

RNAlib-2.2.6

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>ViennaRNA Package core - RNAlib</b>	<b>1</b>
1.1	Introduction . . . . .	2
<b>2</b>	<b>Parsing and Comparing - Functions to Manipulate Structures</b>	<b>3</b>
<b>3</b>	<b>Utilities - Odds and Ends</b>	<b>7</b>
3.1	Producing secondary structure graphs . . . . .	7
3.2	Producing (colored) dot plots for base pair probabilities . . . . .	8
3.3	Producing (colored) alignments . . . . .	9
3.4	RNA sequence related utilities . . . . .	9
3.5	RNA secondary structure related utilities . . . . .	9
3.6	Miscellaneous Utilities . . . . .	10
<b>4</b>	<b>RNAlib API v3.0</b>	<b>13</b>
4.1	Introduction . . . . .	13
4.2	What are the major changes? . . . . .	13
4.3	How to port your program to the new API . . . . .	13
4.4	Some Examples using RNAlib API v3.0 . . . . .	14
<b>5</b>	<b>Scripting Language interface(s)</b>	<b>15</b>
5.1	Introduction . . . . .	15
5.1.1	Function renaming scheme . . . . .	15
5.1.2	Object oriented Interface for data structures . . . . .	15
5.2	Examples . . . . .	15
5.2.1	Perl Examples . . . . .	15
5.2.1.1	Using the Flat Interface . . . . .	15
5.2.1.2	Using the Object Oriented (OO) Interface . . . . .	16
5.2.2	Python Examples . . . . .	16

<b>6</b>	<b>Input / Output File Formats</b>	<b>17</b>
6.1	File formats for Secondary Structure Constraints . . . . .	17
6.1.1	Constraints Definition File . . . . .	17
6.1.1.1	Constraint commands . . . . .	17
6.1.1.2	Specification of the loop type context . . . . .	17
6.1.1.3	Controlling the orientation of base pairing . . . . .	18
6.1.1.4	Sequence coordinates . . . . .	18
6.1.1.5	Valid constraint commands . . . . .	18
<b>7</b>	<b>Example - A Small Example Program</b>	<b>21</b>
<b>8</b>	<b>Deprecated List</b>	<b>23</b>
<b>9</b>	<b>Bug List</b>	<b>33</b>
<b>10</b>	<b>Module Index</b>	<b>35</b>
10.1	Modules . . . . .	35
<b>11</b>	<b>Data Structure Index</b>	<b>37</b>
11.1	Data Structures . . . . .	37
<b>12</b>	<b>File Index</b>	<b>39</b>
12.1	File List . . . . .	39
<b>13</b>	<b>Module Documentation</b>	<b>43</b>
13.1	RNA Secondary Structure Prediction . . . . .	43
13.1.1	Detailed Description . . . . .	44
13.2	Inverse Secondary Structure Prediction . . . . .	45
13.2.1	Detailed Description . . . . .	45
13.2.2	Function Documentation . . . . .	45
13.2.2.1	inverse_fold(char *start, const char *target) . . . . .	45
13.2.2.2	inverse_pf_fold(char *start, const char *target) . . . . .	46
13.2.3	Variable Documentation . . . . .	46
13.2.3.1	final_cost . . . . .	46

13.2.3.2	give_up	46
13.2.3.3	inv_verbose	46
13.3	Refolding paths between secondary structures	47
13.3.1	Detailed Description	47
13.4	Free Energy Evaluation for given Sequence / Structure Pairs	48
13.4.1	Detailed Description	49
13.4.2	Function Documentation	49
13.4.2.1	vrna_eval_structure(vrna_fold_compound_t *vc, const char *structure)	49
13.4.2.2	vrna_eval_covar_structure(vrna_fold_compound_t *vc, const char *structure)	50
13.4.2.3	vrna_eval_structure_simple(const char *string, const char *structure)	50
13.4.2.4	vrna_eval_structure_verbose(vrna_fold_compound_t *vc, const char *structure, FILE *file)	51
13.4.2.5	vrna_eval_structure_simple_verbose(const char *string, const char *structure, FILE *file)	51
13.4.2.6	vrna_eval_structure_pt(vrna_fold_compound_t *vc, const short *pt)	52
13.4.2.7	vrna_eval_structure_pt_simple(const char *string, const short *pt)	52
13.4.2.8	vrna_eval_structure_pt_verbose(vrna_fold_compound_t *vc, const short *pt, FILE *file)	53
13.4.2.9	vrna_eval_structure_pt_simple_verbose(const char *string, const short *pt, FILE *file)	53
13.4.2.10	vrna_eval_loop_pt(vrna_fold_compound_t *vc, int i, const short *pt)	54
13.4.2.11	vrna_eval_move(vrna_fold_compound_t *vc, const char *structure, int m1, int m2)	54
13.4.2.12	vrna_eval_move_pt(vrna_fold_compound_t *vc, short *pt, int m1, int m2)	55
13.4.2.13	energy_of_structure(const char *string, const char *structure, int verbosity_level)	55
13.4.2.14	energy_of_struct_par(const char *string, const char *structure, vrna_param_t *parameters, int verbosity_level)	56
13.4.2.15	energy_of_circ_structure(const char *string, const char *structure, int verbosity_level)	57
13.4.2.16	energy_of_circ_struct_par(const char *string, const char *structure, vrna_param_t *parameters, int verbosity_level)	57
13.4.2.17	energy_of_structure_pt(const char *string, short *ptable, short *s, short *s1, int verbosity_level)	58
13.4.2.18	energy_of_struct_pt_par(const char *string, short *ptable, short *s, short *s1, vrna_param_t *parameters, int verbosity_level)	58

13.4.2.19	energy_of_move(const char *string, const char *structure, int m1, int m2) . . . . .	59
13.4.2.20	energy_of_move_pt(short *pt, short *s, short *s1, int m1, int m2) . . . . .	59
13.4.2.21	loop_energy(short *ptable, short *s, short *s1, int i) . . . . .	60
13.4.2.22	energy_of_struct(const char *string, const char *structure) . . . . .	60
13.4.2.23	energy_of_struct_pt(const char *string, short *ptable, short *s, short *s1) . . . . .	61
13.4.2.24	energy_of_circ_struct(const char *string, const char *structure) . . . . .	62
13.4.2.25	vrna_eval_hp_loop(vrna_fold_compound_t *vc, int i, int j) . . . . .	62
13.5	Processing and Evaluating Decomposed Loops . . . . .	64
13.5.1	Detailed Description . . . . .	65
13.5.2	Function Documentation . . . . .	65
13.5.2.1	E_ExtLoop(int type, int si1, int sj1, vrna_param_t *P) . . . . .	65
13.5.2.2	exp_E_ExtLoop(int type, int si1, int sj1, vrna_exp_param_t *P) . . . . .	65
13.5.2.3	E_Stem(int type, int si1, int sj1, int extLoop, vrna_param_t *P) . . . . .	66
13.5.2.4	exp_E_Stem(int type, int si1, int sj1, int extLoop, vrna_exp_param_t *P) . . . . .	67
13.5.2.5	get_gquad_matrix(short *S, vrna_param_t *P) . . . . .	67
13.5.2.6	parse_gquad(const char *struc, int *L, int l[3]) . . . . .	68
13.5.2.7	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) . . . . .	68
13.5.2.8	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) . . . . .	68
13.5.2.9	E_Hairpin(int size, int type, int si1, int sj1, const char *string, vrna_param_t *P) . . . . .	69
13.5.2.10	exp_E_Hairpin(int u, int type, short si1, short sj1, const char *string, vrna_exp_param_t *P) . . . . .	70
13.5.2.11	vrna_E_hp_loop(vrna_fold_compound_t *vc, int i, int j) . . . . .	70
13.5.2.12	vrna_E_ext_hp_loop(vrna_fold_compound_t *vc, int i, int j) . . . . .	71
13.5.2.13	vrna_exp_E_hp_loop(vrna_fold_compound_t *vc, int i, int j) . . . . .	71
13.5.2.14	vrna_BT_hp_loop(vrna_fold_compound_t *vc, int i, int j, int en, vrna_bp_stack_t *bp_stack, int *stack_count) . . . . .	71
13.5.2.15	E_IntLoop(int n1, int n2, int type, int type_2, int si1, int sj1, int sp1, int sq1, vrna_param_t *P) . . . . .	71
13.5.2.16	exp_E_IntLoop(int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short sq1, vrna_exp_param_t *P) . . . . .	73
13.5.2.17	E_mb_loop_stack(int i, int j, vrna_fold_compound_t *vc) . . . . .	73

13.5.2.18 vrna_BT_mb_loop(vrna_fold_compound_t *vc, int *i, int *j, int *k, int en, int *component1, int *component2)	74
13.6 Energy Parameter Sets and Boltzmann Factors	75
13.6.1 Detailed Description	76
13.6.2 Data Structure Documentation	77
13.6.2.1 struct vrna_param_s	77
13.6.2.2 struct vrna_exp_param_s	77
13.6.3 Typedef Documentation	78
13.6.3.1 paramT	78
13.6.3.2 pf_paramT	78
13.6.4 Function Documentation	78
13.6.4.1 vrna_params(vrna_md_t *md)	78
13.6.4.2 vrna_params_copy(vrna_param_t *par)	79
13.6.4.3 vrna_exp_params(vrna_md_t *md)	79
13.6.4.4 vrna_exp_params_comparative(unsigned int n_seq, vrna_md_t *md)	80
13.6.4.5 vrna_exp_params_copy(vrna_exp_param_t *par)	80
13.6.4.6 vrna_params_subst(vrna_fold_compound_t *vc, vrna_param_t *par)	81
13.6.4.7 vrna_exp_params_subst(vrna_fold_compound_t *vc, vrna_exp_param_t *params)	81
13.6.4.8 vrna_exp_params_rescale(vrna_fold_compound_t *vc, double *mfe)	81
13.6.4.9 vrna_params_reset(vrna_fold_compound_t *vc, vrna_md_t *md_p)	82
13.6.4.10 vrna_exp_params_reset(vrna_fold_compound_t *vc, vrna_md_t *md_p)	83
13.6.4.11 get_scaled_pf_parameters(void)	83
13.6.4.12 get_boltzmann_factors(double temperature, double betaScale, vrna_md_t md, double pf_scale)	83
13.6.4.13 get_boltzmann_factor_copy(vrna_exp_param_t *parameters)	84
13.6.4.14 get_scaled_alipf_parameters(unsigned int n_seq)	84
13.6.4.15 get_boltzmann_factors_alipf(unsigned int n_seq, double temperature, double betaScale, vrna_md_t md, double pf_scale)	85
13.6.4.16 scale_parameters(void)	85
13.6.4.17 get_scaled_parameters(double temperature, vrna_md_t md)	85
13.7 Manipulation of the Prediction Models	87
13.7.1 Detailed Description	91

13.7.2	Data Structure Documentation	91
13.7.2.1	struct vrna_md_s	91
13.7.3	Macro Definition Documentation	93
13.7.3.1	VRNA_MODEL_DEFAULT_TEMPERATURE	93
13.7.3.2	VRNA_MODEL_DEFAULT_PF_SCALE	93
13.7.3.3	VRNA_MODEL_DEFAULT_BETA_SCALE	93
13.7.3.4	VRNA_MODEL_DEFAULT_DANGLES	93
13.7.3.5	VRNA_MODEL_DEFAULT_SPECIAL_HP	94
13.7.3.6	VRNA_MODEL_DEFAULT_NO_LP	94
13.7.3.7	VRNA_MODEL_DEFAULT_NO_GU	94
13.7.3.8	VRNA_MODEL_DEFAULT_NO_GU_CLOSURE	94
13.7.3.9	VRNA_MODEL_DEFAULT_CIRC	94
13.7.3.10	VRNA_MODEL_DEFAULT_GQUAD	95
13.7.3.11	VRNA_MODEL_DEFAULT_UNIQ_ML	95
13.7.3.12	VRNA_MODEL_DEFAULT_ENERGY_SET	95
13.7.3.13	VRNA_MODEL_DEFAULT_BACKTRACK	95
13.7.3.14	VRNA_MODEL_DEFAULT_BACKTRACK_TYPE	95
13.7.3.15	VRNA_MODEL_DEFAULT_COMPUTE_BPP	96
13.7.3.16	VRNA_MODEL_DEFAULT_MAX_BP_SPAN	96
13.7.3.17	VRNA_MODEL_DEFAULT_WINDOW_SIZE	96
13.7.3.18	VRNA_MODEL_DEFAULT_LOG_ML	96
13.7.3.19	VRNA_MODEL_DEFAULT_ALI_OLD_EN	96
13.7.3.20	VRNA_MODEL_DEFAULT_ALI_RIBO	97
13.7.3.21	VRNA_MODEL_DEFAULT_ALI_CV_FACT	97
13.7.3.22	VRNA_MODEL_DEFAULT_ALI_NC_FACT	97
13.7.4	Function Documentation	97
13.7.4.1	vrna_md_set_default(vrna_md_t *md)	97
13.7.4.2	vrna_md_update(vrna_md_t *md)	97
13.7.4.3	vrna_md_option_string(vrna_md_t *md)	98
13.7.4.4	vrna_md_defaults_reset(vrna_md_t *md_p)	98



13.7.4.5	<code>vrna_md_defaults_temperature(double T)</code>	99
13.7.4.6	<code>vrna_md_defaults_temperature_get(void)</code>	99
13.7.4.7	<code>vrna_md_defaults_betaScale(double b)</code>	99
13.7.4.8	<code>vrna_md_defaults_betaScale_get(void)</code>	100
13.7.4.9	<code>vrna_md_defaults_dangles(int d)</code>	100
13.7.4.10	<code>vrna_md_defaults_dangles_get(void)</code>	100
13.7.4.11	<code>vrna_md_defaults_special_hp(int flag)</code>	101
13.7.4.12	<code>vrna_md_defaults_special_hp_get(void)</code>	102
13.7.4.13	<code>vrna_md_defaults_noLP(int flag)</code>	102
13.7.4.14	<code>vrna_md_defaults_noLP_get(void)</code>	102
13.7.4.15	<code>vrna_md_defaults_noGU(int flag)</code>	103
13.7.4.16	<code>vrna_md_defaults_noGU_get(void)</code>	103
13.7.4.17	<code>vrna_md_defaults_noGUclosure(int flag)</code>	103
13.7.4.18	<code>vrna_md_defaults_noGUclosure_get(void)</code>	104
13.7.4.19	<code>vrna_md_defaults_logML(int flag)</code>	104
13.7.4.20	<code>vrna_md_defaults_logML_get(void)</code>	104
13.7.4.21	<code>vrna_md_defaults_circ(int flag)</code>	105
13.7.4.22	<code>vrna_md_defaults_circ_get(void)</code>	106
13.7.4.23	<code>vrna_md_defaults_gquad(int flag)</code>	106
13.7.4.24	<code>vrna_md_defaults_gquad_get(void)</code>	106
13.7.4.25	<code>vrna_md_defaults_uniq_ML(int flag)</code>	107
13.7.4.26	<code>vrna_md_defaults_uniq_ML_get(void)</code>	107
13.7.4.27	<code>vrna_md_defaults_energy_set(int e)</code>	107
13.7.4.28	<code>vrna_md_defaults_energy_set_get(void)</code>	108
13.7.4.29	<code>vrna_md_defaults_backtrack(int flag)</code>	108
13.7.4.30	<code>vrna_md_defaults_backtrack_get(void)</code>	108
13.7.4.31	<code>vrna_md_defaults_backtrack_type(char t)</code>	109
13.7.4.32	<code>vrna_md_defaults_backtrack_type_get(void)</code>	109
13.7.4.33	<code>vrna_md_defaults_compute_bpp(int flag)</code>	109
13.7.4.34	<code>vrna_md_defaults_compute_bpp_get(void)</code>	110

13.7.4.35 vrna_md_defaults_max_bp_span(int span) . . . . .	110
13.7.4.36 vrna_md_defaults_max_bp_span_get(void) . . . . .	110
13.7.4.37 vrna_md_defaults_min_loop_size(int size) . . . . .	111
13.7.4.38 vrna_md_defaults_min_loop_size_get(void) . . . . .	111
13.7.4.39 vrna_md_defaults_window_size(int size) . . . . .	111
13.7.4.40 vrna_md_defaults_window_size_get(void) . . . . .	111
13.7.4.41 vrna_md_defaults_oldAliEn(int flag) . . . . .	112
13.7.4.42 vrna_md_defaults_oldAliEn_get(void) . . . . .	112
13.7.4.43 vrna_md_defaults_ribo(int flag) . . . . .	112
13.7.4.44 vrna_md_defaults_ribo_get(void) . . . . .	113
13.7.4.45 vrna_md_defaults_cv_fact(double factor) . . . . .	113
13.7.4.46 vrna_md_defaults_cv_fact_get(void) . . . . .	113
13.7.4.47 vrna_md_defaults_nc_fact(double factor) . . . . .	114
13.7.4.48 vrna_md_defaults_nc_fact_get(void) . . . . .	114
13.7.4.49 vrna_md_defaults_sfact(double factor) . . . . .	114
13.7.4.50 vrna_md_defaults_sfact_get(void) . . . . .	114
13.7.4.51 set_model_details(vrna_md_t *md) . . . . .	115
13.7.5 Variable Documentation . . . . .	115
13.7.5.1 temperature . . . . .	115
13.7.5.2 pf_scale . . . . .	115
13.7.5.3 dangles . . . . .	116
13.7.5.4 tetra_loop . . . . .	116
13.7.5.5 noLonelyPairs . . . . .	116
13.7.5.6 canonicalBPonly . . . . .	116
13.7.5.7 energy_set . . . . .	116
13.7.5.8 do_backtrack . . . . .	117
13.7.5.9 backtrack_type . . . . .	117
13.7.5.10 nonstandards . . . . .	117
13.7.5.11 max_bp_span . . . . .	117
13.8 Constraining the Secondary Structure Predictions and Evaluations . . . . .	118

13.8.1 Detailed Description . . . . .	120
13.8.2 Macro Definition Documentation . . . . .	122
13.8.2.1 VRNA_CONSTRAINT_FILE . . . . .	122
13.8.2.2 VRNA_CONSTRAINT_SOFT_MFE . . . . .	122
13.8.2.3 VRNA_CONSTRAINT_SOFT_PF . . . . .	122
13.8.2.4 VRNA_DECOMP_PAIR_HP . . . . .	122
13.8.2.5 VRNA_DECOMP_PAIR_IL . . . . .	123
13.8.2.6 VRNA_DECOMP_PAIR_ML . . . . .	124
13.8.2.7 VRNA_DECOMP_ML_ML_ML . . . . .	124
13.8.2.8 VRNA_DECOMP_ML_STEM . . . . .	125
13.8.2.9 VRNA_DECOMP_ML_ML . . . . .	125
13.8.2.10 VRNA_DECOMP_ML_UP . . . . .	126
13.8.2.11 VRNA_DECOMP_ML_ML_STEM . . . . .	126
13.8.2.12 VRNA_DECOMP_ML_COAXIAL . . . . .	127
13.8.2.13 VRNA_DECOMP_EXT_EXT . . . . .	127
13.8.2.14 VRNA_DECOMP_EXT_UP . . . . .	127
13.8.2.15 VRNA_DECOMP_EXT_STEM . . . . .	128
13.8.2.16 VRNA_DECOMP_EXT_EXT_EXT . . . . .	128
13.8.2.17 VRNA_DECOMP_EXT_STEM_EXT . . . . .	128
13.8.2.18 VRNA_DECOMP_EXT_EXT_STEM . . . . .	129
13.8.2.19 VRNA_DECOMP_EXT_EXT_STEM1 . . . . .	129
13.8.3 Function Documentation . . . . .	129
13.8.3.1 vrna_constraints_add(vrna_fold_compound_t *vc, const char *constraint, unsigned int options) . . . . .	129
13.8.3.2 vrna_message_constraint_options(unsigned int option) . . . . .	130
13.8.3.3 vrna_message_constraint_options_all(void) . . . . .	131
13.9 Data Structures and Preprocessor Macros . . . . .	132
13.9.1 Detailed Description . . . . .	133
13.9.2 Data Structure Documentation . . . . .	133
13.9.2.1 struct vrna_basepair_s . . . . .	133
13.9.2.2 struct vrna_plist_s . . . . .	133

13.9.2.3	struct vrna_cpair_s . . . . .	134
13.9.2.4	struct vrna_sect_s . . . . .	134
13.9.2.5	struct vrna_bp_stack_s . . . . .	134
13.9.2.6	struct pu_contrib . . . . .	134
13.9.2.7	struct interact . . . . .	134
13.9.2.8	struct pu_out . . . . .	135
13.9.2.9	struct constrain . . . . .	135
13.9.2.10	struct duplexT . . . . .	135
13.9.2.11	struct node . . . . .	135
13.9.2.12	struct snoopT . . . . .	135
13.9.2.13	struct dupVar . . . . .	135
13.9.3	Typedef Documentation . . . . .	135
13.9.3.1	PAIR . . . . .	135
13.9.3.2	plist . . . . .	136
13.9.3.3	cpair . . . . .	136
13.9.3.4	sect . . . . .	136
13.9.3.5	bondT . . . . .	136
13.10	Utilities . . . . .	137
13.10.1	Detailed Description . . . . .	139
13.10.2	Macro Definition Documentation . . . . .	139
13.10.2.1	VRNA_INPUT_FASTA_HEADER . . . . .	139
13.10.2.2	VRNA_INPUT_CONSTRAINT . . . . .	139
13.10.3	Function Documentation . . . . .	139
13.10.3.1	vrna_alloc(unsigned size) . . . . .	139
13.10.3.2	vrna_realloc(void *p, unsigned size) . . . . .	140
13.10.3.3	vrna_message_error(const char message[]) . . . . .	140
13.10.3.4	vrna_message_warning(const char message[]) . . . . .	140
13.10.3.5	vrna_urn(void) . . . . .	141
13.10.3.6	vrna_int_urn(int from, int to) . . . . .	141
13.10.3.7	vrna_time_stamp(void) . . . . .	141

13.10.3.8	get_line(FILE *fp)	142
13.10.3.9	get_input_line(char **string, unsigned int options)	142
13.10.3.10	vrna_message_input_seq_simple(void)	143
13.10.3.11	vrna_message_input_seq(const char *s)	143
13.10.3.12	vrna_idx_row_wise(unsigned int length)	143
13.10.3.13	vrna_idx_col_wise(unsigned int length)	144
13.10.4	Variable Documentation	144
13.10.4.1	xsubi	144
13.11	Computing Minimum Free Energy (MFE) Structures	145
13.11.1	Detailed Description	145
13.11.2	Function Documentation	146
13.11.2.1	vrna_mfe(vrna_fold_compound_t *vc, char *structure)	146
13.12	Computing Partition Functions and Pair Probabilities	147
13.12.1	Detailed Description	148
13.12.2	Function Documentation	149
13.12.2.1	vrna_pf(vrna_fold_compound_t *vc, char *structure)	149
13.12.2.2	vrna_pf_fold(const char *seq, char *structure, vrna_plist_t **pl)	149
13.12.2.3	vrna_pf_circfold(const char *seq, char *structure, vrna_plist_t **pl)	150
13.12.2.4	vrna_mean_bp_distance_pr(int length, FLT_OR_DBL *pr)	150
13.12.2.5	vrna_mean_bp_distance(vrna_fold_compound_t *vc)	151
13.12.2.6	vrna_stack_prob(vrna_fold_compound_t *vc, double cutoff)	151
13.12.2.7	pf_fold_par(const char *sequence, char *structure, vrna_exp_param_t *parameters, int calculate_bppm, int is_constrained, int is_circular)	152
13.12.2.8	pf_fold(const char *sequence, char *structure)	153
13.12.2.9	pf_circ_fold(const char *sequence, char *structure)	154
13.12.2.10	free_pf_arrays(void)	154
13.12.2.11	update_pf_params(int length)	155
13.12.2.12	update_pf_params_par(int length, vrna_exp_param_t *parameters)	155
13.12.2.13	export_bppm(void)	156
13.12.2.14	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)	156

13.12.2.15	mean_bp_distance(int length)	157
13.12.2.16	mean_bp_distance_pr(int length, FLT_OR_DBL *pr)	157
13.12.2.17	vrna_plist_from_probs(vrna_fold_compound_t *vc, double cut_off)	158
13.12.2.18	assign_plist_from_pr(vrna_plist_t **pl, FLT_OR_DBL *probs, int length, double cutoff)	158
13.13	Compute the structure with maximum expected accuracy (MEA)	159
13.14	Compute the centroid structure	160
13.14.1	Detailed Description	160
13.14.2	Function Documentation	160
13.14.2.1	vrna_centroid(vrna_fold_compound_t *vc, double *dist)	160
13.14.2.2	vrna_centroid_from_plist(int length, double *dist, vrna_plist_t *pl)	161
13.14.2.3	vrna_centroid_from_probs(int length, double *dist, FLT_OR_DBL *probs)	161
13.15	Enumerating Suboptimal Structures	162
13.15.1	Detailed Description	162
13.16	Suboptimal structures according to Zuker et al. 1989	163
13.16.1	Detailed Description	163
13.16.2	Function Documentation	163
13.16.2.1	vrna_subopt_zuker(vrna_fold_compound_t *vc)	163
13.16.2.2	zukersubopt(const char *string)	164
13.16.2.3	zukersubopt_par(const char *string, vrna_param_t *parameters)	164
13.17	Suboptimal structures within an energy band around the MFE	165
13.17.1	Detailed Description	165
13.17.2	Function Documentation	165
13.17.2.1	vrna_subopt(vrna_fold_compound_t *vc, int delta, int sorted, FILE *fp)	165
13.17.2.2	subopt(char *seq, char *structure, int delta, FILE *fp)	166
13.17.2.3	subopt_circ(char *seq, char *sequence, int delta, FILE *fp)	166
13.18	Stochastic backtracking in the Ensemble	168
13.18.1	Detailed Description	168
13.18.2	Function Documentation	168
13.18.2.1	vrna_pbacktrack5(vrna_fold_compound_t *vc, int length)	168
13.18.2.2	vrna_pbacktrack(vrna_fold_compound_t *vc)	169

13.18.2.3 pbacktrack(char *sequence) . . . . .	169
13.18.2.4 pbacktrack_circ(char *sequence) . . . . .	170
13.18.3 Variable Documentation . . . . .	170
13.18.3.1 st_back . . . . .	170
13.19 Calculate Secondary Structures of two RNAs upon Dimerization . . . . .	171
13.19.1 Detailed Description . . . . .	171
13.20 MFE Structures of single Nucleic Acid Sequences . . . . .	172
13.20.1 Detailed Description . . . . .	172
13.20.2 Function Documentation . . . . .	173
13.20.2.1 vrna_fold(const char *string, char *structure) . . . . .	173
13.20.2.2 vrna_circfold(const char *string, char *structure) . . . . .	173
13.20.2.3 fold_par(const char *sequence, char *structure, vrna_param_t *parameters, int is_constrained, int is_circular) . . . . .	174
13.20.2.4 fold(const char *sequence, char *structure) . . . . .	175
13.20.2.5 circfold(const char *sequence, char *structure) . . . . .	175
13.20.2.6 free_arrays(void) . . . . .	176
13.20.2.7 update_fold_params(void) . . . . .	176
13.20.2.8 update_fold_params_par(vrna_param_t *parameters) . . . . .	176
13.20.2.9 export_fold_arrays(int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **indx_p, char **ptype_p) . . . . .	176
13.20.2.10 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) . . . . .	177
13.20.2.11 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) . . . . .	177
13.20.2.12 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) . . . . .	177
13.20.2.13 LoopEnergy(int n1, int n2, int type, int type_2, int si1, int sj1, int sp1, int sq1) . . . . .	177
13.20.2.14 HairpinE(int size, int type, int si1, int sj1, const char *string) . . . . .	177
13.20.2.15 initialize_fold(int length) . . . . .	177
13.21 MFE Structures of two hybridized Sequences . . . . .	178
13.21.1 Detailed Description . . . . .	179
13.21.2 Function Documentation . . . . .	179

13.21.2.1	<code>vrna_cofold(const char *string, char *structure)</code>	179
13.21.2.2	<code>cofold(const char *sequence, char *structure)</code>	179
13.21.2.3	<code>cofold_par(const char *string, char *structure, vrna_param_t *parameters, int is_constrained)</code>	180
13.21.2.4	<code>free_co_arrays(void)</code>	180
13.21.2.5	<code>update_cofold_params(void)</code>	180
13.21.2.6	<code>update_cofold_params_par(vrna_param_t *parameters)</code>	180
13.21.2.7	<code>export_cofold_arrays_gg(int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **fc_p, int **ggg_p, int **indx_p, char **ptype_p)</code>	181
13.21.2.8	<code>export_cofold_arrays(int **f5_p, int **c_p, int **fML_p, int **fM1_p, int **fc_p, int **indx_p, char **ptype_p)</code>	181
13.21.2.9	<code>get_monomere_mfes(float *e1, float *e2)</code>	182
13.21.2.10	<code>initialize_cofold(int length)</code>	182
13.21.2.11	<code>vrna_mfe_dimer(vrna_fold_compound_t *vc, char *structure)</code>	182
13.22	Partition Function for two hybridized Sequences	184
13.22.1	Detailed Description	185
13.22.2	Data Structure Documentation	185
13.22.2.1	<code>struct vrna_dimer_pf_s</code>	185
13.22.2.2	<code>struct vrna_dimer_conc_s</code>	185
13.22.3	Function Documentation	186
13.22.3.1	<code>vrna_pf_dimer(vrna_fold_compound_t *vc, char *structure)</code>	186
13.22.3.2	<code>vrna_pf_dimer_probs(double FAB, double FA, double FB, vrna_plist_t *prAB, const vrna_plist_t *prA, const vrna_plist_t *prB, int Alength, const vrna_exp_param_t *exp_params)</code>	186
13.22.3.3	<code>vrna_pf_dimer_concentrations(double FcAB, double FcAA, double FcBB, double FEA, double FEB, const double *startconc, const vrna_exp_param_t *exp_params)</code>	187
13.23	Partition Function for two hybridized Sequences as a stepwise Process	188
13.23.1	Detailed Description	188
13.23.2	Function Documentation	189
13.23.2.1	<code>pf_unstru(char *sequence, int max_w)</code>	189
13.23.2.2	<code>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)</code>	189
13.24	Predicting Consensus Structures from Alignment(s)	191



13.24.1 Detailed Description	192
13.24.2 Function Documentation	192
13.24.2.1 energy_of_alistruct(const char **sequences, const char *structure, int n_seq, float *energy)	192
13.24.2.2 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)	193
13.24.2.3 update_alifold_params(void)	194
13.24.2.4 vrna_aln_mpi(char *Aseq[], int n_seq, int length, int *mini)	194
13.24.2.5 get_mpi(char *Aseq[], int n_seq, int length, int *mini)	194
13.24.2.6 encode_ali_sequence(const char *sequence, short *S, short *s5, short *s3, char *ss, unsigned short *as, int circ)	195
13.24.2.7 alloc_sequence_arrays(const char **sequences, short ***S, short ***S5, short ***S3, unsigned short ***a2s, char ***Ss, int circ)	195
13.24.2.8 free_sequence_arrays(unsigned int n_seq, short ***S, short ***S5, short ***S3, unsigned short ***a2s, char ***Ss)	196
13.24.3 Variable Documentation	196
13.24.3.1 cv_fact	196
13.24.3.2 nc_fact	196
13.25 MFE Consensus Structures for Sequence Alignment(s)	197
13.25.1 Detailed Description	197
13.25.2 Function Documentation	197
13.25.2.1 vrna_alifold(const char **ssequences, char *structure)	197
13.25.2.2 vrna_circalifold(const char **ssequences, char *structure)	198
13.25.2.3 alifold(const char **strings, char *structure)	198
13.25.2.4 circalifold(const char **strings, char *structure)	199
13.25.2.5 free_alifold_arrays(void)	200
13.26 Partition Function and Base Pair Probabilities for Sequence Alignment(s)	201
13.26.1 Detailed Description	201
13.26.2 Function Documentation	201
13.26.2.1 vrna_pf_alifold(const char **strings, char *structure, vrna_plist_t **pl)	201
13.26.2.2 vrna_pf_circalifold(const char **sequences, char *structure, vrna_plist_t **pl)	202

13.26.2.3	<code>alipf_fold_par(const char **sequences, char *structure, vrna_plist_t **pl, vrna_↵ _exp_param_t *parameters, int calculate_bppm, int is_constrained, int is_circular)</code>	203
13.26.2.4	<code>alipf_fold(const char **sequences, char *structure, vrna_plist_t **pl)</code>	203
13.26.2.5	<code>alipf_circ_fold(const char **sequences, char *structure, vrna_plist_t **pl)</code>	204
13.26.2.6	<code>export_ali_bppm(void)</code>	204
13.26.2.7	<code>free_alipf_arrays(void)</code>	204
13.27	Stochastic Backtracking of Consensus Structures from Sequence Alignment(s)	205
13.27.1	Detailed Description	205
13.27.2	Function Documentation	205
13.27.2.1	<code>alipbacktrack(double *prob)</code>	205
13.28	Predicting Locally stable structures of large sequences	206
13.28.1	Detailed Description	206
13.29	Local MFE structure Prediction and Z-scores	207
13.29.1	Detailed Description	207
13.29.2	Function Documentation	207
13.29.2.1	<code>vrna_Lfold(const char *string, int window_size, FILE *file)</code>	207
13.29.2.2	<code>vrna_Lfoldz(const char *string, int window_size, double min_z, FILE *file)</code>	208
13.29.2.3	<code>Lfold(const char *string, char *structure, int maxdist)</code>	208
13.29.2.4	<code>Lfoldz(const char *string, char *structure, int maxdist, int zsc, double min_z)</code>	209
13.29.2.5	<code>vrna_mfe_window(vrna_fold_compound_t *vc, FILE *file)</code>	209
13.29.2.6	<code>vrna_mfe_window_zscore(vrna_fold_compound_t *vc, double min_z, FILE *file)</code>	209
13.30	Partition functions for locally stable secondary structures	211
13.30.1	Detailed Description	211
13.30.2	Function Documentation	211
13.30.2.1	<code>update_pf_paramsLP(int length)</code>	211
13.30.2.2	<code>pfl_fold(char *sequence, int winSize, int pairSize, float cutoffb, double **pU, plist **dpp2, FILE *pUfp, FILE *spup)</code>	212
13.30.2.3	<code>putoutpU_prob(double **pU, int length, int ulength, FILE *fp, int energies)</code>	212
13.30.2.4	<code>putoutpU_prob_bin(double **pU, int length, int ulength, FILE *fp, int energies)</code>	213
13.31	Local MFE consensus structures for Sequence Alignments	214
13.31.1	Detailed Description	214

13.31.2 Function Documentation . . . . .	214
13.31.2.1 aliLfold(const char **strings, char *structure, int maxdist) . . . . .	214
13.32 Reading/Writing Energy Parameter Sets from/to File . . . . .	215
13.32.1 Detailed Description . . . . .	215
13.32.2 Function Documentation . . . . .	215
13.32.2.1 read_parameter_file(const char fname[]) . . . . .	215
13.32.2.2 write_parameter_file(const char fname[]) . . . . .	216
13.33 Converting Energy Parameter Files . . . . .	217
13.33.1 Detailed Description . . . . .	218
13.33.2 Macro Definition Documentation . . . . .	218
13.33.2.1 VRNA_CONVERT_OUTPUT_ALL . . . . .	218
13.33.2.2 VRNA_CONVERT_OUTPUT_HP . . . . .	218
13.33.2.3 VRNA_CONVERT_OUTPUT_STACK . . . . .	218
13.33.2.4 VRNA_CONVERT_OUTPUT_MM_HP . . . . .	218
13.33.2.5 VRNA_CONVERT_OUTPUT_MM_INT . . . . .	218
13.33.2.6 VRNA_CONVERT_OUTPUT_MM_INT_1N . . . . .	218
13.33.2.7 VRNA_CONVERT_OUTPUT_MM_INT_23 . . . . .	219
13.33.2.8 VRNA_CONVERT_OUTPUT_MM_MULTI . . . . .	219
13.33.2.9 VRNA_CONVERT_OUTPUT_MM_EXT . . . . .	219
13.33.2.10 VRNA_CONVERT_OUTPUT_DANGLE5 . . . . .	219
13.33.2.11 VRNA_CONVERT_OUTPUT_DANGLE3 . . . . .	219
13.33.2.12 VRNA_CONVERT_OUTPUT_INT_11 . . . . .	219
13.33.2.13 VRNA_CONVERT_OUTPUT_INT_21 . . . . .	219
13.33.2.14 VRNA_CONVERT_OUTPUT_INT_22 . . . . .	219
13.33.2.15 VRNA_CONVERT_OUTPUT_BULGE . . . . .	220
13.33.2.16 VRNA_CONVERT_OUTPUT_INT . . . . .	220
13.33.2.17 VRNA_CONVERT_OUTPUT_ML . . . . .	220
13.33.2.18 VRNA_CONVERT_OUTPUT_MISC . . . . .	220
13.33.2.19 VRNA_CONVERT_OUTPUT_SPECIAL_HP . . . . .	220
13.33.2.20 VRNA_CONVERT_OUTPUT_VANILLA . . . . .	220

13.33.2.21	<code>VRNA_CONVERT_OUTPUT_NINIO</code>	220
13.33.2.22	<code>VRNA_CONVERT_OUTPUT_DUMP</code>	221
13.33.3	Function Documentation	221
13.33.3.1	<code>convert_parameter_file(const char *iname, const char *oname, unsigned int options)</code>	221
13.34	Classified Dynamic Programming	222
13.34.1	Detailed Description	222
13.35	Distance based partitioning of the Secondary Structure Space	223
13.35.1	Detailed Description	223
13.36	Calculating MFE representatives of a Distance Based Partitioning	224
13.36.1	Detailed Description	225
13.36.2	Data Structure Documentation	225
13.36.2.1	<code>struct vrna_sol_TwoD_t</code>	225
13.36.2.2	<code>struct TwoDfold_vars</code>	225
13.36.3	Typedef Documentation	227
13.36.3.1	<code>vrna_sol_TwoD_t</code>	227
13.36.3.2	<code>TwoDfold_vars</code>	227
13.36.4	Function Documentation	227
13.36.4.1	<code>vrna_mfe_TwoD(vrna_fold_compound_t *vc, int distance1, int distance2)</code>	227
13.36.4.2	<code>vrna_backtrack5_TwoD(vrna_fold_compound_t *vc, int k, int l, unsigned int j)</code>	228
13.36.4.3	<code>get_TwoDfold_variables(const char *seq, const char *structure1, const char *structure2, int circ)</code>	228
13.36.4.4	<code>destroy_TwoDfold_variables(TwoDfold_vars *our_variables)</code>	229
13.36.4.5	<code>TwoDfoldList(TwoDfold_vars *vars, int distance1, int distance2)</code>	229
13.36.4.6	<code>TwoDfold_backtrack_f5(unsigned int j, int k, int l, TwoDfold_vars *vars)</code>	230
13.37	Calculate Partition Functions of a Distance Based Partitioning	231
13.37.1	Detailed Description	231
13.37.2	Data Structure Documentation	232
13.37.2.1	<code>struct vrna_sol_TwoD_pf_t</code>	232
13.37.3	Typedef Documentation	232
13.37.3.1	<code>vrna_sol_TwoD_pf_t</code>	232

13.37.4 Function Documentation . . . . .	232
13.37.4.1 vrna_pf_TwoD(vrna_fold_compound_t *vc, int maxDistance1, int maxDistance2) . . . . .	232
13.38 Stochastic Backtracking of Structures from Distance Based Partitioning . . . . .	234
13.38.1 Detailed Description . . . . .	234
13.38.2 Function Documentation . . . . .	234
13.38.2.1 vrna_pbacktrack_TwoD(vrna_fold_compound_t *vc, int d1, int d2) . . . . .	234
13.38.2.2 vrna_pbacktrack5_TwoD(vrna_fold_compound_t *vc, int d1, int d2, unsigned int length) . . . . .	235
13.39 Compute the Density of States . . . . .	236
13.39.1 Detailed Description . . . . .	236
13.39.2 Variable Documentation . . . . .	236
13.39.2.1 density_of_states . . . . .	236
13.40 Hard Constraints . . . . .	237
13.40.1 Detailed Description . . . . .	238
13.40.2 Data Structure Documentation . . . . .	239
13.40.2.1 struct vrna_hc_s . . . . .	239
13.40.2.2 struct vrna_hc_up_s . . . . .	240
13.40.3 Macro Definition Documentation . . . . .	240
13.40.3.1 VRNA_CONSTRAINT_DB . . . . .	240
13.40.3.2 VRNA_CONSTRAINT_DB_ENFORCE_BP . . . . .	240
13.40.3.3 VRNA_CONSTRAINT_DB_PIPE . . . . .	241
13.40.3.4 VRNA_CONSTRAINT_DB_DOT . . . . .	241
13.40.3.5 VRNA_CONSTRAINT_DB_X . . . . .	241
13.40.3.6 VRNA_CONSTRAINT_DB_RND_BRACK . . . . .	241
13.40.3.7 VRNA_CONSTRAINT_DB_INTRAMOL . . . . .	242
13.40.3.8 VRNA_CONSTRAINT_DB_INTERMOL . . . . .	242
13.40.3.9 VRNA_CONSTRAINT_DB_GQUAD . . . . .	242
13.40.3.10 VRNA_CONSTRAINT_DB_DEFAULT . . . . .	243
13.40.4 Typedef Documentation . . . . .	243
13.40.4.1 vrna_callback_hc_evaluate . . . . .	243
13.40.5 Function Documentation . . . . .	244

13.40.5.1 vrna_hc_init(vrna_fold_compound_t *vc) . . . . .	244
13.40.5.2 vrna_hc_add_up(vrna_fold_compound_t *vc, int i, char option) . . . . .	244
13.40.5.3 vrna_hc_add_up_batch(vrna_fold_compound_t *vc, vrna_hc_up_t *constraints) . . . . .	244
13.40.5.4 vrna_hc_add_bp(vrna_fold_compound_t *vc, int i, int j, char option) . . . . .	245
13.40.5.5 vrna_hc_add_bp_nonspecific(vrna_fold_compound_t *vc, int i, int d, char option) . . . . .	245
13.40.5.6 vrna_hc_free(vrna_hc_t *hc) . . . . .	246
13.40.5.7 vrna_hc_add_from_db(vrna_fold_compound_t *vc, const char *constraint, unsigned int options) . . . . .	246
13.41 Soft Constraints . . . . .	247
13.41.1 Detailed Description . . . . .	248
13.41.2 Data Structure Documentation . . . . .	248
13.41.2.1 struct vrna_sc_s . . . . .	248
13.41.3 Typedef Documentation . . . . .	249
13.41.3.1 vrna_callback_sc_energy . . . . .	249
13.41.3.2 vrna_callback_sc_exp_energy . . . . .	250
13.41.3.3 vrna_callback_sc_backtrack . . . . .	250
13.41.4 Function Documentation . . . . .	251
13.41.4.1 vrna_sc_init(vrna_fold_compound_t *vc) . . . . .	251
13.41.4.2 vrna_sc_add_bp(vrna_fold_compound_t *vc, const FLT_OR_DBL **constraints, unsigned int options) . . . . .	252
13.41.4.3 vrna_sc_add_up(vrna_fold_compound_t *vc, const FLT_OR_DBL *constraints, unsigned int options) . . . . .	252
13.41.4.4 vrna_sc_remove(vrna_fold_compound_t *vc) . . . . .	252
13.41.4.5 vrna_sc_free(vrna_sc_t *sc) . . . . .	252
13.41.4.6 vrna_sc_add_data(vrna_fold_compound_t *vc, void *data, vrna_callback_free↵_auxdata *free_data) . . . . .	253
13.41.4.7 vrna_sc_add_f(vrna_fold_compound_t *vc, vrna_callback_sc_energy *f) . . . . .	253
13.41.4.8 vrna_sc_add_bt(vrna_fold_compound_t *vc, vrna_callback_sc_backtrack *f) . . . . .	253
13.41.4.9 vrna_sc_add_exp_f(vrna_fold_compound_t *vc, vrna_callback_sc_exp_energy *exp_f) . . . . .	254
13.42 Incorporating SHAPE reactivity data . . . . .	255
13.42.1 Detailed Description . . . . .	255
13.42.2 Function Documentation . . . . .	256

13.42.2.1 vrna_sc_add_SHAPE_deigan(vrna_fold_compound_t *vc, const double *reactivities, double m, double b, unsigned int options) . . . . .	256
13.42.2.2 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) . . . . .	256
13.42.2.3 vrna_sc_add_SHAPE_zarringham(vrna_fold_compound_t *vc, const double *reactivities, double b, double default_value, const char *shape_conversion, unsigned int options) . . . . .	257
13.42.2.4 vrna_sc_SHAPE_to_pr(const char *shape_conversion, double *values, int length, double default_value) . . . . .	257
13.43 Incorporating ligands binding to specific sequence/structure motifs . . . . .	259
13.43.1 Detailed Description . . . . .	259
13.43.2 Function Documentation . . . . .	259
13.43.2.1 vrna_sc_add_hi_motif(vrna_fold_compound_t *vc, const char *seq, const char *structure, FLT_OR_DBL energy, unsigned int options) . . . . .	259
13.44 Generate soft constraints from data . . . . .	261
13.44.1 Detailed Description . . . . .	262
13.44.2 Macro Definition Documentation . . . . .	262
13.44.2.1 VRNA_OBJECTIVE_FUNCTION_QUADRATIC . . . . .	262
13.44.2.2 VRNA_OBJECTIVE_FUNCTION_ABSOLUTE . . . . .	262
13.44.2.3 VRNA_MINIMIZER_CONJUGATE_FR . . . . .	262
13.44.2.4 VRNA_MINIMIZER_CONJUGATE_PR . . . . .	262
13.44.2.5 VRNA_MINIMIZER_VECTOR_BFGS . . . . .	263
13.44.2.6 VRNA_MINIMIZER_VECTOR_BFGS2 . . . . .	263
13.44.2.7 VRNA_MINIMIZER_STEEPEST_DESCENT . . . . .	263
13.44.3 Typedef Documentation . . . . .	263
13.44.3.1 progress_callback . . . . .	263
13.44.4 Function Documentation . . . . .	263
13.44.4.1 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 initialStep←Size, double minStepSize, double minImprovement, double minimizerTolerance, progress_callback callback) . . . . .	263
13.45 The Fold Compound . . . . .	265
13.45.1 Detailed Description . . . . .	266

13.45.2 Data Structure Documentation . . . . .	266
13.45.2.1 struct vrna_fc_s . . . . .	266
13.45.3 Macro Definition Documentation . . . . .	273
13.45.3.1 VRNA_STATUS_MFE_PRE . . . . .	273
13.45.3.2 VRNA_STATUS_MFE_POST . . . . .	273
13.45.3.3 VRNA_STATUS_PF_PRE . . . . .	273
13.45.3.4 VRNA_STATUS_PF_POST . . . . .	274
13.45.3.5 VRNA_OPTION_MFE . . . . .	274
13.45.3.6 VRNA_OPTION_PF . . . . .	274
13.45.3.7 VRNA_OPTION_EVAL_ONLY . . . . .	274
13.45.4 Typedef Documentation . . . . .	274
13.45.4.1 vrna_callback_free_auxdata . . . . .	274
13.45.4.2 vrna_callback_recursion_status . . . . .	275
13.45.5 Enumeration Type Documentation . . . . .	275
13.45.5.1 vrna_fc_type_e . . . . .	275
13.45.6 Function Documentation . . . . .	275
13.45.6.1 vrna_fold_compound(const char *sequence, vrna_md_t *md_p, unsigned int options) . . . . .	275
13.45.6.2 vrna_fold_compound_comparative(const char **sequences, vrna_md_t *md_p, unsigned int options) . . . . .	276
13.45.6.3 vrna_fold_compound_free(vrna_fold_compound_t *vc) . . . . .	277
13.45.6.4 vrna_fold_compound_add_auxdata(vrna_fold_compound_t *vc, void *data, vrna_callback_free_auxdata *f) . . . . .	278
13.45.6.5 vrna_fold_compound_add_callback(vrna_fold_compound_t *vc, vrna_callback↔_recursion_status *f) . . . . .	278
13.46 The Dynamic Programming Matrices . . . . .	280
13.46.1 Detailed Description . . . . .	280
13.46.2 Data Structure Documentation . . . . .	281
13.46.2.1 struct vrna_mx_mfe_s . . . . .	281
13.46.2.2 struct vrna_mx_pf_s . . . . .	283
13.46.3 Enumeration Type Documentation . . . . .	284
13.46.3.1 vrna_mx_type_e . . . . .	284



