRNA/walk.h, [875](#)
- vrna\_alifold
  - Global MFE Prediction, [294](#)
- vrna\_alignment\_s, [478](#)
- vrna\_alloc
  - Utilities, [438](#)
- vrna\_aln\_consensus\_mis
  - Multiple Sequence Alignment Utilities, [524](#)
- vrna\_aln\_consensus\_sequence
  - Multiple Sequence Alignment Utilities, [524](#)
- vrna\_aln\_conservation\_col
  - Multiple Sequence Alignment Utilities, [523](#)
- vrna\_aln\_conservation\_struct
  - Multiple Sequence Alignment Utilities, [522](#)
- vrna\_aln\_copy
  - Multiple Sequence Alignment Utilities, [522](#)
- vrna\_aln\_free
  - Multiple Sequence Alignment Utilities, [521](#)
- vrna\_aln\_mpi
  - Multiple Sequence Alignment Utilities, [519](#)
- vrna\_aln\_pinfo
  - Multiple Sequence Alignment Utilities, [520](#)
- vrna\_aln\_slice
  - Multiple Sequence Alignment Utilities, [520](#)
- vrna\_aln\_toRNA
  - Multiple Sequence Alignment Utilities, [521](#)
- vrna\_aln\_uppercase
  - Multiple Sequence Alignment Utilities, [521](#)
- vrna\_backtrack5
  - Backtracking MFE structures, [302](#)
- vrna\_backtrack5\_TwoD
  - Computing MFE representatives of a Distance Based Partitioning, [361](#)
- vrna\_basepair\_s, [597](#)
- vrna\_boltzmann\_sampling\_callback
  - Random Structure Samples from the Ensemble, [333](#)
- vrna\_bp\_distance
  - Secondary Structure Utilities, [492](#)
- vrna\_bp\_stack\_s, [598](#)
- VRNA\_BRACKETS\_ALPHA
  - Dot-Bracket Notation of Secondary Structures, [495](#)
- VRNA\_BRACKETS\_ANG
  - Dot-Bracket Notation of Secondary Structures, [495](#)
- VRNA\_BRACKETS\_ANY
  - Dot-Bracket Notation of Secondary Structures, [496](#)
- VRNA\_BRACKETS\_CLY
  - Dot-Bracket Notation of Secondary Structures, [495](#)
- VRNA\_BRACKETS\_DEFAULT
  - Dot-Bracket Notation of Secondary Structures, [496](#)
- VRNA\_BRACKETS\_RND
  - Dot-Bracket Notation of Secondary Structures, [495](#)
- VRNA\_BRACKETS\_SQR
  - Dot-Bracket Notation of Secondary Structures, [496](#)
- vrna\_BT\_hp\_loop
  - Backtracking MFE structures, [303](#)
- vrna\_BT\_mb\_loop
  - Backtracking MFE structures, [304](#)
- vrna\_C11\_features
  - (Abstract) Data Structures, [601](#)
- vrna\_callback\_free\_auxdata
  - The Fold Compound, [623](#)
- vrna\_callback\_hc\_evaluate
  - Hard Constraints, [269](#)
- vrna\_callback\_heap\_cmp
  - Heaps, [645](#)
- vrna\_callback\_heap\_get\_pos
  - Heaps, [645](#)
- vrna\_callback\_heap\_set\_pos
  - Heaps, [646](#)
- vrna\_callback\_ht\_compare\_entries
  - Hash Tables, [636](#)
- vrna\_callback\_ht\_free\_entry
  - Hash Tables, [637](#)
- vrna\_callback\_ht\_hash\_function
  - Hash Tables, [636](#)
- vrna\_callback\_move\_update
  - Neighborhood Relation and Move Sets for Secondary Structures, [392](#)
- vrna\_callback\_recursion\_status
  - The Fold Compound, [624](#)
- vrna\_callback\_sc\_backtrack
  - Soft Constraints, [279](#)
- vrna\_callback\_sc\_energy
  - Soft Constraints, [277](#)
- vrna\_callback\_sc\_exp\_energy
  - Soft Constraints, [278](#)
- vrna\_callback\_stream\_output
  - Buffers, [653](#)
- vrna\_callback\_ud\_energy
  - Unstructured Domains, [235](#)
- vrna\_callback\_ud\_exp\_energy
  - Unstructured Domains, [236](#)
- vrna\_callback\_ud\_exp\_production
  - Unstructured Domains, [236](#)
- vrna\_callback\_ud\_probs\_add
  - Unstructured Domains, [237](#)
- vrna\_callback\_ud\_probs\_get
  - Unstructured Domains, [237](#)
- vrna\_callback\_ud\_production
  - Unstructured Domains, [236](#)
- vrna\_centroid
  - Compute the Centroid Structure, [351](#)
- vrna\_centroid\_from\_plist
  - Compute the Centroid Structure, [352](#)
- vrna\_centroid\_from\_probs
  - Compute the Centroid Structure, [352](#)
- vrna\_circalifold

- Global MFE Prediction, [294](#)
- `vrna_circfold`
  - Global MFE Prediction, [293](#)
- `VRNA_CMD_PARSE_DEFAULTS`
  - Command Files, [550](#)
- `VRNA_CMD_PARSE_HC`
  - Command Files, [549](#)
- `VRNA_CMD_PARSE_SC`
  - Command Files, [549](#)
- `VRNA_CMD_PARSE_SD`
  - Command Files, [550](#)
- `VRNA_CMD_PARSE_UD`
  - Command Files, [549](#)
- `vrna_cofold`
  - Global MFE Prediction, [295](#)
- `vrna_color_s`, [597](#)
- `vrna_commands_apply`
  - Command Files, [551](#)
- `vrna_commands_free`
  - Command Files, [552](#)
- `VRNA_CONSTRAINT_DB`
  - Hard Constraints, [265](#)
- `VRNA_CONSTRAINT_DB_ANG_BRACK`
  - `hard.h`, [748](#)
- `VRNA_CONSTRAINT_DB_DEFAULT`
  - Hard Constraints, [268](#)
- `VRNA_CONSTRAINT_DB_DOT`
  - Hard Constraints, [266](#)
- `VRNA_CONSTRAINT_DB_ENFORCE_BP`
  - Hard Constraints, [265](#)
- `VRNA_CONSTRAINT_DB_GQUAD`
  - Hard Constraints, [267](#)
- `VRNA_CONSTRAINT_DB_INTERMOL`
  - Hard Constraints, [267](#)
- `VRNA_CONSTRAINT_DB_INTRAMOL`
  - Hard Constraints, [267](#)
- `VRNA_CONSTRAINT_DB_PIPE`
  - Hard Constraints, [265](#)
- `VRNA_CONSTRAINT_DB_RND_BRACK`
  - Hard Constraints, [266](#)
- `VRNA_CONSTRAINT_DB_WUSS`
  - Hard Constraints, [268](#)
- `VRNA_CONSTRAINT_DB_X`
  - Hard Constraints, [266](#)
- `VRNA_CONSTRAINT_FILE`
  - Constraining the RNA Folding Grammar, [248](#)
- `VRNA_CONSTRAINT_MULTILINE`
  - Nucleic Acid Sequences and Structures, [531](#)
- `VRNA_CONSTRAINT_NO_HEADER`
  - `hard.h`, [748](#)
- `VRNA_CONSTRAINT_SOFT_MFE`
  - Constraining the RNA Folding Grammar, [248](#)
- `VRNA_CONSTRAINT_SOFT_PF`
  - Constraining the RNA Folding Grammar, [249](#)
- `vrna_constraints_add`
  - Constraining the RNA Folding Grammar, [258](#)
- `vrna_convert_dcal_to_kcal`
  - Unit Conversion, [610](#)
- `vrna_convert_energy`
  - Unit Conversion, [609](#)
- `vrna_convert_kcal_to_dcal`
  - Unit Conversion, [610](#)
- `VRNA_CONVERT_OUTPUT_ALL`
  - Converting Energy Parameter Files, [472](#)
- `VRNA_CONVERT_OUTPUT_BULGE`
  - Converting Energy Parameter Files, [474](#)
- `VRNA_CONVERT_OUTPUT_DANGLE3`
  - Converting Energy Parameter Files, [473](#)
- `VRNA_CONVERT_OUTPUT_DANGLE5`
  - Converting Energy Parameter Files, [473](#)
- `VRNA_CONVERT_OUTPUT_DUMP`
  - Converting Energy Parameter Files, [475](#)
- `VRNA_CONVERT_OUTPUT_HP`
  - Converting Energy Parameter Files, [472](#)
- `VRNA_CONVERT_OUTPUT_INT`
  - Converting Energy Parameter Files, [474](#)