13.46.4 Function Documentation . . . . .	285
13.46.4.1 vrna_mx_add(vrna_fold_compound_t *vc, vrna_mx_type_e type, unsigned int options) . . . . .	285
13.46.4.2 vrna_mx_mfe_free(vrna_fold_compound_t *vc) . . . . .	285
13.46.4.3 vrna_mx_pf_free(vrna_fold_compound_t *vc) . . . . .	286
13.47 Direct refolding paths between two secondary structures . . . . .	287
13.47.1 Detailed Description . . . . .	287
13.47.2 Data Structure Documentation . . . . .	288
13.47.2.1 struct vrna_path_s . . . . .	288
13.47.3 Typedef Documentation . . . . .	288
13.47.3.1 path_t . . . . .	288
13.47.4 Function Documentation . . . . .	288
13.47.4.1 vrna_path_findpath_saddle(vrna_fold_compound_t *vc, const char *struc1, const char *struc2, int max) . . . . .	288
13.47.4.2 vrna_path_findpath(vrna_fold_compound_t *vc, const char *s1, const char *s2, int maxkeep) . . . . .	289
13.47.4.3 find_saddle(const char *seq, const char *struc1, const char *struc2, int max) . . . . .	289
13.47.4.4 free_path(vrna_path_t *path) . . . . .	290
13.47.4.5 get_path(const char *seq, const char *s1, const char *s2, int maxkeep) . . . . .	290
13.48 Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings . . . . .	291
13.48.1 Detailed Description . . . . .	292
13.48.2 Macro Definition Documentation . . . . .	292
13.48.2.1 FILENAME_MAX_LENGTH . . . . .	292
13.48.2.2 FILENAME_ID_LENGTH . . . . .	292
13.48.3 Function Documentation . . . . .	292
13.48.3.1 vrna_random_string(int l, const char symbols[]) . . . . .	292
13.48.3.2 vrna_hamming_distance(const char *s1, const char *s2) . . . . .	292
13.48.3.3 vrna_hamming_distance_bound(const char *s1, const char *s2, int n) . . . . .	293
13.48.3.4 vrna_seq_toRNA(char *sequence) . . . . .	293
13.48.3.5 vrna_seq_toupper(char *sequence) . . . . .	293
13.48.3.6 vrna_cut_point_insert(const char *string, int cp) . . . . .	294
13.48.3.7 vrna_cut_point_remove(const char *string, int *cp) . . . . .	294

13.49 Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures . . . . .	295
13.49.1 Detailed Description . . . . .	297
13.49.2 Data Structure Documentation . . . . .	297
13.49.2.1 struct vrna_hx_s . . . . .	297
13.49.3 Function Documentation . . . . .	297
13.49.3.1 b2HIT(const char *structure) . . . . .	297
13.49.3.2 b2C(const char *structure) . . . . .	298
13.49.3.3 b2Shapiro(const char *structure) . . . . .	298
13.49.3.4 add_root(const char *structure) . . . . .	298
13.49.3.5 expand_Shapiro(const char *coarse) . . . . .	299
13.49.3.6 expand_Full(const char *structure) . . . . .	299
13.49.3.7 unexpand_Full(const char *ffull) . . . . .	299
13.49.3.8 unweight(const char *wcoarse) . . . . .	299
13.49.3.9 unexpand_aligned_F(char *align[2]) . . . . .	300
13.49.3.10 parse_structure(const char *structure) . . . . .	300
13.49.3.11 vrna_db_pack(const char *struc) . . . . .	300
13.49.3.12 vrna_db_unpack(const char *packed) . . . . .	301
13.49.3.13 vrna_ptable(const char *structure) . . . . .	301
13.49.3.14 vrna_pt_pk_get(const char *structure) . . . . .	302
13.49.3.15 vrna_ptable_copy(const short *pt) . . . . .	302
13.49.3.16 vrna_pt_snoop_get(const char *structure) . . . . .	302
13.49.3.17 vrna_db_from_ptable(short *pt) . . . . .	302
13.49.3.18 vrna_bp_distance(const char *str1, const char *str2) . . . . .	303
13.49.3.19 vrna_refBPcnt_matrix(const short *reference_pt, unsigned int turn) . . . . .	303
13.49.3.20 vrna_refBPdist_matrix(const short *pt1, const short *pt2, unsigned int turn) . . . . .	303
13.49.3.21 vrna_db_from_bp_stack(vrna_bp_stack_t *bp, unsigned int length) . . . . .	303
13.49.3.22 vrna_plist(const char *struc, float pr) . . . . .	304
13.49.3.23 vrna_db_from_plist(vrna_plist_t *pairs, unsigned int n) . . . . .	304
13.49.3.24 assign_plist_from_db(vrna_plist_t **pl, const char *struc, float pr) . . . . .	305
13.49.3.25 pack_structure(const char *struc) . . . . .	305

13.49.3.26	unpack_structure(const char *packed)	305
13.49.3.27	make_pair_table(const char *structure)	306
13.49.3.28	copy_pair_table(const short *pt)	306
13.49.3.29	alimake_pair_table(const char *structure)	307
13.49.3.30	make_pair_table_snoop(const char *structure)	307
13.49.3.31	bp_distance(const char *str1, const char *str2)	307
13.49.3.32	make_referenceBP_array(short *reference_pt, unsigned int turn)	307
13.49.3.33	compute_BPdifferences(short *pt1, short *pt2, unsigned int turn)	308
13.49.3.34	parenthesis_structure(char *structure, vrna_bp_stack_t *bp, int length)	308
13.49.3.35	parenthesis_zuker(char *structure, vrna_bp_stack_t *bp, int length)	308
13.49.3.36	ppm_to_structure(char *structure, FLT_OR_DBL *pr, unsigned int length)	308
13.49.3.37	ppm_symbol(const float *x)	308
13.50	Various utilities for Sequence Alignments, and Comparative Structure Prediction	309
13.50.1	Detailed Description	309
13.50.2	Data Structure Documentation	309
13.50.2.1	struct vrna_pinfo_s	309
13.50.3	Typedef Documentation	310
13.50.3.1	pair_info	310
13.50.4	Function Documentation	310
13.50.4.1	vrna_aln_pinfo(vrna_fold_compound_t *vc, const char *structure, double threshold)	310
13.51	Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments	312
13.51.1	Detailed Description	313
13.51.2	Macro Definition Documentation	313
13.51.2.1	VRNA_OPTION_MULTILINE	313
13.51.2.2	VRNA_CONSTRAINT_MULTILINE	313
13.51.3	Function Documentation	313
13.51.3.1	vrna_file_helixlist(const char *seq, const char *db, float energy, FILE *file)	313
13.51.3.2	vrna_file_connect(const char *seq, const char *db, float energy, const char *identifier, FILE *file)	314
13.51.3.3	vrna_file_bpseq(const char *seq, const char *db, FILE *file)	314

13.51.3.4 vrna_file_json(const char *seq, const char *db, double energy, const char *identifier, FILE *file)	314
13.51.3.5 vrna_file_fasta_read_record(char **header, char **sequence, char ***rest, FILE *file, unsigned int options)	315
13.51.3.6 vrna_extract_record_rest_structure(const char **lines, unsigned int length, unsigned int option)	316
13.51.3.7 vrna_file_SHAPE_read(const char *file_name, int length, double default_value, char *sequence, double *values)	317
13.51.3.8 vrna_file_constraints_read(const char *filename, unsigned int length, unsigned int options)	317
13.51.3.9 vrna_extract_record_rest_constraint(char **cstruc, const char **lines, unsigned int option)	317
13.51.3.10 read_record(char **header, char **sequence, char ***rest, unsigned int options)	318
13.52 Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More	319
13.52.1 Detailed Description	320
13.52.2 Data Structure Documentation	320
13.52.2.1 struct COORDINATE	320
13.52.3 Macro Definition Documentation	321
13.52.3.1 VRNA_PLOT_TYPE_SIMPLE	321
13.52.3.2 VRNA_PLOT_TYPE_NAVIEW	321
13.52.3.3 VRNA_PLOT_TYPE_CIRCULAR	321
13.52.4 Function Documentation	321
13.52.4.1 aliPS_color_aln(const char *structure, const char *filename, const char *seqs[], const char *names[])	321
13.52.4.2 simple_xy_coordinates(short *pair_table, float *X, float *Y)	321
13.52.4.3 simple_circplot_coordinates(short *pair_table, float *x, float *y)	322
13.52.4.4 vrna_file_PS_rnaplot(const char *seq, const char *structure, const char *file, vrna_md_t *md_p)	322
13.52.4.5 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)	323
13.52.4.6 gmlRNA(char *string, char *structure, char *ssfile, char option)	323
13.52.4.7 ssv_rna_plot(char *string, char *structure, char *ssfile)	324
13.52.4.8 svg_rna_plot(char *string, char *structure, char *ssfile)	324
13.52.4.9 xrna_plot(char *string, char *structure, char *ssfile)	324
13.52.4.10 PS_rna_plot(char *string, char *structure, char *file)	325
13.52.4.11 PS_rna_plot_a(char *string, char *structure, char *file, char *pre, char *post)	325
13.52.4.12 PS_rna_plot_a_gquad(char *string, char *structure, char *ssfile, char *pre, char *post)	325
13.52.4.13 PS_dot_plot_list(char *seq, char *filename, plist *pl, plist *mf, char *comment)	325
13.52.4.14 PS_dot_plot(char *string, char *file)	326
13.52.5 Variable Documentation	326
13.52.5.1 rna_plot_type	326

<b>14 Data Structure Documentation</b>	<b>327</b>
14.1 <code>_struct_en</code> Struct Reference	327
14.2 <code>LIST</code> Struct Reference	327
14.3 <code>LST_BUCKET</code> Struct Reference	328
14.4 <code>Postorder_list</code> Struct Reference	328
14.5 <code>swString</code> Struct Reference	328
14.6 <code>Tree</code> Struct Reference	328
14.7 <code>TwoDpfold_vars</code> Struct Reference	329
14.7.1 Detailed Description	330
14.8 <code>vrna_subopt_sol_s</code> Struct Reference	330
14.8.1 Detailed Description	330
<b>15 File Documentation</b>	<b>331</b>
15.1 <code>ViennaRNA/1.8.4_epars.h</code> File Reference	331
15.1.1 Detailed Description	331
15.2 <code>ViennaRNA/1.8.4_intloops.h</code> File Reference	331
15.2.1 Detailed Description	332
15.3 <code>ViennaRNA/2Dfold.h</code> File Reference	332
15.4 <code>ViennaRNA/2Dpfold.h</code> File Reference	333
15.4.1 Function Documentation	334
15.4.1.1 <code>get_TwoDpfold_variables(const char *seq, const char *structure1, char *structure2, int circ)</code>	334
15.4.1.2 <code>destroy_TwoDpfold_variables(TwoDpfold_vars *vars)</code>	334
15.4.1.3 <code>TwoDpfoldList(TwoDpfold_vars *vars, int maxDistance1, int maxDistance2)</code>	334
15.4.1.4 <code>TwoDpfold_pbacktrack(TwoDpfold_vars *vars, int d1, int d2)</code>	335
15.4.1.5 <code>TwoDpfold_pbacktrack5(TwoDpfold_vars *vars, int d1, int d2, unsigned int length)</code>	336
15.5 <code>ViennaRNA/alifold.h</code> File Reference	336
15.5.1 Detailed Description	338
15.6 <code>ViennaRNA/aln_util.h</code> File Reference	338
15.6.1 Detailed Description	339
15.7 <code>ViennaRNA/alphabet.h</code> File Reference	339

15.7.1 Detailed Description . . . . .	340
15.7.2 Function Documentation . . . . .	340
15.7.2.1 vrna_ptypes(const short *S, vrna_md_t *md) . . . . .	340
15.7.2.2 vrna_nucleotide_encode(char c, vrna_md_t *md) . . . . .	341
15.7.2.3 vrna_nucleotide_decode(int enc, vrna_md_t *md) . . . . .	341
15.8 ViennaRNA/boltzmann_sampling.h File Reference . . . . .	341
15.8.1 Detailed Description . . . . .	342
15.9 ViennaRNA/centroid.h File Reference . . . . .	343
15.9.1 Detailed Description . . . . .	344
15.9.2 Function Documentation . . . . .	344
15.9.2.1 get_centroid_struct_pl(int length, double *dist, vrna_plist_t *pl) . . . . .	344
15.9.2.2 get_centroid_struct_pr(int length, double *dist, FLT_OR_DBL *pr) . . . . .	344
15.10ViennaRNA/cofold.h File Reference . . . . .	344
15.10.1 Detailed Description . . . . .	345
15.11ViennaRNA/constraints.h File Reference . . . . .	345
15.11.1 Detailed Description . . . . .	347
15.12ViennaRNA/constraints_hard.h File Reference . . . . .	347
15.12.1 Detailed Description . . . . .	349
15.12.2 Macro Definition Documentation . . . . .	349
15.12.2.1 VRNA_CONSTRAINT_NO_HEADER . . . . .	349
15.12.2.2 VRNA_CONSTRAINT_DB_ANG_BRACK . . . . .	349
15.12.3 Function Documentation . . . . .	350
15.12.3.1 print_tty_constraint(unsigned int option) . . . . .	350
15.12.3.2 print_tty_constraint_full(void) . . . . .	350
15.12.3.3 constrain_ptypes(const char *constraint, unsigned int length, char *ptype, int *↔ BP, int min_loop_size, unsigned int idx_type) . . . . .	350
15.13ViennaRNA/constraints_SHAPE.h File Reference . . . . .	350
15.13.1 Detailed Description . . . . .	351
15.13.2 Function Documentation . . . . .	351
15.13.2.1 vrna_sc_SHAPE_parse_method(const char *method_string, char *method, float *param_1, float *param_2) . . . . .	351

15.14ViennaRNA/constraints_soft.h File Reference . . . . .	352
15.14.1 Detailed Description . . . . .	353
15.15ViennaRNA/convert_epars.h File Reference . . . . .	353
15.15.1 Detailed Description . . . . .	354
15.16ViennaRNA/data_structures.h File Reference . . . . .	354
15.17ViennaRNA/dist_vars.h File Reference . . . . .	356
15.17.1 Detailed Description . . . . .	357
15.17.2 Variable Documentation . . . . .	357
15.17.2.1 edit_backtrack . . . . .	357
15.17.2.2 cost_matrix . . . . .	357
15.18ViennaRNA/dp_matrices.h File Reference . . . . .	357
15.19ViennaRNA/duplex.h File Reference . . . . .	358
15.19.1 Detailed Description . . . . .	358
15.20ViennaRNA/edit_cost.h File Reference . . . . .	358
15.20.1 Detailed Description . . . . .	358
15.21ViennaRNA/energy_const.h File Reference . . . . .	359
15.21.1 Detailed Description . . . . .	359
15.21.2 Macro Definition Documentation . . . . .	359
15.21.2.1 GASCONST . . . . .	359
15.21.2.2 K0 . . . . .	359
15.21.2.3 INF . . . . .	360
15.21.2.4 FORBIDDEN . . . . .	360
15.21.2.5 BONUS . . . . .	360
15.21.2.6 NBPAIRS . . . . .	360
15.21.2.7 TURN . . . . .	360
15.21.2.8 MAXLOOP . . . . .	360
15.22ViennaRNA/eval.h File Reference . . . . .	360
15.22.1 Detailed Description . . . . .	362
15.23ViennaRNA/exterior_loops.h File Reference . . . . .	362
15.23.1 Detailed Description . . . . .	363

15.24ViennaRNA/file_formats.h File Reference . . . . .	363
15.24.1 Detailed Description . . . . .	364
15.25ViennaRNA/findpath.h File Reference . . . . .	364
15.26ViennaRNA/fold.h File Reference . . . . .	365
15.26.1 Detailed Description . . . . .	366
15.27ViennaRNA/fold_vars.h File Reference . . . . .	366
15.27.1 Detailed Description . . . . .	367
15.27.2 Variable Documentation . . . . .	367
15.27.2.1 RibosumFile . . . . .	367
15.27.2.2 james_rule . . . . .	367
15.27.2.3 logML . . . . .	368
15.27.2.4 cut_point . . . . .	368
15.27.2.5 base_pair . . . . .	368
15.27.2.6 pr . . . . .	368
15.27.2.7 iindx . . . . .	368
15.28ViennaRNA/gquad.h File Reference . . . . .	368
15.28.1 Detailed Description . . . . .	369
15.29ViennaRNA/hairpin_loops.h File Reference . . . . .	369
15.29.1 Detailed Description . . . . .	370
15.30ViennaRNA/interior_loops.h File Reference . . . . .	370
15.30.1 Detailed Description . . . . .	371
15.31ViennaRNA/inverse.h File Reference . . . . .	371
15.31.1 Detailed Description . . . . .	372
15.32ViennaRNA/Lfold.h File Reference . . . . .	372
15.32.1 Detailed Description . . . . .	372
15.33ViennaRNA/ligand.h File Reference . . . . .	373
15.33.1 Detailed Description . . . . .	373
15.34ViennaRNA/loop_energies.h File Reference . . . . .	373
15.34.1 Detailed Description . . . . .	374
15.35ViennaRNA/LPfold.h File Reference . . . . .	374



15.35.1 Detailed Description . . . . .	374
15.35.2 Function Documentation . . . . .	375
15.35.2.1 init_pf_foldLP(int length) . . . . .	375
15.36ViennaRNA/MEA.h File Reference . . . . .	375
15.36.1 Detailed Description . . . . .	375
15.36.2 Function Documentation . . . . .	375
15.36.2.1 MEA(plist *p, char *structure, double gamma) . . . . .	375
15.37ViennaRNA/mfe.h File Reference . . . . .	376
15.37.1 Detailed Description . . . . .	376
15.38ViennaRNA/mm.h File Reference . . . . .	376
15.38.1 Detailed Description . . . . .	377
15.39ViennaRNA/model.h File Reference . . . . .	377
15.39.1 Detailed Description . . . . .	381
15.40ViennaRNA/multibranch_loops.h File Reference . . . . .	381
15.40.1 Detailed Description . . . . .	382
15.41ViennaRNA/naview.h File Reference . . . . .	383
15.42ViennaRNA/params.h File Reference . . . . .	383
15.43ViennaRNA/part_func.h File Reference . . . . .	385
15.43.1 Detailed Description . . . . .	386
15.43.2 Function Documentation . . . . .	387
15.43.2.1 vrna_pf_float_precision(void) . . . . .	387
15.43.2.2 stackProb(double cutoff) . . . . .	387
15.43.2.3 init_pf_fold(int length) . . . . .	387
15.43.2.4 centroid(int length, double *dist) . . . . .	387
15.43.2.5 get_centroid_struct_gquad_pr(int length, double *dist) . . . . .	387
15.43.2.6 mean_bp_dist(int length) . . . . .	388
15.43.2.7 expLoopEnergy(int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short sq1) . . . . .	388
15.43.2.8 expHairpinEnergy(int u, int type, short si1, short sj1, const char *string) . . . . .	388
15.44ViennaRNA/part_func_co.h File Reference . . . . .	388
15.44.1 Detailed Description . . . . .	389

15.44.2 Function Documentation	390
15.44.2.1 co_pf_fold(char *sequence, char *structure)	390
15.44.2.2 co_pf_fold_par(char *sequence, char *structure, vrna_exp_param_t *parameters, int calculate_bppm, int is_constrained)	390
15.44.2.3 get_plist(vrna_plist_t *pl, int length, double cut_off)	391
15.44.2.4 compute_probabilities(double FAB, double FEA, double FEB, vrna_plist_t *prAB, vrna_plist_t *prA, vrna_plist_t *prB, int Alength)	391
15.44.2.5 get_concentrations(double FEAB, double FEAA, double FEBB, double FEA, dou- ble FEB, double *startconc)	391
15.44.2.6 init_co_pf_fold(int length)	392
15.44.2.7 export_co_bppm(void)	392
15.44.2.8 free_co_pf_arrays(void)	393
15.44.2.9 update_co_pf_params(int length)	393
15.44.2.10 update_co_pf_params_par(int length, vrna_exp_param_t *parameters)	393
15.45 ViennaRNA/part_func_up.h File Reference	393
15.45.1 Detailed Description	394
15.46 ViennaRNA/perturbation_fold.h File Reference	394
15.46.1 Detailed Description	395
15.47 ViennaRNA/plot_aln.h File Reference	396
15.47.1 Detailed Description	396
15.48 ViennaRNA/plot_layouts.h File Reference	396
15.48.1 Detailed Description	398
15.49 ViennaRNA/plot_structure.h File Reference	398
15.49.1 Detailed Description	399
15.50 ViennaRNA/profiledist.h File Reference	399
15.50.1 Function Documentation	400
15.50.1.1 profile_edit_distance(const float *T1, const float *T2)	400
15.50.1.2 Make_bp_profile_bppm(FLT_OR_DBL *bppm, int length)	400
15.50.1.3 free_profile(float *T)	400
15.50.1.4 Make_bp_profile(int length)	400
15.51 ViennaRNA/PS_dot.h File Reference	401
15.51.1 Detailed Description	401

15.52ViennaRNA/read_epars.h File Reference . . . . .	401
15.53ViennaRNA/ribo.h File Reference . . . . .	402
15.53.1 Detailed Description . . . . .	402
15.54ViennaRNA/RNAstruct.h File Reference . . . . .	402
15.54.1 Detailed Description . . . . .	403
15.55ViennaRNA/string_utils.h File Reference . . . . .	404
15.55.1 Detailed Description . . . . .	405
15.55.2 Function Documentation . . . . .	405
15.55.2.1 str_uppercase(char *sequence) . . . . .	405
15.55.2.2 str_DNA2RNA(char *sequence) . . . . .	405
15.55.2.3 random_string(int l, const char symbols[]) . . . . .	405
15.55.2.4 hamming(const char *s1, const char *s2) . . . . .	406
15.55.2.5 hamming_bound(const char *s1, const char *s2, int n) . . . . .	406
15.56ViennaRNA/stringdist.h File Reference . . . . .	406
15.56.1 Detailed Description . . . . .	406
15.56.2 Function Documentation . . . . .	406
15.56.2.1 Make_swString(char *string) . . . . .	406
15.56.2.2 string_edit_distance(swString *T1, swString *T2) . . . . .	407
15.57ViennaRNA/structure_utils.h File Reference . . . . .	407
15.57.1 Detailed Description . . . . .	409
15.58ViennaRNA/subopt.h File Reference . . . . .	410
15.58.1 Detailed Description . . . . .	411
15.59ViennaRNA/treedist.h File Reference . . . . .	411
15.59.1 Detailed Description . . . . .	411
15.59.2 Function Documentation . . . . .	411
15.59.2.1 make_tree(char *struc) . . . . .	411
15.59.2.2 tree_edit_distance(Tree *T1, Tree *T2) . . . . .	412
15.59.2.3 free_tree(Tree *t) . . . . .	412
15.60ViennaRNA/utils.h File Reference . . . . .	412
15.60.1 Detailed Description . . . . .	415
15.60.2 Function Documentation . . . . .	415
15.60.2.1 print_tty_input_seq(void) . . . . .	415
15.60.2.2 print_tty_input_seq_str(const char *s) . . . . .	415
15.60.2.3 warn_user(const char message[]) . . . . .	415
15.60.2.4 nrerror(const char message[]) . . . . .	415
15.60.2.5 space(unsigned size) . . . . .	416
15.60.2.6 xrealloc(void *p, unsigned size) . . . . .	416
15.60.2.7 init_rand(void) . . . . .	416
15.60.2.8 urn(void) . . . . .	416
15.60.2.9 int_urn(int from, int to) . . . . .	416
15.60.2.10filecopy(FILE *from, FILE *to) . . . . .	416
15.60.2.11time_stamp(void) . . . . .	416

<b>Bibliography</b>	<b>418</b>
<b>Index</b>	<b>419</b>

# Chapter 1

## ViennaRNA Package core - RNALib

**A Library for folding and comparing RNA secondary structures**

**Date**

1994-2016

**Authors**

Ivo Hofacker, Peter Stadler, Ronny Lorenz and many more

**Table of Contents**

- [Introduction](#)
- [RNA Secondary Structure Prediction](#)
- [Parsing and Comparing - Functions to Manipulate Structures](#)
- [Utilities - Odds and Ends](#)
- [Input / Output File Formats](#)
- [RNALib API v3.0](#)
- [Scripting Language interface\(s\)](#)
- [Example - A Small Example Program](#)

## 1.1 Introduction

The core of the Vienna RNA Package ([9], [7]) 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/>

## Chapter 2

# Parsing and Comparing - Functions to Manipulate Structures

### Representations of Secondary Structures

The standard representation of a secondary structure is the *bracket notation*, where matching brackets symbolize base pairs and unpaired bases are shown as dots. Alternatively, one may use 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 [5] 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 [13], 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'), see the example below.

The following example illustrates the different linear tree representations used by the package. All lines show the same secondary structure.

```
a) .(((...(((...)))...((...))))).
   (U) ((( (U) (U) ((( (U) (U) (U) P) P) P) (U) (U) (( (U) (U) P) P) P) (U) P) P) (U)
b) (U) (( (U2) (( (U3) P3) (U2) (( (U2) P2) P2) (U) P2) (U)
c) (( (H) (H) M) B)
   ((( (( (H) S) (( (H) S) M) S) B) S)
   ((( ((( (H) S) (( (H) S) M) S) B) S) E)
d) ((( ((( (( (H3) S3) (( (H2) S2) M4) S2) B1) S2) E2) R)
```

Above: [Tree](#) representations of secondary structures. a) Full structure: the first line shows the more convenient condensed notation which is used by our programs; the second line shows the rather clumsy expanded notation for completeness, b) HIT structure, c) different versions of coarse grained structures: the second line is exactly Shapiro's representation, the first line is obtained by neglecting the stems. Since each loop is closed by a unique stem, these two lines are equivalent. The third line is an extension taking into account also the external digits. d) weighted coarse structure, this time including the virtual root.

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:

- a) (UU) (P (P (P (P (UU) (UU) (P (P (P (UU) (UU) (UU) P) P) P) (UU) (UU) (P (P (UU) (U...
- b) (UU) (P2 (P2 (U2U2) (P2 (U3U3) P3) (U2U2) (P2 (U2U2) P2) P2) (UU) P2) (UU)
- c) (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)
- d) (R (E2 (S2 (B1 (S2 (M4 (S3 (H3) S3) ( (H2) S2) M4) S2) B1) S2) E2) R)

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

## Parsing and Coarse Graining of Structures

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



## Distance Measures

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

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

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

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

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

Two cost matrices are provided for coarse grained structures:

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

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

The lower matrix uses the costs given in [14]. 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

### 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

### 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

### 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

[Next Page: Utilities](#)

## Chapter 3

# Utilities - Odds and Ends

### Table of Contents

- [Producing secondary structure graphs](#)
- [Producing \(colored\) dot plots for base pair probabilities](#)
- [Producing \(colored\) alignments](#)
- [RNA sequence related utilities](#)
- [RNA secondary structure related utilities](#)
- [Miscellaneous Utilities](#)

### 3.1 Producing secondary structure graphs

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

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

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

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

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

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

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

Produce a secondary structure graph in SStructView format.

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

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

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

Produce a secondary structure plot for further editing in XRNA.

```
int rna_plot_type
```

Switch for changing the secondary structure layout algorithm.

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

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

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

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

See also

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

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

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

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

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

Produce a postscript dot-plot from two pair lists.

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

See also

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

### 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.

### 3.4 RNA sequence related utilities

Several functions provide useful applications to RNA sequences

```
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.

```
void str_DNA2RNA(char *sequence);
```

Convert a DNA input sequence to RNA alphabet.

```
void str_uppercase(char *sequence);
```

Convert an input sequence to uppercase.

### 3.5 RNA secondary structure related utilities

```
char *pack_structure (const char *struc)
```

Pack 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.

### 3.6 Miscellaneous Utilities

```
void print_tty_input_seq (void)
```

Print a line to *stdout* that asks for an input sequence.

```
void print_tty_constraint_full (void)
```

Print structure constraint characters to *stdout* (full constraint support)

```
void print_tty_constraint (unsigned int option)
```

Print structure constraint characters to *stdout*. (constraint support is specified by option parameter)

```
int  *get_iindx (unsigned int length)
```

```
int  *get_indx (unsigned int length)
```

```
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.

```
char  *get_line(FILE *fp);
```

Read a line of arbitrary length from a stream.

```
unsigned int read_record(  
    char **header,  
    char **sequence,  
    char ***rest,  
    unsigned int options);
```

Get a data record from *stdin*.

```
char  *time_stamp (void)
```

Get a timestamp.

```
void warn_user (const char message[])
```

Print a warning message.

```
void nerror (const char message[])
```

Die with an error message.

```
void    init_rand (void)
```

Make random number seeds.

```
unsigned short xsubi[3];
```

Current 48 bit random number.

```
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     *space (unsigned size)
```

Allocate space safely.

```
void     *xrealloc ( void *p,  
                    unsigned size)
```

Reallocate space safely.

**See also**

[utils.h](#) for a complete overview and detailed description of the utility functions

[Next Page: The new RNAlib API v3.0](#)





## Chapter 4

# RNAlib API v3.0

### 4.1 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.

### 4.2 What are the major changes?

...

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

...

## 4.4 Some Examples using RNAlib API v3.0

Below are some example programs and code fragments that show the usage of the new API that is introduced with ViennaRNA version 2.2.

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

#include <ViennaRNA/data_structures.h>
#include <ViennaRNA/params.h>
#include <ViennaRNA/utils.h>
#include <ViennaRNA/eval.h>
#include <ViennaRNA/fold.h>
#include <ViennaRNA/part_func.h>

int main(int argc, char *argv[]){

    char *seq = "
        AGACGACAAGGUUGAAUCGCACCCACAGUCUAUGAGUCGGUGACAACAUUACGAAAGGCUGUAAAAUCAAUUAUUCACCACAGGGGGCCCCCGUGUCUAG";
    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 MFE and PF DP matrices and default model details */
    vrna_fold_compound_t *vc = vrna_fold_compound(seq, NULL,
        VRNA_OPTION_MFE | VRNA_OPTION_PF);

    /* 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;
}
```

## Chapter 5

# Scripting Language interface(s)

### 5.1 Introduction

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

#### 5.1.1 Function renaming scheme

The main difference when using a scripting language interface compared to direct calls of RNAlib C functions is, that the prefix 'vrna\_' is dropped. For instance, when calling the [vrna\\_fold\(\)](#) function, corresponding calls in perl or python are `RNA::fold()`, and `RNA.fold()`, respectively.

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).

#### 5.1.2 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.

## 5.2 Examples

### 5.2.1 Perl Examples

#### 5.2.1.1 Using the Flat Interface

Example 1: "Simple MFE prediction"

```
00001 #!/usr/bin/perl
00002
00003 use warnings;
00004 use strict;
00005
00006 use RNA;
00007
00008 my $seq1 = "CGCAGGGAUACCCGCG";
00009
00010 # compute minimum free energy (mfe) and corresponding structure
00011 my ($ss, $mfe) = RNA::fold($seq1);
00012
00013 # print output
00014 printf "%s [ %6.2f ]\n", $ss, $mfe;
```

### 5.2.1.2 Using the Object Oriented (OO) Interface

The 'fold\_compound' class that serves as an object oriented interface for [vrna\\_fold\\_compound\\_t](#)

Example 1: "Simple MFE prediction"

```
00001 #!/usr/bin/perl
00002
00003 use warnings;
00004 use strict;
00005
00006 use RNA;
00007
00008 my $seq1 = "CGCAGGGAUACCCGCG";
00009
00010 # create new fold_compound object
00011 my $fc = new RNA::fold_compound($seq1);
00012
00013 # compute minimum free energy (mfe) and corresponding structure
00014 my ($ss, $mfe) = $fc->mfe();
00015
00016 # print output
00017 printf "%s [ %6.2f ]\n", $ss, $mfe;
```

## 5.2.2 Python Examples

## Chapter 6

# Input / Output File Formats

### 6.1 File formats for Secondary Structure Constraints

#### 6.1.1 Constraints Definition File

The RNAlib can parse and apply data from constraint definition text files, where each constraint is given as a line of whitespace delimited commands. The syntax we use extends the one used in `mfold` / `UNAFold` where each line begins with a command character followed by a set of positions.

Additionally, 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.

##### 6.1.1.1 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)

##### 6.1.1.2 Specification of the loop type context

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

- `E ...` Exterior loop
- `H ...` Hairpin loop
- `I ...` Interior loop (enclosing pair)
- `i ...` Interior loop (enclosed pair)
- `M ...` Multibranch loop (enclosing pair)
- `m ...` Multibranch loop (enclosed pair)
- `A ...` All loops

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

### 6.1.1.3 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.

### 6.1.1.4 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  $k - l$ .

### 6.1.1.5 Valid constraint commands

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

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

Syntax:

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

Description:

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

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

Syntax:

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

Description:

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

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

Syntax:

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

Description:

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

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

Syntax:

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

Description:

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

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

Syntax:

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

Description:

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

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

Syntax:

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

Description:

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

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

Syntax:

```
C i j k
```

Description:

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

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

Syntax:

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

Description:

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

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

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

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

Syntax:

```
E i 0 k e
```

Description:

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

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

Syntax

`E i j k e`

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



## Chapter 7

# Example - A Small Example Program

The following program exercises most commonly used functions of the library. The program folds two sequences using both the mfe and partition function algorithms and calculates the tree edit and profile distance of the resulting structures and base pairing probabilities.

### Note

This program uses the old API of RNAlib, which is in part already marked deprecated. Please consult the [RNAlib API v3.0](#) page for details of what changes are necessary to port your implementation to the new API.

```
#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="CGCAGGGGAUACCCGCG", *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);
}

```

In a typical Unix environment you would compile this program using:

```
cc ${OPENMP_CFLAGS} -c example.c -I${hpath}
```

and link using

```
cc ${OPENMP_CFLAGS} -o example -L${lpath} -lRNA -lm
```

where `${hpath}` and `${lpath}` point to the location of the header files and library, respectively.

#### Note

As default, the RNAlib is compiled with build-in *OpenMP* multithreading support. Thus, when linking your own object files to the library you have to pass the compiler specific `${OPENMP_CFLAGS}` (e.g. `'-fopenmp'` for **gcc**) even if your code does not use openmp specific code. However, in that case the *OpenMP* flags may be omitted when compiling `example.c`

## Chapter 8

# Deprecated List

Global **alifold** (const char \*\*strings, char \*structure)

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

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

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

Global **alipbacktrack** (double \*prob)

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

Global **alipf\_circ\_fold** (const char \*\*sequences, char \*structure, vrna\_plist\_t \*\*pl)

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

Global **alipf\_fold** (const char \*\*sequences, char \*structure, vrna\_plist\_t \*\*pl)

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

Global **alipf\_fold\_par** (const char \*\*sequences, char \*structure, vrna\_plist\_t \*\*pl, vrna\_exp\_param\_t \*parameters, int calculate\_bppm, int is\_constrained, int is\_circular)

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

Global **assign\_plist\_from\_db** (vrna\_plist\_t \*\*pl, const char \*struc, float pr)

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

Global **assign\_plist\_from\_pr** (vrna\_plist\_t \*\*pl, FLT\_OR\_DBL \*probs, int length, double cutoff)

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

Global **base\_pair**

Do not use this variable anymore!

Global **bondT**

Use [vrna\\_bp\\_stack\\_t](#) instead!

Global **bp\_distance** (const char \*str1, const char \*str2)

Use [vrna\\_bp\\_distance](#) instead

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

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

Global **bppm\_to\_structure** (char \*structure, FLT\_OR\_DBL \*pr, unsigned int length)

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

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

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

Global **circalifold** (const char \*\*strings, char \*structure)

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

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

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

Global **co\_pf\_fold** (char \*sequence, char \*structure)

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

Global **co\_pf\_fold\_par** (char \*sequence, char \*structure, vrna\_exp\_param\_t \*parameters, int calculate\_bppm, int is\_constrained)

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

Global **cofold** (const char \*sequence, char \*structure)

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

Global **cofold\_par** (const char \*string, char \*structure, vrna\_param\_t \*parameters, int is\_constrained)

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

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

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

Global **compute\_probabilities** (double FAB, double FEA, double FEB, vrna\_plist\_t \*prAB, vrna\_plist\_t \*prA, vrna\_plist\_t \*prB, int Alength)

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

Global **constrain\_ptypes** (const char \*constraint, unsigned int length, char \*ptype, int \*BP, int min\_loop\_size, unsigned int idx\_type)

Do not use this function anymore! Structure constraints are now handled through [vrna\\_hc\\_t](#) and related functions.

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

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

Global **cpair**

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

Global **cv\_fact**

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

Global **destroy\_TwoDfold\_variables** (TwoDfold\_vars \*our\_variables)

Use the new API that relies on [vrna\\_fold\\_compound\\_t](#) and the corresponding functions [vrna\\_fold\\_compound\\_TwoD\(\)](#), [vrna\\_mfe\\_TwoD\(\)](#), and [vrna\\_fold\\_compound\\_free\(\)](#) instead!

Global **destroy\_TwoDpfold\_variables** (TwoDpfold\_vars \*vars)

Use the new API that relies on [vrna\\_fold\\_compound\\_t](#) and the corresponding functions [vrna\\_fold\\_compound\\_TwoD\(\)](#), [vrna\\_pf\\_TwoD\(\)](#), and [vrna\\_fold\\_compound\\_free\(\)](#) instead!

Global **energy\_of\_alistruct** (const char \*\*sequences, const char \*structure, int n\_seq, float \*energy)

Usage of this function is discouraged! Use [vrna\\_eval\\_structure\(\)](#), and [vrna\\_eval\\_covar\\_structure\(\)](#) instead!

Global **energy\_of\_circ\_struct** (const char \*string, const char \*structure)

This function is deprecated and should not be used in future programs Use [energy\\_of\\_circ\\_structure\(\)](#) instead!

Global **energy\_of\_circ\_struct\_par** (const char \*string, const char \*structure, vrna\_param\_t \*parameters, int verbosity\_level)

Use [vrna\\_eval\\_structure\(\)](#) or [vrna\\_eval\\_structure\\_verbose\(\)](#) instead!

Global **energy\_of\_circ\_structure** (const char \*string, const char \*structure, int verbosity\_level)

Use [vrna\\_eval\\_structure\(\)](#) or [vrna\\_eval\\_structure\\_verbose\(\)](#) instead!

Global **energy\_of\_move** (const char \*string, const char \*structure, int m1, int m2)

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

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

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

Global **energy\_of\_struct** (const char \*string, const char \*structure)

This function is deprecated and should not be used in future programs! Use [energy\\_of\\_structure\(\)](#) instead!

Global **energy\_of\_struct\_par** (const char \*string, const char \*structure, vrna\_param\_t \*parameters, int verbosity\_level)

Use [vrna\\_eval\\_structure\(\)](#) or [vrna\\_eval\\_structure\\_verbose\(\)](#) instead!

Global **energy\_of\_struct\_pt** (const char \*string, short \*ptable, short \*s, short \*s1)

This function is deprecated and should not be used in future programs! Use [energy\\_of\\_structure\\_pt\(\)](#) instead!

Global **energy\_of\_struct\_pt\_par** (const char \*string, short \*ptable, short \*s, short \*s1, vrna\_param\_t \*parameters, int verbosity\_level)

Use [vrna\\_eval\\_structure\\_pt\(\)](#) or [vrna\\_eval\\_structure\\_pt\\_verbose\(\)](#) instead!

Global **energy\_of\_structure** (const char \*string, const char \*structure, int verbosity\_level)

Use [vrna\\_eval\\_structure\(\)](#) or [vrna\\_eval\\_structure\\_verbose\(\)](#) instead!

Global **energy\_of\_structure\_pt** (const char \*string, short \*ptable, short \*s, short \*s1, int verbosity\_level)

Use [vrna\\_eval\\_structure\\_pt\(\)](#) or [vrna\\_eval\\_structure\\_pt\\_verbose\(\)](#) instead!

Global **expHairpinEnergy** (int u, int type, short si1, short sj1, const char \*string)

Use [exp\\_E\\_Hairpin\(\)](#) from [loop\\_energies.h](#) instead

Global **expLoopEnergy** (int u1, int u2, int type, int type2, short si1, short sj1, short sp1, short sq1)

Use [exp\\_E\\_IntLoop\(\)](#) from [loop\\_energies.h](#) instead

Global **export\_ali\_bppm** (void)

Usage of this function is discouraged! The new [vrna\\_fold\\_compound\\_t](#) allows direct access to the folding matrices, including the pair probabilities! The pair probability array returned here reflects the one of the latest call to [vrna\\_pf\(\)](#), or any of the old API calls for consensus structure partition function folding.

Global **export\_circfold\_arrays** (int \*Fc\_p, int \*FcH\_p, int \*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)

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 \*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)

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

Global **export\_co\_bppm** (void)

This function is deprecated and will be removed soon! The base pair probability array is available through the [vrna\\_fold\\_compound\\_t](#) data structure, and its associated [vrna\\_mx\\_pf\\_t](#) member.

Global **export\_cofold\_arrays** (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*fc\_p, int \*\*indx\_p, char \*\*ptype\_p)

folding matrices now reside within the [vrna\\_fold\\_compound\\_t](#). Thus, this function will only work in conjunction with a prior call to the deprecated functions [cofold\(\)](#) or [cofold\\_par\(\)](#)

Global **export\_cofold\_arrays\_gg** (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*fc\_p, int \*\*ggg\_p, int \*\*indx\_p, char \*\*ptype\_p)

folding matrices now reside within the fold compound. Thus, this function will only work in conjunction with a prior call to [cofold\(\)](#) or [cofold\\_par\(\)](#)

Global **export\_fold\_arrays** (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*indx\_p, char \*\*ptype\_p)

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

Global **export\_fold\_arrays\_par** (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*indx\_p, char \*\*ptype\_p, vrna\_param\_t \*\*P\_p)

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

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

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

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

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

Global **fold\_par** (const char \*sequence, char \*structure, vrna\_param\_t \*parameters, int is\_constrained, int is\_circular)

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

Global **free\_alifold\_arrays** (void)

Usage of this function is discouraged! It only affects memory being free'd that was allocated by an old API function before. Release of memory occupied by the newly introduced [vrna\\_fold\\_compound\\_t](#) is handled by [vrna\\_vrna\\_fold\\_compound\\_free\(\)](#)

Global **free\_alipf\_arrays** (void)

Usage of this function is discouraged! This function only free's memory allocated by old API function calls. Memory allocated by any of the new API calls (starting with vrna\_) will be not affected!

Global **free\_arrays** (void)

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

Global **free\_co\_arrays** (void)

This function will only free memory allocated by a prior call of [cofold\(\)](#) or [cofold\\_par\(\)](#). See [vrna\\_mfe\\_dimer\(\)](#) for how to use the new API

Global **free\_co\_pf\_arrays** (void)

This function will be removed for the new API soon! See [vrna\\_pf\\_dimer\(\)](#), [vrna\\_fold\\_compound\(\)](#), and [vrna\\_fold\\_compound\\_free\(\)](#) for an alternative

Global **free\_pf\_arrays** (void)

See [vrna\\_fold\\_compound\\_t](#) and its related functions for how to free memory occupied by the dynamic programming matrices

Global **get\_alipf\_arrays** (short \*\*\*S\_p, short \*\*\*S5\_p, short \*\*\*S3\_p, unsigned short \*\*\*a2s\_p, char \*\*\*Ss\_p, FLT\_OR\_DBL \*\*qb\_p, FLT\_OR\_DBL \*\*qm\_p, FLT\_OR\_DBL \*\*q1k\_p, FLT\_OR\_DBL \*\*qln\_p, short \*\*pscore)

It is discouraged to use this function! The new [vrna\\_fold\\_compound\\_t](#) allows direct access to all necessary consensus structure prediction related variables!

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

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

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

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

Global **get\_boltzmann\_factors\_ali** (unsigned int n\_seq, double temperature, double betaScale, vrna\_md\_t md, double pf\_scale)

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

Global **get\_centroid\_struct\_gquad\_pr** (int length, double \*dist)

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

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

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

Global **get\_centroid\_struct\_pr** (int length, double \*dist, FLT\_OR\_DBL \*pr)

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

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

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

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

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

Global **get\_mpi** (char \*Aseq[], int n\_seq, int length, int \*mini)

Use [vrna\\_aln\\_mpi\(\)](#) as a replacement

Global **get\_plist** (vrna\_plist\_t \*pl, int length, double cut\_off)

{ This function is deprecated and will be removed soon! } use [assign\\_plist\\_from\\_pr\(\)](#) instead!

Global **get\_scaled\_alipf\_parameters** (unsigned int n\_seq)

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

Global **get\_scaled\_parameters** (double temperature, vrna\_md\_t md)

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

Global **get\_scaled\_pf\_parameters** (void)

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

Global **get\_TwoDfold\_variables** (const char \*seq, const char \*structure1, const char \*structure2, int circ)

Use the new API that relies on [vrna\\_fold\\_compound\\_t](#) and the corresponding functions [vrna\\_fold\\_compound↵\\_TwoD\(\)](#), [vrna\\_mfe\\_TwoD\(\)](#), and [vrna\\_fold\\_compound\\_free\(\)](#) instead!

Global **get\_TwoDpfold\_variables** (const char \*seq, const char \*structure1, char \*structure2, int circ)

Use the new API that relies on [vrna\\_fold\\_compound\\_t](#) and the corresponding functions [vrna\\_fold\\_compound↵\\_TwoD\(\)](#), [vrna\\_pf\\_TwoD\(\)](#), and [vrna\\_fold\\_compound\\_free\(\)](#) instead!

Global **HairpinE** (int size, int type, int si1, int sj1, const char \*string)

{This function is deprecated and will be removed soon. Use [E\\_Hairpin\(\)](#) instead!}

Global **hamming** (const char \*s1, const char \*s2)

Use [vrna\\_hamming\\_distance\(\)](#) instead!

Global **hamming\_bound** (const char \*s1, const char \*s2, int n)

Use [vrna\\_hamming\\_distance\\_bound\(\)](#) instead!

Global **iindx**

Do not use this variable anymore!

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

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

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

This function is obsolete and will be removed soon!

Global **init\_rand** (void)

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

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

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

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

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

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

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

Global **Lfold** (const char \*string, char \*structure, int maxdist)

Use [vrna\\_mfe\\_window\(\)](#) instead!

Global **Lfoldz** (const char \*string, char \*structure, int maxdist, int zsc, double min\_z)

Use [vrna\\_mfe\\_window\\_zscore\(\)](#) instead!

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

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

Global **LoopEnergy** (int n1, int n2, int type, int type\_2, int si1, int sj1, int sp1, int sq1)

{This function is deprecated and will be removed soon. Use [E\\_IntLoop\(\)](#) instead!}

Global **Make\_bp\_profile** (int length)

This function is deprecated and will be removed soon! See [Make\\_bp\\_profile\\_bppm\(\)](#) for a replacement

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

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

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

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

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

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

Global **mean\_bp\_dist** (int length)

This function is not threadsafe and should not be used anymore. Use [mean\\_bp\\_distance\(\)](#) instead!

Global **mean\_bp\_distance** (int length)

Use [vrna\\_mean\\_bp\\_distance\(\)](#) or [vrna\\_mean\\_bp\\_distance\\_pr\(\)](#) instead!

Global **mean\_bp\_distance\_pr** (int length, FLT\_OR\_DBL \*pr)

Use [vrna\\_mean\\_bp\\_distance\(\)](#) or [vrna\\_mean\\_bp\\_distance\\_pr\(\)](#) instead!

Global **nc\_fact**

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

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

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

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

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

Global **PAIR**

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

Global **pair\_info**

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

Global **paramT**

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

Global **parenthesis\_structure** (char \*structure, vrna\_bp\_stack\_t \*bp, int length)

use [vrna\\_parenthesis\\_structure\(\)](#) instead

Global **parenthesis\_zuker** (char \*structure, vrna\_bp\_stack\_t \*bp, int length)

use [vrna\\_parenthesis\\_zuker](#) instead

Global **path\_t**

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

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

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

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

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

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

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

Global **pf\_paramT**

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

Global **plist**

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

Global **pr**

Do not use this variable anymore!

Global **print\_tty\_constraint** (unsigned int option)

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

Global **print\_tty\_constraint\_full** (void)

Use [vrna\\_message\\_constraint\\_options\\_all\(\)](#) instead!



**Global `print_tty_input_seq` (void)**

Use `vrna_message_input_seq_simple()` instead!

**Global `print_tty_input_seq_str` (const char \*s)**

Use `vrna_message_input_seq()` instead!

**Global `PS_dot_plot` (char \*string, char \*file)**

This function is deprecated and will be removed soon! Use `PS_dot_plot_list()` instead!

**Global `PS_rna_plot` (char \*string, char \*structure, char \*file)**

Use `vrna_file_PS_rnaplot()` instead!

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

Use `vrna_file_PS_rnaplot_a()` instead!

**Global `PS_rna_plot_a_gquad` (char \*string, char \*structure, char \*ssfile, char \*pre, char \*post)**

Use `vrna_file_PS_rnaplot_a()` instead!

**Global `random_string` (int l, const char symbols[])**

Use `vrna_random_string()` instead!

**Global `read_record` (char \*\*header, char \*\*sequence, char \*\*\*rest, unsigned int options)**

This function is deprecated! Use `vrna_file_fasta_read_record()` as a replacement.

**Global `scale_parameters` (void)**

Use `vrna_params()` instead!

**Global `sect`**

Use `vrna_sect_t` instead!

**Global `set_model_details` (vrna\_md\_t \*md)**

This function will vanish as soon as backward compatibility of RNALib is dropped (expected in version 3). Use `vrna_md_set_default()` instead!

**Global `space` (unsigned size)**

Use `vrna_alloc()` instead!

**Global `st_back`**

set the `uniq_ML` flag in `vrna_md_t` before passing it to `vrna_fold_compound()`.

**Global `stackProb` (double cutoff)**

Use `vrna_stack_prob()` instead!

**Global `str_DNA2RNA` (char \*sequence)**

Use `vrna_seq_toRNA()` instead!

**Global `str_uppercase` (char \*sequence)**

Use `vrna_seq_toupper()` instead!

**Global `temperature`**

Use `vrna_md_defaults_temperature()`, and `vrna_md_defaults_temperature_get()` to change, and read the global default temperature settings

**Global `time_stamp` (void)**

Use `vrna_time_stamp()` instead!

**Global `TwoDfold_backtrack_f5` (unsigned int j, int k, int l, `TwoDfold_vars` \*vars)**

Use the new API that relies on `vrna_fold_compound_t` and the corresponding functions `vrna_fold_compound_TwoD()`, `vrna_mfe_TwoD()`, `vrna_backtrack5_TwoD()`, and `vrna_fold_compound_free()` instead!

**Global `TwoDfold_vars`**

This data structure will be removed from the library soon! Use `vrna_fold_compound_t` and the corresponding functions `vrna_fold_compound_TwoD()`, `vrna_mfe_TwoD()`, and `vrna_fold_compound_free()` instead!

**Global `TwoDfoldList` (`TwoDfold_vars` \*vars, int distance1, int distance2)**

Use the new API that relies on `vrna_fold_compound_t` and the corresponding functions `vrna_fold_compound↔_TwoD()`, `vrna_mfe_TwoD()`, and `vrna_fold_compound_free()` instead!

**Global `TwoDpfold_pbacktrack` (`TwoDpfold_vars` \*vars, int d1, int d2)**

Use the new API that relies on `vrna_fold_compound_t` and the corresponding functions `vrna_fold_compound↔_TwoD()`, `vrna_pf_TwoD()`, `vrna_pbacktrack_TwoD()`, and `vrna_fold_compound_free()` instead!

**Global `TwoDpfold_pbacktrack5` (`TwoDpfold_vars` \*vars, int d1, int d2, unsigned int length)**

Use the new API that relies on `vrna_fold_compound_t` and the corresponding functions `vrna_fold_compound↔_TwoD()`, `vrna_pf_TwoD()`, `vrna_pbacktrack5_TwoD()`, and `vrna_fold_compound_free()` instead!

**Class `TwoDpfold_vars`**

This data structure will be removed from the library soon! Use `vrna_fold_compound_t` and the corresponding functions `vrna_fold_compound_TwoD()`, `vrna_pf_TwoD()`, and `vrna_fold_compound_free()` instead!

**Global `TwoDpfoldList` (`TwoDpfold_vars` \*vars, int maxDistance1, int maxDistance2)**

Use the new API that relies on `vrna_fold_compound_t` and the corresponding functions `vrna_fold_compound↔_TwoD()`, `vrna_pf_TwoD()`, and `vrna_fold_compound_free()` instead!

**Global `unpack_structure` (const char \*packed)**

Use `vrna_db_unpack()` as a replacement

**Global `update_alifold_params` (void)**

Usage of this function is discouraged! The new API uses `vrna_fold_compound_t` to lump all folding related necessities together, including the energy parameters. Use `vrna_update_fold_params()` to update the energy parameters within a `vrna_fold_compound_t`.

**Global `update_co_pf_params` (int length)**

Use `vrna_exp_params_subst()` instead!

**Global `update_co_pf_params_par` (int length, `vrna_exp_param_t` \*parameters)**

Use `vrna_exp_params_subst()` instead!

**Global `update_cofold_params` (void)**

See `vrna_params_subst()` for an alternative using the new API

**Global `update_cofold_params_par` (`vrna_param_t` \*parameters)**

See `vrna_params_subst()` for an alternative using the new API

**Global `update_fold_params` (void)**

For non-default model settings use the new API with `vrna_params_subst()` and `vrna_mfe()` instead!

**Global `update_fold_params_par` (`vrna_param_t` \*parameters)**

For non-default model settings use the new API with `vrna_params_subst()` and `vrna_mfe()` instead!

**Global `update_pf_params` (int length)**

Use `vrna_exp_params_subst()` instead

**Global `update_pf_params_par` (int length, `vrna_exp_param_t` \*parameters)**

Use `vrna_exp_params_subst()` instead

**Global `urn` (void)**

Use `vrna_urn()` instead!

**Global `VRNA_CONSTRAINT_FILE`**

Use 0 instead!

**Global `VRNA_CONSTRAINT_MULTILINE`**

see `vrna_extract_record_rest_structure()`

**Global `VRNA_CONSTRAINT_NO_HEADER`**

This mode is not supported anymore!

**Global `VRNA_CONSTRAINT_SOFT_MFE`**

This flag has no meaning anymore, since constraints are now always stored!

**Global `VRNA_CONSTRAINT_SOFT_PF`**

Use `VRNA_OPTION_PF` instead!

**Global `vrna_exp_param_s::id`**

This attribute will be removed in version 3

**Global `vrna_extract_record_rest_constraint` (`char **cstruc, const char **lines, unsigned int option`)**

Use `vrna_extract_record_rest_structure()` instead!

**Global `vrna_fc_s::pscore_pf_compat`**

This attribute will vanish in the future!

**Global `vrna_fc_s::ptype_pf_compat`**

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

**Global `warn_user` (`const char message[]`)**

Use `vrna_message_warning()` instead!

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

Use `vrna_realloc()` instead!

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

use `vrna_zukersubopt()` instead

**Global `zukersubopt_par` (`const char *string, vrna_param_t *parameters`)**

use `vrna_zukersubopt()` instead



## Chapter 9

## Bug List

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

Due to resizing, any pre-existing constraints will be lost!



## Chapter 10

# Module Index

### 10.1 Modules

Here is a list of all modules:

RNA Secondary Structure Prediction . . . . .	43
Computing Minimum Free Energy (MFE) Structures . . . . .	145
MFE Structures of single Nucleic Acid Sequences . . . . .	172
MFE Structures of two hybridized Sequences . . . . .	178
MFE Consensus Structures for Sequence Alignment(s) . . . . .	197
Local MFE structure Prediction and Z-scores . . . . .	207
Calculating MFE representatives of a Distance Based Partitioning . . . . .	224
Computing Partition Functions and Pair Probabilities . . . . .	147
Compute the structure with maximum expected accuracy (MEA) . . . . .	159
Compute the centroid structure . . . . .	160
Partition Function for two hybridized Sequences . . . . .	184
Partition Function for two hybridized Sequences as a stepwise Process . . . . .	188
Partition Function and Base Pair Probabilities for Sequence Alignment(s) . . . . .	201
Partition functions for locally stable secondary structures . . . . .	211
Calculate Partition Functions of a Distance Based Partitioning . . . . .	231
Enumerating Suboptimal Structures . . . . .	162
Suboptimal structures according to Zuker et al. 1989 . . . . .	163
Suboptimal structures within an energy band around the MFE . . . . .	165
Stochastic backtracking in the Ensemble . . . . .	168
Stochastic Backtracking of Consensus Structures from Sequence Alignment(s) . . . . .	205
Stochastic Backtracking of Structures from Distance Based Partitioning . . . . .	234
Calculate Secondary Structures of two RNAs upon Dimerization . . . . .	171
MFE Structures of two hybridized Sequences . . . . .	178
Partition Function for two hybridized Sequences . . . . .	184
Partition Function for two hybridized Sequences as a stepwise Process . . . . .	188
Predicting Consensus Structures from Alignment(s) . . . . .	191
MFE Consensus Structures for Sequence Alignment(s) . . . . .	197
Partition Function and Base Pair Probabilities for Sequence Alignment(s) . . . . .	201
Stochastic Backtracking of Consensus Structures from Sequence Alignment(s) . . . . .	205
Local MFE consensus structures for Sequence Alignments . . . . .	214
Predicting Locally stable structures of large sequences . . . . .	206
Local MFE structure Prediction and Z-scores . . . . .	207
Partition functions for locally stable secondary structures . . . . .	211
Local MFE consensus structures for Sequence Alignments . . . . .	214

Classified Dynamic Programming . . . . .	222
Distance based partitioning of the Secondary Structure Space . . . . .	223
Calculating MFE representatives of a Distance Based Partitioning . . . . .	224
Calculate Partition Functions of a Distance Based Partitioning . . . . .	231
Stochastic Backtracking of Structures from Distance Based Partitioning . . . . .	234
Compute the Density of States . . . . .	236
Inverse Secondary Structure Prediction . . . . .	45
Refolding paths between secondary structures . . . . .	47
Direct refolding paths between two secondary structures . . . . .	287
Free Energy Evaluation for given Sequence / Structure Pairs . . . . .	48
Processing and Evaluating Decomposed Loops . . . . .	64
Energy Parameter Sets and Boltzmann Factors . . . . .	75
Reading/Writing Energy Parameter Sets from/to File . . . . .	215
Converting Energy Parameter Files . . . . .	217
Manipulation of the Prediction Models . . . . .	87
Constraining the Secondary Structure Predictions and Evaluations . . . . .	118
Hard Constraints . . . . .	237
Soft Constraints . . . . .	247
Incorporating SHAPE reactivity data . . . . .	255
Incorporating ligands binding to specific sequence/structure motifs . . . . .	259
Generate soft constraints from data . . . . .	261
Data Structures and Preprocessor Macros . . . . .	132
The Fold Compound . . . . .	265
The Dynamic Programming Matrices . . . . .	280
Utilities . . . . .	137
Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings . . . . .	291
Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures . . . . .	295
Various utilities for Sequence Alignments, and Comparative Structure Prediction . . . . .	309
Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments . . . . .	312
Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More . . . . .	319



## Chapter 11

# Data Structure Index

### 11.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">_struct_en</a>	327
<a href="#">LIST</a>	327
<a href="#">LST_BUCKET</a>	328
<a href="#">Postorder_list</a>	328
<a href="#">swString</a>	328
<a href="#">Tree</a>	328
<a href="#">TwoDpfold_vars</a>	
Variables compound for 2Dfold partition function folding	329
<a href="#">vrna_subopt_sol_s</a>	
Solution element from subopt.c	330



## Chapter 12

# File Index

### 12.1 File List

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

ViennaRNA/1.8.4_epars.h	
Free energy parameters for parameter file conversion . . . . .	331
ViennaRNA/1.8.4_intloops.h	
Free energy parameters for interior loop contributions needed by the parameter file conversion functions . . . . .	331
ViennaRNA/2Dfold.h . . . . .	332
ViennaRNA/2Dpfold.h . . . . .	333
ViennaRNA/ali_plex.h . . . . .	??
ViennaRNA/alifold.h	
Compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments . . . . .	336
ViennaRNA/aln_util.h	
Various utility- and helper-functions for sequence alignments and comparative structure prediction . . . . .	338
ViennaRNA/alphabet.h	
Functions to process, convert, and generally handle different nucleotide and/or base pair alphabets . . . . .	339
ViennaRNA/boltzmann_sampling.h	
Boltzmann Sampling of secondary structures from the ensemble . . . . .	341
ViennaRNA/centroid.h	
Centroid structure computation . . . . .	343
ViennaRNA/cofold.h	
MFE version of cofolding routines . . . . .	344
ViennaRNA/constraints.h	
Functions and data structures for constraining secondary structure predictions and evaluation .	345
ViennaRNA/constraints_hard.h	
Functions and data structures for handling of secondary structure hard constraints . . . . .	347
ViennaRNA/constraints_SHAPE.h	
This module provides function to incorporate SHAPE reactivity data into the folding recursions by means of soft constraints . . . . .	350
ViennaRNA/constraints_soft.h	
Functions and data structures for secondary structure soft constraints . . . . .	352
ViennaRNA/convert_epars.h	
Functions and definitions for energy parameter file format conversion . . . . .	353
ViennaRNA/data_structures.h . . . . .	354

ViennaRNA/dist_vars.h	
Global variables for Distance-Package	356
ViennaRNA/dp_matrices.h	357
ViennaRNA/duplex.h	
Duplex folding function declarations..	358
ViennaRNA/edit_cost.h	
Global variables for Edit Costs included by treedist.c and stringdist.c	358
ViennaRNA/energy_const.h	359
ViennaRNA/energy_par.h	??
ViennaRNA/eval.h	
Functions and variables related to energy evaluation of sequence/structure pairs	360
ViennaRNA/exterior_loops.h	
Energy evaluation of exterior loops for MFE and partition function calculations	362
ViennaRNA/file_formats.h	
Functions dealing with file formats for RNA sequences, structures, and alignments	363
ViennaRNA/findpath.h	364
ViennaRNA/fold.h	
MFE calculations for single RNA sequences	365
ViennaRNA/fold_vars.h	
Here all all declarations of the global variables used throughout RNALib	366
ViennaRNA/gquad.h	
Various functions related to G-quadruplex computations	368
ViennaRNA/hairpin_loops.h	
Energy evaluation of hairpin loops for MFE and partition function calculations	369
ViennaRNA/interior_loops.h	
Energy evaluation of interior loops for MFE and partition function calculations	370
ViennaRNA/intl11.h	??
ViennaRNA/intl11dH.h	??
ViennaRNA/intl21.h	??
ViennaRNA/intl21dH.h	??
ViennaRNA/intl22.h	??
ViennaRNA/intl22dH.h	??
ViennaRNA/inverse.h	
Inverse folding routines	371
ViennaRNA/Lfold.h	
Predicting local MFE structures of large sequences	372
ViennaRNA/ligand.h	
Functions for incorporation of ligands binding to haipirn and interior loop motifs	373
ViennaRNA/list.h	??
ViennaRNA/loop_energies.h	
Energy evaluation for MFE and partition function calculations	373
ViennaRNA/LPfold.h	
Function declarations of partition function variants of the Lfold algorithm	374
ViennaRNA/MEA.h	
Computes a MEA (maximum expected accuracy) structure	375
ViennaRNA/mfe.h	
MFE calculations for single RNA sequences	376
ViennaRNA/mm.h	
Several Maximum Matching implementations	376
ViennaRNA/model.h	
The model details data structure and its corresponding modifiers	377
ViennaRNA/move_set.h	??
ViennaRNA/multibranch_loops.h	
Energy evaluation of multibranch loops for MFE and partition function calculations	381
ViennaRNA/naview.h	383
ViennaRNA/pair_mat.h	??
ViennaRNA/params.h	383

ViennaRNA/ <a href="#">part_func.h</a>	
Partition function of single RNA sequences . . . . .	385
ViennaRNA/ <a href="#">part_func_co.h</a>	
Partition function for two RNA sequences . . . . .	388
ViennaRNA/ <a href="#">part_func_up.h</a>	
Partition Function Cofolding as stepwise process . . . . .	393
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 . . . . .	394
ViennaRNA/ <b>PKplex.h</b>	??
ViennaRNA/ <b>plex.h</b>	??
ViennaRNA/ <a href="#">plot_aln.h</a>	
Various functions for plotting Sequence / Structure Alignments . . . . .	396
ViennaRNA/ <a href="#">plot_layouts.h</a>	
Secondary structure plot layout algorithms . . . . .	396
ViennaRNA/ <a href="#">plot_structure.h</a>	
Various functions for plotting RNA secondary structures . . . . .	398
ViennaRNA/ <b>ProfileAln.h</b>	??
ViennaRNA/ <a href="#">profiledist.h</a>	399
ViennaRNA/ <a href="#">PS_dot.h</a>	
Various functions for plotting RNA secondary structures, dot-plots and other visualizations . . .	401
ViennaRNA/ <a href="#">read_epars.h</a>	401
ViennaRNA/ <a href="#">ribo.h</a>	
Parse RiboSum Scoring Matrices for Covariance Scoring of Alignments . . . . .	402
ViennaRNA/ <a href="#">RNAstruct.h</a>	
Parsing and Coarse Graining of Structures . . . . .	402
ViennaRNA/ <b>snoifold.h</b>	??
ViennaRNA/ <b>snoop.h</b>	??
ViennaRNA/ <a href="#">string_utils.h</a>	
General utility- and helper-functions for RNA sequence and structure strings used throughout the ViennaRNA Package . . . . .	404
ViennaRNA/ <a href="#">stringdist.h</a>	
Functions for String Alignment . . . . .	406
ViennaRNA/ <a href="#">structure_utils.h</a>	
Various utility- and helper-functions for secondary structure parsing, converting, etc . . . . .	407
ViennaRNA/ <a href="#">subopt.h</a>	
RNAsubopt and density of states declarations . . . . .	410
ViennaRNA/ <b>svm_utils.h</b>	??
ViennaRNA/ <a href="#">treedist.h</a>	
Functions for <a href="#">Tree</a> Edit Distances . . . . .	411
ViennaRNA/ <a href="#">utils.h</a>	
General utility- and helper-functions used throughout the <i>ViennaRNA Package</i> . . . . .	412



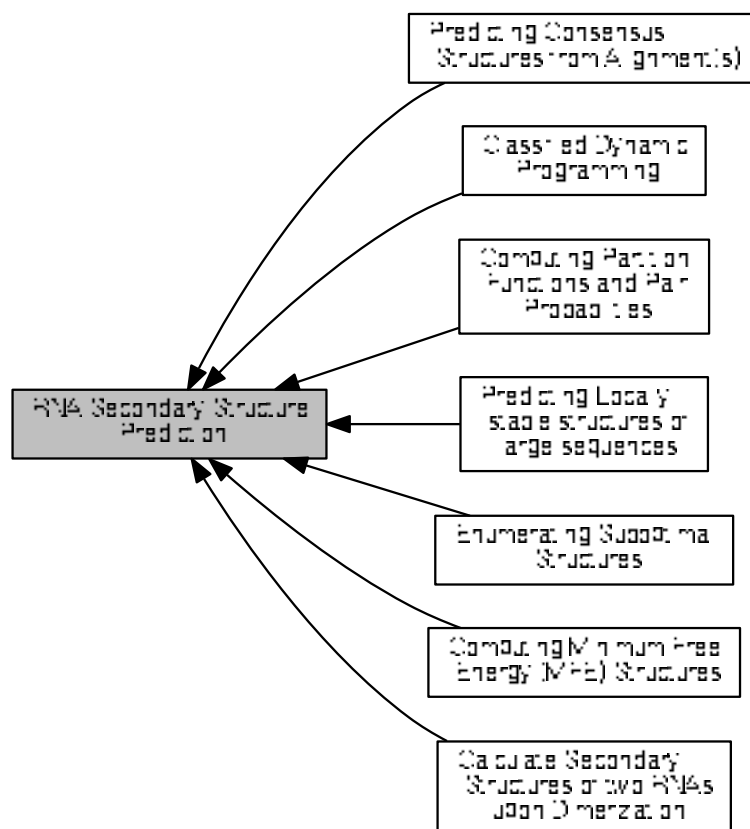
## Chapter 13

# Module Documentation

### 13.1 RNA Secondary Structure Prediction

This module contains all functions related to thermodynamic folding of RNAs.

Collaboration diagram for RNA Secondary Structure Prediction:



## Modules

- [Computing Minimum Free Energy \(MFE\) Structures](#)

*This section covers all functions and variables related to the calculation of minimum free energy (MFE) structures.*

- [Computing Partition Functions and Pair Probabilities](#)

*This section provides information about all functions and variables related to the calculation of the partition function and base pair probabilities.*

- [Enumerating Suboptimal Structures](#)
- [Calculate Secondary Structures of two RNAs upon Dimerization](#)

*Predict structures formed by two molecules upon hybridization.*

- [Predicting Consensus Structures from Alignment\(s\)](#)

*compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments*

- [Predicting Locally stable structures of large sequences](#)
- [Classified Dynamic Programming](#)

## Files

- file [mm.h](#)

*Several Maximum Matching implementations.*

### 13.1.1 Detailed Description

This module contains all functions related to thermodynamic folding of RNAs.



## 13.2 Inverse Secondary Structure Prediction

### 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](#)

### 13.2.1 Detailed Description

We provide two functions that search for sequences with a given structure, thereby inverting the folding routines.

### 13.2.2 Function Documentation

#### 13.2.2.1 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

**13.2.2.2 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

**13.2.3 Variable Documentation****13.2.3.1 float final\_cost**

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

when to stop [inverse\\_pf\\_fold\(\)](#)

**13.2.3.2 int give\_up**

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

default 0: try to minimize structure distance even if no exact solution can be found

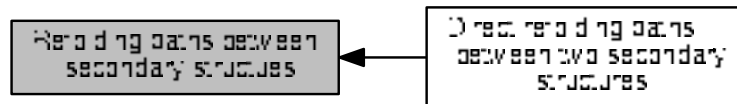
**13.2.3.3 int inv\_verbose**

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

print out substructure on which [inverse\\_fold\(\)](#) fails

## 13.3 Refolding paths between secondary structures

Collaboration diagram for Refolding paths between secondary structures:



### Modules

- [Direct refolding paths between two secondary structures](#)

*Implementation of heuristics to explore optimal (re-)folding paths between two secondary structures.*

### 13.3.1 Detailed Description

## 13.4 Free Energy Evaluation for given Sequence / Structure Pairs

This module contains all functions and variables related to energy evaluation of sequence/structure pairs.

### Functions

- 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\\_simple](#) (const char \*string, const char \*structure)  
*Calculate the free energy of an already folded RNA.*
- 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\\_simple\\_verbose](#) (const char \*string, const char \*structure, FILE \*file)  
*Calculate the free energy of an already folded RNA and print contributions per loop.*
- 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\\_simple](#) (const char \*string, 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\\_simple\\_verbose](#) (const char \*string, const short \*pt, FILE \*file)  
*Calculate the free energy of an already folded RNA.*
- int [vrna\\_eval\\_loop\\_pt](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int i, const short \*pt)  
*Calculate energy of a loop.*
- float [vrna\\_eval\\_move](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*structure, int m1, int m2)  
*Calculate energy of a move (closing or opening of a base pair)*
- int [vrna\\_eval\\_move\\_pt](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, short \*pt, int m1, int m2)  
*Calculate energy of a move (closing or opening of a base pair)*
- float [energy\\_of\\_structure](#) (const char \*string, const char \*structure, int verbosity\_level)  
*Calculate the free energy of an already folded RNA using global model detail settings.*
- float [energy\\_of\\_struct\\_par](#) (const char \*string, const char \*structure, [vrna\\_param\\_t](#) \*parameters, int verbosity\_level)  
*Calculate the free energy of an already folded RNA.*
- float [energy\\_of\\_circ\\_structure](#) (const char \*string, const char \*structure, int verbosity\_level)  
*Calculate the free energy of an already folded circular RNA.*
- float [energy\\_of\\_circ\\_struct\\_par](#) (const char \*string, const char \*structure, [vrna\\_param\\_t](#) \*parameters, int verbosity\_level)  
*Calculate the free energy of an already folded circular RNA.*
- int [energy\\_of\\_structure\\_pt](#) (const char \*string, short \*ptable, short \*s, short \*s1, int verbosity\_level)  
*Calculate the free energy of an already folded RNA.*
- int [energy\\_of\\_struct\\_pt\\_par](#) (const char \*string, short \*ptable, short \*s, short \*s1, [vrna\\_param\\_t](#) \*parameters, int verbosity\_level)  
*Calculate the free energy of an already folded RNA.*
- float [energy\\_of\\_move](#) (const char \*string, const char \*structure, int m1, int m2)  
*Calculate energy of a move (closing or opening of a base pair)*
- int [energy\\_of\\_move\\_pt](#) (short \*pt, short \*s, short \*s1, int m1, int m2)  
*Calculate energy of a move (closing or opening of a base pair)*
- int [loop\\_energy](#) (short \*ptable, short \*s, short \*s1, int i)

*Calculate energy of a loop.*

- float [energy\\_of\\_struct](#) (const char \*string, const char \*structure)
- int [energy\\_of\\_struct\\_pt](#) (const char \*string, short \*ptable, short \*s, short \*s1)
- float [energy\\_of\\_circ\\_struct](#) (const char \*string, const char \*structure)
- int [vrna\\_eval\\_ext\\_hp\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int i, int j)

*Evaluate free energy of an exterior hairpin loop.*

- int [vrna\\_eval\\_hp\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int i, int j)

*Evaluate free energy of a hairpin loop.*

## Variables

- int [cut\\_point](#)  
*set to first pos of second seq for cofolding*
- int [eos\\_debug](#)  
*verbose info from energy\_of\_struct*

### 13.4.1 Detailed Description

This module contains all functions and variables related to energy evaluation of sequence/structure pairs.

### 13.4.2 Function Documentation

13.4.2.1 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\\_VC\\_TYPE\\_SINGLE](#) and [VRNA\\_VC\\_TYPE\\_ALIGNMENT](#)

#### See also

[vrna\\_eval\\_structure\\_pt\(\)](#), [vrna\\_eval\\_structure\\_verbose\(\)](#), [vrna\\_eval\\_structure\\_pt\\_verbose\(\)](#), [vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_comparative\(\)](#), [vrna\\_eval\\_covar\\_structure\(\)](#)

#### Parameters

<i>vc</i>	A <a href="#">vrna_fold_compound_t</a> containing the energy parameters and model details
<i>structure</i>	Secondary structure in dot-bracket notation

**Returns**

The free energy of the input structure given the input sequence in kcal/mol

**13.4.2.2 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 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\\_VC\\_TYPE\\_ALIGNMENT](#) only!

**See also**

[vrna\\_fold\\_compound\\_comparative\(\)](#), [vrna\\_eval\\_structure\(\)](#)

**Parameters**

<i>vc</i>	A <a href="#">vrna_fold_compound_t</a> containing the energy parameters and model details
<i>structure</i>	Secondary (consensus) structure in dot-bracket notation

**Returns**

The covariance pseudo energy score of the input structure given the input sequence alignment in kcal/mol

**13.4.2.3 float 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

13.4.2.4 `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 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. 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>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

13.4.2.5 `float 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 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. 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\\_verbose\(\)](#), [vrna\\_eval\\_structure\\_pt\\_verbose\(\)](#),

## Parameters

<i>string</i>	RNA sequence in uppercase letters
<i>structure</i>	Secondary structure in dot-bracket notation
<i>file</i>	A file handle where this function should print to (may be NULL).

## Returns

The free energy of the input structure given the input sequence in kcal/mol

13.4.2.6 `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

13.4.2.7 `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

13.4.2.8 `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 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.

See also

[vrna\\_ptable\(\)](#), [vrna\\_eval\\_structure\\_pt\(\)](#), [vrna\\_eval\\_structure\\_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
<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

13.4.2.9 `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 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\\_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

13.4.2.10 `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 vrna_fold_compound_t 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

13.4.2.11 `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

13.4.2.12 `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

13.4.2.13 `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

13.4.2.14 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

13.4.2.15 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

13.4.2.16 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

13.4.2.17 `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

13.4.2.18 `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

13.4.2.19 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

13.4.2.20 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

13.4.2.21 `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

13.4.2.22 `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

13.4.2.23 `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

13.4.2.24 `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

13.4.2.25 `int vrna_eval_hp_loop ( vrna_fold_compound_t * vc, int i, int j )`

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

Evaluate free energy of a hairpin loop.

**Note**

This function is polymorphic! The provided `vrna_fold_compound_t` may be of type `VRNA_VC_TYPE_SINGLE` or `VRNA_VC_TYPE_ALIGNMENT`

**Parameters**

<i>vc</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

## 13.5 Processing and Evaluating Decomposed Loops

### Files

- file [exterior\\_loops.h](#)  
*Energy evaluation of exterior loops for MFE and partition function calculations.*
- file [gquad.h](#)  
*Various functions related to G-quadruplex computations.*
- file [hairpin\\_loops.h](#)  
*Energy evaluation of hairpin loops for MFE and partition function calculations.*
- file [interior\\_loops.h](#)  
*Energy evaluation of interior loops for MFE and partition function calculations.*
- file [loop\\_energies.h](#)  
*Energy evaluation for MFE and partition function calculations.*
- file [multibranch\\_loops.h](#)  
*Energy evaluation of multibranch loops for MFE and partition function calculations.*

### Functions

- int [E\\_ExtLoop](#) (int type, int si1, int sj1, [vrna\\_param\\_t](#) \*P)
- [FLT\\_OR\\_DBL exp\\_E\\_ExtLoop](#) (int type, int si1, int sj1, [vrna\\_exp\\_param\\_t](#) \*P)
- int [E\\_Stem](#) (int type, int si1, int sj1, int extLoop, [vrna\\_param\\_t](#) \*P)
- [FLT\\_OR\\_DBL exp\\_E\\_Stem](#) (int type, int si1, int sj1, int extLoop, [vrna\\_exp\\_param\\_t](#) \*P)
- 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)
- 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.*
- 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.*
- int [vrna\\_E\\_hp\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int i, int j)  
*Evaluate the free energy of a hairpin loop and consider possible hard constraints.*
- int [vrna\\_E\\_ext\\_hp\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int i, int j)  
*Evaluate the free energy of an exterior hairpin loop and consider possible hard constraints.*
- [FLT\\_OR\\_DBL vrna\\_exp\\_E\\_hp\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int i, int j)  
*High-Level function for hairpin loop energy evaluation (partition function variant)*
- int [vrna\\_BT\\_hp\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int i, int j, int en, [vrna\\_bp\\_stack\\_t](#) \*bp\_stack, int \*stack\_count)  
*Backtrack a hairpin 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)
- int [E\\_stack](#) (int i, int j, [vrna\\_fold\\_compound\\_t](#) \*vc)  
*Evaluate energy of a base pair stack closed by (i,j)*
- int [vrna\\_BT\\_stack](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, 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` \*vc, int \*i, int \*j, int en, `vrna_bp_stack_t` \*bp\_stack, int \*stack\_count)

*Backtrack an interior loop closed by  $(i, j)$ .*

- int `E_mb_loop_stack` (int i, int j, `vrna_fold_compound_t` \*vc)

*Evaluate energy of a multi branch helices stacking onto closing pair  $(i, j)$*

- int `vrna_BT_mb_loop` (`vrna_fold_compound_t` \*vc, int \*i, int \*j, int \*k, int en, int \*component1, int \*component2)

*Backtrack the decomposition of a multi branch loop closed by  $(i, j)$ .*

### 13.5.1 Detailed Description

### 13.5.2 Function Documentation

#### 13.5.2.1 int E\_ExtLoop ( int type, int si1, int sj1, vrna\_param\_t \* P )

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

Compute the Energy contribution of an Exterior loop stem

This definition is a wrapper for the `E_Stem()` funtion. It is substituted by an `E_Stem()` funtion call with argument `extLoop=1`, so the energy contribution returned reflects a stem introduced in an exterior-loop.

As for the parameters `si1` and `sj1` of the substituted `E_Stem()` function, you can inhibit to take 5'-, 3'-dangles or mismatch contributions to be taken into account by passing -1 to these parameters.

See also

[E\\_Stem\(\)](#)

#### Parameters

<i>type</i>	The pair type of the stem-closing pair
<i>si1</i>	The 5'-mismatching nucleotide
<i>sj1</i>	The 3'-mismatching nucleotide
<i>P</i>	The datastructure containing scaled energy parameters

#### Returns

The energy contribution of the introduced exterior-loop stem

#### 13.5.2.2 FLT\_OR\_DBL exp\_E\_ExtLoop ( int type, int si1, int sj1, vrna\_exp\_param\_t \* P )

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

This is the partition function variant of `E_ExtLoop()`

See also

[E\\_ExtLoop\(\)](#)

Returns

The Boltzmann weighted energy contribution of the introduced exterior-loop stem

13.5.2.3 `int E_Stem ( int type, int si1, int sj1, int extLoop, vrna_param_t * P )`

```
#include <ViennaRNA/exterior_loops.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 sij 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

Parameters

<i>type</i>	The pair type of the first base pair un the stem
<i>si1</i>	The 5'-mismatching nucleotide
<i>sj1</i>	The 3'-mismatching nucleotide
<i>extLoop</i>	A flag that indicates whether the contribution reflects the one of an exterior loop or not
<i>P</i>	The datastructure containing scaled energy parameters

**Returns**

The Free energy of the branch off the loop in dcal/mol

**13.5.2.4 FLT\_OR\_DBL exp\_E\_Stem ( int *type*, int *si1*, int *sj1*, int *extLoop*, vrna\_exp\_param\_t \* *P* )**

```
#include <ViennaRNA/exterior_loops.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

**13.5.2.5 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

### 13.5.2.6 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; ... ;

### 13.5.2.7 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 contibutions

#### Returns

1 on success, 0 if no gquad found

### 13.5.2.8 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

**13.5.2.9** `PRIVATE int E_Hairpin ( int size, int type, int si1, int sj1, const char * string, vrna_param_t * P )`

```
#include <ViennaRNA/hairpin_loops.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
<i>P</i>	The datastructure containing scaled energy parameters

**Returns**

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

**13.5.2.10** `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/hairpin_loops.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
<i>P</i>	The datastructure containing scaled Boltzmann weights of the energy parameters

**Returns**

The Boltzmann weight of the Hairpin-loop

**13.5.2.11** `int vrna_E_hp_loop ( vrna_fold_compound_t * vc, int i, int j )`

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

Evaluate the free energy of a hairpin loop and consider possible hard constraints.

**Note**

This function is polymorphic! The provided [vrna\\_fold\\_compound\\_t](#) may be of type [VRNA\\_VC\\_TYPE\\_SINGLE](#) or [VRNA\\_VC\\_TYPE\\_ALIGNMENT](#)

13.5.2.12 `int vrna_E_ext_hp_loop ( vrna_fold_compound_t * vc, int i, int j )`

```
#include <ViennaRNA/hairpin_loops.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_VC_TYPE_SINGLE` or `VRNA_VC_TYPE_ALIGNMENT`

13.5.2.13 `FLT_OR_DBL vrna_exp_E_hp_loop ( vrna_fold_compound_t * vc, int i, int j )`

```
#include <ViennaRNA/hairpin_loops.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_VC_TYPE_SINGLE` or `VRNA_VC_TYPE_ALIGNMENT`

13.5.2.14 `int vrna_BT_hp_loop ( vrna_fold_compound_t * vc, int i, int j, int en, vrna_bp_stack_t * bp_stack, int * stack_count )`

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

Backtrack a hairpin loop closed by  $(i, j)$ .

#### Note

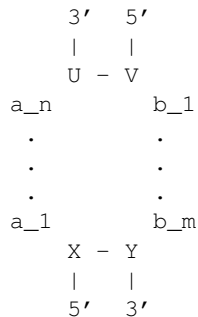
This function is polymorphic! The provided `vrna_fold_compound_t` may be of type `VRNA_VC_TYPE_SINGLE` or `VRNA_VC_TYPE_ALIGNMENT`

13.5.2.15 `int E_IntLoop ( int n1, int n2, int type, int type_2, int si1, int sj1, int sp1, int sq1, vrna_param_t * P )`

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

### Compute the Energy of an interior-loop

This function computes the free energy  $\Delta G$  of an interior-loop with the following structure:



This general structure depicts an interior-loop that is closed by the base pair (X,Y). The enclosed base pair is (V,U) which leaves the unpaired bases  $a_1$ - $a_n$  and  $b_1$ - $b_n$  that constitute the loop. In this example, the length of the interior-loop is  $(n + m)$  where  $n$  or  $m$  may be 0 resulting in a bulge-loop or base pair stack. The mismatching nucleotides for the closing pair (X,Y) are:

5'-mismatch:  $a_1$

3'-mismatch:  $b_m$

and for the enclosed base pair (V,U):

5'-mismatch:  $b_1$

3'-mismatch:  $a_n$

#### Note

Base pairs are always denoted in 5'->3' direction. Thus the enclosed base pair must be 'turned around' when evaluating the free energy of the interior-loop

#### See also

[scale\\_parameters\(\)](#)  
[vrna\\_param\\_t](#)

#### Note

This function is threadsafe

#### Parameters

$n1$	The size of the 'left'-loop (number of unpaired nucleotides)
$n2$	The size of the 'right'-loop (number of unpaired nucleotides)
$type$	The pair type of the base pair closing the interior loop
$type_{\leftarrow 2}$	The pair type of the enclosed base pair
$si1$	The 5'-mismatching nucleotide of the closing pair
$sj1$	The 3'-mismatching nucleotide of the closing pair
$sp1$	The 3'-mismatching nucleotide of the enclosed pair
$sq1$	The 5'-mismatching nucleotide of the enclosed pair
$P$	The datastructure containing scaled energy parameters

**Returns**

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

**13.5.2.16** `PUBLIC 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/interior_loops.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

**13.5.2.17** `int E_mb_loop_stack ( int i, int j, vrna_fold_compound_t * vc )`

```
#include <ViennaRNA/multibranch_loops.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)

13.5.2.18 `int vrna_BT_mb_loop ( vrna_fold_compound_t * vc, int * i, int * j, int * k, int en, int * component1, int * component2 )`

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

Backtrack the decomposition of a multi branch loop closed by  $(i, j)$ .

#### Parameters

<i>vc</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)

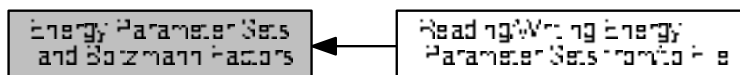
#### Returns

1, if backtracking succeeded, 0 otherwise.

## 13.6 Energy Parameter Sets and Boltzmann Factors

All relevant functions to retrieve and copy precalculated energy parameter sets as well as reading/writing the energy parameter set from/to file(s).

Collaboration diagram for Energy Parameter Sets and Boltzmann Factors:



### Modules

- [Reading/Writing Energy Parameter Sets from/to File](#)  
*Read and Write energy parameter sets from and to text files.*

### Files

- file [params.h](#)

### Data Structures

- struct [vrna\\_param\\_s](#)  
*The datastructure that contains temperature scaled energy parameters. [More...](#)*
- struct [vrna\\_exp\\_param\\_s](#)  
*The datastructure 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.*

### 13.6.1 Detailed Description

All relevant functions to retrieve and copy precalculated energy parameter sets as well as reading/writing the energy parameter set from/to file(s).

This module covers all relevant functions for precalculation 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.

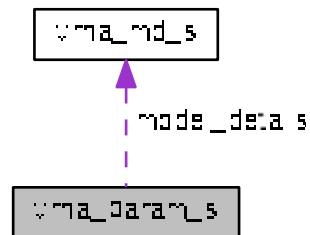


## 13.6.2 Data Structure Documentation

### 13.6.2.1 struct vrna\_param\_s

The datastructure that contains temperature scaled energy parameters.

Collaboration diagram for vrna\_param\_s:



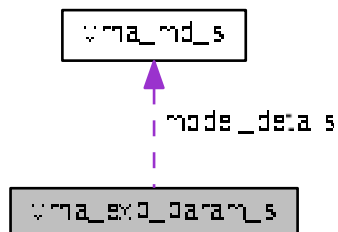
#### Data Fields

- double [temperature](#)  
*Temperature used for loop contribution scaling.*
- [vrna\\_md\\_t model\\_details](#)  
*Model details to be used in the recursions.*

### 13.6.2.2 struct vrna\_exp\_param\_s

The datastructure 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.*

### 13.6.2.2.1 Field Documentation

#### 13.6.2.2.1.1 int vrna\_exp\_param\_s::id

An identifier for the data structure.

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

#### 13.6.2.2.1.2 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  $e^{-E/(\alpha \cdot K \cdot T)}$

## 13.6.3 Typedef Documentation

### 13.6.3.1 typedef struct vrna\_param\_s paramT

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

Old typename of [vrna\\_param\\_s](#).

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

### 13.6.3.2 typedef struct vrna\_exp\_param\_s pf\_paramT

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

Old typename of [#vrna\\_ex\\_param\\_s](#).

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

## 13.6.4 Function Documentation

### 13.6.4.1 vrna\_param\_t\* vrna\_params ( vrna\_md\_t \* md )

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

Get a data structure containing prescaled free energy parameters.

If a NULL pointer is passed for the model details parameter, the default model parameters are stored within the requested [vrna\\_param\\_t](#) structure.

See also

[vrna\\_md\\_t](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_exp\\_params\(\)](#)

## Parameters

<i>md</i>	A pointer to the model details to store inside the structure (Maybe NULL)
-----------	---

## Returns

A pointer to the memory location where the requested parameters are stored

13.6.4.2 `vrna_param_t* vrna_params_copy ( vrna_param_t * par )`

```
#include <ViennaRNA/params.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

13.6.4.3 `vrna_exp_param_t* vrna_exp_params ( vrna_md_t * md )`

```
#include <ViennaRNA/params.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

#### 13.6.4.4 `vrna_exp_param_t* vrna_exp_params_comparative ( unsigned int n_seq, vrna_md_t * md )`

```
#include <ViennaRNA/params.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

#### 13.6.4.5 `vrna_exp_param_t* vrna_exp_params_copy ( vrna_exp_param_t * par )`

```
#include <ViennaRNA/params.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

13.6.4.6 `void vrna_params_subst ( vrna_fold_compound_t * vc, vrna_param_t * par )`

```
#include <ViennaRNA/params.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**

<code>vc</code>	The <code>vrna_fold_compound_t</code> that is about to receive updated energy parameters
<code>par</code>	The energy parameters used to substitute those within <code>vc</code> (Maybe NULL)

13.6.4.7 `void vrna_exp_params_subst ( vrna_fold_compound_t * vc, vrna_exp_param_t * params )`

```
#include <ViennaRNA/params.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 re-computation 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

13.6.4.8 `void vrna_exp_params_rescale ( vrna_fold_compound_t * vc, double * mfe )`

```
#include <ViennaRNA/params.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` datastructure 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` datastructure of `exp_params` in `vc`.

The computed scaling factor  $s$  will be stored as `pf_scale` attribute of the `exp_params` datastructure in `vc`.

#### See also

[vrna\\_exp\\_params\\_subst\(\)](#), [vrna\\_md\\_t](#), [vrna\\_exp\\_param\\_t](#), [vrna\\_fold\\_compound\\_t](#)

#### Parameters

<code>vc</code>	The fold compound data structure
<code>mfe</code>	A pointer to the MFE (in kcal/mol) or <code>NULL</code>

**13.6.4.9** `void vrna_params_reset ( vrna_fold_compound_t * vc, vrna_md_t * md_p )`

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

Reset free energy parameters within a [vrna\\_fold\\_compound\\_t](#) according to provided, or default model details.

This function allow 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</i> ↔ <i>_p</i>	A pointer to the new model details (or NULL for reset to defaults)

13.6.4.10 `void vrna_exp_params_reset ( vrna_fold_compound_t * vc, vrna_md_t * md_p )`

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

Reset Boltzmann factors for partition function computations within a `vrna_fold_compound_t` according to provided, or default model details.

This function allow to rescale Boltzmann factors for subsequent prartition 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

<i>vc</i>	The fold compound data structure
<i>md</i> ↔ <i>_p</i>	A pointer to the new model details (or NULL for reset to defaults)

13.6.4.11 `vrna_exp_param_t* get_scaled_pf_parameters ( void )`

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

get a datastructure 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 datastructure containing Boltzmann weights for use in partition function calculations

13.6.4.12 `vrna_exp_param_t* get_boltzmann_factors ( double temperature, double betaScale, vrna_md_t md, double pf_scale )`

```
#include <ViennaRNA/params.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

13.6.4.13 `vrna_exp_param_t* get_boltzmann_factor_copy ( vrna_exp_param_t * parameters )`

```
#include <ViennaRNA/params.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 dataset

13.6.4.14 `vrna_exp_param_t* get_scaled_alipf_parameters ( unsigned int n_seq )`

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

Get precomputed Boltzmann factors of the loop type dependent energy contributions (alifold variant)

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



13.6.4.15 `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.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!

13.6.4.16 `vrna_param_t* scale_parameters ( void )`

```
#include <ViennaRNA/params.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

13.6.4.17 `vrna_param_t* get_scaled_parameters ( double temperature, vrna_md_t md )`

```
#include <ViennaRNA/params.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

## 13.7 Manipulation of the Prediction 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 multibranch 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 multibranch 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 covariance contribution assessment.*

  - `#define VRNA_MODEL_DEFAULT_ALI_CV_FACT 1.`
- Default model behavior for weighting the covariance 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.*
- `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 multibranch loops using a logarithmic model.*
- `int vrna_md_defaults_logML_get` (void)  
*Get default behavior recomputing free energies of multibranch 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 multibranch loop prediction.*
- `int vrna_md_defaults_uniq_ML_get` (void)  
*Get default behavior for creating additional matrix for unique multibranch 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 covariance scaling factor used in comparative structure prediction.*

  - double `vrna_md_defaults_cv_fact_get` (void)

*Get the default covariance 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 `canonicalBPonly`
- int `uniq_ML`

*do ML decomposition uniquely (for subopt)*
- int `energy_set`

*0 = BP; 1=any mit GC; 2=any mit 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*

### 13.7.1 Detailed Description

### 13.7.2 Data Structure Documentation

#### 13.7.2.1 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`

#### Data Fields

- double `temperature`  
*The temperature used to scale the thermodynamic parameters.*
- double `betaScale`  
*A scaling factor for the thermodynamic temperature of the Boltzmann factors.*
- int `dangles`  
*Specifies the dangle model used in any energy evaluation (0,1,2 or 3)*
- int `special_hp`  
*Include special hairpin contributions for tri, tetra and hexaloops.*
- int `noLP`  
*Only consider canonical structures, i.e. no 'lonely' base pairs.*
- int `noGU`  
*Do not allow GU pairs.*
- int `noGUclosure`  
*Do not allow loops to be closed by GU pair.*
- int `logML`  
*Use logarithmic scaling for multi loops.*
- int `circ`  
*Assume RNA to be circular instead of linear.*
- int `gquad`  
*Include G-quadruplexes in structure prediction.*
- int `canonicalBPonly`  
*remove non-canonical bp's from constraint structures*

- int `uniq_ML`  
*Flag to ensure unique multibranch 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` [33]  
*contains allowed non standard bases*
- int `max_bp_span`  
*maximum allowed base pair span*
- int `min_loop_size`  
*Minimum size of hairpin loops.*
- int `window_size`  
*Size of the sliding window for locally optimal structure prediction.*
- int `oldAliEn`  
*Use old alifold energy model.*
- int `ribo`  
*Use ribosum scoring table in alifold energy model.*
- double `cv_fact`  
*Covariance scaling factor for consensus structure prediction.*
- 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.*

### 13.7.2.1.1 Field Documentation

#### 13.7.2.1.1.1 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 multi-loops. 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 documentaion of the particular recurrences or energy evaluation function for information about the provided dangle model.



#### 13.7.2.1.1.2 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.

### 13.7.3 Macro Definition Documentation

#### 13.7.3.1 #define VRNA\_MODEL\_DEFAULT\_TEMPERATURE 37.0

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

Default temperature for structure prediction and free energy evaluation in °C

#### See also

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

#### 13.7.3.2 #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\(\)](#)

#### 13.7.3.3 #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\(\)](#)

#### 13.7.3.4 #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\(\)](#)

#### 13.7.3.5 `#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\(\)](#)

#### 13.7.3.6 `#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\(\)](#)

#### 13.7.3.7 `#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\(\)](#)

#### 13.7.3.8 `#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\(\)](#)

#### 13.7.3.9 `#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\(\)](#)

**13.7.3.10 #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\(\)](#)

**13.7.3.11 #define VRNA\_MODEL\_DEFAULT\_UNIQ\_ML 0**

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

Default behavior of the model regarding unique multibranch loop decomposition.

See also

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

**13.7.3.12 #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\(\)](#)

**13.7.3.13 #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\(\)](#)

**13.7.3.14 #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\(\)](#)

**13.7.3.15 #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\(\)](#)

**13.7.3.16 #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\(\)](#)

**13.7.3.17 #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\(\)](#)

**13.7.3.18 #define VRNA\_MODEL\_DEFAULT\_LOG\_ML 0**

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

Default model behavior on how to evaluate the energy contribution of multibranch loops.

See also

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

**13.7.3.19 #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\(\)](#)

13.7.3.20 `#define VRNA_MODEL_DEFAULT_ALI_RIBO 0`

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

Default model behavior for consensus structure covariance contribution assessment.

See also

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

13.7.3.21 `#define VRNA_MODEL_DEFAULT_ALI_CV_FACT 1.`

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

Default model behavior for weighting the covariance score in consensus structure prediction.

See also

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

13.7.3.22 `#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\(\)](#)

## 13.7.4 Function Documentation

13.7.4.1 `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
-----------	---

13.7.4.2 `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_t.default()`

#### 13.7.4.3 `char* vrna_md_option_string ( vrna_md_t * md )`

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

Get a corresponding cmdline parameter string of the options in a `vrna_md_t`.

Note

This function is not threadsafe!