- `VRNA_CONVERT_OUTPUT_INT_11`
  - Converting Energy Parameter Files, [473](#)
- `VRNA_CONVERT_OUTPUT_INT_21`
  - Converting Energy Parameter Files, [474](#)
- `VRNA_CONVERT_OUTPUT_INT_22`
  - Converting Energy Parameter Files, [474](#)
- `VRNA_CONVERT_OUTPUT_MISC`
  - Converting Energy Parameter Files, [474](#)
- `VRNA_CONVERT_OUTPUT_ML`
  - Converting Energy Parameter Files, [474](#)
- `VRNA_CONVERT_OUTPUT_MM_EXT`
  - Converting Energy Parameter Files, [473](#)
- `VRNA_CONVERT_OUTPUT_MM_HP`
  - Converting Energy Parameter Files, [472](#)
- `VRNA_CONVERT_OUTPUT_MM_INT`
  - Converting Energy Parameter Files, [472](#)
- `VRNA_CONVERT_OUTPUT_MM_INT_1N`
  - Converting Energy Parameter Files, [472](#)
- `VRNA_CONVERT_OUTPUT_MM_INT_23`
  - Converting Energy Parameter Files, [473](#)
- `VRNA_CONVERT_OUTPUT_MM_MULTI`
  - Converting Energy Parameter Files, [473](#)
- `VRNA_CONVERT_OUTPUT_NINIO`
  - Converting Energy Parameter Files, [475](#)
- `VRNA_CONVERT_OUTPUT_SPECIAL_HP`
  - Converting Energy Parameter Files, [475](#)
- `VRNA_CONVERT_OUTPUT_STACK`
  - Converting Energy Parameter Files, [472](#)
- `VRNA_CONVERT_OUTPUT_VANILLA`
  - Converting Energy Parameter Files, [475](#)
- `vrna_convert_temperature`
  - Unit Conversion, [609](#)
- `vrna_cpair_s`, [597](#)
- `vrna_cstr`
  - Buffers, [653](#)
- `vrna_cstr_close`
  - Buffers, [654](#)
- `vrna_cstr_fflush`
  - Buffers, [654](#)
- `vrna_cstr_free`



- Buffers, [654](#)
- `vrna_cut_point_insert`
  - (Nucleic Acid Sequence) String Utilities, [489](#)
- `vrna_cut_point_remove`
  - (Nucleic Acid Sequence) String Utilities, [490](#)
- `vrna_data_linear_s`, [597](#)
- `vrna_db_flatten`
  - Dot-Bracket Notation of Secondary Structures, [498](#)
- `vrna_db_flatten_to`
  - Dot-Bracket Notation of Secondary Structures, [499](#)
- `vrna_db_from_bp_stack`
  - Secondary Structure Utilities, [493](#)
- `vrna_db_from_plist`
  - Dot-Bracket Notation of Secondary Structures, [500](#)
- `vrna_db_from_ptable`
  - Dot-Bracket Notation of Secondary Structures, [499](#)
- `vrna_db_from_WUSS`
  - Dot-Bracket Notation of Secondary Structures, [500](#)
- `vrna_db_pack`
  - Dot-Bracket Notation of Secondary Structures, [497](#)
- `vrna_db_pk_remove`
  - Dot-Bracket Notation of Secondary Structures, [501](#)
- `vrna_db_to_element_string`
  - Dot-Bracket Notation of Secondary Structures, [501](#)
- `vrna_db_to_tree_string`
  - Tree Representation of Secondary Structures, [514](#)
- `vrna_db_unpack`
  - Dot-Bracket Notation of Secondary Structures, [497](#)
- `VRNA_DECOMP_EXT_EXT`
  - Constraining the RNA Folding Grammar, [254](#)
- `VRNA_DECOMP_EXT_EXT_EXT`
  - Constraining the RNA Folding Grammar, [256](#)
- `VRNA_DECOMP_EXT_EXT_STEM`
  - Constraining the RNA Folding Grammar, [257](#)
- `VRNA_DECOMP_EXT_EXT_STEM1`
  - Constraining the RNA Folding Grammar, [257](#)
- `VRNA_DECOMP_EXT_STEM`
  - Constraining the RNA Folding Grammar, [255](#)
- `VRNA_DECOMP_EXT_STEM_EXT`
  - Constraining the RNA Folding Grammar, [256](#)
- `VRNA_DECOMP_EXT_UP`
  - Constraining the RNA Folding Grammar, [255](#)
- `VRNA_DECOMP_ML_COAXIAL`
  - Constraining the RNA Folding Grammar, [253](#)
- `VRNA_DECOMP_ML_COAXIAL_ENC`
  - Constraining the RNA Folding Grammar, [254](#)
- `VRNA_DECOMP_ML_ML`
  - Constraining the RNA Folding Grammar, [252](#)
- `VRNA_DECOMP_ML_ML_ML`
  - Constraining the RNA Folding Grammar, [251](#)
- `VRNA_DECOMP_ML_ML_STEM`
  - Constraining the RNA Folding Grammar, [253](#)
- `VRNA_DECOMP_ML_STEM`
  - Constraining the RNA Folding Grammar, [251](#)
- `VRNA_DECOMP_ML_UP`
  - Constraining the RNA Folding Grammar, [252](#)
- `VRNA_DECOMP_PAIR_HP`
  - Constraining the RNA Folding Grammar, [249](#)
- `VRNA_DECOMP_PAIR_IL`
  - Constraining the RNA Folding Grammar, [249](#)
- `VRNA_DECOMP_PAIR_ML`
  - Constraining the RNA Folding Grammar, [250](#)
- `vrna_dimer_conc_s`, [722](#)
- `vrna_dimer_pf_s`, [306](#)
- `vrna_dotplot_auxdata_t`, [555](#)
- `vrna_E_ext_hp_loop`
  - Hairpin Loops, [448](#)
- `vrna_E_ext_loop`
  - Exterior Loops, [445](#)
- `vrna_E_ext_stem`
  - Exterior Loops, [444](#)
- `vrna_E_hp_loop`
  - Hairpin Loops, [448](#)
- `vrna_E_mb_loop_stack`
  - Multibranch Loops, [455](#)
- `vrna_elem_prob_s`, [508](#)
- `vrna_ensemble_defect`
  - Predicting various thermodynamic properties, [375](#)
- `vrna_enumerate_necklaces`
  - Combinatorics Algorithms, [588](#)
- `vrna_eval_circ_consensus_structure`
  - Free Energy Evaluation, [147](#)
- `vrna_eval_circ_consensus_structure_v`
  - Free Energy Evaluation, [150](#)
- `vrna_eval_circ_gquad_consensus_structure`
  - Free Energy Evaluation, [148](#)
- `vrna_eval_circ_gquad_consensus_structure_v`
  - Free Energy Evaluation, [152](#)
- `vrna_eval_circ_gquad_structure`
  - Free Energy Evaluation, [141](#)
- `vrna_eval_circ_gquad_structure_v`
  - Free Energy Evaluation, [145](#)
- `vrna_eval_circ_structure`
  - Free Energy Evaluation, [140](#)
- `vrna_eval_circ_structure_v`
  - Free Energy Evaluation, [144](#)
- `vrna_eval_consensus_structure_pt_simple`
  - Free Energy Evaluation, [155](#)
- `vrna_eval_consensus_structure_simple`
  - Free Energy Evaluation, [146](#)
- `vrna_eval_consensus_structure_simple_v`
  - Free Energy Evaluation, [150](#)
- `vrna_eval_consensus_structure_simple_verbose`
  - Free Energy Evaluation, [149](#)
- `vrna_eval_covar_structure`
  - Free Energy Evaluation, [135](#)
- `vrna_eval_gquad_consensus_structure`
  - Free Energy Evaluation, [147](#)
- `vrna_eval_gquad_consensus_structure_v`
  - Free Energy Evaluation, [151](#)
- `vrna_eval_gquad_structure`
  - Free Energy Evaluation, [141](#)
- `vrna_eval_gquad_structure_v`
  - Free Energy Evaluation, [144](#)
- `vrna_eval_hp_loop`
  - Hairpin Loops, [448](#)

- vrna\_eval\_int\_loop
  - Internal Loops, [452](#)
- vrna\_eval\_loop\_pt
  - Energy Evaluation for Individual Loops, [158](#)
- vrna\_eval\_loop\_pt\_v
  - Energy Evaluation for Individual Loops, [158](#)
- vrna\_eval\_move
  - Energy Evaluation for Atomic Moves, [160](#)
- vrna\_eval\_move\_pt