#### 13.7.4.4 `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

<code>md_p</code>	A set of model details to use as global default (if NULL is passed, factory defaults are restored)
-------------------	--

13.7.4.5 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
----------	---------------------------

## 13.7.4.6 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\\_TEMPERATURE](#)

Returns

The global default settings for temperature in centigrade

13.7.4.7 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
----------	------------------------------------

#### 13.7.4.8 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\\_BETA\\_SCALE](#)

Returns

The global default thermodynamic temperature scaling factor

#### 13.7.4.9 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
----------	------------------

#### 13.7.4.10 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



13.7.4.11 `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)
-------------	------------------------------------

13.7.4.12 `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\\_SPECIAL\\_HP](#)

## Returns

The global default settings for the treatment of special hairpin loops

13.7.4.13 `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)
-------------	------------------------------------

13.7.4.14 `int vrna_md_defaults_noLP_get ( 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

13.7.4.15 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)
-------------	------------------------------------

## 13.7.4.16 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

13.7.4.17 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)
-------------	------------------------------------

#### 13.7.4.18 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

#### 13.7.4.19 void vrna\_md\_defaults\_logML ( int *flag* )

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

Set default behavior recomputing free energies of multibranch 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)
-------------	------------------------------------

#### 13.7.4.20 int vrna\_md\_defaults\_logML\_get ( void )

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

Get default behavior recomputing free energies of multibranch 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\\_ML](#)

Returns

The global default settings for logarithmic model in multibranch loop free energy evaluation

13.7.4.21 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)
-------------	------------------------------------

## 13.7.4.22 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

## 13.7.4.23 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)
-------------	------------------------------------

## 13.7.4.24 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

13.7.4.25 void vrna\_md\_defaults\_uniq\_ML ( int *flag* )

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

Set default behavior for creating additional matrix for unique multibranch 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)
-------------	------------------------------------

## 13.7.4.26 int vrna\_md\_defaults\_uniq\_ML\_get ( void )

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

Get default behavior for creating additional matrix for unique multibranch 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 multibranch loop prediction

13.7.4.27 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)
----------	-------------------------

#### 13.7.4.28 `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\\_ENERGY\\_SET](#)

Returns

The global default settings for the energy set

#### 13.7.4.29 `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)
-------------	------------------------------------

#### 13.7.4.30 `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](#)

Returns

The global default settings for backtracking structures



**13.7.4.31 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')
----------	-----------------------------

**13.7.4.32 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

**13.7.4.33 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)
-------------	------------------------------------

#### 13.7.4.34 `int vrna_md_defaults_compute_bpp_get ( void )`

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

Get the default behavior for whether to compute base pair probabilities after partition function computation.

See also

[vrna\\_md\\_defaults\\_compute\\_bpp\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_COMPUTE\\_BPP](#)

Returns

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

#### 13.7.4.35 `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
-------------	------------------------

#### 13.7.4.36 `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\\_MAX\\_BP\\_SPAN](#)

Returns

The global default settings for maximum base pair span

13.7.4.37 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
-------------	--

13.7.4.38 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

13.7.4.39 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
-------------	--------------------------------

13.7.4.40 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\\_WINDOW\\_SIZE](#)

**Returns**

The global default settings for the size of the sliding window

**13.7.4.41 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)
-------------	------------------------------------

**13.7.4.42 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

**13.7.4.43 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)
-------------	------------------------------------

13.7.4.44 `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

13.7.4.45 `void vrna_md_defaults_cv_fact ( double factor )`

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

Set the default covariance 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 covariance factor
---------------	-----------------------

13.7.4.46 `double vrna_md_defaults_cv_fact_get ( void )`

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

Get the default covariance scaling factor used in comparative structure prediction.

## See also

[vrna\\_md\\_defaults\\_cv\\_fact\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_ALI\\_CV\\_FACT](#)

## Returns

The global default settings for the covariance factor

13.7.4.47 void vrna\_md\_defaults\_nc\_fact ( double *factor* )

#include <ViennaRNA/model.h>

See also

[vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_ALI\\_NC\\_FACT](#)

Parameters

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

13.7.4.48 double vrna\_md\_defaults\_nc\_fact\_get ( void )

#include <ViennaRNA/model.h>

See also

[vrna\\_md\\_defaults\\_nc\\_fact\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#), [vrna\\_md\\_set\\_default\(\)](#), [vrna\\_md\\_t](#), [VRNA\\_MODEL\\_DEFAULT\\_ALI\\_NC\\_FACT](#)

Returns

13.7.4.49 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)
---------------	------------------------------------

13.7.4.50 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

13.7.4.51 `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
-----------	---

## 13.7.5 Variable Documentation

13.7.5.1 `double temperature`

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

Rescale energy parameters to a temperature in degC.

Default is 37C. You have to call the `update_..._params()` functions after changing this parameter.

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

See also

[vrna\\_md\\_defaults\\_temperature\(\)](#), [vrna\\_md\\_defaults\\_temperature\\_get\(\)](#), [vrna\\_md\\_defaults\\_reset\(\)](#)

13.7.5.2 `double pf_scale`

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

A scaling factor used by [pf\\_fold\(\)](#) to avoid overflows.

Should be set to approximately  $\exp((-F/kT)/length)$ , where  $F$  is an estimate for the ensemble free energy, for example the minimum free energy. You must call [update\\_pf\\_params\(\)](#) after changing this parameter.

If `pf_scale` is -1 (the default), an estimate will be provided automatically when computing partition functions, e.g. [pf\\_fold\(\)](#). The automatic estimate is usually insufficient for sequences more than a few hundred bases long.

### 13.7.5.3 int dangles

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

Switch the energy model for dangling end contributions (0, 1, 2, 3)

If set to 0 no stabilizing energies are assigned to bases adjacent to helices in free ends and multiloops (so called dangling ends). Normally (dangles = 1) dangling end energies are assigned only to unpaired bases and a base cannot participate simultaneously in two dangling ends. In the partition function algorithm `pf_fold()` these checks are neglected. If `dangles` is set to 2, all folding routines will follow this convention. This treatment of dangling ends gives more favorable energies to helices directly adjacent to one another, which can be beneficial since such helices often do engage in stabilizing interactions through co-axial stacking.

If dangles = 3 co-axial stacking is explicitly included for adjacent helices in multi-loops. 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

### 13.7.5.4 int tetra\_loop

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

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

default is 1.

### 13.7.5.5 int noLonelyPairs

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

Global switch to avoid/allow helices of length 1.

Disallow all pairs which can only occur as lonely pairs (i.e. as helix of length 1). This avoids lonely base pairs in the predicted structures in most cases.

### 13.7.5.6 int canonicalBPonly

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

Do not use this variable, it will eventually be removed in one of the next versions

### 13.7.5.7 int energy\_set

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

0 = BP; 1=any mit GC; 2=any mit 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.



**13.7.5.8 int do\_backtrack**

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

do backtracking, i.e. compute secondary structures or base pair probabilities

If 0, do not calculate pair probabilities in [pf\\_fold\(\)](#); this is about twice as fast. Default is 1.

**13.7.5.9 char backtrack\_type**

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

A backtrack array marker for [inverse\\_fold\(\)](#)

If set to 'C': force (1,N) to be paired, 'M' fold as if the sequence were inside a multi-loop. Otherwise ('F') the usual mfe structure is computed.

**13.7.5.10 char\* nonstandards**

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

contains allowed non standard base pairs

Lists additional base pairs that will be allowed to form in addition to GC, CG, AU, UA, GU and UG. Nonstandard base pairs are given a stacking energy of 0.

**13.7.5.11 int max\_bp\_span**

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

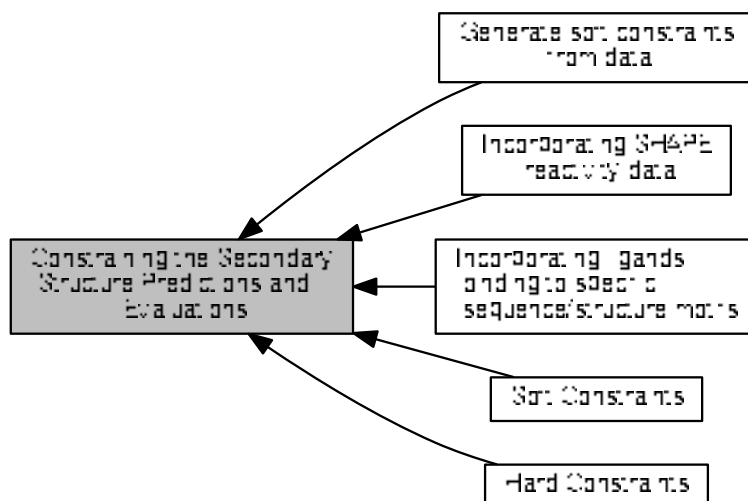
Maximum allowed base pair span.

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

## 13.8 Constraining the Secondary Structure Predictions and Evaluations

This module covers all functions and variables related to the problem of incorporating secondary structure constraints into the folding recursions.

Collaboration diagram for Constraining the Secondary Structure Predictions and Evaluations:



### 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.*

- [Incorporating SHAPE reactivity data](#)

*Incorporate SHAPE reactivity structure probing data into the folding recursions by means of soft constraints.*

- [Incorporating ligands binding to specific sequence/structure motifs](#)

*This module covers functions that enable the incorporation of ligand binding free energies to specific hairpin/interior loop motifs by means of generic 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.*

### Files

- file [constraints.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 1`  
*Flag passed to generic softt constraints callback to indicate hairpin loop decomposition step.*
- `#define VRNA_DECOMP_PAIR_IL 2`  
*Indicator for interior loop decomposition step.*
- `#define VRNA_DECOMP_PAIR_ML 3`  
*Indicator for multibranch loop decomposition step.*
- `#define VRNA_DECOMP_ML_ML_ML 5`  
*Indicator for decomposition of multibranch loop part.*
- `#define VRNA_DECOMP_ML_STEM 4`  
*Indicator for decomposition of multibranch loop part.*
- `#define VRNA_DECOMP_ML_ML 6`  
*Indicator for decomposition of multibranch loop part.*
- `#define VRNA_DECOMP_ML_UP 11`  
*Indicator for decomposition of multibranch loop part.*
- `#define VRNA_DECOMP_ML_ML_STEM 20`  
*Indicator for decomposition of multibranch loop part.*
- `#define VRNA_DECOMP_ML_COAXIAL 13`  
*Indicator for decomposition of multibranch loop part.*
- `#define VRNA_DECOMP_EXT_EXT 9`  
*Indicator for decomposition of exterior loop part.*
- `#define VRNA_DECOMP_EXT_UP 8`  
*Indicator for decomposition of exterior loop part.*
- `#define VRNA_DECOMP_EXT_STEM 14`  
*Indicator for decomposition of exterior loop part.*
- `#define VRNA_DECOMP_EXT_EXT_EXT 15`  
*Indicator for decomposition of exterior loop part.*
- `#define VRNA_DECOMP_EXT_STEM_EXT 16`  
*Indicator for decomposition of exterior loop part.*
- `#define VRNA_DECOMP_EXT_STEM_OUTSIDE 17`  
*Indicator for decomposition of exterior loop part.*
- `#define VRNA_DECOMP_EXT_EXT_STEM 18`  
*Indicator for decomposition of exterior loop part.*
- `#define VRNA_DECOMP_EXT_EXT_STEM1 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)*

### 13.8.1 Detailed Description

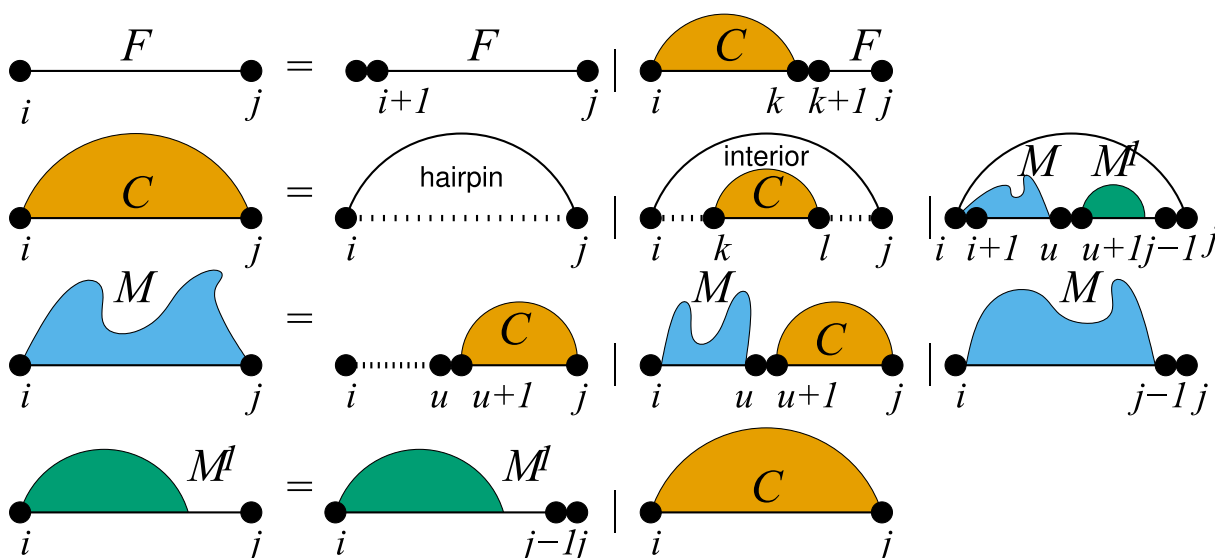
This module covers all functions and variables related to the problem of incorporating secondary structure constraints into the folding recursions.

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` -
- `#VRNA_CONSTRAINT_CONTEXT_NO_REMOVE` -
- `VRNA_CONSTRAINT_CONTEXT_ALL_LOOPS` - Hard constraints flag, shortcut for all base pairs.

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 avaluation 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

- [Incorporating SHAPE reactivity data](#) and
- [Incorporating ligands binding to specific sequence/structure motifs.](#)

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](#).

## 13.8.2 Macro Definition Documentation

### 13.8.2.1 `#define VRNA_CONSTRAINT_FILE 0`

```
#include <ViennaRNA/constraints.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!

### 13.8.2.2 `#define VRNA_CONSTRAINT_SOFT_MFE 0`

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

Indicate generation of constraints for MFE folding.

**Deprecated** This flag has no meaning anymore, since constraints are now always stored!

### 13.8.2.3 `#define VRNA_CONSTRAINT_SOFT_PF VRNA_OPTION_PF`

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

Indicate generation of constraints for partition function computation.

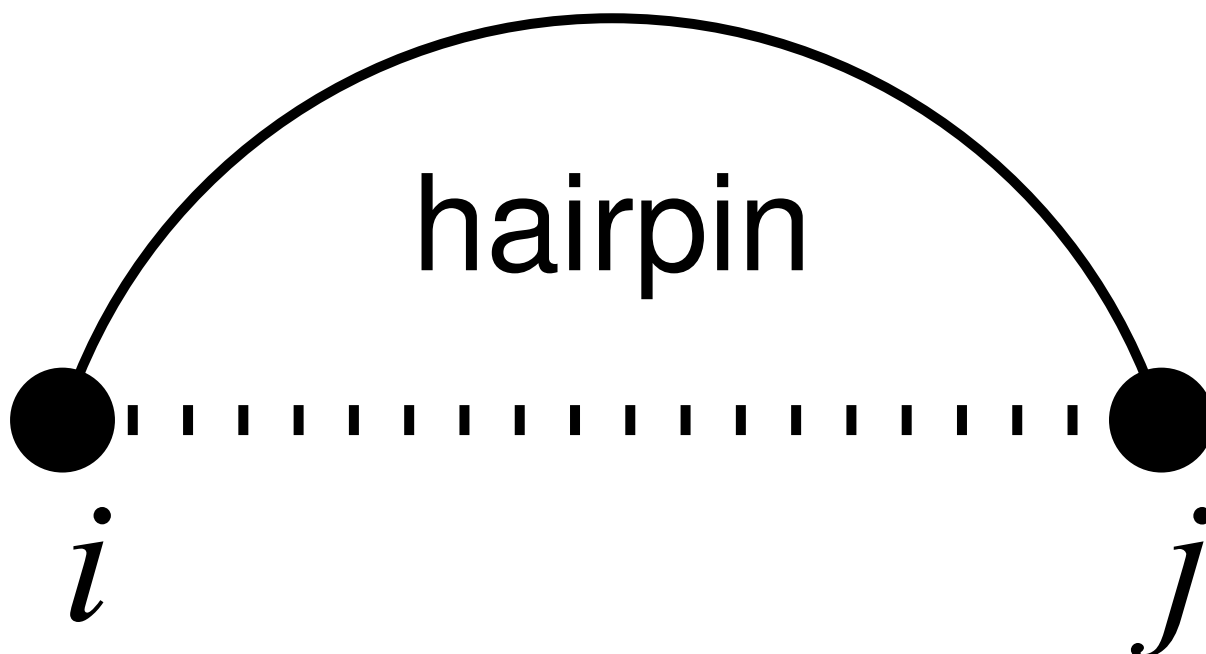
**Deprecated** Use `VRNA_OPTION_PF` instead!

### 13.8.2.4 `#define VRNA_DECOMP_PAIR_HP 1`

```
#include <ViennaRNA/constraints.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  $(i, j)$ .

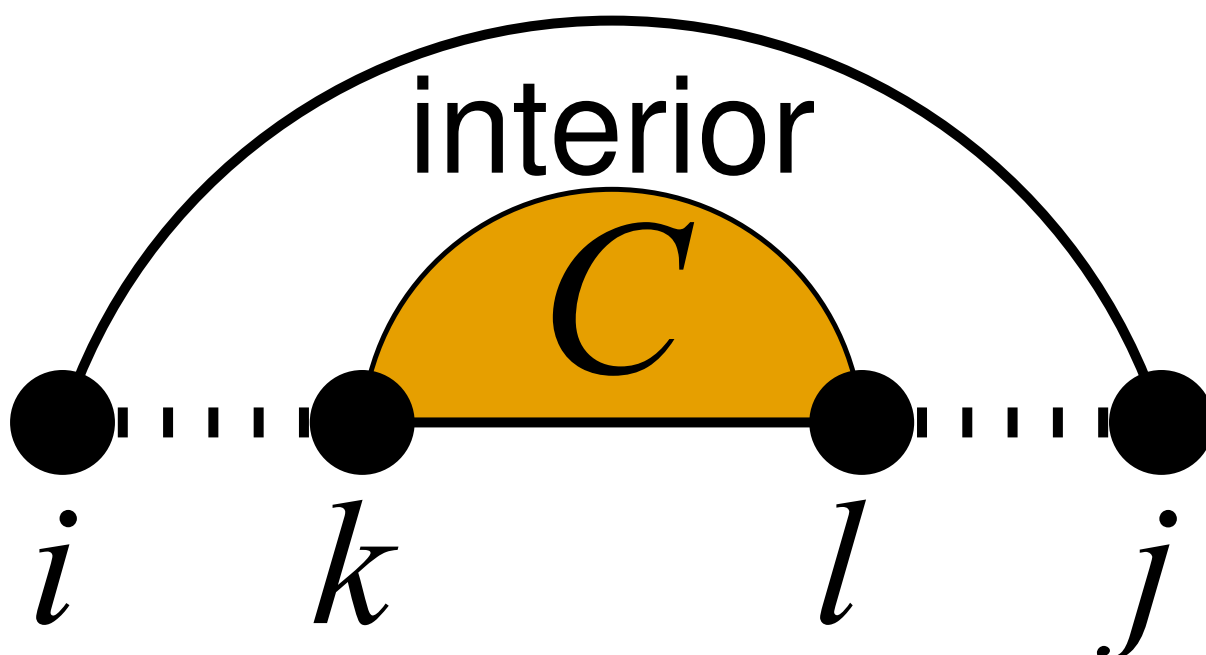


13.8.2.5 `#define VRNA_DECOMP_PAIR_IL 2`

`#include <ViennaRNA/constraints.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  $(k, l)$ .

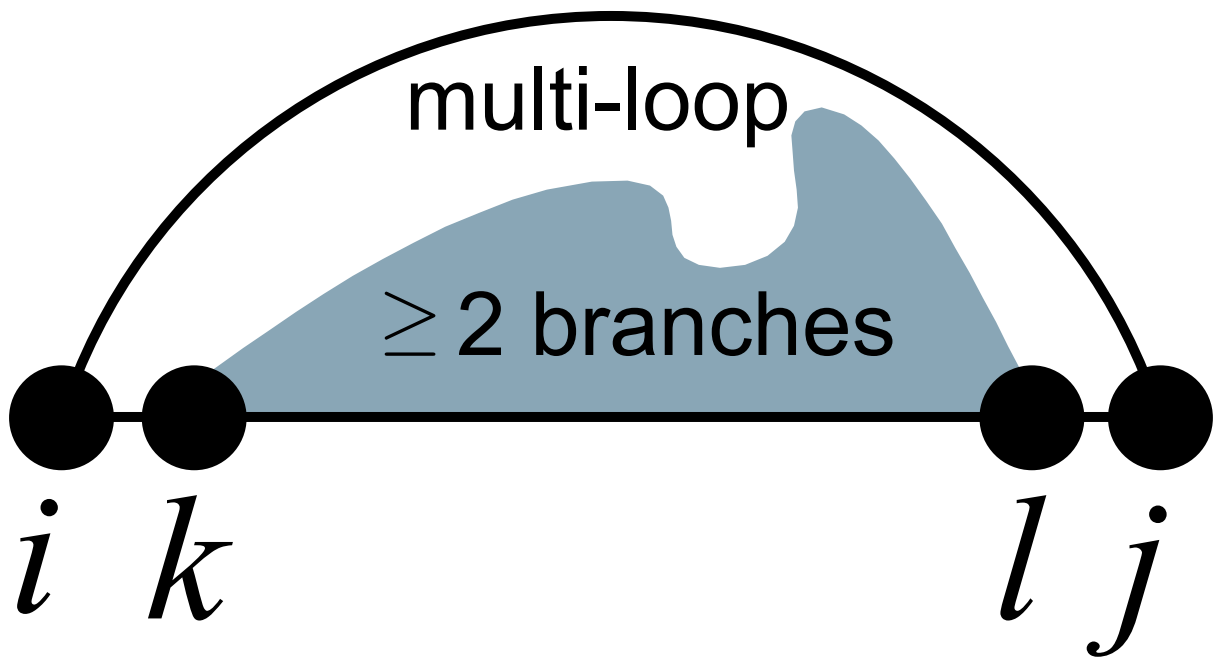


13.8.2.6 `#define VRNA_DECOMP_PAIR_ML 3`

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

Indicator for multibranch loop decomposition step.

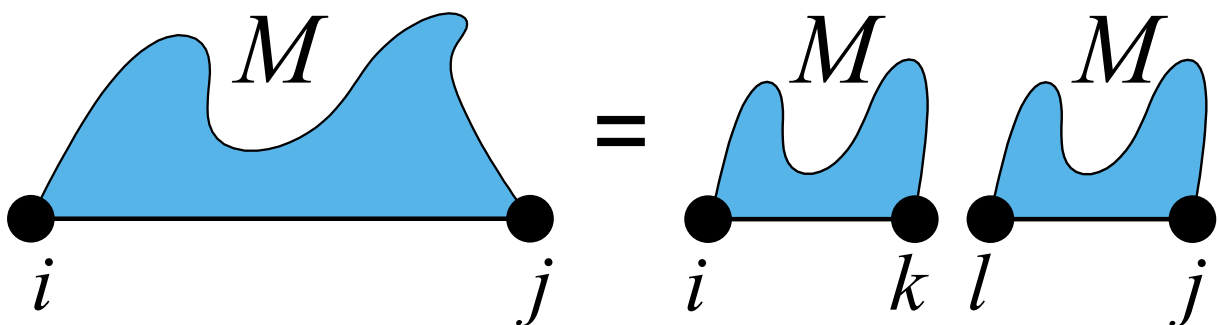
This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multi-branch loop enclosed by the base pair  $(i, j)$ , and consisting of some enclosed multi loop content from  $k$  to  $l$ .

13.8.2.7 `#define VRNA_DECOMP_ML_ML_ML 5`

```
#include <ViennaRNA/constraints.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  $[l : j]$ .



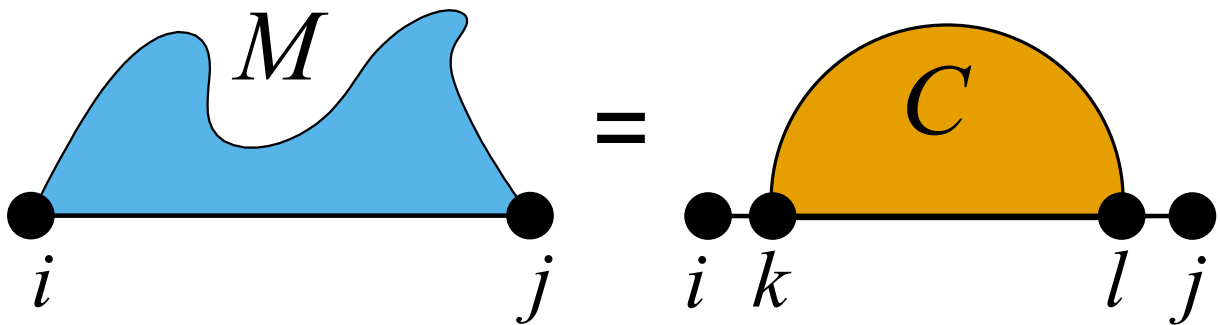


## 13.8.2.8 #define VRNA\_DECOMP\_ML\_STEM 4

```
#include <ViennaRNA/constraints.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  $(k, l)$ .

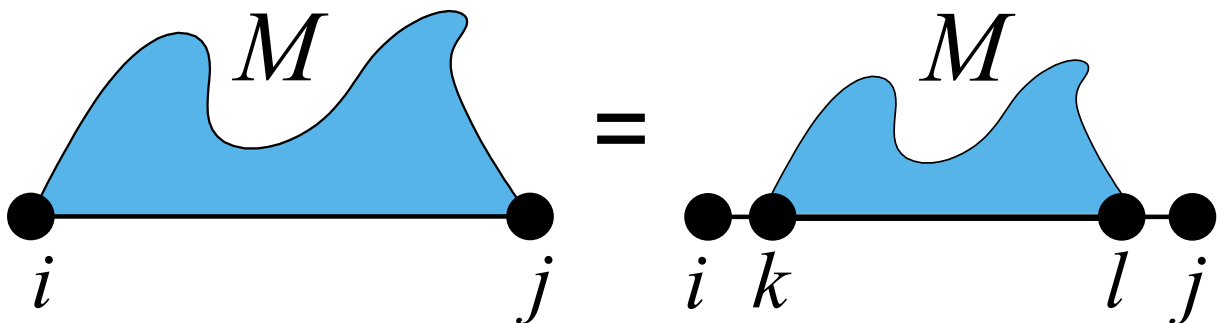


## 13.8.2.9 #define VRNA\_DECOMP\_ML\_ML 6

```
#include <ViennaRNA/constraints.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  $[k : l]$ .

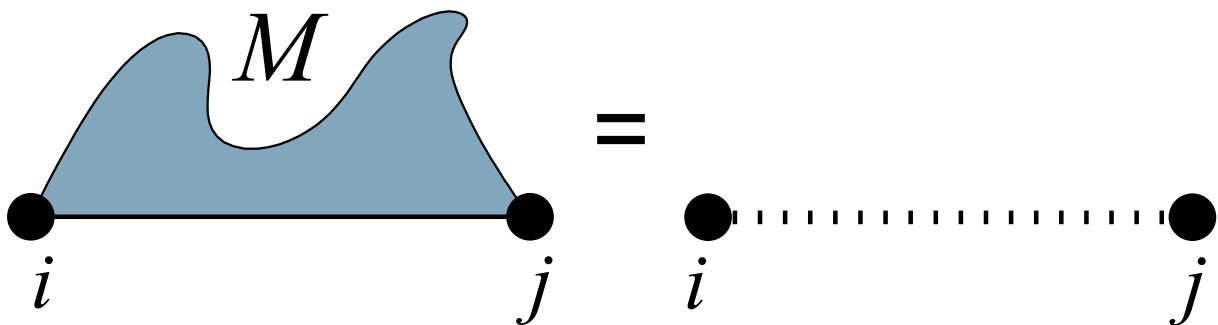


13.8.2.10 `#define VRNA_DECOMP_ML_UP 11`

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

Indicator for decomposition of multibranch loop part.

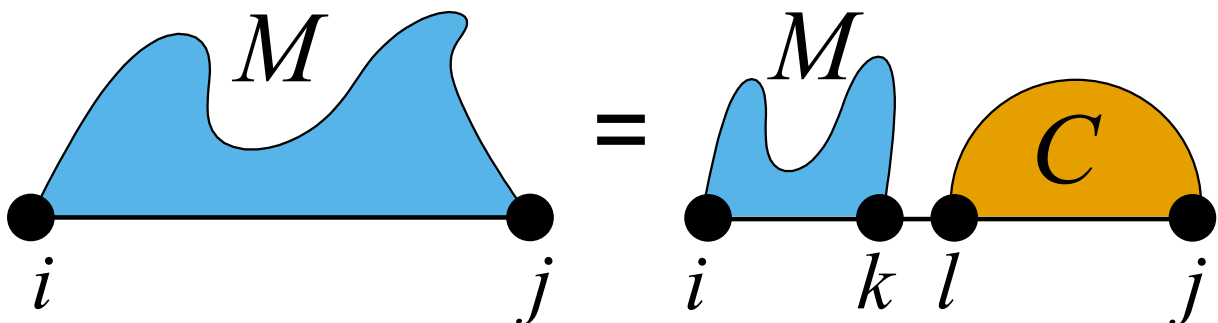
This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multibranch loop part in the interval  $[i : j]$ , which will be considered a multibranch loop part that only consists of unpaired nucleotides.

13.8.2.11 `#define VRNA_DECOMP_ML_ML_STEM 20`

```
#include <ViennaRNA/constraints.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  $(l, j)$ .

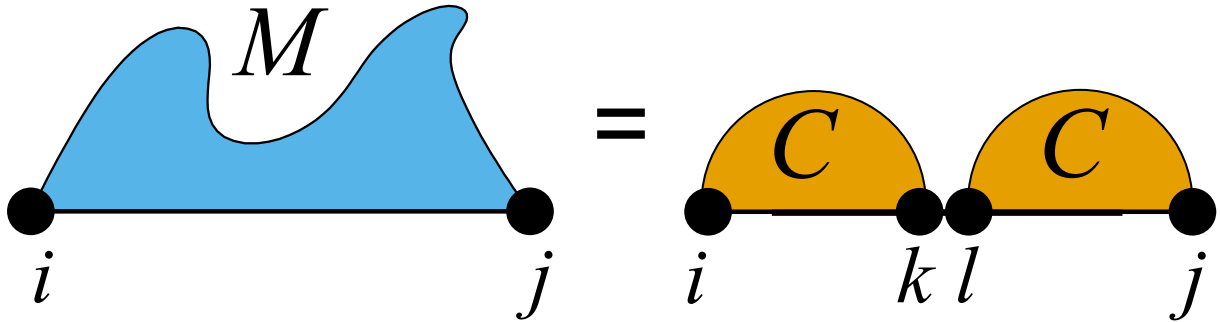


13.8.2.12 `#define VRNA_DECOMP_ML_COAXIAL 13`

`#include <ViennaRNA/constraints.h>`

Indicator for decomposition of multibranch loop part.

This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates a multibranch loop part in the interval  $[i : j]$ , where two stems with enclosing pairs  $(i, k)$  and  $(l, j)$  are coaxially stacking onto each other.

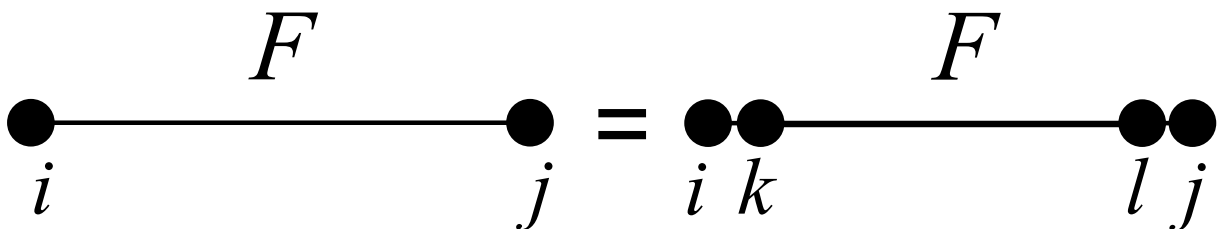


13.8.2.13 `#define VRNA_DECOMP_EXT_EXT 9`

`#include <ViennaRNA/constraints.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  $[k : l]$ .

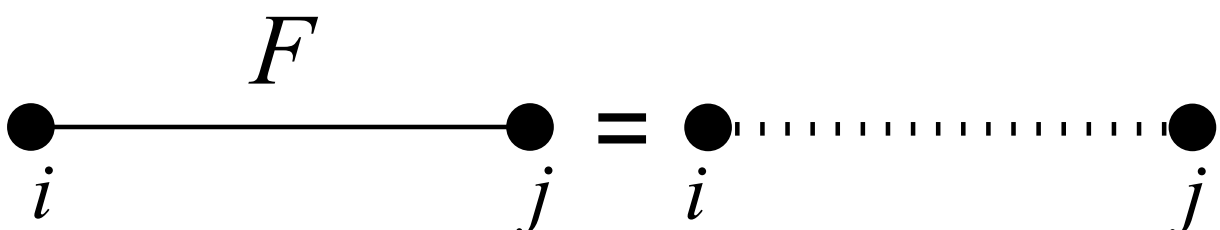


13.8.2.14 `#define VRNA_DECOMP_EXT_UP 8`

`#include <ViennaRNA/constraints.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.

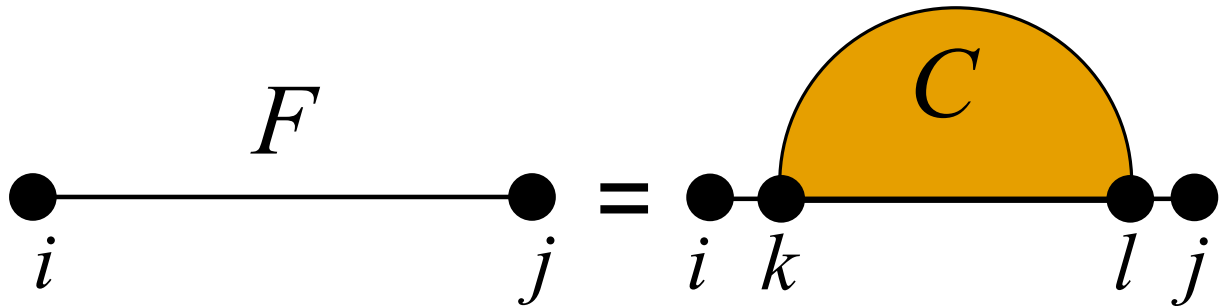


13.8.2.15 `#define VRNA_DECOMP_EXT_STEM 14`

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

Indicator for decomposition of exterior loop part.

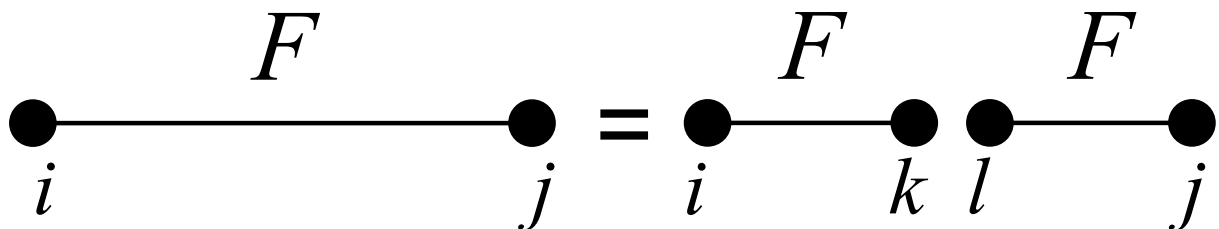
This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval  $[i : j]$ , which will be considered a stem with enclosing pair  $(k, l)$ .

13.8.2.16 `#define VRNA_DECOMP_EXT_EXT_EXT 15`

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

Indicator for decomposition of exterior loop part.

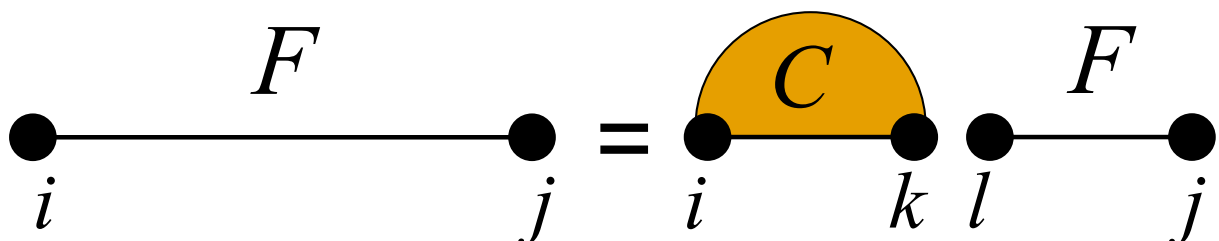
This flag notifies the soft or hard constraint callback function that the current decomposition step evaluates an exterior loop part in the interval  $[i : j]$ , which will be decomposed into two exterior loop parts  $[i : k]$  and  $[l : j]$ .

13.8.2.17 `#define VRNA_DECOMP_EXT_STEM_EXT 16`

```
#include <ViennaRNA/constraints.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  $[l : j]$ .

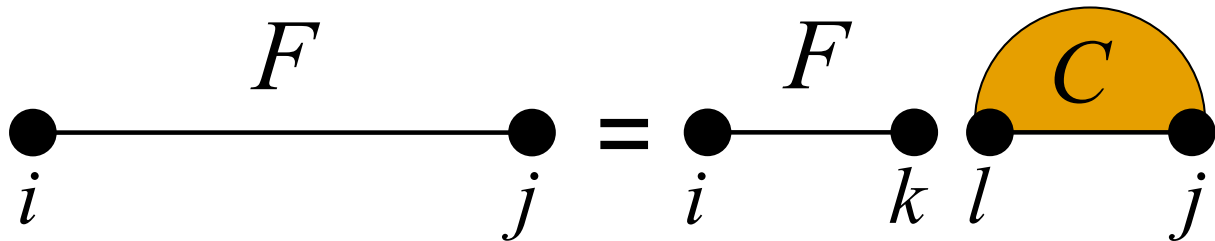


13.8.2.18 `#define VRNA_DECOMP_EXT_EXT_STEM 18`

```
#include <ViennaRNA/constraints.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  $(l, j)$ .

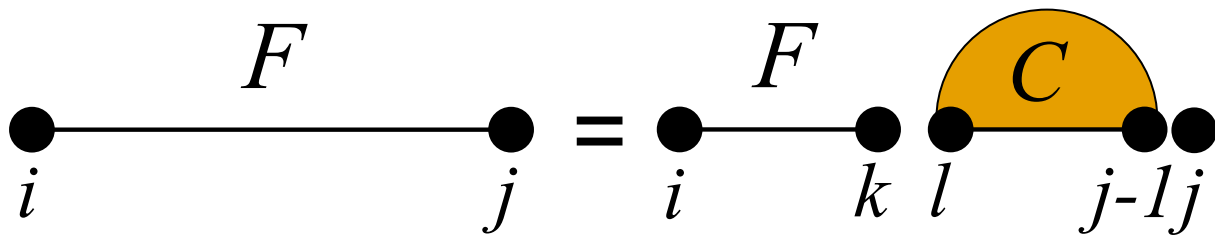


13.8.2.19 `#define VRNA_DECOMP_EXT_EXT_STEM1 19`

```
#include <ViennaRNA/constraints.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  $(l, j - 1)$ .



### 13.8.3 Function Documentation

13.8.3.1 `void vrna_constraints_add ( vrna_fold_compound_t * vc, const char * constraint, unsigned int options )`

```
#include <ViennaRNA/constraints.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\\_add\\_up\(\)](#), [vrna\\_sc\\_add\\_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;
vrna_constraints_add(vc, (const char *)structure, constraint_options);
```

In contrast to the above, constraints may also be read from file:

```
vrna_constraints_add(vc, constraints_file,
VRNA_OPTION_MFE | ((pf) ? VRNA_OPTION_PF : 0));
```

## 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

13.8.3.2 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
---------------	--

13.8.3.3 void vrna\_message\_constraint\_options\_all ( void )

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

Print structure constraint characters to stdout (full constraint support)

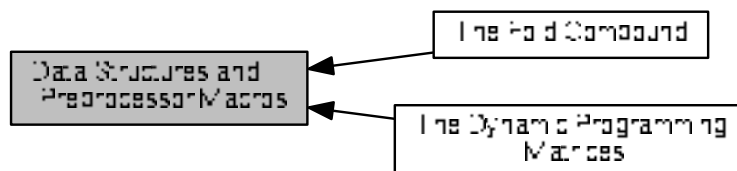
See also

[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](#)

## 13.9 Data Structures and Preprocessor Macros

All datastructures and typedefs shared among the Vienna RNA Package can be found here.

Collaboration diagram for Data Structures and Preprocessor Macros:



### 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.*

### Data Structures

- struct [vrna\\_basepair\\_s](#)  
*Base pair data structure used in subopt.c. [More...](#)*
- struct [vrna\\_plist\\_s](#)  
*this datastructure is used as input parameter in functions of [PS\\_dot.h](#) and others [More...](#)*
- struct [vrna\\_cpair\\_s](#)  
*this datastructure is used as input parameter in functions of [PS\\_dot.c](#) [More...](#)*
- 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](#)
- struct [pu\\_out](#)  
*Collection of all free\_energy of beeing unpaired values for output. [More...](#)*
- struct [constrain](#)  
*constraints for cofolding [More...](#)*
- struct [duplexT](#)
- struct [node](#)
- struct [snoopT](#)
- struct [dupVar](#)



## Typedefs

- typedef struct [vrna\\_basepair\\_s](#) [vrna\\_basepair\\_t](#)  
*Typename for the base pair representing data structure [vrna\\_basepair\\_s](#).*
- typedef struct [vrna\\_plist\\_s](#) [vrna\\_plist\\_t](#)  
*Typename for the base pair list representing data structure [vrna\\_plist\\_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\\_plist\\_s](#) plist  
*Old typename of [vrna\\_plist\\_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 [pu\\_out](#) [pu\\_out](#)  
*Collection of all free\_energy of beeing unpaired values for output.*
- typedef struct [constrain](#) [constrain](#)  
*constraints for cofolding*

### 13.9.1 Detailed Description

All datastructures and typedefs shared among the Vienna RNA Package can be found here.

### 13.9.2 Data Structure Documentation

#### 13.9.2.1 struct [vrna\\_basepair\\_s](#)

Base pair data structure used in subopt.c.

#### 13.9.2.2 struct [vrna\\_plist\\_s](#)

this datastructure is used as input parameter in functions of [PS\\_dot.h](#) and others

### 13.9.2.3 struct vrna\_cpair\_s

this datastructure is used as input parameter in functions of PS\_dot.c

### 13.9.2.4 struct vrna\_sect\_s

Stack of partial structures for backtracking.

### 13.9.2.5 struct vrna\_bp\_stack\_s

Base pair stack element.

### 13.9.2.6 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*

### 13.9.2.7 struct interact

#### 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*

## 13.9.2.8 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*

## 13.9.2.9 struct constrain

constraints for cofolding

## 13.9.2.10 struct duplexT

## 13.9.2.11 struct node

Collaboration diagram for node:



## 13.9.2.12 struct snoopT

## 13.9.2.13 struct dupVar

## 13.9.3 Typedef Documentation

## 13.9.3.1 typedef struct vrna\_basepair\_s PAIR

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

Old typename of `vrna_basepair_s`.

**Deprecated** Use `vrna_basepair_t` instead!

### 13.9.3.2 typedef struct vrna\_plist\_s plist

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

Old typename of [vrna\\_plist\\_s](#).

**Deprecated** Use [vrna\\_plist\\_t](#) instead!

### 13.9.3.3 typedef struct vrna\_cpair\_s cpair

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

Old typename of [vrna\\_cpair\\_s](#).

**Deprecated** Use [vrna\\_cpair\\_t](#) instead!

### 13.9.3.4 typedef struct vrna\_sect\_s sect

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

Old typename of [vrna\\_sect\\_s](#).

**Deprecated** Use [vrna\\_sect\\_t](#) instead!

### 13.9.3.5 typedef struct vrna\_bp\_stack\_s bondT

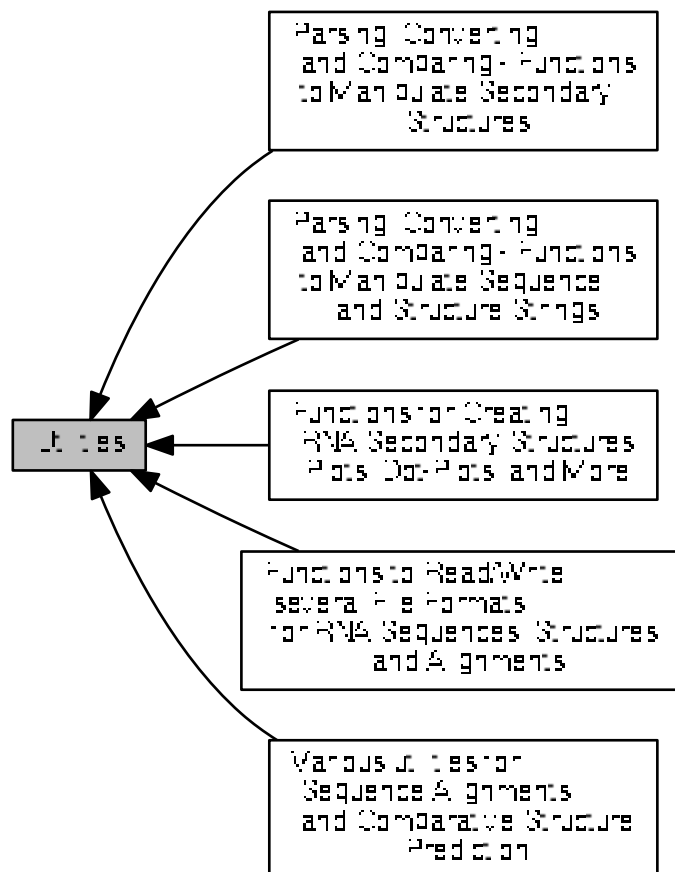
```
#include <ViennaRNA/data_structures.h>
```

Old typename of [vrna\\_bp\\_stack\\_s](#).

**Deprecated** Use [vrna\\_bp\\_stack\\_t](#) instead!

## 13.10 Utilities

Collaboration diagram for Utilities:



### Modules

- [Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings](#)
- [Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures](#)
- [Various utilities for Sequence Alignments, and Comparative Structure Prediction](#)
- [Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments](#)
- [Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More](#)

### Files

- file [alphabet.h](#)  
*Functions to process, convert, and generally handle different nucleotide and/or base pair alphabets.*
- file [utils.h](#)  
*General utility- and helper-functions used throughout the ViennaRNA Package.*

## 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_message_error` (const char message[])  
Die with an error message.
- `void vrna_message_warning` (const char message[])  
Print a warning message.
- `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.*
- void `vrna_file_copy` (FILE \*from, FILE \*to)  
*Inefficient 'cp'.*
- char \* `vrna_time_stamp` (void)  
*Get a timestamp.*
- char \* `get_line` (FILE \*fp)  
*Read a line of arbitrary length from a stream.*
- unsigned int `get_input_line` (char \*\*string, unsigned int options)
- 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.*
- 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 (idx) for accessing the energy matrices, e.g. in MFE related functions.*

## Variables

- unsigned short `xsubi` [3]  
*Current 48 bit random number.*

### 13.10.1 Detailed Description

### 13.10.2 Macro Definition Documentation

#### 13.10.2.1 `#define VRNA_INPUT_FASTA_HEADER 8U`

```
#include <ViennaRNA/utils.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

#### 13.10.2.2 `#define VRNA_INPUT_CONSTRAINT 32U`

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

Input flag for `get_input_line()`:

Tell `get_input_line()` that we assume to read a structure constraint.

### 13.10.3 Function Documentation

#### 13.10.3.1 `void* vrna_alloc ( unsigned size )`

```
#include <ViennaRNA/utils.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

**13.10.3.2** `void* vrna_realloc ( void * p, unsigned size )`

```
#include <ViennaRNA/utils.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

**13.10.3.3** `void vrna_message_error ( const char message[] )`

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

Die with an error message.

**See also**

[vrna\\_message\\_warning\(\)](#)

**Parameters**

<i>message</i>	The error message to be printed before exiting with 'FAILURE'
----------------	---

**13.10.3.4** `void vrna_message_warning ( const char message[] )`

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

Print a warning message.

Print a warning message to *stderr*



## Parameters

<i>message</i>	The warning message
----------------	---------------------

13.10.3.5 `double vrna_urn ( void )`

```
#include <ViennaRNA/utils.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]

13.10.3.6 `int vrna_int_urn ( int from, int to )`

```
#include <ViennaRNA/utils.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]

13.10.3.7 `char* vrna_time_stamp ( void )`

```
#include <ViennaRNA/utils.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

#### 13.10.3.8 char\* get\_line ( FILE \* fp )

```
#include <ViennaRNA/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

#### 13.10.3.9 unsigned int get\_input\_line ( char \*\* string, unsigned int options )

```
#include <ViennaRNA/utils.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 to switch between different reading modes.

Currently available options are:

#VRNA\_INPUT\_NOPRINT\_COMMENTS, [VRNA\\_INPUT\\_NOSKIP\\_COMMENTS](#), #VRNA\_INPUT\_NOELIM\_WS\_SUFFIX

pass a collection of options as one value like this:

```
get_input_line(string, option_1 | option_2 | option_n)
```

If the function recognizes the type of input, it will report it in the return value. It also reports if a user defined 'quit' command (-sign on 'stdin') was given. Possible return values are:

[VRNA\\_INPUT\\_FASTA\\_HEADER](#), [VRNA\\_INPUT\\_ERROR](#), [VRNA\\_INPUT\\_MISC](#), [VRNA\\_INPUT\\_QUIT](#)

#### Parameters

<i>string</i>	A pointer to the character array that contains the line read
<i>options</i>	A collection of options for switching the functions behavior

**Returns**

A flag with information about what has been read

**13.10.3.10 void vrna\_message\_input\_seq\_simple ( void )**

```
#include <ViennaRNA/utils.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

**13.10.3.11 void vrna\_message\_input\_seq ( const char \* s )**

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

Print a line with a user defined string and a ruler to stdout.

(usually this is used to ask for user input) There will also be a ruler (scale line) printed that helps orientation of the sequence positions

**Parameters**

s	A user defined string that will be printed to stdout
---	--

**13.10.3.12 int\* vrna\_idx\_row\_wise ( unsigned int length )**

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

Get an index mapper array (iidx) 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) ~ iidx[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**

length	The length of the RNA sequence
--------	--------------------------------

**Returns**

The mapper array

**13.10.3.13** `int* vrna_idx_col_wise ( unsigned int length )`

```
#include <ViennaRNA/utils.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

**13.10.4 Variable Documentation****13.10.4.1** `unsigned short xsubi[3]`

```
#include <ViennaRNA/utils.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\(\)](#).

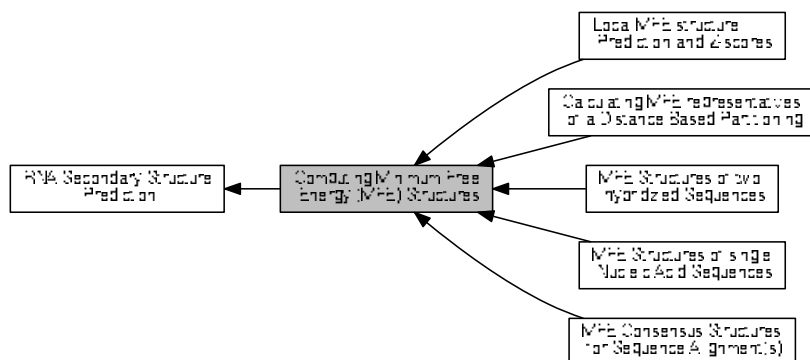
**See also**

[vrna\\_urn\(\)](#)

## 13.11 Computing Minimum Free Energy (MFE) Structures

This section covers all functions and variables related to the calculation of minimum free energy (MFE) structures.

Collaboration diagram for Computing Minimum Free Energy (MFE) Structures:



### Modules

- [MFE Structures of single Nucleic Acid Sequences](#)

*This module contains all functions and variables related to the calculation of global minimum free energy structures for single sequences.*

- [MFE Structures of two hybridized Sequences](#)
- [MFE Consensus Structures for Sequence Alignment\(s\)](#)
- [Local MFE structure Prediction and Z-scores](#)
- [Calculating 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.*

### Functions

- 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.*

#### 13.11.1 Detailed Description

This section covers all functions and variables related to the calculation of minimum free energy (MFE) structures.

The library provides a fast dynamic programming minimum free energy folding algorithm as described in [20]. All relevant parts that directly implement the "Zuker & Stiegler" algorithm for single sequences are described in this section.

Folding of circular RNA sequences is handled as a post-processing step of the forward recursions. See [8] for further details.

Nevertheless, the RNaLib also provides interfaces for the prediction of consensus MFE structures of sequence alignments, MFE structure for two hybridized sequences, local optimal structures and many more. For those more specialized variants of MFE folding routines, please consult the appropriate subsections (Modules) as listed above.

### 13.11.2 Function Documentation

#### 13.11.2.1 `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_VC_TYPE_SINGLE`, and `VRNA_VC_TYPE_ALIGNMENT`.

#### See also

`vrna_fold_compound_t`, `vrna_fold_compound()`, `vrna_fold()`, `vrna_circfold()`, `vrna_fold_compound_comparative()`, `vrna_alifold()`, `vrna_circalifold()`

#### Parameters

<i>vc</i>	fold compound
<i>structure</i>	A pointer to the character array where the secondary structure in dot-bracket notation will be written to (Maybe <code>NULL</code> )

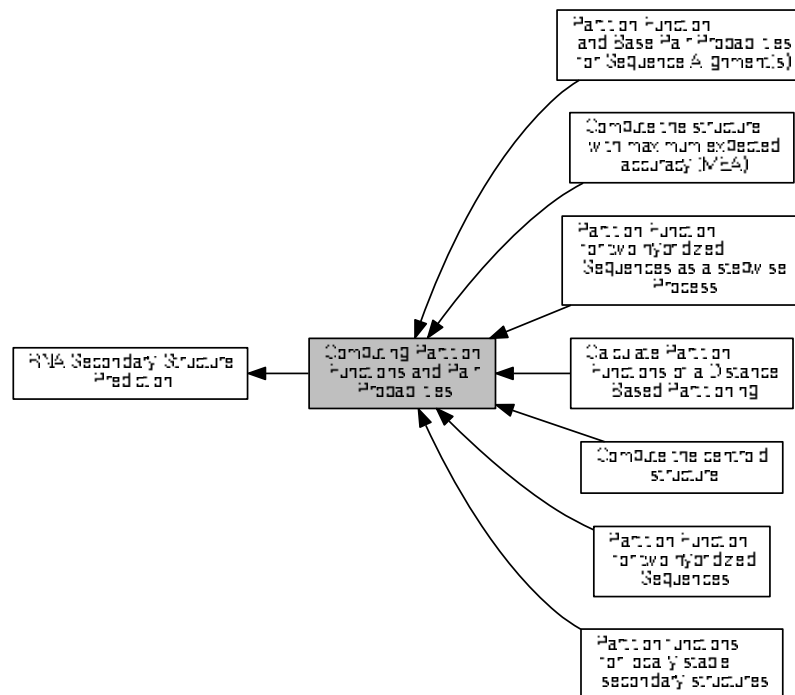
#### Returns

the minimum free energy (MFE) in kcal/mol

## 13.12 Computing Partition Functions and Pair Probabilities

This section provides information about all functions and variables related to the calculation of the partition function and base pair probabilities.

Collaboration diagram for Computing Partition Functions and Pair Probabilities:



### Modules

- [Compute the structure with maximum expected accuracy \(MEA\)](#)
- [Compute the centroid structure](#)
- [Partition Function for two hybridized Sequences](#)  
*Partition Function Cofolding.*
- [Partition Function for two hybridized Sequences as a stepwise Process](#)  
*Partition Function Cofolding as a stepwise process.*
- [Partition Function and Base Pair Probabilities for Sequence Alignment\(s\)](#)
- [Partition functions for locally stable secondary structures](#)
- [Calculate 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.*

### Files

- file [boltzmann\\_sampling.h](#)  
*Boltzmann Sampling of secondary structures from the ensemble.*
- file [part\\_func.h](#)  
*Partition function of single RNA sequences.*

## Functions

- float [vrna\\_pf](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, char \*structure)  
*Compute the partition function  $Q$  for a given RNA sequence, or sequence alignment.*
- float [vrna\\_pf\\_fold](#) (const char \*seq, char \*structure, [vrna\\_plist\\_t](#) \*\*pl)  
*Compute Partition function  $Q$  (and base pair probabilities) for an RNA sequence using a comparative method.*
- float [vrna\\_pf\\_circfold](#) (const char \*seq, char \*structure, [vrna\\_plist\\_t](#) \*\*pl)  
*Compute Partition function  $Q$  (and base pair probabilities) for a circular RNA sequences using a comparative method.*
- 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.*
- [vrna\\_plist\\_t](#) \* [vrna\\_stack\\_prob](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, double cutoff)  
*Compute stacking probabilities.*
- float [pf\\_fold\\_par](#) (const char \*sequence, char \*structure, [vrna\\_exp\\_param\\_t](#) \*parameters, int calculate\_↔  
bppm, int is\_constrained, int is\_circular)  
*Compute the partition function  $Q$  for a given RNA sequence.*
- float [pf\\_fold](#) (const char \*sequence, char \*structure)  
*Compute the partition function  $Q$  of an RNA sequence.*
- float [pf\\_circ\\_fold](#) (const char \*sequence, char \*structure)  
*Compute the partition function of a circular RNA sequence.*
- void [free\\_pf\\_arrays](#) (void)  
*Free arrays for the partition function recursions.*
- void [update\\_pf\\_params](#) (int length)  
*Recalculate energy parameters.*
- void [update\\_pf\\_params\\_par](#) (int length, [vrna\\_exp\\_param\\_t](#) \*parameters)  
*Recalculate energy parameters.*
- [FLT\\_OR\\_DBL](#) \* [export\\_bppm](#) (void)  
*Get a pointer to the base pair probability array  
Accessing the base pair probabilities for a pair (i,j) is achieved by.*
- 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 [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\\_plist\\_t](#) \* [vrna\\_plist\\_from\\_probs](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, double cut\_off)  
*Create a [vrna\\_plist\\_t](#) from base pair probability matrix.*
- void [assign\\_plist\\_from\\_pr](#) ([vrna\\_plist\\_t](#) \*\*pl, [FLT\\_OR\\_DBL](#) \*probs, int length, double cutoff)  
*Create a [vrna\\_plist\\_t](#) from a probability matrix.*

### 13.12.1 Detailed Description

This section provides information about all functions and variables related to the calculation of the partition function and base pair probabilities.

Instead of the minimum free energy structure the partition function of all possible structures and from that the pairing probability for every possible pair can be calculated, using a dynamic programming algorithm as described in [12].



### 13.12.2 Function Documentation

#### 13.12.2.1 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 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.

#### Note

This function is polymorphic. It accepts `vrna_fold_compound_t` of type `VRNA_VC_TYPE_SINGLE`, and `VRNA_VC_TYPE_ALIGNMENT`.

#### See also

`vrna_fold_compound_t`, `vrna_fold_compound()`, `vrna_pf_fold()`, `vrna_pf_circfold()`, `vrna_fold_compound_comparative()`, `vrna_pf_alifold()`, `vrna_pf_circalifold()`, `vrna_db_from_probs()`, `vrna_exp_params()`, `vrna_aln_pinfo()`

#### Parameters

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

#### Returns

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

#### 13.12.2.2 float vrna\_pf\_fold ( const char \* seq, char \* structure, vrna\_plist\_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>sequences</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_plist_t</a> to store pairing probabilities (Maybe NULL)

## Returns

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

13.12.2.3 `float vrna_pf_circfold ( const char * seq, char * structure, vrna_plist_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 [8] for further details.

## See also

[vrna\\_pf\\_fold\(\)](#), [vrna\\_pf\(\)](#), [vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_t](#)

## Parameters

<i>sequences</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_plist_t</a> to store pairing probabilities (Maybe NULL)

## Returns

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

13.12.2.4 `double vrna_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 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

#### 13.12.2.5 `double vrna_mean_bp_distance ( vrna_fold_compound_t * vc )`

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

Get the mean base pair distance in the thermodynamic ensemble.

$$\langle d \rangle = \sum_{a,b} p_a p_b d(S_a, S_b)$$

this can be computed from the pair probs  $p_{ij}$  as

$$\langle d \rangle = \sum_{ij} p_{ij} (1 - p_{ij})$$

#### Parameters

<i>vc</i>	The fold compound data structure
-----------	----------------------------------

#### Returns

The mean pair distance of the structure ensemble

#### 13.12.2.6 `vrna_plist_t* vrna_stack_prob ( vrna_fold_compound_t * vc, double cutoff )`

```
#include <ViennaRNA/part_func.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  $p$

13.12.2.7 `float pf_fold_par ( const char * sequence, char * structure, vrna_exp_param_t * parameters, int calculate_bppm, int is_constrained, int is_circular )`

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

Compute the partition function  $Q$  for a given RNA sequence.

If *structure* is not a NULL pointer on input, it contains on return a string consisting of the letters ". , | { } ( ) " denoting bases that are essentially unpaired, weakly paired, strongly paired without preference, weakly upstream (downstream) paired, or strongly up- (down-)stream paired bases, respectively. If *fold\_constrained* is not 0, the *structure* string is interpreted on input as a list of constraints for the folding. The character "x" marks bases that must be unpaired, matching brackets " ( ) " denote base pairs, all other characters are ignored. Any pairs conflicting with the constraint will be forbidden. This is usually sufficient to ensure the constraints are honored. If the parameter *calculate\_bppm* is set to 0 base pairing probabilities will not be computed (saving CPU time), otherwise after calculations took place *pr* will contain the probability that bases  $i$  and  $j$  pair.

**Deprecated** Use *vrna\_pf()* instead

**Note**

The global array *pr* is deprecated and the user who wants the calculated base pair probabilities for further computations is advised to use the function *export\_bppm()*

**Postcondition**

After successful run the hidden folding matrices are filled with the appropriate Boltzmann factors. Depending on whether the global variable *do\_backtrack* was set the base pair probabilities are already computed and may be accessed for further usage via the *export\_bppm()* function. A call of *free\_pf\_arrays()* will free all memory allocated by this function. Successive calls will first free previously allocated memory before starting the computation.

**See also**

*vrna\_pf()*, *bppm\_to\_structure()*, *export\_bppm()*, *vrna\_exp\_params()*, *free\_pf\_arrays()*

**Parameters**

in	<i>sequence</i>	The RNA sequence input
in, out	<i>structure</i>	A pointer to a char array where a base pair probability information can be stored in a pseudo-dot-bracket notation (may be NULL, too)
in	<i>parameters</i>	Data structure containing the precalculated Boltzmann factors
in	<i>calculate_bppm</i>	Switch to Base pair probability calculations on/off (0==off)
in	<i>is_constrained</i>	Switch to indicate that a structure constraint is passed via the structure argument (0==off)
in	<i>is_circular</i>	Switch to (de-)activate postprocessing steps in case RNA sequence is circular (0==off)

**Returns**

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

**13.12.2.8 float pf\_fold ( const char \* *sequence*, char \* *structure* )**

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

Compute the partition function  $Q$  of an RNA sequence.

If *structure* is not a NULL pointer on input, it contains on return a string consisting of the letters ". , | { } ( ) " denoting bases that are essentially unpaired, weakly paired, strongly paired without preference, weakly upstream (downstream) paired, or strongly up- (down-)stream paired bases, respectively. If [fold\\_constrained](#) is not 0, the *structure* string is interpreted on input as a list of constraints for the folding. The character "x" marks bases that must be unpaired, matching brackets " ( ) " denote base pairs, all other characters are ignored. Any pairs conflicting with the constraint will be forbidden. This is usually sufficient to ensure the constraints are honored. If [do\\_backtrack](#) has been set to 0 base pairing probabilities will not be computed (saving CPU time), otherwise [pr](#) will contain the probability that bases  $i$  and  $j$  pair.

**Note**

The global array [pr](#) is deprecated and the user who wants the calculated base pair probabilities for further computations is advised to use the function [export\\_bppm\(\)](#).

**OpenMP:** This function is not entirely threadsafe. While the recursions are working on their own copies of data the model details for the recursions are determined from the global settings just before entering the recursions. Consider using [pf\\_fold\\_par\(\)](#) for a really threadsafe implementation.

**Precondition**

This function takes its model details from the global variables provided in *RNAlib*

**Postcondition**

After successful run the hidden folding matrices are filled with the appropriate Boltzmann factors. Depending on whether the global variable [do\\_backtrack](#) was set the base pair probabilities are already computed and may be accessed for further usage via the [export\\_bppm\(\)](#) function. A call of [free\\_pf\\_arrays\(\)](#) will free all memory allocated by this function. Successive calls will first free previously allocated memory before starting the computation.

**See also**

[pf\\_fold\\_par\(\)](#), [pf\\_circ\\_fold\(\)](#), [bppm\\_to\\_structure\(\)](#), [export\\_bppm\(\)](#)

**Parameters**

<i>sequence</i>	The RNA sequence input
<i>structure</i>	A pointer to a char array where a base pair probability information can be stored in a pseudo-dot-bracket notation (may be NULL, too)

**Returns**

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

**13.12.2.9 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>	<code>sequence</code>	The RNA sequence input
<code>in, out</code>	<code>structure</code>	A pointer to a char array where a base pair probability information can be stored in a pseudo-dot-bracket notation (may be NULL, too)

**Returns**

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

**13.12.2.10 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()`

#### 13.12.2.11 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

#### 13.12.2.12 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

### 13.12.2.13 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.

```
00001 FLT_OR_DBL *pr = export_bppm();
00002 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

### 13.12.2.14 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 <sup>B</sup> matrix
out	<i>qm_p</i>	A pointer to the Q <sup>M</sup> 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

13.12.2.15 `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

13.12.2.16 `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

13.12.2.17 `vrna_plist_t* vrna_plist_from_probs ( vrna_fold_compound_t * vc, double cut_off )`

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

Create a `vrna_plist_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>cutoff</i>	The cutoff value

#### Returns

A pointer to the plist that is to be created

13.12.2.18 `void assign_plist_from_pr ( vrna_plist_t ** pl, FLT_OR_DBL * probs, int length, double cutoff )`

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

Create a `vrna_plist_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

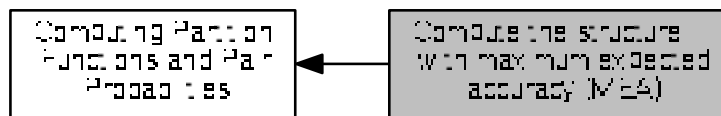
**Deprecated** Use `vrna_plist_from_probs()` instead!

#### Parameters

out	<i>pl</i>	A pointer to the <code>vrna_plist_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

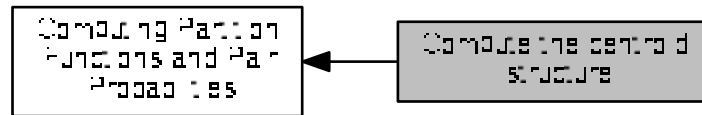
### 13.13 Compute the structure with maximum expected accuracy (MEA)

Collaboration diagram for Compute the structure with maximum expected accuracy (MEA):



## 13.14 Compute the centroid structure

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\\_plist\\_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.*

### 13.14.1 Detailed Description

### 13.14.2 Function Documentation

#### 13.14.2.1 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 adresssed 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

### 13.14.2.2 `char* vrna_centroid_from_plist ( int length, double * dist, vrna_plist_t * pl )`

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

Get the centroid structure of the ensemble.

This function is a threadsafe replacement for `centroid()` with a `vrna_plist_t` input

The centroid is the structure with the minimal average distance to all other structures

$$\langle d(S) \rangle = \sum_{(i,j) \in S} (1 - p_{ij}) + \sum_{(i,j) \notin S} p_{ij}$$

Thus, the centroid is simply the structure containing all pairs with  $p_{ij} > 0.5$ . The distance of the centroid to the ensemble is written to the memory addressed by `dist`.

#### Parameters

in	<i>length</i>	The length of the sequence
out	<i>dist</i>	A pointer to the distance variable where the centroid distance will be written to
in	<i>pl</i>	A pair list containing base pair probability information about the ensemble

#### Returns

The centroid structure of the ensemble in dot-bracket notation

### 13.14.2.3 `char* vrna_centroid_from_probs ( int length, double * dist, FLT_OR_DBL * probs )`

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

Get the centroid structure of the ensemble.

This function is a threadsafe replacement for `centroid()` with a probability array input

The centroid is the structure with the minimal average distance to all other structures

$$\langle d(S) \rangle = \sum_{(i,j) \in S} (1 - p_{ij}) + \sum_{(i,j) \notin S} p_{ij}$$

Thus, the centroid is simply the structure containing all pairs with  $p_{ij} > 0.5$ . The distance of the centroid to the ensemble is written to the memory addressed by `dist`.

#### Parameters

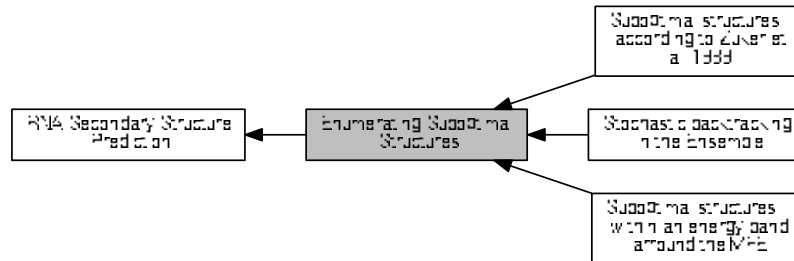
in	<i>length</i>	The length of the sequence
out	<i>dist</i>	A pointer to the distance variable where the centroid distance will be written to
in	<i>probs</i>	An upper triangular matrix containing base pair probabilities (access via <code>iindx vrna_idx_row_wise()</code> )

#### Returns

The centroid structure of the ensemble in dot-bracket notation

## 13.15 Enumerating Suboptimal Structures

Collaboration diagram for Enumerating Suboptimal Structures:



### Modules

- [Suboptimal structures according to Zuker et al. 1989](#)
- [Suboptimal structures within an energy band around the MFE](#)
- [Stochastic backtracking in the Ensemble](#)

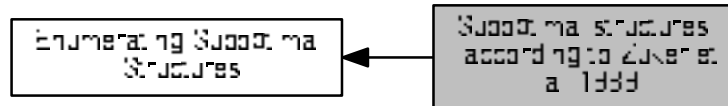
### Files

- file [subopt.h](#)  
*RNAsubopt and density of states declarations.*

### 13.15.1 Detailed Description

## 13.16 Suboptimal structures according to Zuker et al. 1989

Collaboration diagram for Suboptimal structures according to 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.

### 13.16.1 Detailed Description

### 13.16.2 Function Documentation

#### 13.16.2.1 `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 [19], 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**

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

**Returns**

List of zucker suboptimal structures

**13.16.2.2 SOLUTION\*** `zuckersubopt ( const char * string )`

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

Compute Zuker type suboptimal structures.

Compute Suboptimal structures according to M. Zuker, i.e. for every possible base pair the minimum energy structure containing the resp. base pair. Returns a list of these structures and their energies.

**Deprecated** use `vrna_zuckersubopt()` instead

**Parameters**

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

**Returns**

List of zucker suboptimal structures

**13.16.2.3 SOLUTION\*** `zuckersubopt_par ( const char * string, vrna_param_t * parameters )`

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

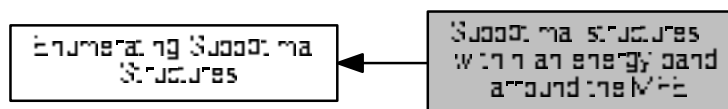
Compute Zuker type suboptimal structures.

**Deprecated** use `vrna_zuckersubopt()` instead



## 13.17 Suboptimal structures within an energy band around the MFE

Collaboration diagram for Suboptimal structures within an energy band around the MFE:



### 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.*
- [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.*

#### 13.17.1 Detailed Description

#### 13.17.2 Function Documentation

13.17.2.1 [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 [17]. 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.

See also

[vrna\\_subopt\\_zuker\(\)](#)

## Parameters

<i>vc</i>	
<i>delta</i>	
<i>sorted</i>	Sort results by energy in ascending order
<i>fp</i>	

## Returns

13.17.2.2 **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

13.17.2.3 **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

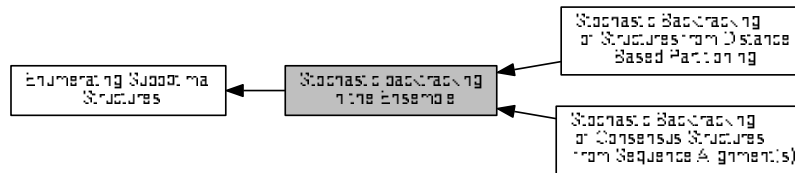
## Parameters

<i>seq</i>	
<i>sequence</i>	
<i>delta</i>	
<i>fp</i>	

Returns

## 13.18 Stochastic backtracking in the Ensemble

Collaboration diagram for Stochastic backtracking in the Ensemble:



### Modules

- [Stochastic Backtracking of Consensus Structures from Sequence Alignment\(s\)](#)
- [Stochastic Backtracking of Structures from Distance Based Partitioning](#)

*Contains functions related to stochastic backtracking from a specified distance class.*

### Functions

- `char * vrna_pbacktrack5 (vrna_fold_compound_t *vc, int length)`  
*Sample a secondary structure of a subsequence from the Boltzmann ensemble according its probability.*
- `char * vrna_pbacktrack (vrna_fold_compound_t *vc)`  
*Sample a secondary structure (consensus structure) from the Boltzmann ensemble according its probability.*
- `char * pbacktrack (char *sequence)`  
*Sample a secondary structure from the Boltzmann ensemble according its probability.*
- `char * pbacktrack_circ (char *sequence)`  
*Sample a secondary structure of a circular RNA from the Boltzmann ensemble according its probability.*

### Variables

- `int st_back`  
*Flag indicating that auxiliary arrays are needed throughout the computations. This is essential for stochastic backtracking.*

#### 13.18.1 Detailed Description

#### 13.18.2 Function Documentation

##### 13.18.2.1 `char* vrna_pbacktrack5 ( vrna_fold_compound_t * vc, int length )`

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

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

#### Precondition

The fold compound has to be obtained using the #VRNA\_OPTION\_HYBRID option in `vrna_fold_compound()` `vrna_pf()` has to be called first to fill the partition function matrices

## Parameters

<i>vc</i>	The fold compound data structure
<i>length</i>	The length of the subsequence to consider (starting with 5' end)

## Returns

A sampled secondary structure in dot-bracket notation

**13.18.2.2** `char* vrna_pbacktrack ( vrna_fold_compound_t * vc )`

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

Sample a secondary structure (consensus structure) from the Boltzmann ensemble according its probability.

## Precondition

The dynamic programming (DP) matrices have to allow for unique multibranch loop decomposition, i.e. the `vrna_md_t.uniq_ML` flag has to be non-zero before calling `vrna_fold_compound()`  
`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_VC_TYPE_SINGLE`, and `VRNA_VC_TYPE_ALIGNMENT`.  
 The function will automagically detect cicular RNAs based on the `model_details` in `exp_params` as provided via the `vrna_fold_compound_t`

## Parameters

<i>vc</i>	The fold compound data structure
<i>length</i>	The length of the subsequence to consider (starting with 5' end)

## Returns

A sampled secondary structure in dot-bracket notation

**13.18.2.3** `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

13.18.2.4 `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

## 13.18.3 Variable Documentation

13.18.3.1 `int st_back`

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

Flag indicating that auxiliary arrays are needed throughout the computations. This is essential for stochastic backtracking.

Set this variable to 1 prior to a call of [pf\\_fold\(\)](#) to ensure that all matrices needed for stochastic backtracking are filled in the forward recursions

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

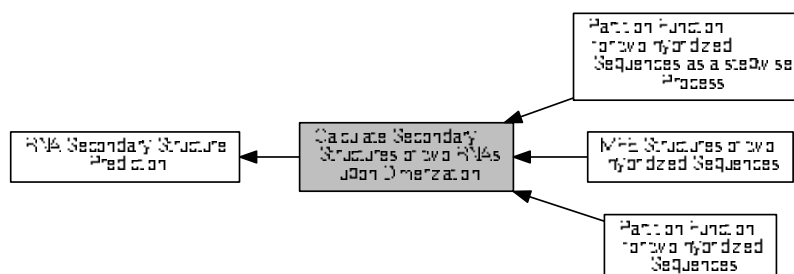
## See also

[pbacktrack\(\)](#), [pbacktrack\\_circ](#)

## 13.19 Calculate Secondary Structures of two RNAs upon Dimerization

Predict structures formed by two molecules upon hybridization.

Collaboration diagram for Calculate Secondary Structures of two RNAs upon Dimerization:



### Modules

- [MFE Structures of two hybridized Sequences](#)
- [Partition Function for two hybridized Sequences](#)  
*Partition Function Cofolding.*
- [Partition Function for two hybridized Sequences as a stepwise Process](#)  
*Partition Function Cofolding as a stepwise process.*

### 13.19.1 Detailed Description

Predict structures formed by two molecules upon hybridization.

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

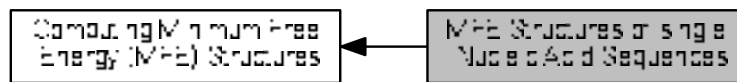
One approach to co-folding two RNAs consists of concatenating the two sequences and keeping track of the concatenation point in all energy evaluations. Correspondingly, many of the [cofold\(\)](#) and [co\\_pf\\_fold\(\)](#) routines below take one sequence string as argument and use the the global variable [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).

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.

## 13.20 MFE Structures of single Nucleic Acid Sequences

This module contains all functions and variables related to the calculation of global minimum free energy structures for single sequences.

Collaboration diagram for MFE Structures of single Nucleic Acid Sequences:



### Functions

- float [vrna\\_fold](#) (const char \*string, char \*structure)  
*Compute Minimum Free Energy (MFE), and a corresponding secondary structure for an RNA sequence.*
- float [vrna\\_circfold](#) (const char \*string, char \*structure)  
*Compute Minimum Free Energy (MFE), and a corresponding secondary structure for a circular RNA sequence.*
- float [fold\\_par](#) (const char \*sequence, char \*structure, [vrna\\_param\\_t](#) \*parameters, int is\_constrained, int is\_circular)  
*Compute minimum free energy and an appropriate secondary structure of an RNA sequence.*
- float [fold](#) (const char \*sequence, char \*structure)  
*Compute minimum free energy and an appropriate secondary structure of an RNA sequence.*
- float [circfold](#) (const char \*sequence, char \*structure)  
*Compute minimum free energy and an appropriate secondary structure of a circular RNA sequence.*
- void [free\\_arrays](#) (void)  
*Free arrays for mfe folding.*
- void [update\\_fold\\_params](#) (void)  
*Recalculate energy parameters.*
- void [update\\_fold\\_params\\_par](#) ([vrna\\_param\\_t](#) \*parameters)  
*Recalculate energy parameters.*
- void [export\\_fold\\_arrays](#) (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*indx\_p, char \*\*ptype\_p)
- void [export\\_fold\\_arrays\\_par](#) (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*indx\_p, char \*\*ptype\_p, [vrna\\_param\\_t](#) \*\*P\_p)
- void [export\\_circfold\\_arrays](#) (int \*Fc\_p, int \*FcH\_p, int \*FcI\_p, int \*FcM\_p, int \*\*fM2\_p, int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*indx\_p, char \*\*ptype\_p)
- void [export\\_circfold\\_arrays\\_par](#) (int \*Fc\_p, int \*FcH\_p, int \*FcI\_p, int \*FcM\_p, int \*\*fM2\_p, int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*indx\_p, char \*\*ptype\_p, [vrna\\_param\\_t](#) \*\*P\_p)
- int [LoopEnergy](#) (int n1, int n2, int type, int type\_2, int si1, int sj1, int sp1, int sq1)
- int [HairpinE](#) (int size, int type, int si1, int sj1, const char \*string)
- void [initialize\\_fold](#) (int length)

### 13.20.1 Detailed Description

This module contains all functions and variables related to the calculation of global minimum free energy structures for single sequences.

The library provides a fast dynamic programming minimum free energy folding algorithm as described by "Zuker & Stiegler (1981)" [20].



## 13.20.2 Function Documentation

### 13.20.2.1 float `vrna_fold` ( const char \* *string*, char \* *structure* )

```
#include <ViennaRNA/fold.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\(\)](#), [vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_t](#)

#### 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

### 13.20.2.2 float `vrna_circfold` ( const char \* *string*, char \* *structure* )

```
#include <ViennaRNA/fold.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 [8] 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\(\)](#), [vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_t](#)

## 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

**13.20.2.3** `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 prescaled 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 postprocessing steps in case RNA sequence is circular (0==off)

**Returns**

the minimum free energy (MFE) in kcal/mol

**13.20.2.4 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

**13.20.2.5 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

## 13.20.2.6 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!

## 13.20.2.7 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!

## 13.20.2.8 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!

## 13.20.2.9 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!

13.20.2.10 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!

13.20.2.11 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* )

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

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

13.20.2.12 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* )

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

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

13.20.2.13 int LoopEnergy ( int *n1*, int *n2*, int *type*, int *type\_2*, int *si1*, int *sj1*, int *sp1*, int *sq1* )

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

**Deprecated** {This function is deprecated and will be removed soon. Use [E\\_IntLoop\(\)](#) instead!}

13.20.2.14 int HairpinE ( int *size*, int *type*, int *si1*, int *sj1*, const char \* *string* )

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

**Deprecated** {This function is deprecated and will be removed soon. Use [E\\_Hairpin\(\)](#) instead!}

13.20.2.15 void initialize\_fold ( int *length* )

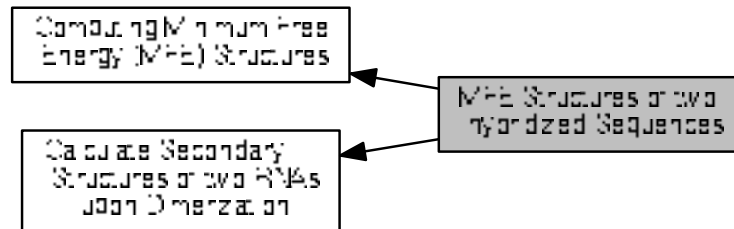
```
#include <ViennaRNA/fold.h>
```

Allocate arrays for folding

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

## 13.21 MFE Structures of two hybridized Sequences

Collaboration diagram for MFE Structures of two hybridized Sequences:



### Files

- file [cofold.h](#)  
*MFE version of cofolding routines.*

### Functions

- float [vrna\\_cofold](#) (const char \*string, char \*structure)  
*Compute Minimum Free Energy (MFE), and a corresponding secondary structure for two dimerized RNA sequences.*
- 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 [vrna\\_mfe\\_dimer](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, char \*structure)  
*Compute the minimum free energy of two interacting RNA molecules.*

### 13.21.1 Detailed Description

### 13.21.2 Function Documentation

#### 13.21.2.1 float vrna\_cofold ( const char \* *string*, char \* *structure* )

```
#include <ViennaRNA/cofold.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

#### 13.21.2.2 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

13.21.2.3 `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

13.21.2.4 `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\(\)](#)

13.21.2.5 `void update_cofold_params ( void )`

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

Recalculate parameters.

**Deprecated** See [vrna\\_params\\_subst\(\)](#) for an alternative using the new API

13.21.2.6 `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



13.21.2.7 `void export_cofold_arrays_gg ( int ** f5_p, int ** c_p, int ** fML_p, int ** fM1_p, int ** fc_p, int ** ggg_p, int ** indx_p, char ** ptype_p )`

```
#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

<code>f5_p</code>	A pointer to the 'f5' array, i.e. array containing best free energy in interval [1..j]
<code>c_p</code>	A pointer to the 'c' array, i.e. array containing best free energy in interval [i..j] given that i pairs with j
<code>fML_p</code>	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
<code>fM1_p</code>	A pointer to the 'M1' array, i.e. array containing best free energy in interval [i..j] for multiloop segment with exactly one stem
<code>fc_p</code>	A pointer to the 'fc' array, i.e. array ...
<code>ggg_p</code>	A pointer to the 'ggg' array, i.e. array containing best free energy of a gquadruplex delimited by [i..j]
<code>indx_p</code>	A pointer to the indexing array used for accessing the energy matrices
<code>ptype↔ _p</code>	A pointer to the ptype array containing the base pair types for each possibility (i,j)

13.21.2.8 `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>pptype←_p</i>	A pointer to the ptype array containing the base pair types for each possibility (i,j)

13.21.2.9 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

13.21.2.10 void initialize\_cofold ( int *length* )

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

allocate arrays for folding

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

13.21.2.11 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

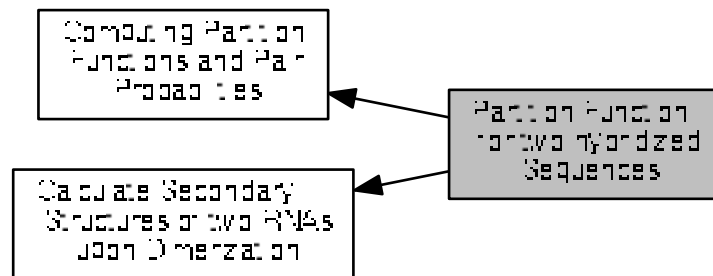
## Returns

minimum free energy of the structure

## 13.22 Partition Function for two hybridized Sequences

Partition Function Cofolding.

Collaboration diagram for Partition Function for two hybridized Sequences:



### Files

- file [part\\_func\\_co.h](#)  
*Partition function for two RNA sequences.*

### Data Structures

- struct [vrna\\_dimer\\_pf\\_s](#)
- struct [vrna\\_dimer\\_conc\\_s](#)

### 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\\_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\(\)](#)*

### Functions

- [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.*
- void [vrna\\_pf\\_dimer\\_probs](#) (double FAB, double FA, double FB, [vrna\\_plist\\_t](#) \*prAB, const [vrna\\_plist\\_t](#) \*prA, const [vrna\\_plist\\_t](#) \*prB, int Alength, const [vrna\\_exp\\_param\\_t](#) \*exp\_params)  
*Compute Boltzmann probabilities of dimerization without homodimers.*
- [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.*

## Variables

- int [mirnatog](#)  
*Toggles no intrabp in 2nd mol.*
- double [F\\_monomer](#) [2]  
*Free energies of the two monomers.*

### 13.22.1 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.

### 13.22.2 Data Structure Documentation

#### 13.22.2.1 struct vrna\_dimer\_pf\_s

##### 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*

#### 13.22.2.2 struct vrna\_dimer\_conc\_s

##### Data Fields

- double [A0](#)  
*start concentration A*
- double [B0](#)  
*start concentration B*
- double [ABc](#)  
*End concentration AB.*

### 13.22.3 Function Documentation

#### 13.22.3.1 `vrna_dimer_pf_t vrna_pf_dimer ( vrna_fold_compound_t * vc, char * structure )`

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

Calculate partition function and base pair probabilities of nucleic acid/nucleic acid dimers.

This is the cofold partition function folding.

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.

#### 13.22.3.2 `void vrna_pf_dimer_probs ( double FAB, double FA, double FB, vrna_plist_t * prAB, const vrna_plist_t * prA, const vrna_plist_t * prB, int Alength, const vrna_exp_param_t * exp_params )`

```
#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 [vrna\\_plist\\_from\\_probs\(\)](#), the dimer probabilities 'prAB' are modified in place.

#### 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
<i>exp_params</i>	The precomputed Boltzmann factors

13.22.3.3 `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/part_func_co.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>FEAB</i>	Free energy of AB dimer ( <i>FcAB</i> entry)
<i>FEAA</i>	Free energy of AA dimer ( <i>FcAB</i> entry)
<i>FEBB</i>	Free energy of BB dimer ( <i>FcAB</i> 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

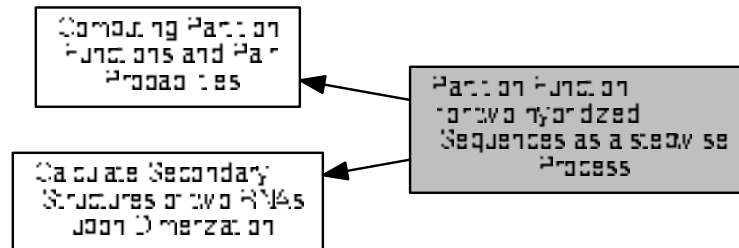
#### Returns

`vrna_dimer_conc_t` array containing the equilibrium energies and start concentrations

## 13.23 Partition Function for two hybridized Sequences as a stepwise Process

Partition Function Cofolding as a stepwise process.

Collaboration diagram for Partition Function for two hybridized Sequences as a stepwise Process:



### Files

- file [part\\_func\\_up.h](#)

*Partition Function Cofolding as 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\(\)](#).*

### 13.23.1 Detailed Description

Partition Function Cofolding as a stepwise process.



### 13.23.2 Function Documentation

#### 13.23.2.1 `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

<code>sequence</code>	
<code>max_w</code>	

#### Returns

#### 13.23.2.2 `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 constraints 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  $w = w + \text{incr5} + \text{incr3}$  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 `Gikjl` 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'. `Gikjl_wo` is `Gikjl` without the probability of being unpaired.

Use `free_interact()` to free the returned structure, all other stuff is freed inside `pf_interact()`.

**Parameters**

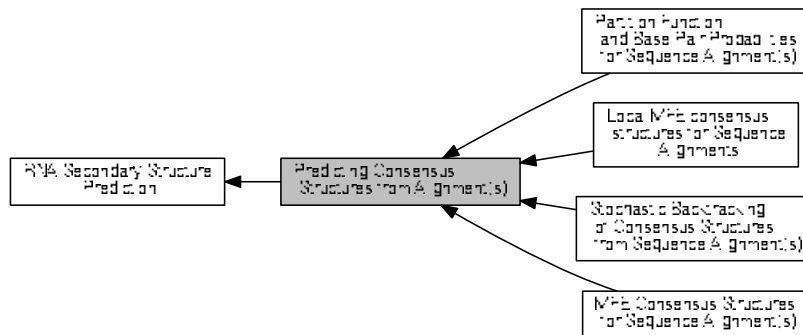
<i>s1</i>	
<i>s2</i>	
<i>p_c</i>	
<i>p_c2</i>	
<i>max<sub>←→</sub></i> <i>_w</i>	
<i>cstruc</i>	
<i>incr3</i>	
<i>incr5</i>	

**Returns**

## 13.24 Predicting Consensus Structures from Alignment(s)

compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments

Collaboration diagram for Predicting Consensus Structures from Alignment(s):



### Modules

- [MFE Consensus Structures for Sequence Alignment\(s\)](#)
- [Partition Function and Base Pair Probabilities for Sequence Alignment\(s\)](#)
- [Stochastic Backtracking of Consensus Structures from Sequence Alignment\(s\)](#)
- [Local MFE consensus structures for Sequence Alignments](#)

### Files

- file [alifold.h](#)  
*compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments*

### 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.*
- 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.*
- void [update\\_alifold\\_params](#) (void)  
*Update the energy parameters for alifold function.*
- int [vrna\\_aln\\_mpi](#) (char \*Aseq[], int n\_seq, int length, int \*mini)  
*Get the mean pairwise identity in steps from ?to?(ident)*
- 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.*
- float \*\* [get\\_ribosum](#) (const char \*\*Alseq, int n\_seq, int length)  
*Retrieve a RiboSum Scoring Matrix for a given Alignment.*

## 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.*

### 13.24.1 Detailed Description

compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments

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 [\[6\]](#) and [\[1\]](#).

### 13.24.2 Function Documentation

13.24.2.1 float [energy\\_of\\_alistruct](#) ( const char \*\* sequences, const char \* structure, int n\_seq, float \* energy )

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

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

13.24.2.2 `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<sub>p</sub></code>	A pointer to the 'S5' array
<code>S3<sub>p</sub></code>	A pointer to the 'S3' array
<code>a2s<sub>p</sub></code>	A pointer to the pair type matrix
<code>Ss_p</code>	A pointer to the 'Ss' array
<code>qb_p</code>	A pointer to the $Q^B$ matrix
<code>qm<sub>p</sub></code>	A pointer to the $Q^M$ matrix
<code>q1k<sub>p</sub></code>	A pointer to the 5' slice of the Q matrix ( $q1k(k) = Q(1, k)$ )
<code>qln<sub>p</sub></code>	A pointer to the 3' slice of the Q matrix ( $qln(l) = Q(l, n)$ )

## Returns

Non Zero if everything went fine, 0 otherwise

### 13.24.2.3 void update\_alifold\_params ( void )

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

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](#).

### 13.24.2.4 int vrna\_aln\_mpi ( char \* *Alseq*[], int *n\_seq*, int *length*, int \* *mini* )

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

Get the mean pairwise identity in steps from ?to?(ident)

#### 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

### 13.24.2.5 int get\_mpi ( char \* *Alseq*[], int *n\_seq*, int *length*, int \* *mini* )

```
#include <ViennaRNA/aln_util.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

**13.24.2.6** `void encode_aln_sequence ( const char * sequence, short * S, short * s5, short * s3, char * ss, unsigned short * as, int circ )`

```
#include <ViennaRNA/aln_util.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 ( <i>circ</i> =0)

**13.24.2.7** `void alloc_sequence_arrays ( const char ** sequences, short *** S, short *** S5, short *** S3, unsigned short *** a2s, char *** Ss, int circ )`

```
#include <ViennaRNA/aln_util.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 ( <i>circ</i> =0)

**13.24.2.8** `void free_sequence_arrays ( unsigned int n_seq, short *** S, short *** S5, short *** S3, unsigned short *** a2s, char *** Ss )`

```
#include <ViennaRNA/aln_util.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

### 13.24.3 Variable Documentation

**13.24.3.1** `double cv_fact`

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

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.

**13.24.3.2** `double nc_fact`

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

This variable controls the magnitude of the penalty for non-compatible sequences in the covariance term of alignment folding algorithms.

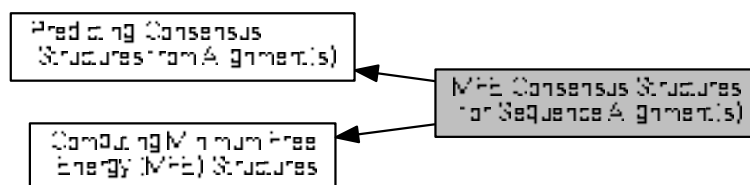
**Deprecated** See [#vrna\\_md\\_t.nc\\_fact](#), and [vrna\\_mfe\(\)](#) to avoid using global variables

Default is 1.



## 13.25 MFE Consensus Structures for Sequence Alignment(s)

Collaboration diagram for MFE Consensus Structures for Sequence Alignment(s):



### Functions

- 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 [alifold](#) (const char \*\*strings, char \*structure)  
Compute MFE and according consensus structure of an alignment of sequences.
- float [circalifold](#) (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.

### 13.25.1 Detailed Description

### 13.25.2 Function Documentation

#### 13.25.2.1 float vrna\_alifold ( const char \*\* sequences, char \* structure )

```
#include <ViennaRNA/alifold.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\(\)](#), [vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_t](#)

**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

**13.25.2.2 float vrna\_circalfold ( const char \*\* *ssequences*, char \* *structure* )**

```
#include <ViennaRNA/alifold.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 [8] 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\(\)](#), [vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_t](#)

**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

**13.25.2.3 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

13.25.2.4 `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

### 13.25.2.5 void free\_alifold\_arrays ( void )

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

Free the memory occupied by MFE alifold functions.

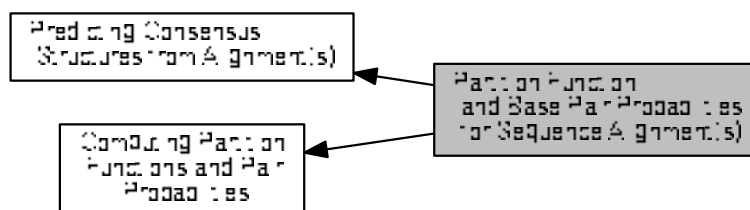
**Deprecated** Usage of this function is discouraged! It only affects memory being free'd that was allocated by an old API function before. Release of memory occupied by the newly introduced [vrna\\_fold\\_compound\\_t](#) is handled by [vrna\\_vrna\\_fold\\_compound\\_free\(\)](#)

#### See also

[vrna\\_vrna\\_fold\\_compound\\_free\(\)](#)

## 13.26 Partition Function and Base Pair Probabilities for Sequence Alignment(s)

Collaboration diagram for Partition Function and Base Pair Probabilities for Sequence Alignment(s):



### Functions

- float [vrna\\_pf\\_alifold](#) (const char \*\*strings, char \*structure, [vrna\\_plist\\_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\\_plist\\_t](#) \*\*pl)
 

Compute Partition function  $Q$  (and base pair probabilities) for an alignment of circular RNA sequences using a comparative method.
- float [alipf\\_fold\\_par](#) (const char \*\*sequences, char \*structure, [vrna\\_plist\\_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\\_plist\\_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\\_plist\\_t](#) \*\*pl)
- [FLT\\_OR\\_DBL](#) \* [export\\_alipf\\_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.

### 13.26.1 Detailed Description

### 13.26.2 Function Documentation

13.26.2.1 float [vrna\\_pf\\_alifold](#) ( const char \*\* strings, char \* structure, [vrna\\_plist\\_t](#) \*\* pl )

```
#include <ViennaRNA/alifold.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_plist_t</a> to store pairing probabilities (Maybe NULL)