  - Energy Evaluation for Atomic Moves, [161](#)
- vrna\_eval\_structure
  - Free Energy Evaluation, [135](#)
- vrna\_eval\_structure\_pt
  - Free Energy Evaluation, [137](#)
- vrna\_eval\_structure\_pt\_simple
  - Free Energy Evaluation, [153](#)
- vrna\_eval\_structure\_pt\_simple\_v
  - Free Energy Evaluation, [154](#)
- vrna\_eval\_structure\_pt\_simple\_verbose
  - Free Energy Evaluation, [154](#)
- vrna\_eval\_structure\_pt\_v
  - Free Energy Evaluation, [139](#)
- vrna\_eval\_structure\_pt\_verbose
  - Free Energy Evaluation, [138](#)
- vrna\_eval\_structure\_simple
  - Free Energy Evaluation, [140](#)
- vrna\_eval\_structure\_simple\_v
  - Free Energy Evaluation, [143](#)
- vrna\_eval\_structure\_simple\_verbose
  - Free Energy Evaluation, [142](#)
- vrna\_eval\_structure\_v
  - Free Energy Evaluation, [137](#)
- vrna\_eval\_structure\_verbose
  - Free Energy Evaluation, [136](#)
- vrna\_exp\_E\_ext\_stem
  - Exterior Loops, [445](#)
- vrna\_exp\_E\_hp\_loop
  - Hairpin Loops, [451](#)
- vrna\_exp\_param\_s, [218](#)
  - alpha, [219](#)
  - id, [219](#)
- vrna\_exp\_params
  - Energy Parameters, [221](#)
- vrna\_exp\_params\_comparative
  - Energy Parameters, [221](#)
- vrna\_exp\_params\_copy
  - Energy Parameters, [222](#)
- vrna\_exp\_params\_rescale
  - Energy Parameters, [224](#)
- vrna\_exp\_params\_reset
  - Energy Parameters, [225](#)
- vrna\_exp\_params\_subst
  - Energy Parameters, [223](#)
- vrna\_extract\_record\_rest\_constraint
  - Nucleic Acid Sequences and Structures, [536](#)
- vrna\_extract\_record\_rest\_structure
  - Nucleic Acid Sequences and Structures, [535](#)
- vrna\_fc\_s, [613](#)
  - auxdata, [617](#)
  - cons\_seq, [619](#)
  - free\_auxdata, [617](#)
  - n\_seq, [619](#)
  - pscore, [620](#)
  - pscore\_local, [620](#)
  - pscore\_pf\_compat, [621](#)
  - ptype, [617](#)
  - ptype\_pf\_compat, [618](#)
  - S, [619](#)
  - S3, [620](#)
  - S5, [620](#)
  - S\_cons, [619](#)
  - sc, [618](#)
  - scs, [621](#)
  - sequence, [617](#)
  - sequence\_encoding, [617](#)
  - sequences, [618](#)
  - stat\_cb, [616](#)
  - type, [616](#)
- VRNA\_FC\_TYPE\_COMPARATIVE
  - The Fold Compound, [624](#)
- vrna\_fc\_type\_e
  - The Fold Compound, [624](#)
- VRNA\_FC\_TYPE\_SINGLE
  - The Fold Compound, [624](#)
- vrna\_file\_bpseq
  - Nucleic Acid Sequences and Structures, [532](#)
- vrna\_file\_commands\_apply
  - Command Files, [551](#)
- vrna\_file\_commands\_read
  - Command Files, [550](#)
- vrna\_file\_connect
  - Nucleic Acid Sequences and Structures, [532](#)
- vrna\_file\_exists
  - Files and I/O, [529](#)
- vrna\_file\_fasta\_read\_record
  - Nucleic Acid Sequences and Structures, [533](#)
- VRNA\_FILE\_FORMAT\_MSA\_APPEND
  - Multiple Sequence Alignments, [541](#)
- VRNA\_FILE\_FORMAT\_MSA\_CLUSTAL
  - Multiple Sequence Alignments, [539](#)
- VRNA\_FILE\_FORMAT\_MSA\_DEFAULT
  - Multiple Sequence Alignments, [540](#)
- VRNA\_FILE\_FORMAT\_MSA\_FASTA
  - Multiple Sequence Alignments, [539](#)
- VRNA\_FILE\_FORMAT\_MSA\_MAF
  - Multiple Sequence Alignments, [540](#)
- VRNA\_FILE\_FORMAT\_MSA\_MIS
  - Multiple Sequence Alignments, [540](#)
- VRNA\_FILE\_FORMAT\_MSA\_NOCHECK
  - Multiple Sequence Alignments, [541](#)
- VRNA\_FILE\_FORMAT\_MSA\_QUIET
  - Multiple Sequence Alignments, [542](#)
- VRNA\_FILE\_FORMAT\_MSA\_SILENT
  - Multiple Sequence Alignments, [542](#)
- VRNA\_FILE\_FORMAT\_MSA\_STOCKHOLM
  - Multiple Sequence Alignments, [539](#)



- VRNA\_FILE\_FORMAT\_MSA\_UNKNOWN
  - Multiple Sequence Alignments, [541](#)
- vrna\_file\_helixlist
  - Nucleic Acid Sequences and Structures, [531](#)
- vrna\_file\_json
  - Nucleic Acid Sequences and Structures, [533](#)
- vrna\_file\_msa\_detect\_format
  - Multiple Sequence Alignments, [545](#)
- vrna\_file\_msa\_read
  - Multiple Sequence Alignments, [542](#)
- vrna\_file\_msa\_read\_record
  - Multiple Sequence Alignments, [544](#)
- vrna\_file\_msa\_write
  - Multiple Sequence Alignments, [546](#)
- vrna\_file\_PS\_aln
  - Alignment Plots, [581](#)
- vrna\_file\_PS\_aln\_slice
  - Alignment Plots, [582](#)
- vrna\_file\_PS\_rnaplot
  - Plotting, [556](#)
- vrna\_file\_PS\_rnaplot\_a
  - Plotting, [557](#)
- vrna\_file\_SHAPE\_read
  - Nucleic Acid Sequences and Structures, [535](#)
- vrna\_filename\_sanitize
  - Files and I/O, [527](#)
- vrna\_fold
  - Global MFE Prediction, [292](#)
- vrna\_fold\_compound
  - The Fold Compound, [625](#)
- vrna\_fold\_compound\_add\_auxdata
  - The Fold Compound, [627](#)
- vrna\_fold\_compound\_add\_callback
  - The Fold Compound, [628](#)
- vrna\_fold\_compound\_comparative
  - The Fold Compound, [626](#)
- vrna\_fold\_compound\_free
  - The Fold Compound, [627](#)
- vrna\_gr\_aux\_s, [179](#)
- vrna\_hamming\_distance
  - (Nucleic Acid Sequence) String Utilities, [487](#)
- vrna\_hamming\_distance\_bound
  - (Nucleic Acid Sequence) String Utilities, [488](#)
- vrna\_hash\_table\_t
  - Hash Tables, [636](#)
- vrna\_hc\_add\_bp
  - Hard Constraints, [271](#)
- vrna\_hc\_add\_bp\_nonspecific
  - Hard Constraints, [271](#)
- vrna\_hc\_add\_data
  - hard.h, [749](#)
- vrna\_hc\_add\_from\_db
  - Hard Constraints, [272](#)
- vrna\_hc\_add\_up
  - Hard Constraints, [270](#)
- vrna\_hc\_add\_up\_batch
  - Hard Constraints, [270](#)
- VRNA\_HC\_DEFAULT
  - hard.h, [749](#)
- vrna\_hc\_free
  - Hard Constraints, [272](#)
- vrna\_hc\_init
  - Hard Constraints, [269](#)
- vrna\_hc\_s, [264](#)
  - free\_data, [264](#)
- vrna\_hc\_type\_e
  - hard.h, [748](#)
- vrna\_hc\_up\_s, [265](#)
- VRNA\_HC\_WINDOW
  - hard.h, [749](#)
- vrna\_heap\_free
  - Heaps, [647](#)
- vrna\_heap\_init
  - Heaps, [646](#)
- vrna\_heap\_insert
  - Heaps, [648](#)
- vrna\_heap\_pop
  - Heaps, [648](#)
- vrna\_heap\_remove
  - Heaps, [649](#)
- vrna\_heap\_size
  - Heaps, [648](#)
- vrna\_heap\_t
  - Heaps, [645](#)
- vrna\_heap\_top
  - Heaps, [649](#)
- vrna\_heap\_update
  - Heaps, [650](#)
- vrna\_heat\_capacity
  - Predicting various thermodynamic properties, [378](#)
- vrna\_heat\_capacity\_callback
  - Predicting various thermodynamic properties, [373](#)