**Returns**

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

**13.26.2.2** `float vrna_pf_circalifold ( const char ** sequences, char * structure, vrna_plist_t ** pl )`

```
#include <ViennaRNA/alifold.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 [8] for further details.

**See also**

[vrna\\_pf\\_alifold\(\)](#), [vrna\\_pf\(\)](#), [vrna\\_fold\\_compound\\_comparative\(\)](#), [vrna\\_fold\\_compound\\_t](#)

**Parameters**

<i>sequences</i>	Sequence alignment of circular RNAs
<i>structure</i>	A pointer to the character array where position-wise pairing propensity will be stored. (Maybe NULL)
<i>pl</i>	A pointer to a list of <a href="#">vrna_plist_t</a> to store pairing probabilities (Maybe NULL)

## Returns

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

13.26.2.3 float alipf\_fold\_par ( const char \*\* *sequences*, char \* *structure*, vrna\_plist\_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

13.26.2.4 float alipf\_fold ( const char \*\* *sequences*, char \* *structure*, vrna\_plist\_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

### 13.26.2.5 float alipf\_circ\_fold ( const char \*\* *sequences*, char \* *structure*, vrna\_plist\_t \*\* *pl* )

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

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

#### Parameters

<i>sequences</i>	
<i>structure</i>	
<i>pl</i>	

#### Returns

### 13.26.2.6 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

### 13.26.2.7 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!

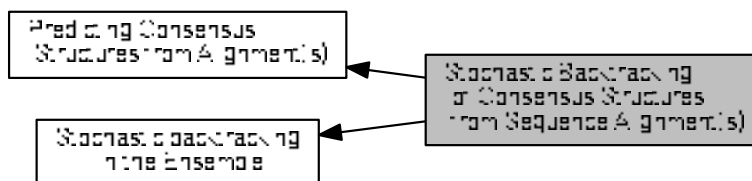
#### See also

[vrna\\_fold\\_compound\\_t](#), [vrna\\_vrna\\_fold\\_compound\\_free\(\)](#)



## 13.27 Stochastic Backtracking of Consensus Structures from Sequence Alignment(s)

Collaboration diagram for Stochastic Backtracking of Consensus Structures from Sequence Alignment(s):



### Functions

- `char * alipbacktrack (double *prob)`

*Sample a consensus secondary structure from the Boltzmann ensemble according its probability.*

#### 13.27.1 Detailed Description

#### 13.27.2 Function Documentation

13.27.2.1 `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

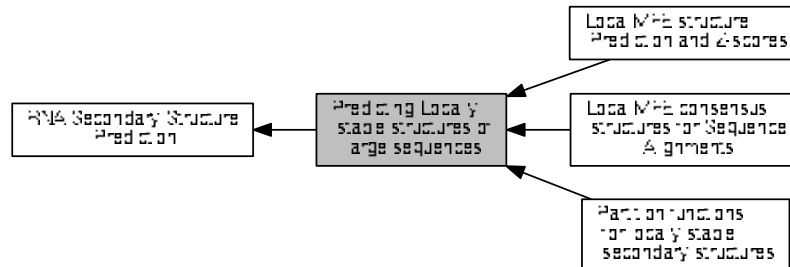
<i>prob</i>	to be described (berni)
-------------	-------------------------

#### Returns

A sampled consensus secondary structure in dot-bracket notation

## 13.28 Predicting Locally stable structures of large sequences

Collaboration diagram for Predicting Locally stable structures of large sequences:



### Modules

- [Local MFE structure Prediction and Z-scores](#)
- [Partition functions for locally stable secondary structures](#)
- [Local MFE consensus structures for Sequence Alignments](#)

### Files

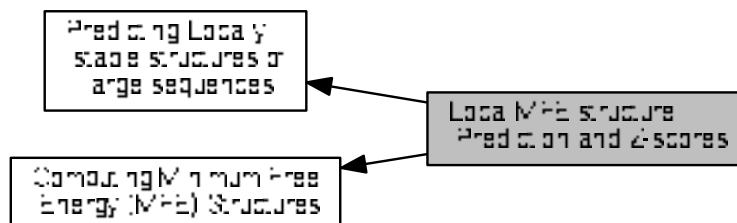
- file [Lfold.h](#)  
*Predicting local MFE structures of large sequences.*

### 13.28.1 Detailed Description

Local structures can be predicted by a modified version of the [fold\(\)](#) algorithm that restricts the span of all base pairs.

## 13.29 Local MFE structure Prediction and Z-scores

Collaboration diagram for Local MFE structure Prediction and Z-scores:



### Functions

- float [vrna\\_Lfold](#) (const char \*string, int window\_size, FILE \*file)  
*Local MFE prediction using a sliding window approach (simplified interface)*
- 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 [Lfold](#) (const char \*string, char \*structure, int maxdist)  
*The local analog to [fold\(\)](#).*
- float [Lfoldz](#) (const char \*string, char \*structure, int maxdist, int zsc, double min\_z)
- float [vrna\\_mfe\\_window](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, FILE \*file)  
*Local MFE prediction using a sliding window approach.*
- 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)*

### 13.29.1 Detailed Description

### 13.29.2 Function Documentation

#### 13.29.2.1 float vrna\_Lfold ( const char \* string, int window\_size, FILE \* file )

```
#include <ViennaRNA/Lfold.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\(\)](#), [vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_t](#)

## 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)

13.29.2.2 `float vrna_Lfoldz ( const char * string, int window_size, double min_z, FILE * file )`

```
#include <ViennaRNA/Lfold.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\(\)](#), [vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_t](#)

## 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)

13.29.2.3 `float Lfold ( const char * string, 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!

13.29.2.4 `float Lfoldz ( const char * string, char * structure, int maxdist, int zsc, double min_z )`

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

**Deprecated** Use `vrna_mfe_window_zscore()` instead!

13.29.2.5 `float vrna_mfe_window ( vrna_fold_compound_t * vc, FILE * file )`

```
#include <ViennaRNA/mfe.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)

13.29.2.6 `float vrna_mfe_window_zscore ( vrna_fold_compound_t * vc, double min_z, FILE * file )`

```
#include <ViennaRNA/mfe.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

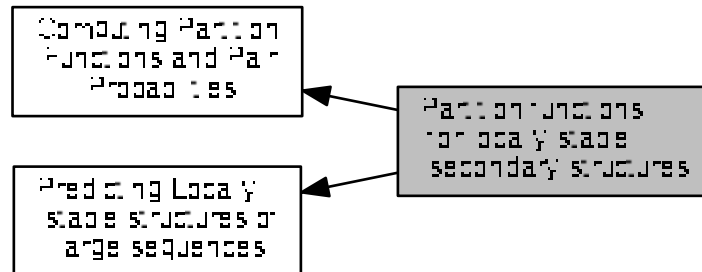
`vrna_fold_compound()`, `vrna_mfe_window_zscore()`, `vrna_mfe()`, `vrna_Lfold()`, `vrna_Lfoldz()`, `#VRNA_OPTION_WINDOW`, `vrna_md_t.max_bp_span`, `vrna_md_t.window_size`

## 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)

## 13.30 Partition functions for locally stable secondary structures

Collaboration diagram for Partition functions for locally stable secondary structures:



### Files

- file [LPfold.h](#)

*Function declarations of partition function variants of the Lfold algorithm.*

### Functions

- void [update\\_pf\\_paramsLP](#) (int length)
- [plist](#) \* [pfl\\_fold](#) (char \*sequence, int winSize, int pairSize, float cutoffb, double \*\*pU, [plist](#) \*\*dpp2, FILE \*pUfp, FILE \*spup)  
*Compute partition functions for locally stable secondary structures.*
- [plist](#) \* [pfl\\_fold\\_par](#) (char \*sequence, int winSize, int pairSize, float cutoffb, double \*\*pU, [plist](#) \*\*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.*

### 13.30.1 Detailed Description

### 13.30.2 Function Documentation

#### 13.30.2.1 void update\_pf\_paramsLP ( int length )

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

## Parameters

<i>length</i>	
---------------	--

**13.30.2.2** `plist* pfl_fold ( char * sequence, int winSize, int pairSize, float cutoffb, double ** pU, plist ** 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

**13.30.2.3** `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

13.30.2.4 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.

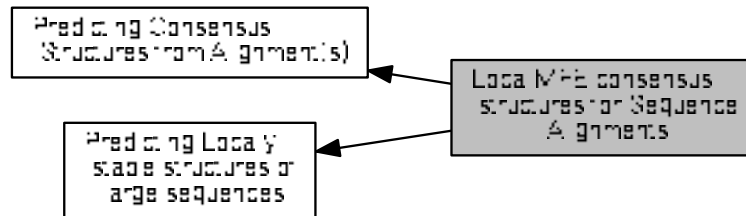
Can write either the unpaired probabilities (accessibilities) pU or the opening energies  $-\log(pU)kT$  into a file

## 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

### 13.31 Local MFE consensus structures for Sequence Alignments

Collaboration diagram for Local MFE consensus structures for Sequence Alignments:



#### Functions

- float [aliLfold](#) (const char \*\*strings, char \*structure, int maxdist)

#### 13.31.1 Detailed Description

#### 13.31.2 Function Documentation

13.31.2.1 float aliLfold ( const char \*\* *strings*, char \* *structure*, int *maxdist* )

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

##### Parameters

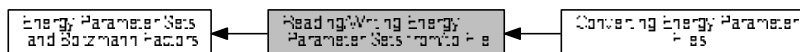
<i>strings</i>	
<i>structure</i>	
<i>maxdist</i>	

##### Returns

## 13.32 Reading/Writing Energy Parameter Sets from/to File

Read and Write energy parameter sets from and to text files.

Collaboration diagram for Reading/Writing Energy Parameter Sets from/to File:



### Modules

- [Converting Energy Parameter Files](#)  
*Convert energy parameter files into the latest format.*

### Files

- file [read\\_epars.h](#)

### Functions

- 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.*

### 13.32.1 Detailed Description

Read and Write energy parameter sets from and to text files.

A default set of parameters, identical to the one described in [11] and [15], is compiled into the library.

### 13.32.2 Function Documentation

#### 13.32.2.1 void read\_parameter\_file ( const char fname[ ] )

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

Read energy parameters from a file.

#### Parameters

<i>fname</i>	The path to the file containing the energy parameters
--------------	---

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

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

Write energy parameters to a file.

#### Parameters

<i>fname</i>	A filename (path) for the file where the current energy parameters will be written to
--------------	---

## 13.33 Converting Energy Parameter Files

Convert energy parameter files into the latest format.

Collaboration diagram for Converting Energy Parameter Files:



### Files

- file [convert\\_epars.h](#)

*Functions and definitions for energy parameter file format conversion.*

### 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)

### 13.33.1 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

### 13.33.2 Macro Definition Documentation

#### 13.33.2.1 `#define VRNA_CONVERT_OUTPUT_ALL 1U`

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

Flag to indicate printing of a complete parameter set

#### 13.33.2.2 `#define VRNA_CONVERT_OUTPUT_HP 2U`

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

Flag to indicate printing of hairpin contributions

#### 13.33.2.3 `#define VRNA_CONVERT_OUTPUT_STACK 4U`

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

Flag to indicate printing of base pair stack contributions

#### 13.33.2.4 `#define VRNA_CONVERT_OUTPUT_MM_HP 8U`

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

Flag to indicate printing of hairpin mismatch contribution

#### 13.33.2.5 `#define VRNA_CONVERT_OUTPUT_MM_INT 16U`

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

Flag to indicate printing of interior loop mismatch contribution

#### 13.33.2.6 `#define VRNA_CONVERT_OUTPUT_MM_INT_1N 32U`

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

Flag to indicate printing of 1:n interior loop mismatch contribution

**13.33.2.7** `#define VRNA_CONVERT_OUTPUT_MM_INT_23 64U`

`#include <ViennaRNA/convert_epars.h>`

Flag to indicate printing of 2:3 interior loop mismatch contribution

**13.33.2.8** `#define VRNA_CONVERT_OUTPUT_MM_MULTI 128U`

`#include <ViennaRNA/convert_epars.h>`

Flag to indicate printing of multi loop mismatch contribution

**13.33.2.9** `#define VRNA_CONVERT_OUTPUT_MM_EXT 256U`

`#include <ViennaRNA/convert_epars.h>`

Flag to indicate printing of exterior loop mismatch contribution

**13.33.2.10** `#define VRNA_CONVERT_OUTPUT_DANGLE5 512U`

`#include <ViennaRNA/convert_epars.h>`

Flag to indicate printing of 5' dangle contribution

**13.33.2.11** `#define VRNA_CONVERT_OUTPUT_DANGLE3 1024U`

`#include <ViennaRNA/convert_epars.h>`

Flag to indicate printing of 3' dangle contribution

**13.33.2.12** `#define VRNA_CONVERT_OUTPUT_INT_11 2048U`

`#include <ViennaRNA/convert_epars.h>`

Flag to indicate printing of 1:1 interior loop contribution

**13.33.2.13** `#define VRNA_CONVERT_OUTPUT_INT_21 4096U`

`#include <ViennaRNA/convert_epars.h>`

Flag to indicate printing of 2:1 interior loop contribution

**13.33.2.14** `#define VRNA_CONVERT_OUTPUT_INT_22 8192U`

`#include <ViennaRNA/convert_epars.h>`

Flag to indicate printing of 2:2 interior loop contribution

**13.33.2.15** `#define VRNA_CONVERT_OUTPUT_BULGE 16384U`

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

Flag to indicate printing of bulge loop contribution

**13.33.2.16** `#define VRNA_CONVERT_OUTPUT_INT 32768U`

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

Flag to indicate printing of interior loop contribution

**13.33.2.17** `#define VRNA_CONVERT_OUTPUT_ML 65536U`

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

Flag to indicate printing of multi loop contribution

**13.33.2.18** `#define VRNA_CONVERT_OUTPUT_MISC 131072U`

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

Flag to indicate printing of misc contributions (such as terminalAU)

**13.33.2.19** `#define VRNA_CONVERT_OUTPUT_SPECIAL_HP 262144U`

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

Flag to indicate printing of special hairpin contributions (tri-, tetra-, hexa-loops)

**13.33.2.20** `#define VRNA_CONVERT_OUTPUT_VANILLA 524288U`

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

Flag to indicate printing of given parameters only

#### Note

This option overrides all other output options, except `VRNA_CONVERT_OUTPUT_DUMP` !

**13.33.2.21** `#define VRNA_CONVERT_OUTPUT_NINIO 1048576U`

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

Flag to indicate printing of interior loop asymmetry contribution



13.33.2.22 `#define VRNA_CONVERT_OUTPUT_DUMP 2097152U`

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

Flag to indicate dumping the energy contributions from the library instead of an input file

### 13.33.3 Function Documentation

13.33.3.1 `void convert_parameter_file ( const char * iname, const char * oname, unsigned int options )`

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

Convert/dump a Vienna 1.8.4 formatted energy parameter file

The options argument allows 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_OUT↵
PUT_MM_EXT
VRNA_CONVERT_OUTPUT_DANGLE5, VRNA_CONVERT_OUTPUT_DANGLE3, VRNA_CONVERT_OUTPU↵
T_INT_11
VRNA_CONVERT_OUTPUT_INT_21, VRNA_CONVERT_OUTPUT_INT_22, VRNA_CONVERT_OUTPUT_BU↵
LGE
VRNA_CONVERT_OUTPUT_INT, VRNA_CONVERT_OUTPUT_ML, VRNA_CONVERT_OUTPUT_MISC
VRNA_CONVERT_OUTPUT_SPECIAL_HP, VRNA_CONVERT_OUTPUT_VANILLA, VRNA_CONVERT_OUT↵
PUT_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:

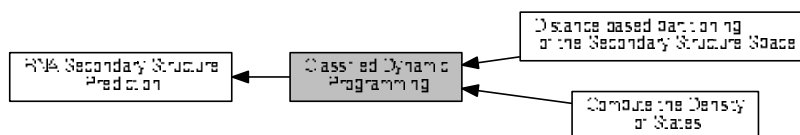
```
convert_parameter_file(ifile, ofile, option_1 | option_2 | option_n)
```

#### 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)

## 13.34 Classified Dynamic Programming

Collaboration diagram for Classified Dynamic Programming:



### Modules

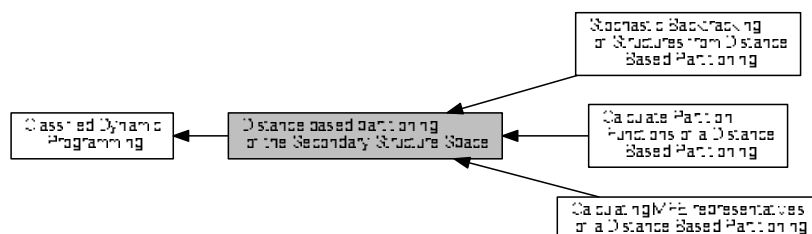
- [Distance based partitioning of the Secondary Structure Space](#)  
*Compute Thermodynamic properties for a Distance Class Partitioning of the Secondary Structure Space.*
- [Compute the Density of States](#)

### 13.34.1 Detailed Description

## 13.35 Distance based partitioning of the Secondary Structure Space

Compute Thermodynamic properties for a Distance Class Partitioning of the Secondary Structure Space.

Collaboration diagram for Distance based partitioning of the Secondary Structure Space:



### Modules

- [Calculating 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.
- [Calculate 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.

### 13.35.1 Detailed Description

Compute Thermodynamic properties for a Distance Class Partitioning of the Secondary Structure Space.

All functions related to this group implement the basic recursions for MFE folding, partition function computation and stochastic backtracking with a *classified dynamic programming* approach. 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

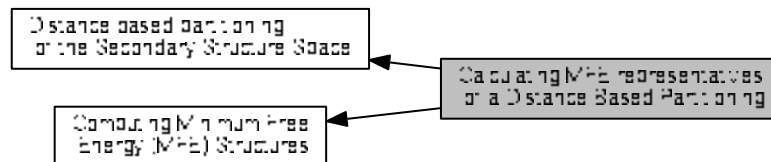
#### See also

For further details, we refer to Lorenz et al. 2009 [10]

## 13.36 Calculating 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.

Collaboration diagram for Calculating MFE representatives of a Distance Based Partitioning:



### Files

- file [2Dfold.h](#)

### Data Structures

- struct [vrna\\_sol\\_TwoD\\_t](#)  
*Solution element returned from [vrna\\_mfe\\_TwoD\(\)](#) [More...](#)*
- struct [TwoDfold\\_vars](#)  
*Variables compound for 2Dfold MFE folding. [More...](#)*

### Typedefs

- typedef struct [vrna\\_sol\\_TwoD\\_t](#) [vrna\\_sol\\_TwoD\\_t](#)  
*Solution element returned from [vrna\\_mfe\\_TwoD\(\)](#)*
- typedef struct [TwoDfold\\_vars](#) [TwoDfold\\_vars](#)  
*Variables compound for 2Dfold MFE folding.*

### Functions

- [vrna\\_sol\\_TwoD\\_t](#) \* [vrna\\_mfe\\_TwoD](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int distance1, int distance2)  
*Compute MFE's and representative for distance partitioning.*
- char \* [vrna\\_backtrack5\\_TwoD](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int k, int l, unsigned int j)  
*Backtrack a minimum free energy structure from a 5' section of specified length.*
- [TwoDfold\\_vars](#) \* [get\\_TwoDfold\\_variables](#) (const char \*seq, const char \*structure1, const char \*structure2, int circ)  
*Get a structure of type [TwoDfold\\_vars](#) prefilled with current global settings.*
- void [destroy\\_TwoDfold\\_variables](#) ([TwoDfold\\_vars](#) \*our\_variables)  
*Destroy a [TwoDfold\\_vars](#) datastructure without memory loss.*
- [vrna\\_sol\\_TwoD\\_t](#) \* [TwoDfoldList](#) ([TwoDfold\\_vars](#) \*vars, int distance1, int distance2)  
*Compute MFE's and representative for distance partitioning.*
- char \* [TwoDfold\\_backtrack\\_f5](#) (unsigned int j, int k, int l, [TwoDfold\\_vars](#) \*vars)  
*Backtrack a minimum free energy structure from a 5' section of specified length.*

### 13.36.1 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 [10]

### 13.36.2 Data Structure Documentation

#### 13.36.2.1 struct vrna\_sol\_TwoD\_t

Solution element returned from [vrna\\_mfe\\_TwoD\(\)](#)

This element contains free energy and structure for the appropriate kappa (k), lambda (l) neighborhood. The data-structure contains two integer attributes 'k' and 'l' as well as an attribute 'en' of type float representing the free energy in kcal/mol and an attribute 's' of type char\* containing the secondary structure representative,

A value of [INF](#) in k denotes the end of a list

See also

[vrna\\_mfe\\_TwoD\(\)](#)

#### Data Fields

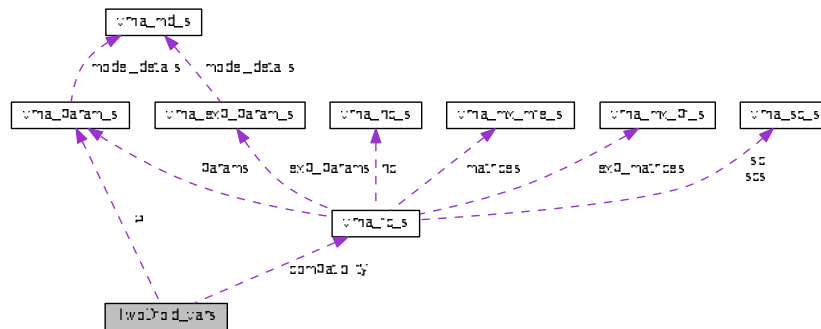
- int [k](#)  
*Distance to first reference.*
- int [l](#)  
*Distance to second reference.*
- float [en](#)  
*Free energy in kcal/mol.*
- char \* [s](#)  
*MFE representative structure in dot-bracket notation.*

#### 13.36.2.2 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 distancy 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].*

### 13.36.3 Typedef Documentation

#### 13.36.3.1 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 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\(\)](#)

#### 13.36.3.2 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!

### 13.36.4 Function Documentation

#### 13.36.4.1 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

13.36.4.2 `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 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

13.36.4.3 `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

13.36.4.4 `void destroy_TwoDfold_variables ( TwoDfold_vars * our_variables )`

```
#include <ViennaRNA/2Dfold.h>
```

Destroy a [TwoDfold\\_vars](#) datastructure without memory loss.

This function free's all allocated memory that depends on the datastructure given.

**Deprecated** Use the new API that relies on [vrna\\_fold\\_compound\\_t](#) and the corresponding functions [vrna\\_fold\\_compound\\_TwoD\(\)](#), [vrna\\_mfe\\_TwoD\(\)](#), and [vrna\\_fold\\_compound\\_free\(\)](#) instead!

**Parameters**

<i>our_variables</i>	A pointer to the datastructure to be destroyed
----------------------	--

13.36.4.5 `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)

13.36.4.6 `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 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!

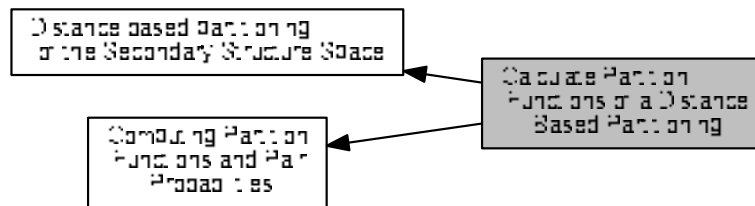
#### Parameters

<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

## 13.37 Calculate 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.

Collaboration diagram for Calculate Partition Functions of a Distance Based Partitioning:



### Files

- file [2Dpfold.h](#)

### 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.

#### 13.37.1 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.

### 13.37.2 Data Structure Documentation

#### 13.37.2.1 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*

### 13.37.3 Typedef Documentation

#### 13.37.3.1 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\(\)](#)

### 13.37.4 Function Documentation

#### 13.37.4.1 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)

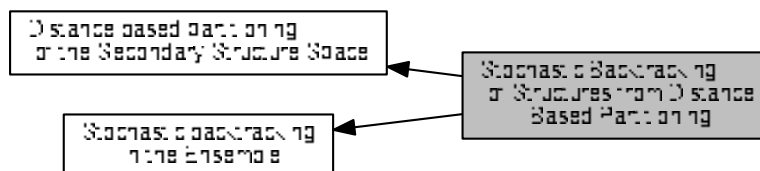
**Returns**

A list of partition funtions for the corresponding distance classes

## 13.38 Stochastic Backtracking of Structures from Distance Based Partitioning

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.*

### 13.38.1 Detailed Description

Contains functions related to stochastic backtracking from a specified distance class.

### 13.38.2 Function Documentation

13.38.2.1 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	<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

**13.38.2.2** `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	<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

## 13.39 Compute the Density of States

Collaboration diagram for Compute the Density of States:



### Variables

- int [density\\_of\\_states](#) [MAXDOS+1]

*The Density of States.*

### 13.39.1 Detailed Description

### 13.39.2 Variable Documentation

#### 13.39.2.1 int density\_of\_states[MAXDOS+1]

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

The Density of States.

This array contains the density of states for an RNA sequences after a call to [subopt\\_par\(\)](#), [subopt\(\)](#) or [subopt\\_circ\(\)](#).

#### Precondition

Call one of the functions [subopt\\_par\(\)](#), [subopt\(\)](#) or [subopt\\_circ\(\)](#) prior accessing the contents of this array

#### See also

[subopt\\_par\(\)](#), [subopt\(\)](#), [subopt\\_circ\(\)](#)



## 13.40 Hard Constraints

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

Collaboration diagram for Hard Constraints:



### Files

- file [constraints\\_hard.h](#)  
*Functions and data structures for handling of secondary structure hard constraints.*

### Data Structures

- struct [vrna\\_hc\\_s](#)  
*The hard constraints data structure. [More...](#)*
- struct [vrna\\_hc\\_up\\_s](#)  
*A single hard constraint for a single nucleotide. [More...](#)*

### Macros

- `#define VRNA_CONSTRAINT_DB 16384U`  
*Flag for [vrna\\_constraints\\_add\(\)](#) to indicate that constraint is passed in pseudo dot-bracket notation.*
- `#define VRNA_CONSTRAINT_DB_ENFORCE_BP 32768U`  
*Switch for dot-bracket structure constraint to enforce base pairs.*
- `#define VRNA_CONSTRAINT_DB_PIPE 65536U`  
*Flag that is used to indicate the pipe '|' sign in pseudo dot-bracket notation of hard constraints.*
- `#define VRNA_CONSTRAINT_DB_DOT 131072U`  
*dot '.' switch for structure constraints (no constraint at all)*
- `#define VRNA_CONSTRAINT_DB_X 262144U`  
*'x' switch for structure constraint (base must not pair)*
- `#define VRNA_CONSTRAINT_DB_RND_BRACK 1048576U`  
*round brackets '(',')' switch for structure constraint (base i pairs base j)*
- `#define VRNA_CONSTRAINT_DB_INTRAMOL 2097152U`  
*Flag that is used to indicate the character 'I' in pseudo dot-bracket notation of hard constraints.*
- `#define VRNA_CONSTRAINT_DB_INTERMOL 4194304U`  
*Flag that is used to indicate the character 'e' in pseudo dot-bracket notation of hard constraints.*
- `#define VRNA_CONSTRAINT_DB_GQUAD 8388608U`  
*'+' switch for structure constraint (base is involved in a quad)*

- `#define VRNA_CONSTRAINT_DB_DEFAULT`  
*Switch for dot-bracket structure constraint with default symbols.*
- `#define VRNA_CONSTRAINT_CONTEXT_EXT_LOOP (char)0x01`  
*Hard constraints flag, base pair in the exterior loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_HP_LOOP (char)0x02`  
*Hard constraints flag, base pair encloses hairpin loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_INT_LOOP (char)0x04`  
*Hard constraints flag, base pair encloses an interior loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_INT_LOOP_ENC (char)0x08`  
*Hard constraints flag, base pair encloses a multi branch loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_MB_LOOP (char)0x10`  
*Hard constraints flag, base pair is enclosed in an interior loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_MB_LOOP_ENC (char)0x20`  
*Hard constraints flag, base pair is enclosed in a multi branch loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_ALL_LOOPS`  
*Hard constraints flag, shortcut for all base pairs.*

## 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 char( vrna_callback_hc_evaluate) (int i, int j, int k, int l, 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, char option)`  
*Make a certain nucleotide unpaired.*
- `int vrna_hc_add_up_batch (vrna_fold_compound_t *vc, vrna_hc_up_t *constraints)`  
*Apply a list of hard constraints for single nucleotides.*
- `void vrna_hc_add_bp (vrna_fold_compound_t *vc, int i, int j, 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, 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.*

### 13.40.1 Detailed Description

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

## 13.40.2 Data Structure Documentation

### 13.40.2.1 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 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\\_EXT\\_LOOP\\_ENC](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_MB\\_LOOP](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_MB\\_LOOP\\_ENC](#)

#### Data Fields

- char \* [matrix](#)  
*Upper triangular matrix that encodes where a base pair or unpaired nucleotide is allowed.*
- 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.*

### 13.40.2.1.1 Field Documentation

#### 13.40.2.1.1.1 `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.

### 13.40.2.2 `struct vrna_hc_up_s`

A single hard constraint for a single nucleotide.

#### Data Fields

- int `position`  
*The sequence position (1-based)*
- char `options`  
*The hard constraint option.*

## 13.40.3 Macro Definition Documentation

### 13.40.3.1 `#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()`

### 13.40.3.2 `#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()`

**13.40.3.3 #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\(\)](#)

**13.40.3.4 #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\(\)](#)

**13.40.3.5 #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\(\)](#)

**13.40.3.6 #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\(\)](#)

**13.40.3.7 #define VRNA\_CONSTRAINT\_DB\_INTRAMOL 2097152U**

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

Flag that is used to indicate the character 'l' in pseudo dot-bracket notation of hard constraints.

Use this definition to indicate the usage of 'l' character (intramolecular pairs only)

See also

[vrna\\_hc\\_add\\_from\\_db\(\)](#), [vrna\\_constraints\\_add\(\)](#), [vrna\\_message\\_constraint\\_options\(\)](#), [vrna\\_message\\_constraint\\_options\\_all\(\)](#)

**13.40.3.8 #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\(\)](#)

**13.40.3.9 #define VRNA\_CONSTRAINT\_DB\_GQUAD 8388608U**

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

'+' switch for structure constraint (base is involved in a quad)

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.

## 13.40.3.10 #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\(\)](#)

## 13.40.4 Typedef Documentation

## 13.40.4.1 typedef char( vrna\_callback\_hc\_evaluate)(int i, int j, int k, int l, char d, void \*data)

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

Callback to evaluate whether or not a particular decomposition step is contributing to the solution space.

This is the prototype for callback functions used by the folding recursions to evaluate generic hard constraints. The first four parameters passed indicate the delimiting nucleotide positions of the decomposition, and the parameter `denotes` the decomposition step. The last parameter `data` is the auxiliary data structure associated to the hard constraints via `vrna_hc_add_data()`, or NULL if no auxiliary data was added.

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
Generated by Auxiliary data	

## Returns

Pseudo energy contribution in deka-kalories per mol

## 13.40.5 Function Documentation

## 13.40.5.1 void vrna\_hc\_init ( vrna\_fold\_compound\_t \* vc )

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

Initialize/Reset hard constraints to default values.

This function resets the hard constraints to their default values, i.e. all positions may be unpaired in all contexts, and base pairs are allowed in all contexts, if they resemble canonical pairs. Previously set hard constraints will be removed before initialization.

## See also

[vrna\\_hc\\_add\\_bp\(\)](#), [vrna\\_hc\\_add\\_bp\\_nonspecific\(\)](#), [vrna\\_hc\\_add\\_up\(\)](#)

## Parameters

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

## 13.40.5.2 void vrna\_hc\_add\_up ( vrna\_fold\_compound\_t \* vc, int i, 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\\_E↔XT\\_LOOP](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_HP\\_LOOP](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_INT\\_LOOP](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_MB\\_LOOP](#), [VRNA\\_CONSTRAINT\\_CONTEXT\\_ALL\\_LOOPS](#)

## Parameters

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

## 13.40.5.3 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

13.40.5.4 `void vrna_hc_add_bp ( vrna_fold_compound_t * vc, int i, int j, 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\\_INT\\_LOOP\\_ENC](#), [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

13.40.5.5 `void vrna_hc_add_bp_nonspecific ( vrna_fold_compound_t * vc, int i, int d, 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\\_INT\\_LOOP\\_ENC](#), [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

#### 13.40.5.6 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](#)

#### 13.40.5.7 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\\_ANG\\_BRACK](#), [VRNA\\_CONSTRAINT\\_DB\\_RND\\_BRACK](#), [VRNA\\_CONSTRAINT\\_DB\\_INTRAMOL](#), [VRNA\\_CONSTRAINT\\_DB\\_INTERMOL](#), [VRNA\\_CONSTRAINT\\_DB\\_GQUAD](#)

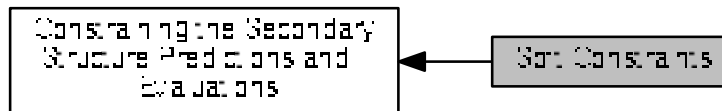
#### Parameters

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

## 13.41 Soft Constraints

Functions and data structures for secondary structure soft constraints.

Collaboration diagram for Soft Constraints:



### Files

- file [constraints\\_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, 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, 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, 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\\_add\\_bp](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const [FLT\\_OR\\_DBL](#) \*\*constraints, unsigned int options)  
*Add soft constraints for paired nucleotides.*
- void [vrna\\_sc\\_add\\_up](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const [FLT\\_OR\\_DBL](#) \*constraints, 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)*

### 13.41.1 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.

### 13.41.2 Data Structure Documentation

#### 13.41.2.1 struct vrna\_sc\_s

The soft constraints data structure.

#### Data Fields

- int \*\* `energy_up`  
*Energy contribution for stretches of unpaired nucleotides.*
- int \* `energy_bp`  
*Energy contribution for base pairs.*
- FLT\_OR\_DBL \*\* `exp_energy_up`  
*Boltzmann Factors of the energy contributions for unpaired sequence stretches.*
- FLT\_OR\_DBL \* `exp_energy_bp`  
*Boltzmann Factors of the energy contribution for base pairs.*
- 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.*

## 13.41.2.1.1 Field Documentation

13.41.2.1.1.1 `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\(\)](#)

13.41.2.1.1.2 `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\(\)](#)

13.41.2.1.1.3 `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\(\)](#)

## 13.41.3 Typedef Documentation

13.41.3.1 `typedef int( vrna_callback_sc_energy) (int i, int j, int k, int l, 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.

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

**13.41.3.2** `typedef FLT_OR_DBL( vrna_callback_sc_exp_energy)(int i, int j, int k, int l, 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.

**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

**13.41.3.3** `typedef vrna_basepair_t*( vrna_callback_sc_backtrack)(int i, int j, int k, int l, char d, void *data)`

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

Callback to retrieve auxiliary base pairs for soft constraint feature.

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

### 13.41.4 Function Documentation

#### 13.41.4.1 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\\_VC\\_TYPE\\_SINGLE](#) and [VRNA\\_VC\\_TYPE\\_ALIGNMENT](#)

See also

[vrna\\_sc\\_add\\_bp\(\)](#), [vrna\\_sc\\_add\\_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

vc	The <a href="#">vrna_fold_compound_t</a> where an empty soft constraint feature is to be added to
----	---

13.41.4.2 `void vrna_sc_add_bp ( vrna_fold_compound_t * vc, const FLT_OR_DBL ** constraints, unsigned int options )`

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

Add soft constraints for paired nucleotides.

#### Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> the soft constraints are associated with
<i>constraints</i>	A two-dimensional array of pseudo free energies in <i>kcal/mol</i>
<i>options</i>	The options flag indicating how/where to store the soft constraints

13.41.4.3 `void vrna_sc_add_up ( vrna_fold_compound_t * vc, const FLT_OR_DBL * constraints, unsigned int options )`

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

Add soft constraints for unpaired nucleotides.

#### Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> the soft constraints are associated with
<i>constraints</i>	A vector of pseudo free energies in <i>kcal/mol</i>
<i>options</i>	The options flag indicating how/where to store the soft constraints

13.41.4.4 `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\\_VC\\_TYPE\\_SINGLE](#) and [VRNA\\_VC\\_TYPE\\_ALIGNMENT](#)

#### Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> possibly containing soft constraints
-----------	---

13.41.4.5 `void vrna_sc_free ( vrna_sc_t * sc )`

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

Free memory occupied by a [vrna\\_sc\\_t](#) data structure.



## Parameters

sc	The data structure to free from memory
----	--

13.41.4.6 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

vc	The fold compound the generic soft constraint function should be bound to
data	A pointer to the data structure that holds required data for function 'f'
free_data	A pointer to a function that free's the memory occupied by (Maybe NULL)

13.41.4.7 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 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

vc	The fold compound the generic soft constraint function should be bound to
f	A pointer to the function that evaluates the generic soft constraint feature

13.41.4.8 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 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

13.41.4.9 `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 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()`

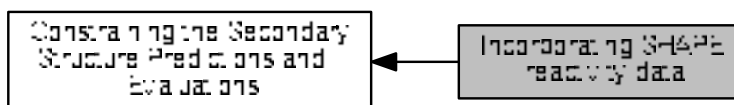
#### Parameters

<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

## 13.42 Incorporating SHAPE reactivity data

Incorporate SHAPE reactivity structure probing data into the folding recursions by means of soft constraints.

Collaboration diagram for Incorporating SHAPE reactivity data:



### Files

- file [constraints\\_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\\_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\\_to\\_pr](#) (const char \*shape\_conversion, double \*values, int length, double default\_value)

*Convert SHAPE reactivity values to probabilities for being unpaired.*

### 13.42.1 Detailed Description

Incorporate SHAPE reactivity structure probing data into the folding recursions by means of soft constraints.

### 13.42.2 Function Documentation

13.42.2.1 `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 [3].

[vrna\\_sc\\_remove\(\)](#), [vrna\\_sc\\_add\\_SHAPE\\_zarringhalam\(\)](#), [vrna\\_sc\\_minimize\\_perturbation\(\)](#)

#### 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

13.42.2.2 `int vrna_sc_add_SHAPE_deigan_ali ( vrna_fold_compound_t * vc, const char ** shape_files, const int * shape_file_association, double m, double b, unsigned int options )`

```
#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

13.42.2.3 `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 [18]

[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>options</i>	The options flag indicating how/where to store the soft constraints

## Returns

1 on successful extraction of the method, 0 on errors

13.42.2.4 `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\(\)](#)

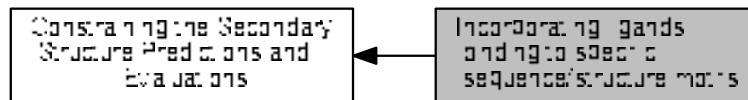
## Parameters

<i>shape_conversion</i>	String definining 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

## 13.43 Incorporating ligands binding to specific sequence/structure motifs

This module covers functions that enable the incorporation of ligand binding free energies to specific hairpin/interior loop motifs by means of generic soft constraints.

Collaboration diagram for Incorporating ligands binding to specific sequence/structure motifs:



### Files

- file [ligand.h](#)

*Functions for incorporation of ligands binding to hairpin and interior loop motifs.*

### 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.*

#### 13.43.1 Detailed Description

This module covers functions that enable the incorporation of ligand binding free energies to specific hairpin/interior loop motifs by means of generic soft constraints.

#### 13.43.2 Function Documentation

13.43.2.1 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/ligand.h>
```

Add soft constraints for hairpin or interior loop binding motif.

Here is an example that adds a theophylline binding motif. Free energy contribution is derived from  $k_d = 0.32 \mu\text{mol/l}$ , taken from Jenison et al. 1994

```
vrna_sc_add_hi_motif( vc,
    "GAUACCG&CCCUUGGCAGC",
    "...((((&)...))...)",
    -9.22, VRNA_OPTION_DEFAULT);
```

**Parameters**

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> the motif is applied to
<i>seq</i>	The sequence motif (may be interspaced by '&' character)
<i>structure</i>	The structure motif (may be interspaced by '&' character)
<i>energy</i>	The free energy of the motif (e.g. binding free energy)
<i>options</i>	Options

**Returns**

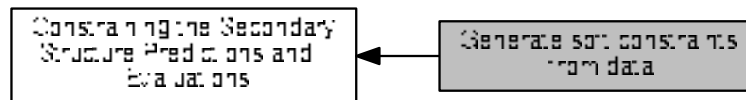
non-zero value if application of the motif using soft constraints was successful



## 13.44 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.

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.*

### 13.44.1 Detailed Description

Find a vector of perturbation energies that minimizes the discrepancies between predicted and observed pairing probabilities and the amount of necessary adjustments.

### 13.44.2 Macro Definition Documentation

#### 13.44.2.1 `#define VRNA_OBJECTIVE_FUNCTION_QUADRATIC 0`

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

Use the sum of squared aberrations as objective function.

$$F(\vec{\epsilon}) = \sum_{i=1}^n \frac{\epsilon_i^2}{\tau^2} + \sum_{i=1}^n \frac{(p_i(\vec{\epsilon}) - q_i)^2}{\sigma^2} \rightarrow \min$$

#### 13.44.2.2 `#define VRNA_OBJECTIVE_FUNCTION_ABSOLUTE 1`

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

Use the sum of absolute aberrations as objective function.

$$F(\vec{\epsilon}) = \sum_{i=1}^n \frac{|\epsilon_i|}{\tau^2} + \sum_{i=1}^n \frac{|p_i(\vec{\epsilon}) - q_i|}{\sigma^2} \rightarrow \min$$

#### 13.44.2.3 `#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

#### 13.44.2.4 `#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

13.44.2.5 `#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

13.44.2.6 `#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

13.44.2.7 `#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

## 13.44.3 Typedef Documentation

13.44.3.1 `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

## 13.44.4 Function Documentation

13.44.4.1 `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 )`

```
#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 making 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 [16].

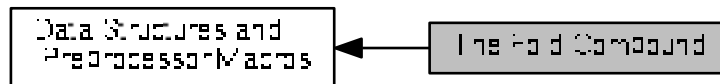
#### Parameters

<code>vc</code>	Pointer to a fold compound
<code>q_prob_unpaired</code>	Pointer to an array containing the probability to be unpaired for each nucleotide
<code>objective_function</code>	The type of objective function to be used (VRNA_OBJECTIVE_FUNCTION_QUADRATIC / VRNA_OBJECTIVE_FUNCTION_LINEAR)
<code>sigma_squared</code>	A factor used for weighting the objective function. More weight on this factor will lead to a solution close to the null vector.
<code>tau_squared</code>	A factor used for weighting the objective function. More weight on this factor will lead to a solution close to the data provided in <code>q_prob_unpaired</code> .
<code>algorithm</code>	The minimization algorithm (VRNA_MINIMIZER_*)
<code>sample_size</code>	The number of sampled sequences used for estimating the pairing probabilities. A value $\leq 0$ will lead to an exact evaluation.
<code>epsilon</code>	A pointer to an array used for storing the calculated vector of perturbation energies
<code>callback</code>	A pointer to a callback function used for reporting the current minimization progress

## 13.45 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.

Collaboration diagram for The Fold Compound:



### 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\\_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\\_EVAL\\_ONLY](#) 8U  
*Option flag to specify that neither MFE, nor PF DP matrices are required.*

### 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

## Functions

- [vrna\\_fold\\_compound\\_t \\* vrna\\_fold\\_compound](#) (const char \*sequence, [vrna\\_md\\_t](#) \*md\_p, unsigned int options)  
Retrieve a [vrna\\_fold\\_compound\\_t](#) data structure for single sequences and hybridizing sequences.
- [vrna\\_fold\\_compound\\_t \\* vrna\\_fold\\_compound\\_comparative](#) (const char \*\*sequences, [vrna\\_md\\_t](#) \*md\_p, unsigned int options)  
Retrieve a [vrna\\_fold\\_compound\\_t](#) data structure for sequence alignments.
- void [vrna\\_fold\\_compound\\_free](#) ([vrna\\_fold\\_compound\\_t](#) \*vc)  
Free memory occupied by a [vrna\\_fold\\_compound\\_t](#).
- void [vrna\\_fold\\_compound\\_add\\_auxdata](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, void \*data, [vrna\\_callback\\_free\\_t](#) auxdata \*f)  
Add auxiliary data to the [vrna\\_fold\\_compound\\_t](#).
- void [vrna\\_fold\\_compound\\_add\\_callback](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_callback\\_recursion\\_status](#) \*f)  
Add a recursion status callback to the [vrna\\_fold\\_compound\\_t](#).

### 13.45.1 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.

### 13.45.2 Data Structure Documentation

#### 13.45.2.1 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.

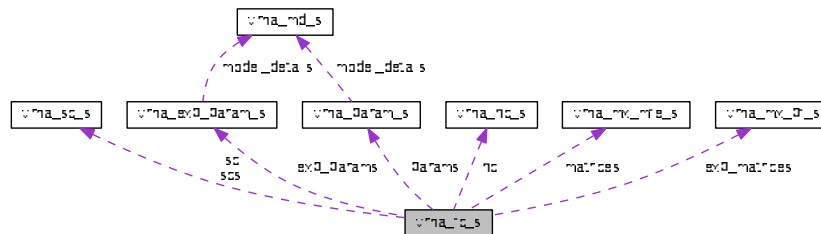
#### Warning

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_VC_TYPE_SINGLE, VRNA_VC_TYPE_ALIGNMENT
```

Collaboration diagram for vrna\_fc\_s:



### Data Fields

## Common data fields

- `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.*
- `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.*

### Data fields available for single/hybrid structure prediction

- char \* [sequence](#)  
*The input sequence string.*
- short \* [sequence\\_encoding](#)  
*Numerical encoding of the input sequence.*
- short \* **sequence\_encoding2**
- char \* [ptype](#)  
*Pair type array.*
- char \* [ptype\\_pf\\_compat](#)  
*ptype array indexed via iindx*
- [vrna\\_sc\\_t](#) \* **sc**  
*The soft constraints for usage in structure prediction and evaluation.*

### Data fields for consensus structure prediction

- char \*\* [sequences](#)  
*The aligned sequences.*
- unsigned int [n\\_seq](#)  
*The number of sequences in the alignment.*
- char \* [cons\\_seq](#)  
*The consensus sequence of the aligned sequences.*
- short \* [S\\_cons](#)  
*Numerical encoding of the consensus sequence.*
- short \*\* [S](#)  
*Numerical encoding of the sequences in the alignment.*
- short \*\* [S5](#)  
*S5[s][i] holds next base 5' of i in sequence s.*
- short \*\* [S3](#)  
*S3[s][i] holds next base 3' of i in sequence s.*
- char \*\* **Ss**
- unsigned short \*\* **a2s**
- int \* [pscore](#)  
*Precomputed array of pair types expressed as pairing scores.*
- short \* [pscore\\_pf\\_compat](#)  
*Precomputed array of pair types expressed as pairing scores indexed via iindx.*
- [vrna\\_sc\\_t](#) \*\* **scs**  
*A set of soft constraints (for each sequence in the alignment)*
- int **oldAliEn**

### Additional data fields for Distance Class Partitioning

*These data fields are typically populated with meaningful data only if used in the context of Distance Class Partitioning*

- unsigned int [maxD1](#)  
*Maximum allowed base pair distance to first reference.*
- unsigned int [maxD2](#)  
*Maximum allowed base pair distance to second reference.*
- short \* [reference\\_pt1](#)  
*A pairtable of the first reference structure.*
- short \* [reference\\_pt2](#)  
*A pairtable of the second reference structure.*
- unsigned int \* [referenceBPs1](#)  
*Matrix containing number of basepairs of reference structure1 in interval [i,j].*
- unsigned int \* [referenceBPs2](#)  
*Matrix containing number of basepairs of reference structure2 in interval [i,j].*
- unsigned int \* [bpdist](#)  
*Matrix containing base pair distance of reference structure 1 and 2 on interval [i,j].*
- unsigned int \* [mm1](#)



*Maximum matching matrix, reference struct 1 disallowed.*

- unsigned int \* [mm2](#)

*Maximum matching matrix, reference struct 2 disallowed.*

### Additional data fields for local folding

*These data fields are typically populated with meaningful data only if used in the context of local folding*

- int [window\\_size](#)  
*window size for local folding sliding window approach*
- char \*\* [ptype\\_local](#)  
*Pair type array (for local folding)*

#### 13.45.2.1.1 Field Documentation

##### 13.45.2.1.1.1 `vrna_fc_type_e vrna_fc_s::type`

The type of the [vrna\\_fold\\_compound\\_t](#).

Currently possible values are [VRNA\\_VC\\_TYPE\\_SINGLE](#), and [VRNA\\_VC\\_TYPE\\_ALIGNMENT](#)

#### 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.

##### 13.45.2.1.1.2 `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\(\)](#)

##### 13.45.2.1.1.3 `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](#)

##### 13.45.2.1.1.4 `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\(\)](#)

#### 13.45.2.1.1.5 `char* vrna_fc_s::sequence`

The input sequence string.

##### Warning

Only available if

```
type==VRNA_VC_TYPE_SINGLE
```

#### 13.45.2.1.1.6 `short* vrna_fc_s::sequence_encoding`

Numerical encoding of the input sequence.

##### See also

`vrna_sequence_encode()`

##### Warning

Only available if

```
type==VRNA_VC_TYPE_SINGLE
```

#### 13.45.2.1.1.7 `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_VC_TYPE_SINGLE
```

##### See also

[vrna\\_idx\\_col\\_wise\(\)](#), [vrna\\_ptypes\(\)](#)

#### 13.45.2.1.1.8 `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_VC_TYPE_SINGLE
```

**13.45.2.1.1.9 vrna\_sc\_t\* vrna\_fc\_s::sc**

The soft constraints for usage in structure prediction and evaluation.

**Warning**

Only available if

```
type==VRNA_VC_TYPE_SINGLE
```

**13.45.2.1.1.10 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_VC_TYPE_ALIGNMENT
```

**13.45.2.1.1.11 unsigned int vrna\_fc\_s::n\_seq**

The number of sequences in the alignment.

**Warning**

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

**13.45.2.1.1.12 char\* vrna\_fc\_s::cons\_seq**

The consensus sequence of the aligned sequences.

**Warning**

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

**13.45.2.1.1.13 short\* vrna\_fc\_s::S\_cons**

Numerical encoding of the consensus sequence.

**Warning**

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

**13.45.2.1.1.14 short\*\* vrna\_fc\_s::S**

Numerical encoding of the sequences in the alignment.

**Warning**

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

**13.45.2.1.1.15 short\*\* vrna\_fc\_s::S5**

S5[s][i] holds next base 5' of i in sequence s.

**Warning**

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

**13.45.2.1.1.16 short\*\* vrna\_fc\_s::S3**

S3[s][i] holds next base 3' of i in sequence s.

**Warning**

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

**13.45.2.1.1.17 int\* vrna\_fc\_s::pscore**

Precomputed array of pair types expressed as pairing scores.

**Warning**

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

**13.45.2.1.1.18 short\* vrna\_fc\_s::pscore\_pf\_compat**

Precomputed array of pair types expressed as pairing scores indexed via iidx.

**Deprecated** This attribute will vanish in the future!

**Warning**

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

#### 13.45.2.1.1.19 `vrna_sc_t** vrna_fc_s::scs`

A set of soft constraints (for each sequence in the alignment)

#### Warning

Only available if

```
type==VRNA_VC_TYPE_ALIGNMENT
```

### 13.45.3 Macro Definition Documentation

#### 13.45.3.1 `#define VRNA_STATUS_MFE_PRE (unsigned char)1`

```
#include <ViennaRNA/data_structures.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\(\)](#)

#### 13.45.3.2 `#define VRNA_STATUS_MFE_POST (unsigned char)2`

```
#include <ViennaRNA/data_structures.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\(\)](#)

#### 13.45.3.3 `#define VRNA_STATUS_PF_PRE (unsigned char)3`

```
#include <ViennaRNA/data_structures.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\(\)](#)

#### 13.45.3.4 `#define VRNA_STATUS_PF_POST (unsigned char)4`

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

Status message indicating that Partition function computations are finished.

See also

[vrna\\_fold\\_compound\\_t.stat\\_cb](#), [vrna\\_callback\\_recursion\\_status\(\)](#), [vrna\\_pf\(\)](#)

#### 13.45.3.5 `#define VRNA_OPTION_MFE 1U`

```
#include <ViennaRNA/data_structures.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](#)

#### 13.45.3.6 `#define VRNA_OPTION_PF 2U`

```
#include <ViennaRNA/data_structures.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](#)

#### 13.45.3.7 `#define VRNA_OPTION_EVAL_ONLY 8U`

```
#include <ViennaRNA/data_structures.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\(\)](#)

### 13.45.4 Typedef Documentation

#### 13.45.4.1 `typedef void( vrna_callback_free_auxdata)(void *data)`

```
#include <ViennaRNA/data_structures.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.

## Parameters

<i>data</i>	The data that needs to be free'd
-------------	----------------------------------

13.45.4.2 `typedef void( vrna_callback_recursion_status)( unsigned char status, void *data)`

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

Callback to perform specific user-defined actions before, or after recursive computations.

## See also

[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

## 13.45.5 Enumeration Type Documentation

13.45.5.1 `enum vrna_fc_type_e`

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

An enumerator that is used to specify the type of a [vrna\\_fold\\_compound\\_t](#).

## Enumerator

***VRNA\_VC\_TYPE\_SINGLE*** Type is suitable for single, and hybridizing sequences

***VRNA\_VC\_TYPE\_ALIGNMENT*** Type is suitable for sequence alignments (consensus structure prediction)

## 13.45.6 Function Documentation

13.45.6.1 `vrna_fold_compound_t* vrna_fold_compound( const char * sequence, vrna_md_t * md_p, unsigned int options )`

```
#include <ViennaRNA/data_structures.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. Use the macros:

- `VRNA_OPTION_MFE`
- `VRNA_OPTION_PF`
- `#VRNA_OPTION_WINDOW`
- `VRNA_OPTION_EVAL_ONLY`
- `#VRNA_OPTION_DEFAULT`

to specify the required type of computations that will be performed with the `vrna_fold_compound_t`.

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`, `VRNA_OPTION_MFE`, `VRNA_OPTION_PF`, `VRNA_OPTION_EVAL_ONLY`, `#VRNA_OPTION_WINDOW`

#### 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` that can be readily used for computations

**13.45.6.2** `vrna_fold_compound_t* vrna_fold_compound_comparative ( const char ** sequences, vrna_md_t * md_p, unsigned int options )`

```
#include <ViennaRNA/data_structures.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. Use the macros:

- `VRNA_OPTION_MFE`
- `VRNA_OPTION_PF`
- `VRNA_OPTION_EVAL_ONLY`
- `#VRNA_OPTION_DEFAULT`

to specify the required type of computations that will be performed with the `vrna_fold_compound_t`.

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` that can be readily used for computations

**13.45.6.3** `void vrna_fold_compound_free ( vrna_fold_compound_t * vc )`

```
#include <ViennaRNA/data_structures.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

vc	The <a href="#">vrna_fold_compound_t</a> that is to be erased from memory
----	---

13.45.6.4 void [vrna\\_fold\\_compound\\_add\\_auxdata](#) ( [vrna\\_fold\\_compound\\_t](#) \* vc, void \* data, [vrna\\_callback\\_free\\_auxdata](#) \* f )

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

Add auxiliary data to the [vrna\\_fold\\_compound\\_t](#).

This function allows 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

vc	The fold_compound the arbitrary data pointer should be associated with
data	A pointer to an arbitrary data structure
f	A pointer to function that free's memory occupied by the arbitrary data (May be NULL)

13.45.6.5 void [vrna\\_fold\\_compound\\_add\\_callback](#) ( [vrna\\_fold\\_compound\\_t](#) \* vc, [vrna\\_callback\\_recursion\\_status](#) \* f )

```
#include <ViennaRNA/data_structures.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 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](#)

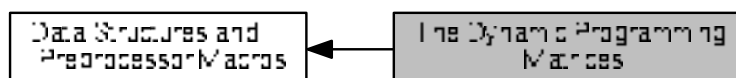
## Parameters

<i>vc</i>	The fold_compound the callback function should be attached to
<i>f</i>	The pointer to the recursion status callback function

## 13.46 The Dynamic Programming Matrices

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

### 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.

#### 13.46.1 Detailed Description

This module provides interfaces that deal with creation and destruction of dynamic programming matrices used within the RNAlib.

## 13.46.2 Data Structure Documentation

### 13.46.2.1 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
```

- int \* [c](#)  
*Energy array, given that i-j pair.*
- int \* [f5](#)  
*Energy of 5' end.*
- int \* [f3](#)  
*Energy of 3' end.*
- int \* [fc](#)  
*Energy from i to cutpoint (and vice versa if i > cut)*
- int \* [fML](#)  
*Multi-loop auxiliary energy array.*
- int \* [fM1](#)  
*Second ML array, only for unique multibranch loop decomposition.*
- int \* [fM2](#)  
*Energy for a multibranch loop region with exactly two stems, extending to 3' end.*
- int \* [ggg](#)  
*Energies of g-quadruplexes.*
- int [Fc](#)  
*Minimum Free Energy of entire circular RNA.*
- int [FcH](#)
- int [FcI](#)
- int [FcM](#)

##### Local Folding DP matrices using window approach

###### Note

*These data fields are available if*

```
vrna_mx_mfe_t.type == VRNA_MX_WINDOW
```

- int \*\* [c\\_local](#)  
*Energy array, given that i-j pair.*
- int \* [f3\\_local](#)  
*Energy of 5' end.*
- int \*\* [fML\\_local](#)  
*Multi-loop auxiliary energy array.*
- int \*\* [ggg\\_local](#)  
*Energies of g-quadruplexes.*

##### Distance Class DP matrices

*Note*

*These data fields are available if*

```
vrna_mx_mfe_t.type == VRNA_MX_2DFOLD
```

- int \*\*\* E\_F5
- int \*\* I\_min\_F5
- int \*\* I\_max\_F5
- int \* k\_min\_F5
- int \* k\_max\_F5
- int \*\*\* E\_F3
- int \*\* I\_min\_F3
- int \*\* I\_max\_F3
- int \* k\_min\_F3
- int \* k\_max\_F3
- int \*\*\* E\_C
- int \*\* I\_min\_C
- int \*\* I\_max\_C
- int \* k\_min\_C
- int \* k\_max\_C
- int \*\*\* E\_M
- int \*\* I\_min\_M
- int \*\* I\_max\_M
- int \* k\_min\_M
- int \* k\_max\_M
- int \*\*\* E\_M1
- int \*\* I\_min\_M1
- int \*\* I\_max\_M1
- int \* k\_min\_M1
- int \* k\_max\_M1
- int \*\*\* E\_M2
- int \*\* I\_min\_M2
- int \*\* I\_max\_M2
- int \* k\_min\_M2
- int \* k\_max\_M2
- int \*\* E\_Fc
- int \* I\_min\_Fc
- int \* I\_max\_Fc
- int k\_min\_Fc
- int k\_max\_Fc
- int \*\* E\_FcH
- int \* I\_min\_FcH
- int \* I\_max\_FcH
- int k\_min\_FcH
- int k\_max\_FcH
- int \*\* E\_Fcl
- int \* I\_min\_Fcl
- int \* I\_max\_Fcl
- int k\_min\_Fcl
- int k\_max\_Fcl
- int \*\* E\_FcM
- int \* I\_min\_FcM
- int \* I\_max\_FcM
- int k\_min\_FcM
- int k\_max\_FcM
- int \* E\_F5\_rem
- int \* E\_F3\_rem
- int \* E\_C\_rem
- int \* E\_M\_rem
- int \* E\_M1\_rem
- int \* E\_M2\_rem
- int E\_Fc\_rem
- int E\_FcH\_rem
- int E\_Fcl\_rem
- int E\_FcM\_rem

## 13.46.2.2 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
```

- [FLT\\_OR\\_DBL](#) \* **q**
- [FLT\\_OR\\_DBL](#) \* **qb**
- [FLT\\_OR\\_DBL](#) \* **qm**
- [FLT\\_OR\\_DBL](#) \* **qm1**
- [FLT\\_OR\\_DBL](#) \* **probs**
- [FLT\\_OR\\_DBL](#) \* **q1k**
- [FLT\\_OR\\_DBL](#) \* **qln**
- [FLT\\_OR\\_DBL](#) \* **G**
- [FLT\\_OR\\_DBL](#) **qo**
- [FLT\\_OR\\_DBL](#) \* **qm2**
- [FLT\\_OR\\_DBL](#) **qho**
- [FLT\\_OR\\_DBL](#) **qio**
- [FLT\\_OR\\_DBL](#) **qmo**

## Distance Class DP matrices

## Note

*These data fields are available if*

```
vrna_mx_pf_t.type == VRNA_MX_2DFOLD
```

- [FLT\\_OR\\_DBL](#) \*\*\* **Q**
- int \*\* **l\_min\_Q**
- int \*\* **l\_max\_Q**
- int \* **k\_min\_Q**
- int \* **k\_max\_Q**
- [FLT\\_OR\\_DBL](#) \*\*\* **Q\_B**
- int \*\* **l\_min\_Q\_B**
- int \*\* **l\_max\_Q\_B**
- int \* **k\_min\_Q\_B**
- int \* **k\_max\_Q\_B**
- [FLT\\_OR\\_DBL](#) \*\*\* **Q\_M**
- int \*\* **l\_min\_Q\_M**
- int \*\* **l\_max\_Q\_M**
- int \* **k\_min\_Q\_M**
- int \* **k\_max\_Q\_M**
- [FLT\\_OR\\_DBL](#) \*\*\* **Q\_M1**
- int \*\* **l\_min\_Q\_M1**
- int \*\* **l\_max\_Q\_M1**
- int \* **k\_min\_Q\_M1**

- `int * k_max_Q_M1`
- `FLT_OR_DBL *** Q_M2`
- `int ** l_min_Q_M2`
- `int ** l_max_Q_M2`
- `int * k_min_Q_M2`
- `int * k_max_Q_M2`
- `FLT_OR_DBL ** Q_c`
- `int * l_min_Q_c`
- `int * l_max_Q_c`
- `int k_min_Q_c`
- `int k_max_Q_c`
- `FLT_OR_DBL ** Q_cH`
- `int * l_min_Q_cH`
- `int * l_max_Q_cH`
- `int k_min_Q_cH`
- `int k_max_Q_cH`
- `FLT_OR_DBL ** Q_cl`
- `int * l_min_Q_cl`
- `int * l_max_Q_cl`
- `int k_min_Q_cl`
- `int k_max_Q_cl`
- `FLT_OR_DBL ** Q_cM`
- `int * l_min_Q_cM`
- `int * l_max_Q_cM`
- `int k_min_Q_cM`
- `int k_max_Q_cM`
- `FLT_OR_DBL * Q_rem`
- `FLT_OR_DBL * Q_B_rem`
- `FLT_OR_DBL * Q_M_rem`
- `FLT_OR_DBL * Q_M1_rem`
- `FLT_OR_DBL * Q_M2_rem`
- `FLT_OR_DBL Q_c_rem`
- `FLT_OR_DBL Q_cH_rem`
- `FLT_OR_DBL Q_cl_rem`
- `FLT_OR_DBL Q_cM_rem`

### 13.46.3 Enumeration Type Documentation

#### 13.46.3.1 `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

[vrna\\_mfe\\_window\(\)](#), [vrna\\_mfe\\_window\\_zscore\(\)](#), [pfl\\_fold\(\)](#)

**`VRNA_MX_2DFOLD`** DP matrices suitable for distance class partitioned structure prediction.

See also

[vrna\\_mfe\\_TwoD\(\)](#), [vrna\\_pf\\_TwoD\(\)](#)



### 13.46.4 Function Documentation

13.46.4.1 `int vrna_mx_add ( vrna_fold_compound_t * vc, vrna_mx_type_e type, unsigned int options )`

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

Add Dynamic Programming (DP) matrices (allocate memory)

This function adds DP matrices of a specific type to the provided `vrna_fold_compound_t`, such that successive DP recursion can be applied. The function caller has to specify which type of DP matrix is requested, see `vrna_mx_type_e`, and what kind of recursive algorithm will be applied later on, using the parameters `type`, and `options`, respectively. For the latter, Minimum free energy (MFE), and Partition function (PF) computations are distinguished. A third option that may be passed is `#VRNA_OPTION_HYBRID`, indicating that auxiliary DP arrays are required for RNA-RNA interaction prediction.

#### Note

Usually, there is no need to call this function, since the constructors of `vrna_fold_compound_t` are handling all the DP matrix memory allocation.

#### See also

`vrna_mx_mfe_add()`, `vrna_mx_pf_add()`, `vrna_fold_compound()`, `vrna_fold_compound_comparative()`, `vrna_fold_compound_free()`, `vrna_mx_pf_free()`, `vrna_mx_mfe_free()`, `vrna_mx_type_e`, `VRNA_OPTION_MFE`, `VRNA_OPTION_PF`, `#VRNA_OPTION_HYBRID`, `VRNA_OPTION_EVAL_ONLY`

#### Parameters

<code>vc</code>	The <code>vrna_fold_compound_t</code> that holds pointers to the DP matrices
<code>type</code>	The type of DP matrices requested
<code>options</code>	Option flags that specify the kind of DP matrices, such as MFE or PF arrays, and auxiliary requirements

#### Returns

1 if DP matrices were properly allocated and attached, 0 otherwise

13.46.4.2 `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

<code>vc</code>	The <code>vrna_fold_compound_t</code> storing the MFE DP matrices that are to be erased from memory
-----------------	---

13.46.4.3 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\(\)](#)

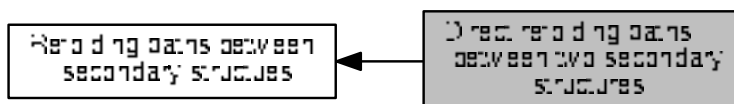
Parameters

vc	The <a href="#">vrna_fold_compound_t</a> storing the PF DP matrices that are to be erased from memory
----	---

## 13.47 Direct refolding paths between two secondary structures

Implementation of heuristics to explore optimal (re-)folding paths between two secondary structures.

Collaboration diagram for Direct refolding paths between two secondary structures:



### Data Structures

- struct [vrna\\_path\\_s](#)  
An element of a refolding path list. [More...](#)

### Typedefs

- typedef struct [vrna\\_path\\_s](#) [vrna\\_path\\_t](#)  
Typename for the refolding path data structure [vrna\\_path\\_s](#).
- typedef struct [vrna\\_path\\_s](#) [path\\_t](#)  
Old typename of [vrna\\_path\\_s](#).

### Functions

- int [vrna\\_path\\_findpath\\_saddle](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*struc1, const char \*struc2, int max)  
Find energy of a saddle point between 2 structures (search only direct path)
- [vrna\\_path\\_t](#) \* [vrna\\_path\\_findpath](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*s1, const char \*s2, int maxkeep)  
Find refolding path between 2 structures (search only direct path)
- int [find\\_saddle](#) (const char \*seq, const char \*struc1, const char \*struc2, int max)  
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 maxkeep)  
Find refolding path between 2 structures (search only direct path)

#### 13.47.1 Detailed Description

Implementation of heuristics to explore optimal (re-)folding paths between two secondary structures.

## 13.47.2 Data Structure Documentation

### 13.47.2.1 struct vrna\_path\_s

An element of a refolding path list.

See also

[vrna\\_path\\_findpath\(\)](#)

#### Data Fields

- double [en](#)  
*Free energy of current structure.*
- char \* [s](#)  
*Secondary structure in dot-bracket notation.*

## 13.47.3 Typedef Documentation

### 13.47.3.1 typedef struct vrna\_path\_s path\_t

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

Old typename of [vrna\\_path\\_s](#).

**Deprecated** Use [vrna\\_path\\_t](#) instead!

## 13.47.4 Function Documentation

### 13.47.4.1 int vrna\_path\_findpath\_saddle ( vrna\_fold\_compound\_t \* vc, const char \* struc1, const char \* struc2, int max )

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

Find energy of a saddle point between 2 structures (search only direct path)

This function uses an implementation of the *findpath* algorithm [4] for near-optimal direct refolding path prediction.

Model details, and energy parameters are used as provided via the parameter 'vc'. The [vrna\\_fold\\_compound\\_t](#) does not require memory for any DP matrices, but requires all most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_EVAL_ONLY);
```

See also

[vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_t](#), [vrna\\_path\\_findpath\(\)](#)

## Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> with precomputed sequence encoding and model details
<i>struc1</i>	The start structure in dot-bracket notation
<i>struc2</i>	The target structure in dot-bracket notation
<i>max</i>	A number specifying how many structures are being kept at each step during the search

## Returns

The saddle energy in 10cal/mol

13.47.4.2 `vrna_path_t* vrna_path_findpath ( vrna_fold_compound_t * vc, const char * s1, const char * s2, int maxkeep )`

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

Find refolding path between 2 structures (search only direct path)

This function uses an implementation of the *findpath* algorithm [4] for near-optimal direct refolding path prediction.

Model details, and energy parameters are used as provided via the parameter 'vc'. The [vrna\\_fold\\_compound\\_t](#) does not require memory for any DP matrices, but requires all most basic init values as one would get from a call like this:

```
vc = vrna_fold_compound(sequence, NULL, VRNA_OPTION_EVAL_ONLY);
```

## See also

[vrna\\_fold\\_compound\(\)](#), [vrna\\_fold\\_compound\\_t](#), [vrna\\_path\\_findpath\\_saddle\(\)](#)

## Parameters

<i>vc</i>	The <a href="#">vrna_fold_compound_t</a> with precomputed sequence encoding and model details
<i>struc1</i>	The start structure in dot-bracket notation
<i>struc2</i>	The target structure in dot-bracket notation
<i>max</i>	A number specifying how many structures are being kept at each step during the search

## Returns

The saddle energy in 10cal/mol

13.47.4.3 `int find_saddle ( const char * seq, const char * struc1, const char * struc2, int max )`

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

Find energy of a saddle point between 2 structures (search only direct path)

## Parameters

<i>seq</i>	RNA sequence
<i>struc1</i>	A pointer to the character array where the first secondary structure in dot-bracket notation will be written to
<i>struc2</i>	A pointer to the character array where the second secondary structure in dot-bracket notation will be written to
<i>max</i>	integer how many strutures are being kept during the search

## Returns

the saddle energy in 10cal/mol

13.47.4.4 `void free_path ( vrna_path_t * path )`

#include <ViennaRNA/findpath.h>

Free memory allocated by `get_path()` function.

## Parameters

<i>path</i>	pointer to memory to be freed
-------------	-------------------------------

13.47.4.5 `vrna_path_t* get_path ( const char * seq, const char * s1, const char * s2, int maxkeep )`

#include <ViennaRNA/findpath.h>

Find refolding path between 2 structures (search only direct path)

## Parameters

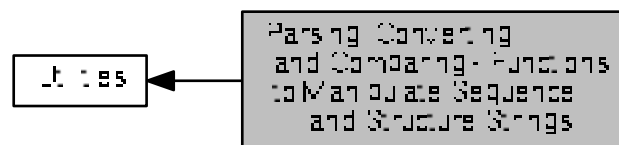
<i>seq</i>	RNA sequence
<i>s1</i>	A pointer to the character array where the first secondary structure in dot-bracket notation will be written to
<i>s2</i>	A pointer to the character array where the second secondary structure in dot-bracket notation will be written to
<i>maxkeep</i>	integer how many strutures are being kept during the search

## Returns

direct refolding path between two structures

## 13.48 Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings

Collaboration diagram for Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings:



### Files

- file [string\\_utils.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_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_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.*

### 13.48.1 Detailed Description

### 13.48.2 Macro Definition Documentation

#### 13.48.2.1 `#define FILENAME_MAX_LENGTH 80`

```
#include <ViennaRNA/string_utils.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.

#### 13.48.2.2 `#define FILENAME_ID_LENGTH 42`

```
#include <ViennaRNA/string_utils.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

### 13.48.3 Function Documentation

#### 13.48.3.1 `char* vrna_random_string ( int l, const char symbols[] )`

```
#include <ViennaRNA/string_utils.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

#### 13.48.3.2 `int vrna_hamming_distance ( const char * s1, const char * s2 )`

```
#include <ViennaRNA/string_utils.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*

**13.48.3.3** `int vrna_hamming_distance_bound ( const char * s1, const char * s2, int n )`

```
#include <ViennaRNA/string_utils.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

**Returns**

The hamming distance between *s1* and *s2*

**13.48.3.4** `void vrna_seq_toRNA ( char * sequence )`

```
#include <ViennaRNA/string_utils.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
-----------------	------------------------------

**13.48.3.5** `void vrna_seq_toupper ( char * sequence )`

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

Convert an input sequence to uppercase.

**Parameters**

<i>sequence</i>	The sequence to be converted
-----------------	------------------------------

**13.48.3.6** `char* vrna_cut_point_insert ( const char * string, int cp )`

```
#include <ViennaRNA/string_utils.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

**13.48.3.7** `char* vrna_cut_point_remove ( const char * string, int * cp )`

```
#include <ViennaRNA/string_utils.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**

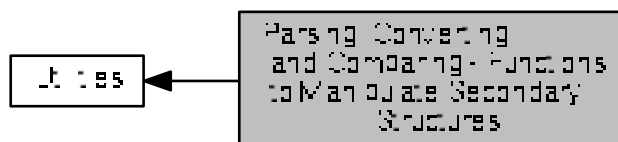
<i>string</i>	The original string
<i>cp</i>	The cut-point position

**Returns**

A copy of the input string with the '&' being sliced out

## 13.49 Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures

Collaboration diagram for Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures:



### Files

- file [RNAstruct.h](#)  
*Parsing and Coarse Graining of Structures.*
- file [structure\\_utils.h](#)  
*Various utility- and helper-functions for secondary structure parsing, converting, etc.*

### Data Structures

- struct [vrna\\_hx\\_s](#)

### 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 \* [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\(\)](#)*

  - short \* [vrna\\_ptable](#) (const char \*structure)
- Create a pair table of a secondary structure.*

  - 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)*

  - int \* [vrna\\_loopidx\\_from\\_ptable](#) (const short \*pt)
- Get a loop index representation of a structure.*

  - char \* [vrna\\_db\\_from\\_ptable](#) (short \*pt)
- Convert a pair table into dot-parenthesis notation.*

  - 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.*

  - [vrna\\_plist\\_t](#) \* [vrna\\_plist](#) (const char \*struc, float pr)
- Create a [vrna\\_plist\\_t](#) from a dot-bracket string.*

  - char \* [vrna\\_db\\_from\\_plist](#) ([vrna\\_plist\\_t](#) \*pairs, unsigned int n)
- Convert a list of base pairs into dot-bracket notation.*

  - void [assign\\_plist\\_from\\_db](#) ([vrna\\_plist\\_t](#) \*\*pl, const char \*struc, float pr)
- Create a [vrna\\_plist\\_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)
- Create a pair table of a secondary structure (snoop version)*

  - short \* [make\\_pair\\_table\\_snoop](#) (const char \*structure)
- Compute the "base pair" distance between two secondary structures s1 and s2.*

  - int [bp\\_distance](#) (const char \*str1, const char \*str2)

- 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](#) [STRUC]  
*contains a list of all loop sizes. `loop_size[0]` contains the number of external bases.*
- int [helix\\_size](#) [STRUC]  
*contains a list of all stack sizes.*
- int [loop\\_degree](#) [STRUC]  
*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.*

### 13.49.1 Detailed Description

### 13.49.2 Data Structure Documentation

#### 13.49.2.1 struct `vrna_hx_s`

### 13.49.3 Function Documentation

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

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

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

#### Parameters

<code>structure</code>	
------------------------	--

**Returns****13.49.3.2** `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.

**Parameters**

<i>structure</i>	
------------------	--

**Returns****13.49.3.3** `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.

**Parameters**

<i>structure</i>	
------------------	--

**Returns****13.49.3.4** `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**

**13.49.3.5** 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****13.49.3.6** 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****13.49.3.7** 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****13.49.3.8** char\* unweight ( const char \* *wcoarse* )

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

Strip weights from any weighted tree.

## Parameters

<i>wcoarse</i>	
----------------	--

## Returns

13.49.3.9 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>	
--------------	--

13.49.3.10 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

13.49.3.11 char\* vrna\_db\_pack ( const char \* *struc* )

```
#include <ViennaRNA/structure_utils.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>struct</i>	The secondary structure in dot-bracket notation
---------------	---

## Returns

The binary encoded structure

**13.49.3.12** `char* vrna_db_unpack ( const char * packed )`

```
#include <ViennaRNA/structure_utils.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

**13.49.3.13** `short* vrna_ptable ( const char * structure )`

```
#include <ViennaRNA/structure_utils.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.

## Parameters

<i>structure</i>	The secondary structure in dot-bracket notation
------------------	---

## Returns

A pointer to the created `pair_table`

### 13.49.3.14 `short* vrna_pt_pk_get ( const char * structure )`

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

Create a pair table of a secondary structure (pseudo-knot version)

Returns a newly allocated table, such that `table[i]=j` if (i,j) pair or 0 if i is unpaired, `table[0]` contains the length of the structure.

In contrast to `vrna_ptable()` this function also recognizes the base pairs denoted by '[' and ']' brackets.

#### Parameters

<i>structure</i>	The secondary structure in (extended) dot-bracket notation
------------------	--

#### Returns

A pointer to the created `pair_table`

### 13.49.3.15 `short* vrna_ptable_copy ( const short * pt )`

```
#include <ViennaRNA/structure_utils.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`'

### 13.49.3.16 `short* vrna_pt_snoop_get ( const char * structure )`

```
#include <ViennaRNA/structure_utils.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.

### 13.49.3.17 `char* vrna_db_from_ptable ( short * pt )`

```
#include <ViennaRNA/structure_utils.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

**13.49.3.18** `int vrna_bp_distance ( const char * str1, const char * str2 )`

```
#include <ViennaRNA/structure_utils.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*

**13.49.3.19** `unsigned int* vrna_refBPcnt_matrix ( const short * reference_pt, unsigned int turn )`

```
#include <ViennaRNA/structure_utils.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*!!!

**13.49.3.20** `unsigned int* vrna_refBPdist_matrix ( const short * pt1, const short * pt2, unsigned int turn )`

```
#include <ViennaRNA/structure_utils.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*!!!

**13.49.3.21** `char* vrna_db_from_bp_stack ( vrna_bp_stack_t * bp, unsigned int length )`

```
#include <ViennaRNA/structure_utils.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

**13.49.3.22** `vrna_plist_t* vrna_plist ( const char * struc, float pr )`

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

Create a [vrna\\_plist\\_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

**13.49.3.23** `char* vrna_db_from_plist ( vrna_plist_t * pairs, unsigned int n )`

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

Convert a list of base pairs into dot-bracket notation.

**See also**

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

**Parameters**

<i>pairs</i>	A <a href="#">vrna_plist_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

**13.49.3.24** `void assign_plist_from_db ( vrna_plist_t ** pl, const char * struc, float pr )`

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

Create a `vrna_plist_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_plist_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

**13.49.3.25** `char* pack_structure ( const char * struc )`

```
#include <ViennaRNA/structure_utils.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

**13.49.3.26** `char* unpack_structure ( const char * packed )`

```
#include <ViennaRNA/structure_utils.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

**13.49.3.27** `short* make_pair_table ( const char * structure )`

```
#include <ViennaRNA/structure_utils.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

**13.49.3.28** `short* copy_pair_table ( const short * pt )`

```
#include <ViennaRNA/structure_utils.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'

**13.49.3.29** `short* alimake_pair_table ( const char * structure )`

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

Pair table for snoop align

**Deprecated** Use `vrna_pt_ali_get()` instead!

**13.49.3.30** `short* make_pair_table_snoop ( const char * structure )`

```
#include <ViennaRNA/structure_utils.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!

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

```
#include <ViennaRNA/structure_utils.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

**13.49.3.32** `unsigned int* make_referenceBP_array ( short * reference_pt, unsigned int turn )`

```
#include <ViennaRNA/structure_utils.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

13.49.3.33 `unsigned int* compute_BPdifferences ( short * pt1, short * pt2, unsigned int turn )`

`#include <ViennaRNA/structure_utils.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

13.49.3.34 `void parenthesis_structure ( char * structure, vrna_bp_stack_t * bp, int length )`

`#include <ViennaRNA/structure_utils.h>`

Create a dot-bracket/parenthesis structure from backtracking stack.

**Deprecated** use `vrna_parenthesis_structure()` instead

#### Note

This function is threadsafe

13.49.3.35 `void parenthesis_zuker ( char * structure, vrna_bp_stack_t * bp, int length )`

`#include <ViennaRNA/structure_utils.h>`

Create a dot-bracket/parenthesis structure from backtracking stack obtained by zuker suboptimal calculation in `cofold.c`.

**Deprecated** use `vrna_parenthesis_zuker` instead

#### Note

This function is threadsafe

13.49.3.36 `void bppm_to_structure ( char * structure, FLT_OR_DBL * pr, unsigned int length )`

`#include <ViennaRNA/structure_utils.h>`

Create a dot-bracket like structure string from base pair probability matrix.

**Deprecated** Use `vrna_db_from_probs()` instead!

13.49.3.37 `char bppm_symbol ( const float * x )`

`#include <ViennaRNA/structure_utils.h>`

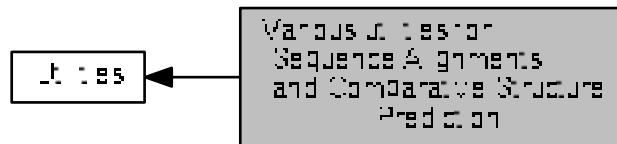
Get a pseudo dot bracket notation for a given probability information.

**Deprecated** Use `vrna_bpp_symbol()` instead!



## 13.50 Various utilities for Sequence Alignments, and Comparative Structure Prediction

Collaboration diagram for Various utilities for Sequence Alignments, and Comparative Structure Prediction:



### Files

- file [aln\\_util.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...](#)*

### 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

- [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.*

#### 13.50.1 Detailed Description

#### 13.50.2 Data Structure Documentation

##### 13.50.2.1 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*

## 13.50.3 Typedef Documentation

## 13.50.3.1 typedef struct vrna\_pinfo\_s pair\_info

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

Old typename of `vrna_pinfo_s`.

**Deprecated** Use `vrna_pinfo_t` instead!

## 13.50.4 Function Documentation

13.50.4.1 `vrna_pinfo_t* vrna_aln_pinfo ( vrna_fold_compound_t * vc, const char * structure, double threshold )`

```
#include <ViennaRNA/aln_util.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

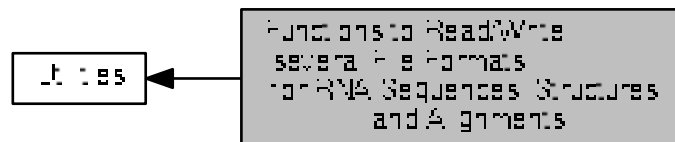
<code>vc</code>	The <code>vrna_fold_compound_t</code> of type <code>VRNA_VC_TYPE_ALIGNMENT</code> with precomputed partition function matrices
<code>structure</code>	An optional structure in dot-bracket notation (Maybe NULL)
<code>threshold</code>	Do not include results with pair probabilities below threshold

**Returns**

The `vrna_pinfo_t` array

## 13.51 Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments

Collaboration diagram for Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments:



### Files

- file [file\\_formats.h](#)  
*Functions dealing with file formats for RNA sequences, structures, and alignments.*
- file [ribo.h](#)  
*Parse RiboSum Scoring Matrices for Covariance Scoring of Alignments.*

### Macros

- `#define VRNA_OPTION_MULTILINE 32U`  
*Tell a function that an input is assumed to span several lines.*
- `#define VRNA_CONSTRAINT_MULTILINE 32U`  
*parse multiline constraint*

### Functions

- void [vrna\\_file\\_helixlist](#) (const char \*seq, const char \*db, float energy, FILE \*file)  
*Print a secondary structure as helix list.*
- void [vrna\\_file\\_connect](#) (const char \*seq, const char \*db, float energy, const char \*identifier, FILE \*file)  
*Print a secondary structure as connect table.*
- void [vrna\\_file\\_bpseq](#) (const char \*seq, const char \*db, FILE \*file)  
*Print a secondary structure in bpseq format.*
- void [vrna\\_file\\_json](#) (const char \*seq, const char \*db, double energy, const char \*identifier, FILE \*file)  
*Print a secondary structure in jsonformat.*
- unsigned int [vrna\\_file\\_fasta\\_read\\_record](#) (char \*\*header, char \*\*sequence, char \*\*\*rest, FILE \*file, unsigned int options)  
*Get a (fasta) data set from a file or stdin.*
- char \* [vrna\\_extract\\_record\\_rest\\_structure](#) (const char \*\*lines, unsigned int length, unsigned int option)  
*Extract a dot-bracket structure string from (multiline)character array.*
- int [vrna\\_file\\_SHAPE\\_read](#) (const char \*file\_name, int length, double default\_value, char \*sequence, double \*values)

*Read data from a given SHAPE reactivity input file.*

- [vrna\\_plist\\_t](#) \* [vrna\\_file\\_constraints\\_read](#) (const char \*filename, unsigned int length, unsigned int options)

*Read constraints from an 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.*

- float \*\* [readribosum](#) (char \*name)

*Read a RiboSum or other user-defined Scoring Matrix and Store into global Memory.*

### 13.51.1 Detailed Description

### 13.51.2 Macro Definition Documentation

#### 13.51.2.1 #define VRNA\_OPTION\_MULTILINE 32U

```
#include <ViennaRNA/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\(\)](#)

#### 13.51.2.2 #define VRNA\_CONSTRAINT\_MULTILINE 32U

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

parse multiline constraint

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

### 13.51.3 Function Documentation

#### 13.51.3.1 void vrna\_file\_helixlist ( const char \* seq, const char \* db, float energy, FILE \* file )

```
#include <ViennaRNA/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>file</i>	The file handle used to print to (print defaults to 'stdout' if(file == NULL) )

**13.51.3.2** void vrna\_file\_connect ( const char \* *seq*, const char \* *db*, float *energy*, const char \* *identifier*, FILE \* *file* )

```
#include <ViennaRNA/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) )

**13.51.3.3** void vrna\_file\_bpseq ( const char \* *seq*, const char \* *db*, FILE \* *file* )

```
#include <ViennaRNA/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) )

**13.51.3.4** void vrna\_file\_json ( const char \* *seq*, const char \* *db*, double *energy*, const char \* *identifier*, FILE \* *file* )

```
#include <ViennaRNA/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) )

**13.51.3.5** `unsigned int vrna_file_fasta_read_record ( char ** header, char ** sequence, char *** rest, FILE * file, unsigned int options )`

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

Get a (fasta) data set from a file or stdin.

This function may be used to obtain complete datasets from a filehandle or stdin. A dataset is always defined to contain at least a sequence. If data starts with a fasta header, i.e. a line like

```
>some header info
```

then `vrna_file_fasta_read_record()` will assume that the sequence that follows the header may span over several lines. To disable this behavior and to assign a single line to the argument 'sequence' one can pass `VRNA_INPUT_NO_SPAN` in the 'options' argument. If no fasta header is read in the beginning of a data block, a sequence must not span over multiple lines!

Unless the options `VRNA_INPUT_NOSKIP_COMMENTS` or `VRNA_INPUT_NOSKIP_BLANK_LINES` are passed, a sequence may be interrupted by lines starting with a comment character or empty lines.

A sequence is regarded as completely read if it was either assumed to not span over multiple lines, a secondary structure or structure constraint follows the sequence on the next line, or a new header marks the beginning of a new sequence...

All lines following the sequence (this includes comments) that do not initiate a new dataset according to the above definition are available through the line-array 'rest'. Here one can usually find the structure constraint or other information belonging to the current dataset. Filling of 'rest' may be prevented by passing `VRNA_INPUT_NO_REST` to the options argument.

## Note

This function will exit any program with an error message if no sequence could be read!

This function is NOT threadsafe! It uses a global variable to store information about the next data block.

The main purpose of this function is to be able to easily parse blocks of data in the header of a loop where all calculations for the appropriate data is done inside the loop. The loop may be then left on certain return values, e.g.:

```
00001 char *id, *seq, **rest;
00002 int i;
00003 id = seq = NULL;
00004 rest = NULL;
00005 while(! (vrna_file_fasta_read_record(&id, &seq, &rest, NULL, 0) & (VRNA_INPUT_ERROR |
VRNA_INPUT_QUIT))) {
00006     if(id) printf("%s\n", id);
00007     printf("%s\n", seq);
00008     if(rest)
00009         for(i=0; rest[i]; i++) {
00010             printf("%s\n", rest[i]);
00011             free(rest[i]);
00012         }
00013     free(rest);
00014     free(seq);
00015     free(id);
00016 }
```

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

**13.51.3.6** `char* vrna_extract_record_rest_structure ( const char ** lines, unsigned int length, unsigned int option )`

```
#include <ViennaRNA/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

13.51.3.7 `int vrna_file_SHAPE_read ( const char * file_name, int length, double default_value, char * sequence, double * values )`

```
#include <ViennaRNA/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

13.51.3.8 `vrna_plist_t* vrna_file_constraints_read ( const char * filename, unsigned int length, unsigned int options )`

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

Read constraints from an input file.

This function reads constraint definitions from a file and converts them into an array of `vrna_plist_t` data structures. The data fields of each individual returned plist entry may adopt the following configurations:

- `plist.i == plist.j` → single nucleotide constraint
- `plist.i != plist.j` → base pair constraint
- `plist.i == 0` → End of list

13.51.3.9 `void vrna_extract_record_rest_constraint ( char ** cstruct, const char ** lines, unsigned int option )`

```
#include <ViennaRNA/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_X`, `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

13.51.3.10 unsigned int read\_record ( char \*\* *header*, char \*\* *sequence*, char \*\*\* *rest*, unsigned int *options* )

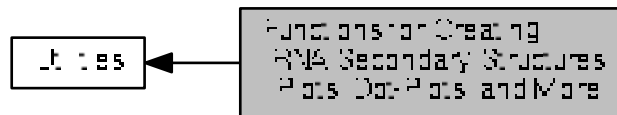
```
#include <ViennaRNA/file_formats.h>
```

Get a data record from stdin.

**Deprecated** This function is deprecated! Use [vrna\\_file\\_fasta\\_read\\_record\(\)](#) as a replacment.

## 13.52 Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More

Collaboration diagram for Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More:



### Files

- file [naview.h](#)
- file [plot\\_aln.h](#)  
*Various functions for plotting Sequence / Structure Alignments.*
- file [plot\\_layouts.h](#)  
*Secondary structure plot layout algorithms.*
- file [plot\\_structure.h](#)  
*Various functions for plotting RNA secondary structures.*
- file [PS\\_dot.h](#)  
*Various functions for plotting RNA secondary structures, dot-plots and other visualizations.*

### Data Structures

- struct [COORDINATE](#)  
*this is a workaround for the SWIG Perl Wrapper RNA plot function that returns an array of type [COORDINATE](#) More...*

### Macros

- `#define` [VRNA\\_PLOT\\_TYPE\\_SIMPLE](#) 0  
*Definition of Plot type simple*
- `#define` [VRNA\\_PLOT\\_TYPE\\_NAVIEW](#) 1  
*Definition of Plot type Naview*
- `#define` [VRNA\\_PLOT\\_TYPE\\_CIRCULAR](#) 2  
*Definition of Plot type Circular*

## Functions

- int [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[])
- 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 [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)*
- 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.*

## Variables

- int [rna\\_plot\\_type](#)  
*Switch for changing the secondary structure layout algorithm.*

### 13.52.1 Detailed Description

### 13.52.2 Data Structure Documentation

#### 13.52.2.1 struct COORDINATE

this is a workaround for the SWIG Perl Wrapper RNA plot function that returns an array of type [COORDINATE](#)

### 13.52.3 Macro Definition Documentation

#### 13.52.3.1 #define VRNA\_PLOT\_TYPE\_SIMPLE 0

```
#include <ViennaRNA/plot_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\(\)](#)

#### 13.52.3.2 #define VRNA\_PLOT\_TYPE\_NAVIEW 1

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

Definition of Plot type *Naview*

This is the plot type definition for several RNA structure plotting functions telling them to use **Naview** plotting algorithm

See also

[rna\\_plot\\_type](#), [vrna\\_file\\_PS\\_rnaplot\\_a\(\)](#), [vrna\\_file\\_PS\\_rnaplot\(\)](#), [svg\\_rna\\_plot\(\)](#), [gmlRNA\(\)](#), [ssv\\_rna\\_plot\(\)](#), [xrna\\_plot\(\)](#)

#### 13.52.3.3 #define VRNA\_PLOT\_TYPE\_CIRCULAR 2

```
#include <ViennaRNA/plot_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\(\)](#)

### 13.52.4 Function Documentation

#### 13.52.4.1 int aliPS\_color\_aln ( const char \* *structure*, const char \* *filename*, const char \* *seqs*[], const char \* *names*[] )

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

PS\_color\_aln for duplexes

#### 13.52.4.2 int simple\_xy\_coordinates ( short \* *pair\_table*, float \* *X*, float \* *Y* )

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

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

See also

[make\\_pair\\_table\(\)](#), [rna\\_plot\\_type](#), [simple\\_circplot\\_coordinates\(\)](#), [naview\\_xy\\_coordinates\(\)](#), [vrna\\_file\\_PS\\_rnaplot\\_a\(\)](#), [vrna\\_file\\_PS\\_rnaplot\(\)](#), [svg\\_rna\\_plot\(\)](#)

## Parameters

<i>pair_table</i>	The pair table of the secondary structure
<i>X</i>	a pointer to an array with enough allocated space to hold the x coordinates
<i>Y</i>	a pointer to an array with enough allocated space to hold the y coordinates

## Returns

length of sequence on success, 0 otherwise

13.52.4.3 `int simple_circplot_coordinates ( short * pair_table, float * x, float * y )`

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

Calculate nucleotide coordinates for *Circular Plot*

This function calculates the coordinates of nucleotides mapped in equal distances onto a unit circle.

## Note

In order to draw nice arcs using quadratic bezier curves that connect base pairs one may calculate a second tangential point  $P^t$  in addition to the actual  $R^2$  coordinates. the simplest way to do so may be to compute a radius scaling factor  $rs$  in the interval  $[0, 1]$  that weights the proportion of base pair span to the actual length of the sequence. This scaling factor can then be used to calculate the coordinates for  $P^t$ , i.e.  $P_x^t[i] = X[i] * rs$  and  $P_y^t[i] = Y[i] * rs$ .

## See also

`make_pair_table()`, `rna_plot_type`, `simple_xy_coordinates()`, `naview_xy_coordinates()`, `vrna_file_PS_↵  
rnaplot_a()`, `vrna_file_PS_rnaplot`, `svg_rna_plot()`

## Parameters

<i>pair_table</i>	The pair table of the secondary structure
<i>x</i>	a pointer to an array with enough allocated space to hold the x coordinates
<i>y</i>	a pointer to an array with enough allocated space to hold the y coordinates

## Returns

length of sequence on success, 0 otherwise

13.52.4.4 `int vrna_file_PS_rnaplot ( const char * seq, const char * structure, const char * file, vrna_md_t * md_p )`

```
#include <ViennaRNA/plot_structure.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

13.52.4.5 `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/plot_structure.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

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

```
#include <ViennaRNA/plot_structure.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

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

```
#include <ViennaRNA/plot_structure.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

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

```
#include <ViennaRNA/plot_structure.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

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

```
#include <ViennaRNA/plot_structure.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

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

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

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

**Deprecated** Use `vrna_file_PS_rnaplot()` instead!

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

```
#include <ViennaRNA/plot_structure.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!