- vrna\_heat\_capacity\_cb
  - Predicting various thermodynamic properties, [379](#)
- vrna\_heat\_capacity\_s, [373](#)
- vrna\_heat\_capacity\_simple
  - Predicting various thermodynamic properties, [380](#)
- vrna\_heat\_capacity\_t
  - Predicting various thermodynamic properties, [374](#)
- vrna\_ht\_clear
  - Hash Tables, [640](#)
- vrna\_ht\_collisions
  - Hash Tables, [638](#)
- vrna\_ht\_db\_comp
  - Hash Tables, [641](#)
- vrna\_ht\_db\_free\_entry
  - Hash Tables, [642](#)
- vrna\_ht\_db\_hash\_func
  - Hash Tables, [642](#)
- vrna\_ht\_entry\_db\_t, [635](#)
  - energy, [635](#)
  - structure, [635](#)
- vrna\_ht\_free
  - Hash Tables, [641](#)
- vrna\_ht\_get
  - Hash Tables, [639](#)

- vrna\_ht\_init
  - Hash Tables, [637](#)
- vrna\_ht\_insert
  - Hash Tables, [639](#)
- vrna\_ht\_remove
  - Hash Tables, [640](#)
- vrna\_ht\_size
  - Hash Tables, [638](#)
- vrna\_hx\_from\_ptable
  - Helix List Representation of Secondary Structures, [510](#)
- vrna\_hx\_s, [510](#)
- vrna\_idx\_col\_wise
  - Utilities, [441](#)
- vrna\_idx\_row\_wise
  - Utilities, [441](#)
- VRNA\_INPUT\_FASTA\_HEADER
  - Utilities, [438](#)
- vrna\_int\_urn
  - Utilities, [439](#)
- vrna\_Lfold
  - Local (sliding window) MFE Prediction, [300](#)
- vrna\_Lfoldz
  - Local (sliding window) MFE Prediction, [301](#)
- vrna\_loopidx\_update
  - Neighborhood Relation and Move Sets for Secondary Structures, [395](#)
- vrna\_maximum\_matching
  - mm.h, [802](#)
- vrna\_maximum\_matching\_simple
  - mm.h, [802](#)
- vrna\_md\_copy
  - Fine-tuning of the Implemented Models, [193](#)
- vrna\_md\_defaults\_backtrack
  - Fine-tuning of the Implemented Models, [204](#)
- vrna\_md\_defaults\_backtrack\_get
  - Fine-tuning of the Implemented Models, [204](#)
- vrna\_md\_defaults\_backtrack\_type
  - Fine-tuning of the Implemented Models, [205](#)
- vrna\_md\_defaults\_backtrack\_type\_get
  - Fine-tuning of the Implemented Models, [205](#)
- vrna\_md\_defaults\_betaScale
  - Fine-tuning of the Implemented Models, [196](#)
- vrna\_md\_defaults\_betaScale\_get
  - Fine-tuning of the Implemented Models, [196](#)
- vrna\_md\_defaults\_circ
  - Fine-tuning of the Implemented Models, [201](#)
- vrna\_md\_defaults\_circ\_get
  - Fine-tuning of the Implemented Models, [201](#)
- vrna\_md\_defaults\_compute\_bpp
  - Fine-tuning of the Implemented Models, [205](#)
- vrna\_md\_defaults\_compute\_bpp\_get
  - Fine-tuning of the Implemented Models, [206](#)
- vrna\_md\_defaults\_cv\_fact
  - Fine-tuning of the Implemented Models, [210](#)
- vrna\_md\_defaults\_cv\_fact\_get
  - Fine-tuning of the Implemented Models, [210](#)
- vrna\_md\_defaults\_dangles
  - Fine-tuning of the Implemented Models, [196](#)
- vrna\_md\_defaults\_dangles\_get
  - Fine-tuning of the Implemented Models, [197](#)
- vrna\_md\_defaults\_energy\_set
  - Fine-tuning of the Implemented Models, [203](#)
- vrna\_md\_defaults\_energy\_set\_get
  - Fine-tuning of the Implemented Models, [204](#)
- vrna\_md\_defaults\_gquad
  - Fine-tuning of the Implemented Models, [202](#)
- vrna\_md\_defaults\_gquad\_get
  - Fine-tuning of the Implemented Models, [202](#)
- vrna\_md\_defaults\_logML
  - Fine-tuning of the Implemented Models, [200](#)
- vrna\_md\_defaults\_logML\_get
  - Fine-tuning of the Implemented Models, [201](#)
- vrna\_md\_defaults\_max\_bp\_span
  - Fine-tuning of the Implemented Models, [206](#)
- vrna\_md\_defaults\_max\_bp\_span\_get
  - Fine-tuning of the Implemented Models, [207](#)
- vrna\_md\_defaults\_min\_loop\_size
  - Fine-tuning of the Implemented Models, [207](#)
- vrna\_md\_defaults\_min\_loop\_size\_get
  - Fine-tuning of the Implemented Models, [207](#)
- vrna\_md\_defaults\_nc\_fact
  - Fine-tuning of the Implemented Models, [211](#)
- vrna\_md\_defaults\_nc\_fact\_get
  - Fine-tuning of the Implemented Models, [211](#)
- vrna\_md\_defaults\_noGU
  - Fine-tuning of the Implemented Models, [199](#)
- vrna\_md\_defaults\_noGU\_get
  - Fine-tuning of the Implemented Models, [199](#)
- vrna\_md\_defaults\_noGUclosure
  - Fine-tuning of the Implemented Models, [199](#)
- vrna\_md\_defaults\_noGUclosure\_get
  - Fine-tuning of the Implemented Models, [200](#)
- vrna\_md\_defaults\_noLP
  - Fine-tuning of the Implemented Models, [198](#)
- vrna\_md\_defaults\_noLP\_get
  - Fine-tuning of the Implemented Models, [198](#)
- vrna\_md\_defaults\_oldAliEn
  - Fine-tuning of the Implemented Models, [208](#)
- vrna\_md\_defaults\_oldAliEn\_get
  - Fine-tuning of the Implemented Models, [209](#)
- vrna\_md\_defaults\_reset
  - Fine-tuning of the Implemented Models, [194](#)
- vrna\_md\_defaults\_ribo
  - Fine-tuning of the Implemented Models, [209](#)
- vrna\_md\_defaults\_ribo\_get
  - Fine-tuning of the Implemented Models, [210](#)
- vrna\_md\_defaults\_sfact
  - Fine-tuning of the Implemented Models, [211](#)
- vrna\_md\_defaults\_sfact\_get
  - Fine-tuning of the Implemented Models, [212](#)
- vrna\_md\_defaults\_special\_hp
  - Fine-tuning of the Implemented Models, [197](#)
- vrna\_md\_defaults\_special\_hp\_get
  - Fine-tuning of the Implemented Models, [198](#)
- vrna\_md\_defaults\_temperature

- Fine-tuning of the Implemented Models, [195](#)
- `vrna_md_defaults_temperature_get`
  - Fine-tuning of the Implemented Models, [195](#)
- `vrna_md_defaults_uniq_ML`
  - Fine-tuning of the Implemented Models, [202](#)
- `vrna_md_defaults_uniq_ML_get`
  - Fine-tuning of the Implemented Models, [203](#)
- `vrna_md_defaults_window_size`
  - Fine-tuning of the Implemented Models, [208](#)
- `vrna_md_defaults_window_size_get`
  - Fine-tuning of the Implemented Models, [208](#)
- `vrna_md_option_string`
  - Fine-tuning of the Implemented Models, [194](#)
- `vrna_md_s`, [184](#)
  - dangles, [187](#)
  - min\_loop\_size, [187](#)
- `vrna_md_set_default`
  - Fine-tuning of the Implemented Models, [193](#)
- `vrna_md_update`
  - Fine-tuning of the Implemented Models, [193](#)
- `vrna_MEA`
  - Compute the Structure with Maximum Expected Accuracy (MEA), [348](#)
- `vrna_MEA_from_plist`
  - Compute the Structure with Maximum Expected Accuracy (MEA), [349](#)
- `vrna_mean_bp_distance`
  - Predicting various thermodynamic properties, [374](#)
- `vrna_mean_bp_distance_pr`
  - Predicting various thermodynamic properties, [374](#)
- `VRNA_MEASURE_SHANNON_ENTROPY`
  - Multiple Sequence Alignment Utilities, [519](#)
- `vrna_message_constraint_options`
  - Constraining the RNA Folding Grammar, [259](#)
- `vrna_message_constraint_options_all`
  - Constraining the RNA Folding Grammar, [261](#)
- `vrna_message_error`
  - Messages, [602](#)
- `vrna_message_info`
  - Messages, [604](#)
- `vrna_message_input_seq`
  - Messages, [606](#)
- `vrna_message_input_seq_simple`
  - Messages, [605](#)
- `vrna_message_verror`
  - Messages, [603](#)
- `vrna_message_vinfo`
  - Messages, [605](#)
- `vrna_message_vwarning`
  - Messages, [604](#)
- `vrna_message_warning`
  - Messages, [603](#)
- `vrna_mfe`
  - Global MFE Prediction, [291](#)
- `vrna_mfe_dimer`
  - Global MFE Prediction, [292](#)
- `vrna_mfe_TwoD`
  - Computing MFE representatives of a Distance Based Partitioning, [360](#)
- `vrna_mfe_window`
  - Local (sliding window) MFE Prediction, [299](#)
- `vrna_mfe_window_callback`
  - Local (sliding window) MFE Prediction, [298](#)
- `vrna_mfe_window_zscore`
  - Local (sliding window) MFE Prediction, [299](#)
- `VRNA_MINIMIZER_CONJUGATE_FR`
  - Generate Soft Constraints from Data, [423](#)
- `VRNA_MINIMIZER_CONJUGATE_PR`
  - Generate Soft Constraints from Data, [423](#)
- `VRNA_MINIMIZER_STEEPEST_DESCENT`
  - Generate Soft Constraints from Data, [424](#)
- `VRNA_MINIMIZER_VECTOR_BFGS`
  - Generate Soft Constraints from Data, [424](#)
- `VRNA_MINIMIZER_VECTOR_BFGS2`
  - Generate Soft Constraints from Data, [424](#)
- `VRNA_MODEL_DEFAULT_ALI_CV_FACT`
  - Fine-tuning of the Implemented Models, [192](#)
- `VRNA_MODEL_DEFAULT_ALI_NC_FACT`
  - Fine-tuning of the Implemented Models, [192](#)
- `VRNA_MODEL_DEFAULT_ALI_OLD_EN`
  - Fine-tuning of the Implemented Models, [192](#)
- `VRNA_MODEL_DEFAULT_ALI_RIBO`
  - Fine-tuning of the Implemented Models, [192](#)
- `VRNA_MODEL_DEFAULT_BACKTRACK`
  - Fine-tuning of the Implemented Models, [190](#)
- `VRNA_MODEL_DEFAULT_BACKTRACK_TYPE`
  - Fine-tuning of the Implemented Models, [190](#)
- `VRNA_MODEL_DEFAULT_BETA_SCALE`
  - Fine-tuning of the Implemented Models, [188](#)
- `VRNA_MODEL_DEFAULT_CIRC`
  - Fine-tuning of the Implemented Models, [189](#)
- `VRNA_MODEL_DEFAULT_COMPUTE_BPP`
  - Fine-tuning of the Implemented Models, [191](#)
- `VRNA_MODEL_DEFAULT_DANGLES`
  - Fine-tuning of the Implemented Models, [188](#)
- `VRNA_MODEL_DEFAULT_ENERGY_SET`
  - Fine-tuning of the Implemented Models, [190](#)
- `VRNA_MODEL_DEFAULT_GQUAD`
  - Fine-tuning of the Implemented Models, [189](#)
- `VRNA_MODEL_DEFAULT_LOG_ML`
  - Fine-tuning of the Implemented Models, [191](#)
- `VRNA_MODEL_DEFAULT_MAX_BP_SPAN`
  - Fine-tuning of the Implemented Models, [191](#)
- `VRNA_MODEL_DEFAULT_NO_GU`
  - Fine-tuning of the Implemented Models, [189](#)
- `VRNA_MODEL_DEFAULT_NO_GU_CLOSURE`
  - Fine-tuning of the Implemented Models, [189](#)
- `VRNA_MODEL_DEFAULT_NO_LP`
  - Fine-tuning of the Implemented Models, [188](#)
- `VRNA_MODEL_DEFAULT_PF_SCALE`
  - Fine-tuning of the Implemented Models, [187](#)
- `VRNA_MODEL_DEFAULT_SPECIAL_HP`
  - Fine-tuning of the Implemented Models, [188](#)
- `VRNA_MODEL_DEFAULT_TEMPERATURE`
  - Fine-tuning of the Implemented Models, [187](#)

- VRNA\_MODEL\_DEFAULT\_UNIQ\_ML
  - Fine-tuning of the Implemented Models, [190](#)
- VRNA\_MODEL\_DEFAULT\_WINDOW\_SIZE
  - Fine-tuning of the Implemented Models, [191](#)
- vrna\_move\_apply
  - Neighborhood Relation and Move Sets for Secondary Structures, [393](#)
- vrna\_move\_compare
  - Neighborhood Relation and Move Sets for Secondary Structures, [394](#)
- vrna\_move\_init
  - Neighborhood Relation and Move Sets for Secondary Structures, [392](#)
- vrna\_move\_is\_insertion
  - Neighborhood Relation and Move Sets for Secondary Structures, [394](#)
- vrna\_move\_is\_removal
  - Neighborhood Relation and Move Sets for Secondary Structures, [393](#)
- vrna\_move\_is\_shift
  - Neighborhood Relation and Move Sets for Secondary Structures, [394](#)
- vrna\_move\_list\_free
  - Neighborhood Relation and Move Sets for Secondary Structures, [393](#)
- vrna\_move\_neighbor\_diff
  - Neighborhood Relation and Move Sets for Secondary Structures, [398](#)
- vrna\_move\_neighbor\_diff\_cb
  - Neighborhood Relation and Move Sets for Secondary Structures, [397](#)
- vrna\_move\_s, [389](#)
- VRNA\_MOVESET\_DEFAULT
  - Neighborhood Relation and Move Sets for Secondary Structures, [391](#)
- VRNA\_MOVESET\_DELETION
  - Neighborhood Relation and Move Sets for Secondary Structures, [390](#)
- VRNA\_MOVESET\_INSERTION
  - Neighborhood Relation and Move Sets for Secondary Structures, [390](#)
- VRNA\_MOVESET\_NO\_LP
  - Neighborhood Relation and Move Sets for Secondary Structures, [390](#)
- VRNA\_MOVESET\_SHIFT
  - Neighborhood Relation and Move Sets for Secondary Structures, [390](#)
- VRNA\_MX\_2DFOLD
  - The Dynamic Programming Matrices, [631](#)
- vrna\_mx\_add
  - The Dynamic Programming Matrices, [631](#)
- VRNA\_MX\_DEFAULT
  - The Dynamic Programming Matrices, [631](#)
- vrna\_mx\_mfe\_free
  - The Dynamic Programming Matrices, [632](#)
- vrna\_mx\_mfe\_s, [630](#)
- vrna\_mx\_pf\_aux\_el\_t
  - Exterior Loops, [444](#)
- vrna\_mx\_pf\_aux\_ml\_t
  - Multibranch Loops, [455](#)
- vrna\_mx\_pf\_free
  - The Dynamic Programming Matrices, [632](#)
- vrna\_mx\_pf\_s, [630](#)
- vrna\_mx\_type\_e
  - The Dynamic Programming Matrices, [631](#)
- VRNA\_MX\_WINDOW
  - The Dynamic Programming Matrices, [631](#)
- VRNA\_NEIGHBOR\_CHANGE
  - Neighborhood Relation and Move Sets for Secondary Structures, [391](#)
- VRNA\_NEIGHBOR\_INVALID
  - Neighborhood Relation and Move Sets for Secondary Structures, [391](#)
- VRNA\_NEIGHBOR\_NEW
  - Neighborhood Relation and Move Sets for Secondary Structures, [391](#)
- vrna\_neighbors
  - Neighborhood Relation and Move Sets for Secondary Structures, [396](#)
- vrna\_neighbors\_successive
  - Neighborhood Relation and Move Sets for Secondary Structures, [396](#)
- vrna\_nucleotide\_decode
  - Utilities to deal with Nucleotide Alphabets, [480](#)
- vrna\_nucleotide\_encode
  - Utilities to deal with Nucleotide Alphabets, [480](#)
- VRNA\_OBJECTIVE\_FUNCTION\_ABSOLUTE
  - Generate Soft Constraints from Data, [423](#)
- VRNA\_OBJECTIVE\_FUNCTION\_QUADRATIC
  - Generate Soft Constraints from Data, [423](#)
- VRNA\_OPTION\_EVAL\_ONLY
  - The Fold Compound, [623](#)
- VRNA\_OPTION\_MFE
  - The Fold Compound, [622](#)
- VRNA\_OPTION\_MULTILINE
  - Nucleic Acid Sequences and Structures, [531](#)
- VRNA\_OPTION\_PF
  - The Fold Compound, [622](#)
- vrna\_ostream\_free
  - Buffers, [655](#)
- vrna\_ostream\_init
  - Buffers, [655](#)
- vrna\_ostream\_provide
  - Buffers, [656](#)
- vrna\_ostream\_request
  - Buffers, [656](#)
- vrna\_param\_s, [218](#)
- VRNA\_PARAMETER\_FORMAT\_DEFAULT
  - Reading/Writing Energy Parameter Sets from/to File, [463](#)
- vrna\_params
  - Energy Parameters, [220](#)
- vrna\_params\_copy
  - Energy Parameters, [220](#)
- vrna\_params\_load

- Reading/Writing Energy Parameter Sets from/to File, [463](#)
- `vrna_params_load_defaults`
  - Reading/Writing Energy Parameter Sets from/to File, [465](#)
- `vrna_params_load_DNA_Mathews1999`
  - Reading/Writing Energy Parameter Sets from/to File, [468](#)
- `vrna_params_load_DNA_Mathews2004`
  - Reading/Writing Energy Parameter Sets from/to File, [468](#)
- `vrna_params_load_from_string`
  - Reading/Writing Energy Parameter Sets from/to File, [464](#)
- `vrna_params_load_RNA_Andronesescu2007`
  - Reading/Writing Energy Parameter Sets from/to File, [466](#)
- `vrna_params_load_RNA_Langdon2018`
  - Reading/Writing Energy Parameter Sets from/to File, [467](#)
- `vrna_params_load_RNA_misc_special_hairpins`
  - Reading/Writing Energy Parameter Sets from/to File, [467](#)
- `vrna_params_load_RNA_Turner1999`
  - Reading/Writing Energy Parameter Sets from/to File, [466](#)
- `vrna_params_load_RNA_Turner2004`
  - Reading/Writing Energy Parameter Sets from/to File, [465](#)
- `vrna_params_reset`
  - Energy Parameters, [225](#)
- `vrna_params_save`
  - Reading/Writing Energy Parameter Sets from/to File, [464](#)
- `vrna_params_subst`
  - Energy Parameters, [223](#)
- `vrna_path`
  - Folding Paths that start at a single Secondary Structure, [414](#)
- `VRNA_PATH_DEFAULT`
  - Folding Paths that start at a single Secondary Structure, [413](#)
- `vrna_path_direct`
  - Direct Refolding Paths between two Secondary Structures, [409](#)
- `vrna_path_direct_ub`
  - Direct Refolding Paths between two Secondary Structures, [410](#)
- `vrna_path_findpath`
  - Direct Refolding Paths between two Secondary Structures, [407](#)
- `vrna_path_findpath_saddle`
  - Direct Refolding Paths between two Secondary Structures, [405](#)
- `vrna_path_findpath_saddle_ub`
  - Direct Refolding Paths between two Secondary Structures, [406](#)
- `vrna_path_findpath_ub`
  - Direct Refolding Paths between two Secondary Structures, [408](#)
- `vrna_path_free`
  - (Re-)folding Paths, Saddle Points, Energy Barriers, and Local Minima, [403](#)
- `vrna_path_gradient`
  - Folding Paths that start at a single Secondary Structure, [415](#)
- `VRNA_PATH_NO_TRANSITION_OUTPUT`
  - Folding Paths that start at a single Secondary Structure, [413](#)
- `vrna_path_options_findpath`
  - Direct Refolding Paths between two Secondary Structures, [409](#)
- `vrna_path_options_free`
  - (Re-)folding Paths, Saddle Points, Energy Barriers, and Local Minima, [403](#)
- `VRNA_PATH_RANDOM`
  - Folding Paths that start at a single Secondary Structure, [413](#)
- `vrna_path_random`
  - Folding Paths that start at a single Secondary Structure, [415](#)
- `vrna_path_s`, [401](#)
  - type, [402](#)
- `VRNA_PATH_STEEPEST_DESCENT`
  - Folding Paths that start at a single Secondary Structure, [412](#)
- `VRNA_PATH_TYPE_DOT_BRACKET`
  - (Re-)folding Paths, Saddle Points, Energy Barriers, and Local Minima, [402](#)
- `VRNA_PATH_TYPE_MOVES`
  - (Re-)folding Paths, Saddle Points, Energy Barriers, and Local Minima, [403](#)
- `vrna_pbacktrack`
  - Random Structure Samples from the Ensemble, [341](#)
- `vrna_pbacktrack5`
  - Random Structure Samples from the Ensemble, [334](#)
- `vrna_pbacktrack5_cb`
  - Random Structure Samples from the Ensemble, [336](#)
- `vrna_pbacktrack5_num`
  - Random Structure Samples from the Ensemble, [335](#)
- `vrna_pbacktrack5_resume`
  - Random Structure Samples from the Ensemble, [337](#)
- `vrna_pbacktrack5_resume_cb`
  - Random Structure Samples from the Ensemble, [339](#)
- `vrna_pbacktrack5_TwoD`
  - Stochastic Backtracking of Structures from Distance Based Partitioning, [369](#)
- `vrna_pbacktrack_cb`
  - Random Structure Samples from the Ensemble, [343](#)

- VRNA\_PBACKTRACK\_DEFAULT
  - Random Structure Samples from the Ensemble, [332](#)
- vrna\_pbacktrack\_mem\_free
  - Random Structure Samples from the Ensemble, [347](#)
- vrna\_pbacktrack\_mem\_t
  - Random Structure Samples from the Ensemble, [333](#)
- VRNA\_PBACKTRACK\_NON\_REDUNDANT
  - Random Structure Samples from the Ensemble, [332](#)
- vrna\_pbacktrack\_num
  - Random Structure Samples from the Ensemble, [341](#)
- vrna\_pbacktrack\_resume
  - Random Structure Samples from the Ensemble, [344](#)
- vrna\_pbacktrack\_resume\_cb
  - Random Structure Samples from the Ensemble, [345](#)
- vrna\_pbacktrack\_TwoD
  - Stochastic Backtracking of Structures from Distance Based Partitioning, [368](#)
- vrna\_pf
  - Global Partition Function and Equilibrium Probabilities, [307](#)
- vrna\_pf\_alifold
  - Global Partition Function and Equilibrium Probabilities, [310](#)
- vrna\_pf\_circalifold
  - Global Partition Function and Equilibrium Probabilities, [310](#)
- vrna\_pf\_circfold
  - Global Partition Function and Equilibrium Probabilities, [309](#)
- vrna\_pf\_co\_fold
  - Global Partition Function and Equilibrium Probabilities, [312](#)
  - Partition Function for Two Hybridized Sequences, [457](#)
- vrna\_pf\_dimer
  - Global Partition Function and Equilibrium Probabilities, [307](#)
- vrna\_pf\_dimer\_concentrations
  - Partition Function for Two Hybridized Sequences, [458](#)
- vrna\_pf\_dimer\_probs
  - Predicting various thermodynamic properties, [377](#)
- vrna\_pf\_float\_precision
  - Partition Function and Equilibrium Properties, [289](#)
- vrna\_pf\_fold
  - Global Partition Function and Equilibrium Probabilities, [308](#)
- vrna\_pf\_TwoD
  - Computing Partition Functions of a Distance Based Partitioning, [366](#)
- vrna\_pfl\_fold
  - Local (sliding window) Partition Function and Equilibrium Probabilities, [318](#)
- vrna\_pfl\_fold\_cb
  - Local (sliding window) Partition Function and Equilibrium Probabilities, [319](#)
- vrna\_pfl\_fold\_up
  - Local (sliding window) Partition Function and Equilibrium Probabilities, [319](#)
- vrna\_pfl\_fold\_up\_cb
  - Local (sliding window) Partition Function and Equilibrium Probabilities, [320](#)
- vrna\_pinfo\_s, [518](#)
- vrna\_plist
  - Pair List Representation of Secondary Structures, [508](#)
- vrna\_plist\_from\_probs
  - Global Partition Function and Equilibrium Probabilities, [311](#)
- vrna\_plot\_coords
  - Layouts and Coordinates, [569](#)
- vrna\_plot\_coords\_circular
  - Layouts and Coordinates, [572](#)
- vrna\_plot\_coords\_circular\_pt
  - Layouts and Coordinates, [573](#)
- vrna\_plot\_coords\_navigate
  - Layouts and Coordinates, [574](#)
- vrna\_plot\_coords\_navigate\_pt
  - Layouts and Coordinates, [574](#)
- vrna\_plot\_coords\_pt
  - Layouts and Coordinates, [570](#)
- vrna\_plot\_coords\_puzzler
  - Layouts and Coordinates, [575](#)
- vrna\_plot\_coords\_puzzler\_pt
  - Layouts and Coordinates, [576](#)
- vrna\_plot\_coords\_simple
  - Layouts and Coordinates, [571](#)
- vrna\_plot\_coords\_simple\_pt
  - Layouts and Coordinates, [571](#)
- vrna\_plot\_coords\_turtle
  - Layouts and Coordinates, [578](#)
- vrna\_plot\_coords\_turtle\_pt
  - Layouts and Coordinates, [579](#)
- vrna\_plot\_layout
  - Layouts and Coordinates, [564](#)
- vrna\_plot\_layout\_circular
  - Layouts and Coordinates, [566](#)
- vrna\_plot\_layout\_free
  - Layouts and Coordinates, [568](#)
- vrna\_plot\_layout\_navigate
  - Layouts and Coordinates, [566](#)
- vrna\_plot\_layout\_puzzler
  - Layouts and Coordinates, [568](#)
- vrna\_plot\_layout\_s, [563](#)
- vrna\_plot\_layout\_simple
  - Layouts and Coordinates, [565](#)
- vrna\_plot\_layout\_t
  - Layouts and Coordinates, [564](#)
- vrna\_plot\_layout\_turtle



- Layouts and Coordinates, [567](#)
- `vrna_plot_options_puzzler`
  - Layouts and Coordinates, [577](#)
- `vrna_plot_options_puzzler_free`
  - Layouts and Coordinates, [577](#)
- `vrna_plot_options_puzzler_t`, [563](#)
- `VRNA_PLOT_TYPE_CIRCULAR`
  - Layouts and Coordinates, [563](#)
- `VRNA_PLOT_TYPE_NAVIEW`
  - Layouts and Coordinates, [563](#)
- `VRNA_PLOT_TYPE_SIMPLE`
  - Layouts and Coordinates, [563](#)
- `vrna_positional_entropy`
  - Predicting various thermodynamic properties, [376](#)
- `vrna_pr_energy`
  - Predicting various thermodynamic properties, [378](#)
- `vrna_pr_structure`
  - Predicting various thermodynamic properties, [378](#)
- `vrna_probs_window`
  - Local (sliding window) Partition Function and Equilibrium Probabilities, [317](#)
- `VRNA_PROBS_WINDOW_BPP`
  - Local (sliding window) Partition Function and Equilibrium Probabilities, [314](#)
- `vrna_probs_window_callback`
  - Local (sliding window) Partition Function and Equilibrium Probabilities, [316](#)
- `VRNA_PROBS_WINDOW_PF`
  - Local (sliding window) Partition Function and Equilibrium Probabilities, [315](#)
- `VRNA_PROBS_WINDOW_STACKP`
  - Local (sliding window) Partition Function and Equilibrium Probabilities, [315](#)
- `VRNA_PROBS_WINDOW_UP`
  - Local (sliding window) Partition Function and Equilibrium Probabilities, [314](#)
- `VRNA_PROBS_WINDOW_UP_SPLIT`
  - Local (sliding window) Partition Function and Equilibrium Probabilities, [315](#)
- `vrna_pt_pk_get`
  - Pair Table Representation of Secondary Structures, [504](#)
- `vrna_pt_pk_remove`
  - Pair Table Representation of Secondary Structures, [505](#)
- `vrna_pt_snoop_get`
  - Pair Table Representation of Secondary Structures, [505](#)
- `vrna_ptable`
  - Pair Table Representation of Secondary Structures, [503](#)
- `vrna_ptable_copy`
  - Pair Table Representation of Secondary Structures, [505](#)
- `vrna_ptable_from_string`
  - Pair Table Representation of Secondary Structures, [504](#)
- `vrna_ptypes`
  - Utilities to deal with Nucleotide Alphabets, [479](#)
- `vrna_random_string`
  - (Nucleic Acid Sequence) String Utilities, [487](#)
- `vrna_read_line`
  - Files and I/O, [527](#)
- `vrna_realloc`
  - Utilities, [439](#)
- `vrna_refBPcnt_matrix`
  - Secondary Structure Utilities, [492](#)
- `vrna_refBPdist_matrix`
  - Secondary Structure Utilities, [493](#)
- `vrna_rotational_symmetry`
  - Combinatorics Algorithms, [591](#)
- `vrna_rotational_symmetry_db`
  - Combinatorics Algorithms, [592](#)
- `vrna_rotational_symmetry_db_pos`
  - Combinatorics Algorithms, [593](#)
- `vrna_rotational_symmetry_num`
  - Combinatorics Algorithms, [589](#)
- `vrna_rotational_symmetry_pos`
  - Combinatorics Algorithms, [591](#)
- `vrna_rotational_symmetry_pos_num`
  - Combinatorics Algorithms, [590](#)
- `vrna_sc_add_bp`
  - Soft Constraints, [281](#)
- `vrna_sc_add_bt`
  - Soft Constraints, [284](#)
- `vrna_sc_add_data`
  - Soft Constraints, [283](#)
- `vrna_sc_add_exp_f`
  - Soft Constraints, [285](#)
- `vrna_sc_add_f`
  - Soft Constraints, [284](#)
- `vrna_sc_add_hi_motif`
  - Incorporating Ligands Binding to Specific Sequence/Structure Motifs using Soft Constraints, [429](#)
- `vrna_sc_add_SHAPE_deigan`
  - SHAPE Reactivity Data, [418](#)
- `vrna_sc_add_SHAPE_deigan_all`
  - SHAPE Reactivity Data, [419](#)
- `vrna_sc_add_SHAPE_zarringhalam`
  - SHAPE Reactivity Data, [420](#)
- `vrna_sc_add_up`
  - Soft Constraints, [282](#)
- `vrna_sc_bp_storage_t`, [723](#)
- `VRNA_SC_DEFAULT`
  - `soft.h`, [756](#)
- `vrna_sc_free`
  - Soft Constraints, [283](#)
- `vrna_sc_init`
  - Soft Constraints, [279](#)
- `vrna_sc_minimize_perturbation`
  - Generate Soft Constraints from Data, [425](#)
- `vrna_sc_motif_s`, [723](#)
- `vrna_sc_motif_t`
  - `ligand.h`, [752](#)
- `vrna_sc_remove`

- Soft Constraints, [282](#)
- `vrna_sc_s`, [275](#)
  - `bt`, [276](#)
  - `exp_f`, [277](#)
  - `f`, [276](#)
- `vrna_sc_set_bp`
  - Soft Constraints, [280](#)
- `vrna_sc_set_up`
  - Soft Constraints, [281](#)
- `vrna_sc_SHAPE_parse_method`
  - SHAPE.h, [753](#)
- `vrna_sc_SHAPE_to_pr`
  - SHAPE Reactivity Data, [421](#)
- `vrna_sc_type_e`
  - soft.h, [755](#)
- `VRNA_SC_WINDOW`
  - soft.h, [756](#)
- `vrna_search_BM_BCT`
  - Search Algorithms, [587](#)
- `vrna_search_BM_BCT_num`
  - Search Algorithms, [586](#)
- `vrna_search_BMH`
  - Search Algorithms, [585](#)
- `vrna_search_BMH_num`
  - Search Algorithms, [584](#)
- `vrna_sect_s`, [597](#)
- `VRNA_SEQ_DNA`
  - Utilities to deal with Nucleotide Alphabets, [479](#)
- `vrna_seq_encode`
  - Utilities to deal with Nucleotide Alphabets, [479](#)
- `VRNA_SEQ_RNA`
  - Utilities to deal with Nucleotide Alphabets, [479](#)
- `vrna_seq_toRNA`
  - (Nucleic Acid Sequence) String Utilites, [488](#)
- `vrna_seq_toupper`
  - (Nucleic Acid Sequence) String Utilites, [488](#)
- `vrna_seq_type_e`
  - Utilities to deal with Nucleotide Alphabets, [479](#)
- `vrna_seq_ungapped`
  - (Nucleic Acid Sequence) String Utilites, [489](#)
- `VRNA_SEQ_UNKNOWN`
  - Utilities to deal with Nucleotide Alphabets, [479](#)
- `vrna_sequence_s`, [478](#)
- `vrna_sol_TwoD_pf_t`, [365](#)
  - Computing Partition Functions of a Distance Based Partitioning, [366](#)
- `vrna_sol_TwoD_t`, [358](#)
  - Computing MFE representatives of a Distance Based Partitioning, [360](#)
- `vrna_stack_prob`
  - Predicting various thermodynamic properties, [376](#)
- `VRNA_STATUS_MFE_POST`
  - The Fold Compound, [621](#)
- `VRNA_STATUS_MFE_PRE`
  - The Fold Compound, [621](#)
- `VRNA_STATUS_PF_POST`
  - The Fold Compound, [622](#)
- `VRNA_STATUS_PF_PRE`
  - The Fold Compound, [622](#)
- `vrna_strcat_printf`
  - (Nucleic Acid Sequence) String Utilites, [485](#)
- `vrna_strcat_vprintf`
  - (Nucleic Acid Sequence) String Utilites, [485](#)
- `vrna_strdup_printf`
  - (Nucleic Acid Sequence) String Utilites, [483](#)
- `vrna_strdup_vprintf`
  - (Nucleic Acid Sequence) String Utilites, [484](#)
- `vrna_strsplit`
  - (Nucleic Acid Sequence) String Utilites, [486](#)
- `VRNA_STRUCTURE_TREE_EXPANDED`
  - Tree Representation of Secondary Structures, [514](#)
- `VRNA_STRUCTURE_TREE_HIT`
  - Tree Representation of Secondary Structures, [512](#)
- `VRNA_STRUCTURE_TREE_SHAPIRO`
  - Tree Representation of Secondary Structures, [513](#)
- `VRNA_STRUCTURE_TREE_SHAPIRO_EXT`
  - Tree Representation of Secondary Structures, [513](#)
- `VRNA_STRUCTURE_TREE_SHAPIRO_SHORT`
  - Tree Representation of Secondary Structures, [513](#)
- `VRNA_STRUCTURE_TREE_SHAPIRO_WEIGHT`
  - Tree Representation of Secondary Structures, [513](#)
- `vrna_structured_domains_s`, [723](#)
- `vrna_subopt`
  - Suboptimal Structures within an Energy Band around the MFE, [327](#)
- `vrna_subopt_callback`
  - Suboptimal Structures within an Energy Band around the MFE, [326](#)
- `vrna_subopt_cb`
  - Suboptimal Structures within an Energy Band around the MFE, [328](#)
- `vrna_subopt_sol_s`, [724](#)
- `vrna_subopt_zuker`
  - Suboptimal Structures sensu Stiegler et al. 1984 / Zuker et al. 1989, [324](#)
- `vrna_time_stamp`
  - Utilities, [440](#)
- `vrna_tree_string_to_db`
  - Tree Representation of Secondary Structures, [515](#)
- `vrna_tree_string_unweight`
  - Tree Representation of Secondary Structures, [515](#)
- `vrna_ud_add_motif`
  - Unstructured Domains, [239](#)
- `vrna_ud_motifs_centroid`
  - Unstructured Domains, [237](#)
- `vrna_ud_motifs_MEA`
  - Unstructured Domains, [238](#)
- `vrna_ud_motifs_MFE`
  - Unstructured Domains, [238](#)
- `vrna_ud_remove`
  - Unstructured Domains, [240](#)
- `vrna_ud_set_data`
  - Unstructured Domains, [240](#)
- `vrna_ud_set_exp_prod_rule_cb`
  - Unstructured Domains, [242](#)
- `vrna_ud_set_prob_cb`



- unstructured\_domains.h, [869](#)
- vrna\_ud\_set\_prod\_rule\_cb
  - Unstructured Domains, [241](#)
- VRNA\_UNIT\_CAL
  - Unit Conversion, [608](#)
- VRNA\_UNIT\_CAL\_IT
  - Unit Conversion, [608](#)
- VRNA\_UNIT\_DACAL
  - Unit Conversion, [608](#)
- VRNA\_UNIT\_DACAL\_IT
  - Unit Conversion, [608](#)
- VRNA\_UNIT\_DEG\_C
  - Unit Conversion, [609](#)
- VRNA\_UNIT\_DEG\_DE
  - Unit Conversion, [609](#)
- VRNA\_UNIT\_DEG\_F
  - Unit Conversion, [609](#)
- VRNA\_UNIT\_DEG\_N
  - Unit Conversion, [609](#)
- VRNA\_UNIT\_DEG\_R
  - Unit Conversion, [609](#)
- VRNA\_UNIT\_DEG\_RE
  - Unit Conversion, [609](#)
- VRNA\_UNIT\_DEG\_RO
  - Unit Conversion, [609](#)
- vrna\_unit\_energy\_e
  - Unit Conversion, [608](#)
- VRNA\_UNIT\_EV
  - Unit Conversion, [608](#)
- VRNA\_UNIT\_G\_TNT
  - Unit Conversion, [608](#)
- VRNA\_UNIT\_J
  - Unit Conversion, [608](#)
- VRNA\_UNIT\_K
  - Unit Conversion, [609](#)
- VRNA\_UNIT\_KCAL
  - Unit Conversion, [608](#)
- VRNA\_UNIT\_KCAL\_IT
  - Unit Conversion, [608](#)
- VRNA\_UNIT\_KG\_TNT
  - Unit Conversion, [608](#)
- VRNA\_UNIT\_KJ
  - Unit Conversion, [608](#)
- VRNA\_UNIT\_KWH
  - Unit Conversion, [608](#)
- VRNA\_UNIT\_T\_TNT
  - Unit Conversion, [608](#)
- vrna\_unit\_temperature\_e
  - Unit Conversion, [608](#)
- VRNA\_UNIT\_WH
  - Unit Conversion, [608](#)
- vrna\_unstructured\_domain\_motif\_s, [724](#)
- vrna\_unstructured\_domain\_s, [234](#)
  - prod\_cb, [235](#)
- vrna\_urn
  - Utilities, [439](#)
- warn\_user
  - basic.h, [819](#)
- write\_parameter\_file
  - Reading/Writing Energy Parameter Sets from/to File, [470](#)
- xrealloc
  - basic.h, [820](#)
- xrna\_plot
  - Plotting, [559](#)
- xsubi
  - Utilities, [442](#)
- zukersubopt
  - Suboptimal Structures sensu Stiegler et al. 1984 / Zuker et al. 1989, [325](#)
- zukersubopt\_par
  - Suboptimal Structures sensu Stiegler et al. 1984 / Zuker et al. 1989, [325](#)