13.52.4.12 `int PS_rna_plot_a_gquad ( char * string, char * structure, char * ssfile, char * pre, char * post )`

```
#include <ViennaRNA/plot_structure.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!

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

```
#include <ViennaRNA/PS_dot.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

**13.52.4.14** `int PS_dot_plot ( char * string, char * file )`

```
#include <ViennaRNA/PS_dot.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!

## 13.52.5 Variable Documentation

**13.52.5.1** `int rna_plot_type`

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

Switch for changing the secondary structure layout algorithm.

Current possibility are 0 for a simple radial drawing or 1 for the modified radial drawing taken from the *naview* program of Brucoleri & Heinrich (1988).

## 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](#)

## Chapter 14

# Data Structure Documentation

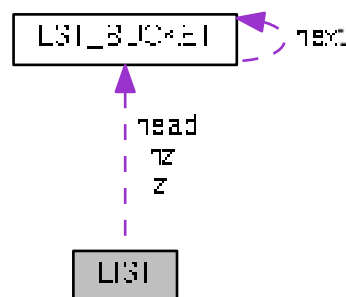
### 14.1 `_struct_en` Struct Reference

The documentation for this struct was generated from the following file:

- ViennaRNA/move\_set.h

### 14.2 `LIST` Struct Reference

Collaboration diagram for `LIST`:



The documentation for this struct was generated from the following file:

- ViennaRNA/list.h

### 14.3 LST\_BUCKET Struct Reference

Collaboration diagram for LST\_BUCKET:



The documentation for this struct was generated from the following file:

- ViennaRNA/list.h

### 14.4 Postorder\_list Struct Reference

The documentation for this struct was generated from the following file:

- ViennaRNA/[dist\\_vars.h](#)

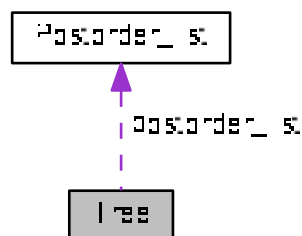
### 14.5 swString Struct Reference

The documentation for this struct was generated from the following file:

- ViennaRNA/[dist\\_vars.h](#)

### 14.6 Tree Struct Reference

Collaboration diagram for Tree:



The documentation for this struct was generated from the following file:

- ViennaRNA/[dist\\_vars.h](#)



### 14.7.1 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](#)

## 14.8 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.*

### 14.8.1 Detailed Description

Solution element from subopt.c.

The documentation for this struct was generated from the following file:

- [ViennaRNA/subopt.h](#)

# Chapter 15

## File Documentation

### 15.1 ViennaRNA/1.8.4\_epars.h File Reference

Free energy parameters for parameter file conversion.

#### 15.1.1 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-Neughbor Parameters for GU Mismatches...." Biochemistry 1991, 30 11124-11132

A.E. Peritz, R. Kierzek, N, Sugimoto, D.H. Turner "Thermodynamic Study of Internal Loops in Oligoribonucleotides..." Biochemistry 1991, 30, 6428-6435

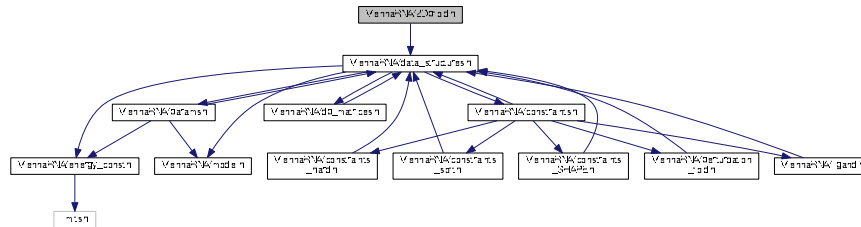
### 15.2 ViennaRNA/1.8.4\_intloops.h File Reference

Free energy parameters for interior loop contributions needed by the parameter file conversion functions.





Include dependency graph for 2Dpfold.h:



- struct `vrna_sol_TwoD_pf_t`  
*Solution element returned from `vrna_pf_TwoD()` [More...](#)*
- struct `TwoDpfold_vars`  
*Variables compound for 2Dfold partition function folding*

- typedef struct `vrna_sol_TwoD_pf_t` `vrna_sol_TwoD_pf_t`  
*Solution element returned from `vrna_pf_TwoD()`*

- `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.*

### 15.4.1 Function Documentation

**15.4.1.1** `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

**15.4.1.2** `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
-------------	--------------------------------

**15.4.1.3** `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\(\)](#), [#TwoDpfold\\_solution](#)

#### 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

#### 15.4.1.4 `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

#### 15.4.1.5 `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

## 15.5 ViennaRNA/alifold.h File Reference

compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments

[illegible]

- 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\\_pf\\_alifold](#) (const char \*\*strings, char \*structure, [vrna\\_plist\\_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\\_plist\\_t](#) \*\*pl)  
*Compute Partition function  $Q$  (and base pair probabilities) for an alignment of circular RNA sequences using a comparative method.*
- float [alifold](#) (const char \*\*strings, char \*structure)  
*Compute MFE and according consensus structure of an alignment of sequences.*
- float [circalifold](#) (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 [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.*
- float [alipf\\_fold\\_par](#) (const char \*\*sequences, char \*structure, [vrna\\_plist\\_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\\_plist\\_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\\_plist\\_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.*

- void [update\\_alifold\\_params](#) (void)

*Update the energy parameters for alifold function.*

## 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.*

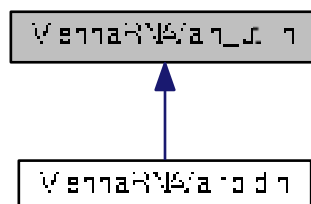
### 15.5.1 Detailed Description

compute various properties (consensus MFE structures, partition function, Boltzmann distributed stochastic samples, ...) for RNA sequence alignments

## 15.6 ViennaRNA/aln\_util.h File Reference

Various utility- and helper-functions for sequence alignments and comparative structure prediction.

This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [vrna\\_pinfo\\_s](#)

*A base pair info structure. [More...](#)*

## 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](#) (char \*Aseq[], int n\_seq, int length, int \*mini)  
*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.*
- 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.*

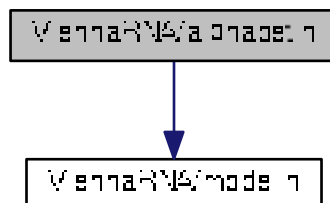
### 15.6.1 Detailed Description

Various utility- and helper-functions for sequence alignments and comparative structure prediction.

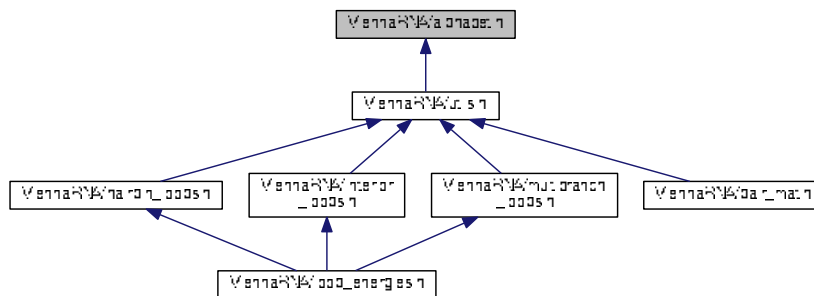
## 15.7 ViennaRNA/alphabet.h File Reference

Functions to process, convert, and generally handle different nucleotide and/or base pair alphabets.

Include dependency graph for alphabet.h:



This graph shows which files directly or indirectly include this file:



## Functions

- `char * vrna_ptypes` (`const short *S`, `vrna_md_t *md`)  
Get an array of the numerical encoding for each possible base pair (i,j)
- `short * vrna_seq_encode` (`const char *sequence`, `vrna_md_t *md`)  
Get a numerical representation of the nucleotide sequence.
- `short * vrna_seq_encode_simple` (`const char *sequence`, `vrna_md_t *md`)  
Get a numerical representation of the nucleotide sequence (simple version)
- `int vrna_nucleotide_encode` (`char c`, `vrna_md_t *md`)  
Encode a nucleotide character to numerical value.
- `char vrna_nucleotide_decode` (`int enc`, `vrna_md_t *md`)  
Decode a numerical representation of a nucleotide back into nucleotide alphabet.

### 15.7.1 Detailed Description

Functions to process, convert, and generally handle different nucleotide and/or base pair alphabets.

### 15.7.2 Function Documentation

#### 15.7.2.1 `char* vrna_ptypes ( const short * S, vrna_md_t * md )`

Get an array of the numerical encoding for each possible base pair (i,j)

#### Note

This array is always indexed in column-wise order, in contrast to previously different indexing between mfe and pf variants!

#### See also

[vrna\\_idx\\_col\\_wise\(\)](#), [vrna\\_fold\\_compound\\_t](#)



### 15.7.2.2 `int vrna_nucleotide_encode ( char c, vrna_md_t * md )`

Encode a nucleotide character to numerical value.

This function encodes a nucleotide character to its numerical representation as required by many functions in R<sub>NALib</sub>.

See also

[vrna\\_nucleotide\\_decode\(\)](#), [vrna\\_seq\\_encode\(\)](#)

#### Parameters

<i>c</i>	The nucleotide character to encode
<i>md</i>	The model details that determine the kind of encoding

#### Returns

The encoded nucleotide

### 15.7.2.3 `char vrna_nucleotide_decode ( int enc, vrna_md_t * md )`

Decode a numerical representation of a nucleotide back into nucleotide alphabet.

This function decodes a numerical representation of a nucleotide character back into nucleotide alphabet

See also

[vrna\\_nucleotide\\_encode\(\)](#), [vrna\\_seq\\_encode\(\)](#)

#### Parameters

<i>enc</i>	The encoded nucleotide
<i>md</i>	The model details that determine the kind of decoding

#### Returns

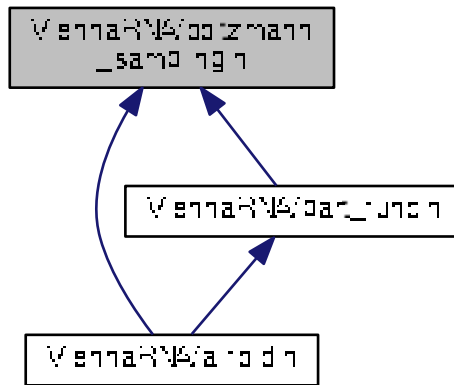
The decoded nucleotide character

## 15.8 ViennaRNA/boltzmann\_sampling.h File Reference

Boltzmann Sampling of secondary structures from the ensemble.

```

graph TD
    A[V_התאמת_המבנה] --> B[V_התאמת_התוכן]
    A --> C[V_התאמת_הסגנון]
    B --> C
    C --> A
    C --> B
  
```



- `char * vrna_pbacktrack5 (vrna_fold_compound_t *vc, int length)`  
*Sample a secondary structure of a subsequence from the Boltzmann ensemble according its probability.*
- `char * vrna_pbacktrack (vrna_fold_compound_t *vc)`  
*Sample a secondary structure (consensus structure) from the Boltzmann ensemble according its probability.*

Boltzmann Sampling of secondary structures from the ensemble.

Generated by Doxygen

Centroid structure computation.

[illegible]

```

graph BT
    A[VerrnaRNA/aligner] --> B[VerrnaRNA/paral_uni]
    B --> C[VerrnaRNA/aligner]

```

- 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\_plist\_t \*pl)  
*Get the centroid structure of the ensemble.*
- char \* [vrna\\_centroid\\_from\\_probs](#) (int length, double \*dist, FLT\_OR\_DBL \*probs)  
*Get the centroid structure of the ensemble.*
- char \* [get\\_centroid\\_struct\\_pl](#) (int length, double \*dist, vrna\_plist\_t \*pl)  
*Get the centroid structure of the ensemble.*
- char \* [get\\_centroid\\_struct\\_pr](#) (int length, double \*dist, FLT\_OR\_DBL \*pr)  
*Get the centroid structure of the ensemble.*



## Functions

- float `vrna_cofold` (const char \*string, char \*structure)  
*Compute Minimum Free Energy (MFE), and a corresponding secondary structure for two dimerized RNA sequences.*
- float `cofold` (const char \*sequence, char \*structure)  
*Compute the minimum free energy of two interacting RNA molecules.*
- float `cofold_par` (const char \*string, char \*structure, `vrna_param_t` \*parameters, int is\_constrained)  
*Compute the minimum free energy of two interacting RNA molecules.*
- void `free_co_arrays` (void)  
*Free memory occupied by `cofold()`*
- void `update_cofold_params` (void)  
*Recalculate parameters.*
- void `update_cofold_params_par` (`vrna_param_t` \*parameters)  
*Recalculate parameters.*
- void `export_cofold_arrays_gq` (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*fc\_p, int \*\*ggg\_p, int \*\*indx\_p, char \*\*ptype\_p)  
*Export the arrays of partition function cofold (with gquadruplex support)*
- void `export_cofold_arrays` (int \*\*f5\_p, int \*\*c\_p, int \*\*fML\_p, int \*\*fM1\_p, int \*\*fc\_p, int \*\*indx\_p, char \*\*ptype\_p)  
*Export the arrays of partition function cofold.*
- void `get_monomere_mfes` (float \*e1, float \*e2)  
*get\_monomer\_free\_energies*
- void `initialize_cofold` (int length)

### 15.10.1 Detailed Description

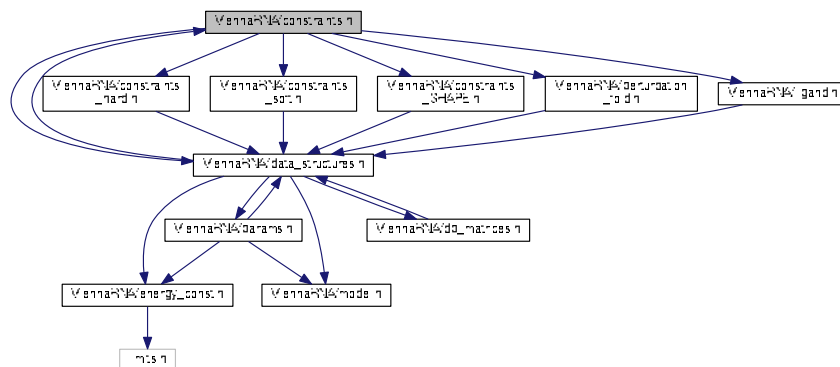
MFE version of cofolding routines.

This file includes (almost) all function declarations within the **RNALib** that are related to MFE Cofolding... This also includes the Zuker suboptimals calculations, since they are implemented using the cofold routines.

## 15.11 ViennaRNA/constraints.h File Reference

### Functions and data structures for constraining secondary structure predictions and evaluation.

Include dependency graph for constraints.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 1`  
Flag passed to generic softt constraints callback to indicate hairpin loop decomposition step.
- `#define VRNA_DECOMP_PAIR_IL 2`  
Indicator for interior loop decomposition step.
- `#define VRNA_DECOMP_PAIR_ML 3`  
Indicator for multibranch loop decomposition step.
- `#define VRNA_DECOMP_ML_ML_ML 5`  
Indicator for decomposition of multibranch loop part.
- `#define VRNA_DECOMP_ML_STEM 4`  
Indicator for decomposition of multibranch loop part.
- `#define VRNA_DECOMP_ML_ML 6`  
Indicator for decomposition of multibranch loop part.
- `#define VRNA_DECOMP_ML_UP 11`  
Indicator for decomposition of multibranch loop part.
- `#define VRNA_DECOMP_ML_ML_STEM 20`  
Indicator for decomposition of multibranch loop part.
- `#define VRNA_DECOMP_ML_COAXIAL 13`  
Indicator for decomposition of multibranch loop part.
- `#define VRNA_DECOMP_EXT_EXT 9`  
Indicator for decomposition of exterior loop part.
- `#define VRNA_DECOMP_EXT_UP 8`  
Indicator for decomposition of exterior loop part.
- `#define VRNA_DECOMP_EXT_STEM 14`  
Indicator for decomposition of exterior loop part.
- `#define VRNA_DECOMP_EXT_EXT_EXT 15`  
Indicator for decomposition of exterior loop part.
- `#define VRNA_DECOMP_EXT_STEM_EXT 16`  
Indicator for decomposition of exterior loop part.
- `#define VRNA_DECOMP_EXT_STEM_OUTSIDE 17`  
Indicator for decomposition of exterior loop part.
- `#define VRNA_DECOMP_EXT_EXT_STEM 18`  
Indicator for decomposition of exterior loop part.
- `#define VRNA_DECOMP_EXT_EXT_STEM1 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.*

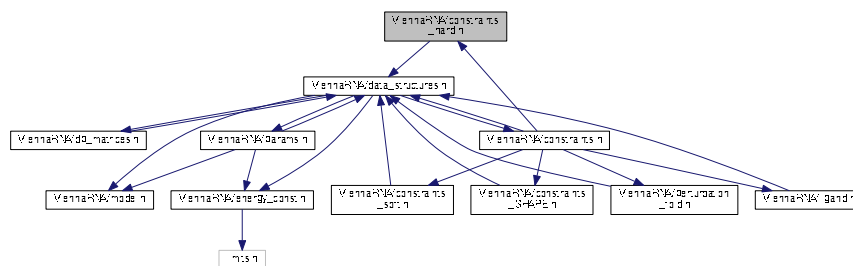
### 15.11.1 Detailed Description

Functions and data structures for constraining secondary structure predictions and evaluation.

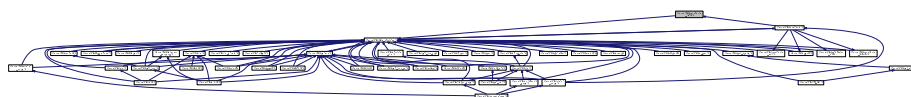
## 15.12 ViennaRNA/constraints\_hard.h File Reference

Functions and data structures for handling of secondary structure hard constraints.

Include dependency graph for constraints\_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 gquad)*
- `#define VRNA_CONSTRAINT_DB_DEFAULT`  
*Switch for dot-bracket structure constraint with default symbols.*
- `#define VRNA_CONSTRAINT_CONTEXT_EXT_LOOP (char)0x01`  
*Hard constraints flag, base pair in the exterior loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_HP_LOOP (char)0x02`  
*Hard constraints flag, base pair encloses hairpin loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_INT_LOOP (char)0x04`  
*Hard constraints flag, base pair encloses an interior loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_INT_LOOP_ENC (char)0x08`  
*Hard constraints flag, base pair encloses a multi branch loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_MB_LOOP (char)0x10`  
*Hard constraints flag, base pair is enclosed in an interior loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_MB_LOOP_ENC (char)0x20`  
*Hard constraints flag, base pair is enclosed in a multi branch loop.*
- `#define VRNA_CONSTRAINT_CONTEXT_ALL_LOOPS`  
*Hard constraints flag, shortcut for all base pairs.*

## 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 char( vrna_callback_hc_evaluate)(int i, int j, int k, int l, char d, void *data)`  
*Callback to evaluate whether or not a particular decomposition step is contributing to the solution space.*



## 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, char option)  
*Make a certain nucleotide unpaired.*
- int [vrna\\_hc\\_add\\_up\\_batch](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_hc\\_up\\_t](#) \*constraints)  
*Apply a list of hard constraints for single nucleotides.*
- void [vrna\\_hc\\_add\\_bp](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int i, int j, 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, 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.*
- 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.*

### 15.12.1 Detailed Description

Functions and data structures for handling of secondary structure hard constraints.

### 15.12.2 Macro Definition Documentation

#### 15.12.2.1 `#define VRNA_CONSTRAINT_NO_HEADER 0`

do not print the header information line

**Deprecated** This mode is not supported anymore!

#### 15.12.2.2 `#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\(\)](#)

### 15.12.3 Function Documentation

#### 15.12.3.1 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
---------------	--

#### 15.12.3.2 void print\_tty\_constraint\_full ( void )

Print structure constraint characters to stdout (full constraint support)

**Deprecated** Use [vrna\\_message\\_constraint\\_options\\_all\(\)](#) instead!

#### 15.12.3.3 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)

## 15.13 ViennaRNA/constraints\_SHAPE.h File Reference

This module provides function to incorporate SHAPE reactivity data into the folding recursions by means of soft constraints.

- `int vrna_sc_add_SHAPE_deigan` (`vrna_fold_compound_t` \*vc, const double \*reactivities, double m, double b, unsigned int options)  
*Add SHAPE reactivity data as soft constraints (Deigan et al. method)*
- `int vrna_sc_add_SHAPE_deigan_ali` (`vrna_fold_compound_t` \*vc, const char \*\*shape\_files, const int \*shape\_file\_association, double m, double b, unsigned int options)  
*Add SHAPE reactivity data from files as soft constraints for consensus structure prediction (Deigan et al. method)*
- `int vrna_sc_add_SHAPE_zarringhalam` (`vrna_fold_compound_t` \*vc, const double \*reactivities, double b, double default\_value, const char \*shape\_conversion, unsigned int options)  
*Add SHAPE reactivity data as soft constraints (Zarringhalam et al. method)*
- `int vrna_sc_SHAPE_parse_method` (const char \*method\_string, char \*method, float \*param\_1, float \*param\_2)  
*Parse a character string and extract the encoded SHAPE reactivity conversion method and possibly the parameters for conversion into pseudo free energies.*
- `int vrna_sc_SHAPE_to_pr` (const char \*shape\_conversion, double \*values, int length, double default\_value)  
*Convert SHAPE reactivity values to probabilities for being unpaired.*

This module provides function to incorporate SHAPE reactivity data into the folding recursions by means of soft constraints.

**15.13.2.1** `int vrna_sc SHAPE parse method ( const char * method string, char * method, float * param 1, float * param 2 )`

Generated by Doxygen



## 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_add_bp` (`vrna_fold_compound_t` \*vc, const `FLT_OR_DBL` \*\*constraints, unsigned int options)  
*Add soft constraints for paired nucleotides.*
- void `vrna_sc_add_up` (`vrna_fold_compound_t` \*vc, const `FLT_OR_DBL` \*constraints, unsigned int options)  
*Add soft constraints for unpaired nucleotides.*
- void `vrna_sc_remove` (`vrna_fold_compound_t` \*vc)  
*Remove soft constraints from `vrna_fold_compound_t`.*
- void `vrna_sc_free` (`vrna_sc_t` \*sc)  
*Free memory occupied by a `vrna_sc_t` data structure.*
- void `vrna_sc_add_data` (`vrna_fold_compound_t` \*vc, void \*data, `vrna_callback_free_auxdata` \*free\_data)  
*Add an auxiliary data structure for the generic soft constraints callback function.*
- void `vrna_sc_add_f` (`vrna_fold_compound_t` \*vc, `vrna_callback_sc_energy` \*f)  
*Bind a function pointer for generic soft constraint feature (MFE version)*
- void `vrna_sc_add_bt` (`vrna_fold_compound_t` \*vc, `vrna_callback_sc_backtrack` \*f)  
*Bind a backtracking function pointer for generic soft constraint feature.*
- void `vrna_sc_add_exp_f` (`vrna_fold_compound_t` \*vc, `vrna_callback_sc_exp_energy` \*exp\_f)  
*Bind a function pointer for generic soft constraint feature (PF version)*

### 15.14.1 Detailed Description

Functions and data structures for secondary structure soft constraints.

## 15.15 ViennaRNA/convert\_epars.h File Reference

Functions and definitions for energy parameter file format conversion.

## 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`



## 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\\_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\\_EVAL\\_ONLY](#) 8U  
*Option flag to specify that neither MFE, nor PF DP matrices are required.*

## Typedefs

- typedef struct [vrna\\_fc\\_s](#) [vrna\\_fold\\_compound\\_t](#)  
*Typename for the fold\_compound data structure [vrna\\_fc\\_s](#).*
- typedef struct [vrna\\_basepair\\_s](#) [vrna\\_basepair\\_t](#)  
*Typename for the base pair representing data structure [vrna\\_basepair\\_s](#).*
- typedef struct [vrna\\_plist\\_s](#) [vrna\\_plist\\_t](#)  
*Typename for the base pair list representing data structure [vrna\\_plist\\_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 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.*
- typedef struct [vrna\\_basepair\\_s](#) PAIR  
*Old typename of [vrna\\_basepair\\_s](#).*
- typedef struct [vrna\\_plist\\_s](#) plist  
*Old typename of [vrna\\_plist\\_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 [pu\\_out](#) [pu\\_out](#)  
*Collection of all free\_energy of beeing unpaired values for output.*
- typedef struct [constrain](#) [constrain](#)  
*constraints for cofolding*

## Enumerations

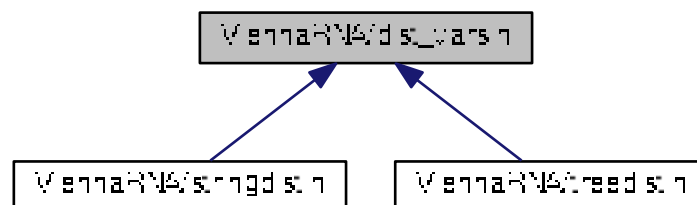
## Functions

- [vrna\\_fold\\_compound\\_t \\* vrna\\_fold\\_compound](#) (const char \*sequence, [vrna\\_md\\_t](#) \*md\_p, unsigned int options)  
Retrieve a [vrna\\_fold\\_compound\\_t](#) data structure for single sequences and hybridizing sequences.
- [vrna\\_fold\\_compound\\_t \\* vrna\\_fold\\_compound\\_comparative](#) (const char \*\*sequences, [vrna\\_md\\_t](#) \*md\_p, unsigned int options)  
Retrieve a [vrna\\_fold\\_compound\\_t](#) data structure for sequence alignments.
- void [vrna\\_fold\\_compound\\_free](#) ([vrna\\_fold\\_compound\\_t](#) \*vc)  
Free memory occupied by a [vrna\\_fold\\_compound\\_t](#).
- void [vrna\\_fold\\_compound\\_add\\_auxdata](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, void \*data, [vrna\\_callback\\_free\\_t](#) auxdata \*f)  
Add auxiliary data to the [vrna\\_fold\\_compound\\_t](#).
- void [vrna\\_fold\\_compound\\_add\\_callback](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, [vrna\\_callback\\_recursion\\_status](#) \*f)  
Add a recursion status callback to the [vrna\\_fold\\_compound\\_t](#).

## 15.17 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](#)
- struct [Tree](#)
- struct [swString](#)

## 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.

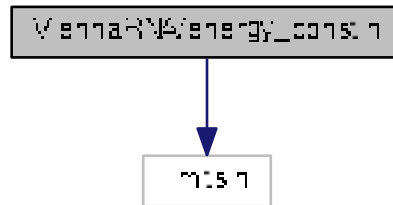






## 15.21 ViennaRNA/energy\_const.h File Reference

Include dependency graph for energy\_const.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`

### 15.21.1 Detailed Description

energy constants

### 15.21.2 Macro Definition Documentation

#### 15.21.2.1 `#define GASCONST 1.98717 /* in [cal/K] */`

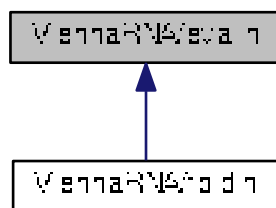
The gas constant

#### 15.21.2.2 `#define K0 273.15`

0 deg Celsius in Kelvin



This graph shows which files directly or indirectly include this file:



## Functions

- 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\\_simple](#) (const char \*string, const char \*structure)  
Calculate the free energy of an already folded RNA.
- 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\\_simple\\_verbose](#) (const char \*string, const char \*structure, FILE \*file)  
Calculate the free energy of an already folded RNA and print contributions per loop.
- 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\\_simple](#) (const char \*string, 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\\_simple\\_verbose](#) (const char \*string, const short \*pt, FILE \*file)  
Calculate the free energy of an already folded RNA.
- int [vrna\\_eval\\_loop\\_pt](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int i, const short \*pt)  
Calculate energy of a loop.
- float [vrna\\_eval\\_move](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, const char \*structure, int m1, int m2)  
Calculate energy of a move (closing or opening of a base pair)
- int [vrna\\_eval\\_move\\_pt](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, short \*pt, int m1, int m2)  
Calculate energy of a move (closing or opening of a base pair)
- float [energy\\_of\\_structure](#) (const char \*string, const char \*structure, int verbosity\_level)  
Calculate the free energy of an already folded RNA using global model detail settings.
- float [energy\\_of\\_struct\\_par](#) (const char \*string, const char \*structure, [vrna\\_param\\_t](#) \*parameters, int verbosity\_level)  
Calculate the free energy of an already folded RNA.
- float [energy\\_of\\_circ\\_structure](#) (const char \*string, const char \*structure, int verbosity\_level)  
Calculate the free energy of an already folded circular RNA.
- float [energy\\_of\\_circ\\_struct\\_par](#) (const char \*string, const char \*structure, [vrna\\_param\\_t](#) \*parameters, int verbosity\_level)

*Calculate the free energy of an already folded circular RNA.*

- int [energy\\_of\\_structure\\_pt](#) (const char \*string, short \*ptable, short \*s, short \*s1, int verbosity\_level)

*Calculate the free energy of an already folded RNA.*

- int [energy\\_of\\_struct\\_pt\\_par](#) (const char \*string, short \*ptable, short \*s, short \*s1, [vrna\\_param\\_t](#) \*parameters, int verbosity\_level)

*Calculate the free energy of an already folded RNA.*

- float [energy\\_of\\_move](#) (const char \*string, const char \*structure, int m1, int m2)

*Calculate energy of a move (closing or opening of a base pair)*

- int [energy\\_of\\_move\\_pt](#) (short \*pt, short \*s, short \*s1, int m1, int m2)

*Calculate energy of a move (closing or opening of a base pair)*

- int [loop\\_energy](#) (short \*ptable, short \*s, short \*s1, int i)

*Calculate energy of a loop.*

- float [energy\\_of\\_struct](#) (const char \*string, const char \*structure)
- int [energy\\_of\\_struct\\_pt](#) (const char \*string, short \*ptable, short \*s, short \*s1)
- float [energy\\_of\\_circ\\_struct](#) (const char \*string, const char \*structure)

## Variables

- int [cut\\_point](#)  
*set to first pos of second seq for cofolding*
- int [eos\\_debug](#)  
*verbose info from energy\_of\_struct*

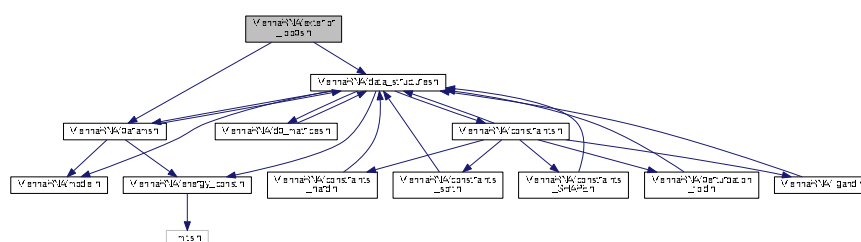
### 15.22.1 Detailed Description

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

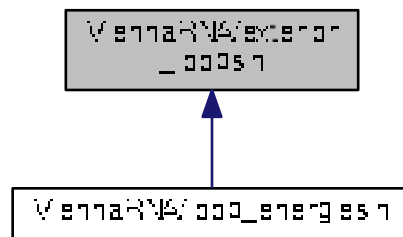
## 15.23 ViennaRNA/exterior\_loops.h File Reference

Energy evaluation of exterior loops for MFE and partition function calculations.

Include dependency graph for exterior\_loops.h:



This graph shows which files directly or indirectly include this file:



## Functions

- [int E\\_ExtLoop](#) (int type, int si1, int sj1, [vrna\\_param\\_t \\*P](#))
- [FLT\\_OR\\_DBL exp\\_E\\_ExtLoop](#) (int type, int si1, int sj1, [vrna\\_exp\\_param\\_t \\*P](#))
- [int E\\_Stem](#) (int type, int si1, int sj1, int extLoop, [vrna\\_param\\_t \\*P](#))
- [FLT\\_OR\\_DBL exp\\_E\\_Stem](#) (int type, int si1, int sj1, int extLoop, [vrna\\_exp\\_param\\_t \\*P](#))

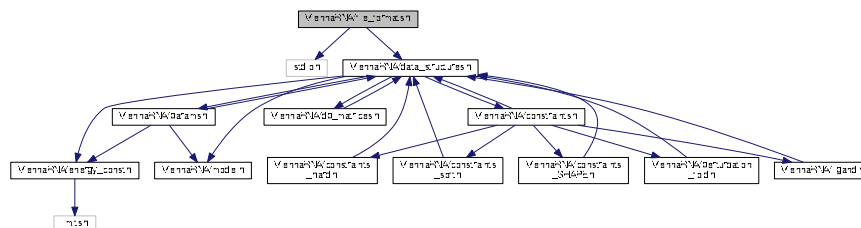
### 15.23.1 Detailed Description

Energy evaluation of exterior loops for MFE and partition function calculations.

## 15.24 ViennaRNA/file\_formats.h File Reference

Functions dealing with file formats for RNA sequences, structures, and alignments.

Include dependency graph for file\_formats.h:



## Macros

- [#define VRNA\\_OPTION\\_MULTILINE](#) 32U  
Tell a function that an input is assumed to span several lines.
- [#define VRNA\\_CONSTRAINT\\_MULTILINE](#) 32U  
parse multiline constraint

## Functions

- void [vrna\\_file\\_helixlist](#) (const char \*seq, const char \*db, float energy, FILE \*file)  
*Print a secondary structure as helix list.*
- void [vrna\\_file\\_connect](#) (const char \*seq, const char \*db, float energy, const char \*identifier, FILE \*file)  
*Print a secondary structure as connect table.*
- void [vrna\\_file\\_bpseq](#) (const char \*seq, const char \*db, FILE \*file)  
*Print a secondary structure in bpseq format.*
- void [vrna\\_file\\_json](#) (const char \*seq, const char \*db, double energy, const char \*identifier, FILE \*file)  
*Print a secondary structure in jsonformat.*
- unsigned int [vrna\\_file\\_fasta\\_read\\_record](#) (char \*\*header, char \*\*sequence, char \*\*\*rest, FILE \*file, unsigned int options)  
*Get a (fasta) data set from a file or stdin.*
- char \* [vrna\\_extract\\_record\\_rest\\_structure](#) (const char \*\*lines, unsigned int length, unsigned int option)  
*Extract a dot-bracket structure string from (multiline)character array.*
- int [vrna\\_file\\_SHAPE\\_read](#) (const char \*file\_name, int length, double default\_value, char \*sequence, double \*values)  
*Read data from a given SHAPE reactivity input file.*
- [vrna\\_plist\\_t](#) \* [vrna\\_file\\_constraints\\_read](#) (const char \*filename, unsigned int length, unsigned int options)  
*Read constraints from an input file.*
- void [vrna\\_extract\\_record\\_rest\\_constraint](#) (char \*\*cstruc, const char \*\*lines, unsigned int option)  
*Extract a hard constraint encoded as pseudo dot-bracket string.*
- unsigned int [read\\_record](#) (char \*\*header, char \*\*sequence, char \*\*\*rest, unsigned int options)  
*Get a data record from stdin.*

### 15.24.1 Detailed Description

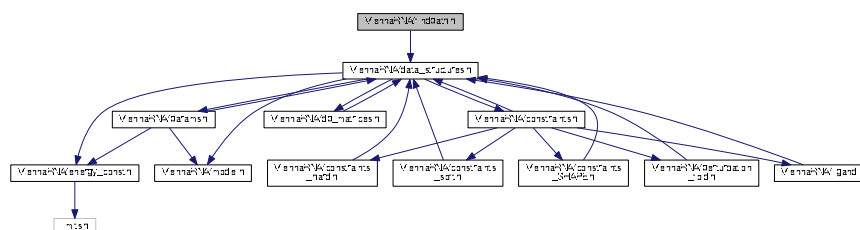
Functions dealing with file formats for RNA sequences, structures, and alignments.

See also

[Constraints Definition File](#)

## 15.25 ViennaRNA/findpath.h File Reference

Include dependency graph for findpath.h:

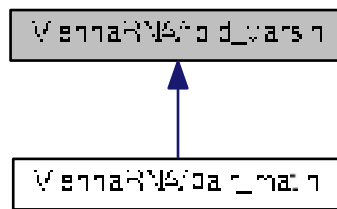








This graph shows which files directly or indirectly include this file:



## Variables

- int [fold\\_constrained](#)  
*Global switch to activate/deactivate folding with structure constraints.*
- int [csv](#)  
*generate comma seperated output*
- char \* [RibosumFile](#)
- int [james\\_rule](#)
- int [logML](#)
- int [cut\\_point](#)  
*Marks the position (starting from 1) of the first nucleotide of the second molecule within the concatenated sequence.*
- [bondT](#) \* [base\\_pair](#)  
*Contains a list of base pairs after a call to [fold\(\)](#).*
- [FLT\\_OR\\_DBL](#) \* [pr](#)  
*A pointer to the base pair probability matrix.*
- int \* [iindx](#)  
*index array to move through pr.*

### 15.27.1 Detailed Description

Here all all declarations of the global variables used throughout RNAlib.

### 15.27.2 Variable Documentation

#### 15.27.2.1 char\* RibosumFile

warning this variable will vanish in the future ribosums will be compiled in instead

#### 15.27.2.2 int james\_rule

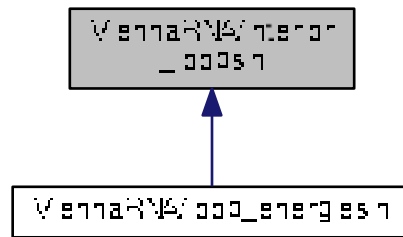
interior loops of size 2 get energy 0.8Kcal and no mismatches, default 1







This graph shows which files directly or indirectly include this file:



## Functions

- 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)
- int [E\\_stack](#) (int i, int j, [vrna\\_fold\\_compound\\_t](#) \*vc)
 

*Evaluate energy of a base pair stack closed by (i,j)*
- int [vrna\\_BT\\_stack](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, 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](#) \*vc, int \*i, int \*j, int en, [vrna\\_bp\\_stack\\_t](#) \*bp\_stack, int \*stack\_count)
 

*Backtrack an interior loop closed by (i,j).*

### 15.30.1 Detailed Description

Energy evaluation of interior loops for MFE and partition function calculations.

## 15.31 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.*







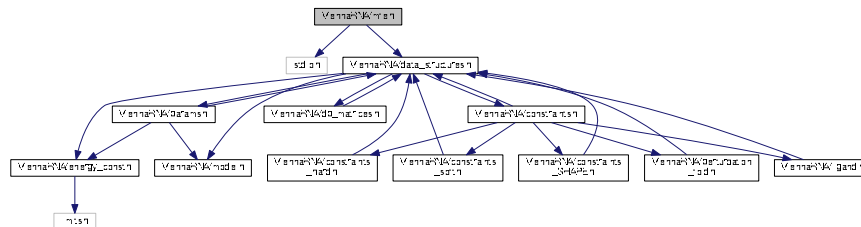




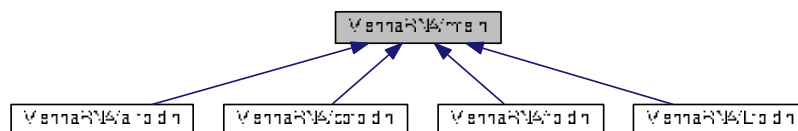
## 15.37 ViennaRNA/mfe.h File Reference

MFE calculations for single RNA sequences.

Include dependency graph for mfe.h:



This graph shows which files directly or indirectly include this file:



### Functions

- 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.*
- float [vrna\\_mfe\\_window](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, FILE \*file)  
*Local MFE prediction using a sliding window approach.*
- 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)*

### 15.37.1 Detailed Description

MFE calculations for single RNA sequences.

This file includes (almost) all function declarations within the RNAlib that are related to MFE folding...

## 15.38 ViennaRNA/mm.h File Reference

Several Maximum Matching implementations.

### 15.38.1 Detailed Description

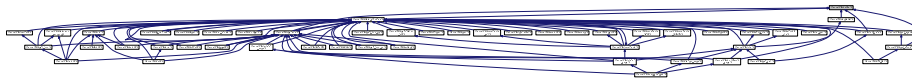
Several Maximum Matching implementations.

This file contains the declarations for several maximum matching implementations

## 15.39 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  $^{\circ}\text{C}$
- `#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 multibranch 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 multibranch 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 covariance contribution assessment.*

  - `#define VRNA_MODEL_DEFAULT_ALI_CV_FACT 1.`
- Default model behavior for weighting the covariance 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.*
- `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 multibranch loops using a logarithmic model.*
- int [vrna\\_md\\_defaults\\_logML\\_get](#) (void)  
*Get default behavior recomputing free energies of multibranch 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 multibranch loop prediction.*
- int [vrna\\_md\\_defaults\\_uniq\\_ML\\_get](#) (void)  
*Get default behavior for creating additional matrix for unique multibranch 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 covariance scaling factor used in comparative structure prediction.*

- double [vrna\\_md\\_defaults\\_cv\\_fact\\_get](#) (void)

*Get the default covariance 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 [canonicalBPonly](#)  
*do ML decomposition uniquely (for subopt)*
- int [uniq\\_ML](#)  
*do ML decomposition uniquely (for subopt)*
- int [energy\\_set](#)  
*0 = BP; 1=any mit GC; 2=any mit 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](#)*

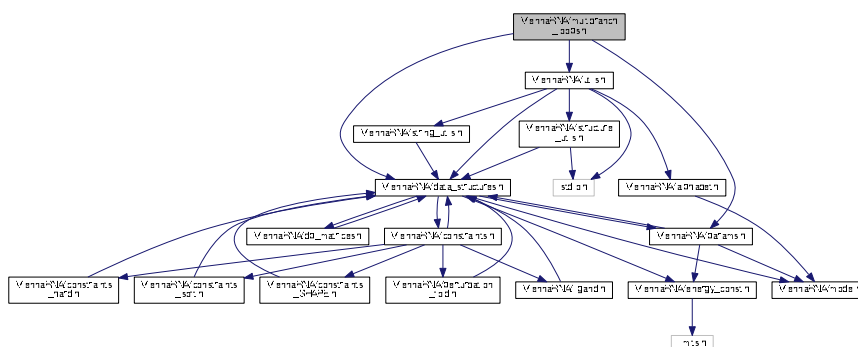
### 15.39.1 Detailed Description

The model details data structure and its corresponding modifiers.

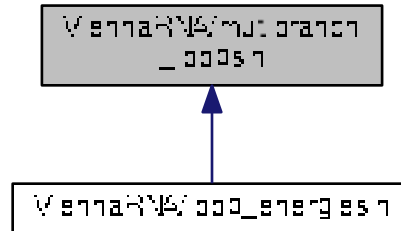
## 15.40 ViennaRNA/multibranch\_loops.h File Reference

Energy evaluation of multibranch loops for MFE and partition function calculations.

Include dependency graph for multibranch\_loops.h:



This graph shows which files directly or indirectly include this file:



## Functions

- int [E\\_mb\\_loop\\_stack](#) (int i, int j, [vrna\\_fold\\_compound\\_t](#) \*vc)

*Evaluate energy of a multi branch helices stacking onto closing pair (i,j)*

- int [vrna\\_BT\\_mb\\_loop](#) ([vrna\\_fold\\_compound\\_t](#) \*vc, int \*i, int \*j, int \*k, int en, int \*component1, int \*component2)

*Backtrack the decomposition of a multi branch loop closed by (i,j).*

### 15.40.1 Detailed Description

Energy evaluation of multibranch loops for MFE and partition function calculations.



## Typedefs

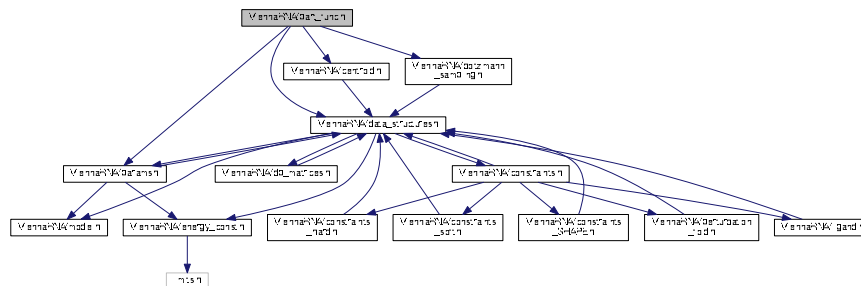
- typedef struct [vrna\\_param\\_s](#) [vrna\\_param\\_t](#)  
*Typename for the free energy parameter data structure [vrna\\_params](#).*
- typedef struct [vrna\\_exp\\_param\\_s](#) [vrna\\_exp\\_param\\_t](#)  
*Typename for the Boltzmann factor data structure [vrna\\_exp\\_params](#).*
- typedef struct [vrna\\_param\\_s](#) paramT  
*Old typename of [vrna\\_param\\_s](#).*
- typedef struct [vrna\\_exp\\_param\\_s](#) pf\_paramT  
*Old typename of [vrna\\_ex\\_param\\_s](#).*

## Functions

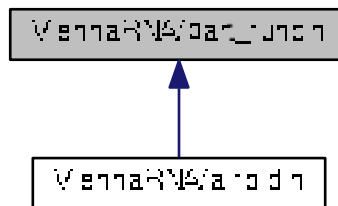
- [vrna\\_param\\_t](#) \* [vrna\\_params](#) ([vrna\\_md\\_t](#) \*md)  
*Get a data structure containing prescaled free energy parameters.*
- [vrna\\_param\\_t](#) \* [vrna\\_params\\_copy](#) ([vrna\\_param\\_t](#) \*par)  
*Get a copy of the provided free energy parameters.*
- [vrna\\_exp\\_param\\_t](#) \* [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)  
*Get precomputed Boltzmann factors of the loop type dependent energy contributions with independent thermodynamic temperature.*
- [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.*

### Partition function of single RNA sequences.

Include dependency graph for part\_func.h:



This graph shows which files directly or indirectly include this file:



- float `vrna_pf` (`vrna_fold_compound_t *vc`, `char *structure`)  
*Compute the partition function  $Q$  for a given RNA sequence, or sequence alignment.*
- float `vrna_pf_fold` (`const char *seq`, `char *structure`, `vrna_plist_t **pl`)  
*Compute Partition function  $Q$  (and base pair probabilities) for an RNA sequence using a comparative method.*
- float `vrna_pf_circfold` (`const char *seq`, `char *structure`, `vrna_plist_t **pl`)  
*Compute Partition function  $Q$  (and base pair probabilities) for a circular RNA sequences using a comparative method.*
- int `vrna_pf_float_precision` (`void`)  
*Find out whether partition function computations are using single precision floating points.*
- 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.*
- `vrna_plist_t * vrna_stack_prob` (`vrna_fold_compound_t *vc`, `double cutoff`)  
*Compute stacking probabilities.*

- float [pf\\_fold\\_par](#) (const char \*sequence, char \*structure, [vrna\\_exp\\_param\\_t](#) \*parameters, int calculate\_bp, int is\_constrained, int is\_circular)  
*Compute the partition function  $Q$  for a given RNA sequence.*
- float [pf\\_fold](#) (const char \*sequence, char \*structure)  
*Compute the partition function  $Q$  of an RNA sequence.*
- float [pf\\_circ\\_fold](#) (const char \*sequence, char \*structure)  
*Compute the partition function of a circular RNA sequence.*
- char \* [pbacktrack](#) (char \*sequence)  
*Sample a secondary structure from the Boltzmann ensemble according its probability.*
- char \* [pbacktrack\\_circ](#) (char \*sequence)  
*Sample a secondary structure of a circular RNA from the Boltzmann ensemble according its probability.*
- void [free\\_pf\\_arrays](#) (void)  
*Free arrays for the partition function recursions.*
- void [update\\_pf\\_params](#) (int length)  
*Recalculate energy parameters.*
- void [update\\_pf\\_params\\_par](#) (int length, [vrna\\_exp\\_param\\_t](#) \*parameters)  
*Recalculate energy parameters.*
- [FLT\\_OR\\_DBL](#) \* [export\\_bp](#) (void)  
*Get a pointer to the base pair probability array  
Accessing the base pair probabilities for a pair (i,j) is achieved by.*
- 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\\_plist\\_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)

## Variables

- int [st\\_back](#)  
*Flag indicating that auxiliary arrays are needed throughout the computations. This is essential for stochastic backtracking.*

### 15.43.1 Detailed Description

Partition function of single RNA sequences.

This file includes (almost) all function declarations within the **RNALib** that are related to Partion function folding...

## 15.43.2 Function Documentation

### 15.43.2.1 `int vrna_pf_float_precision ( void )`

Find out whether partition function computations are using single precision floating points.

See also

[FLT\\_OR\\_DBL](#)

Returns

1 if single precision is used, 0 otherwise

### 15.43.2.2 `vrna_plist_t* stackProb ( double cutoff )`

Get the probability of stacks.

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

### 15.43.2.3 `void init_pf_fold ( int length )`

Allocate space for [pf\\_fold\(\)](#)

**Deprecated** This function is obsolete and will be removed soon!

### 15.43.2.4 `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\(\)](#)

### 15.43.2.5 `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\(\)](#)





## Functions

- [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.*
- void [vrna\\_pf\\_dimer\\_probs](#) (double FAB, double FA, double FB, [vrna\\_plist\\_t](#) \*prAB, const [vrna\\_plist\\_t](#) \*prA, const [vrna\\_plist\\_t](#) \*prB, int Alength, const [vrna\\_exp\\_param\\_t](#) \*exp\_params)  
*Compute Boltzmann probabilities of dimerization without homodimers.*
- [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.*
- [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\\_plist\\_t](#) \* [get\\_plist](#) ([vrna\\_plist\\_t](#) \*pl, int length, double cut\_off)
- void [compute\\_probabilities](#) (double FAB, double FEA, double FEB, [vrna\\_plist\\_t](#) \*prAB, [vrna\\_plist\\_t](#) \*prA, [vrna\\_plist\\_t](#) \*prB, int Alength)  
*Compute Boltzmann probabilities of dimerization without homodimers.*
- [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.*
- 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.*

### 15.44.1 Detailed Description

Partition function for two RNA sequences.

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.

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.

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)

## 15.44.2 Function Documentation

### 15.44.2.1 `vrna_dimer_pf_t co_pf_fold ( char * sequence, char * structure )`

Calculate partition function and base pair probabilities.

This is the cofold partition function folding. The second molecule starts at the [cut\\_point](#) nucleotide.

#### Note

OpenMP: Since this function relies on the global parameters [do\\_backtrack](#), [dangles](#), [temperature](#) and [pf\\_scale](#) it is not threadsafe according to concurrent changes in these variables! Use `co_pf_fold_par()` instead to circumvent this issue.

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

#### Parameters

<i>sequence</i>	Concatenated RNA sequences
<i>structure</i>	Will hold the structure or constraints

#### Returns

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

### 15.44.2.2 `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.

This is the cofold partition function folding. The second molecule starts at the [cut\\_point](#) nucleotide.

**Deprecated** Use [vrna\\_pf\\_dimer\(\)](#) instead!

#### See also

[get\\_boltzmann\\_factors\(\)](#), [co\\_pf\\_fold\(\)](#)

## Parameters

<i>sequence</i>	Concatenated RNA sequences
<i>structure</i>	Pointer to the structure constraint
<i>parameters</i>	Data structure containing the precalculated Boltzmann factors
<i>calculate_bppm</i>	Switch to turn Base pair probability calculations on/off (0==off)
<i>is_constrained</i>	Switch to indicate that a structure constraint is passed via the structure argument (0==off)

## Returns

vrna\_dimer\_pf\_t structure containing a set of energies needed for concentration computations.

15.44.2.3 `vrna_plist_t* get_plist ( vrna_plist_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!

15.44.2.4 `void compute_probabilities ( double FAB, double FEA, double FEB, vrna_plist_t * prAB, vrna_plist_t * prA, vrna_plist_t * prB, int Alength )`

Compute Boltzmann probabilities of dimerization without homodimers.

Given the pair probabilities and free energies (in the null model) for a dimer AB and the two constituent monomers A and B, compute the conditional pair probabilities given that a dimer AB actually forms. Null model pair probabilities are given as a list as produced by [assign\\_plist\\_from\\_pr\(\)](#), the dimer probabilities 'prAB' are modified in place.

**Deprecated** { Use [vrna\\_pf\\_dimer\\_probs\(\)](#) instead!}

## Parameters

<i>FAB</i>	free energy of dimer AB
<i>FEA</i>	free energy of monomer A
<i>FEB</i>	free energy of monomer B
<i>prAB</i>	pair probabilities for dimer
<i>prA</i>	pair probabilities monomer
<i>prB</i>	pair probabilities monomer
<i>Alength</i>	Length of molecule A

15.44.2.5 `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

15.44.2.6 `void init_co_pf_fold ( int length )`

DO NOT USE THIS FUNCTION ANYMORE

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

15.44.2.7 `FLT_OR_DBL* export_co_bppm ( void )`

Get a pointer to the base pair probability array.

Accessing the base pair probabilities for a pair (i,j) is achieved by

```
FLT_OR_DBL *pr = export_bppm(); pr_ij = pr[iindx[i]-j];
```

**Deprecated** This function is deprecated and will be removed soon! The base pair probability array is available through the [vrna\\_fold\\_compound\\_t](#) data structure, and its associated [vrna\\_mx\\_pf\\_t](#) member.

#### See also

[vrna\\_idx\\_row\\_wise\(\)](#)

#### Returns

A pointer to the base pair probability array

## 15.44.2.8 void free\_co\_pf\_arrays ( void )

Free the memory occupied by [co\\_pf\\_fold\(\)](#)

**Deprecated** This function will be removed for the new API soon! See [vrna\\_pf\\_dimer\(\)](#), [vrna\\_fold\\_compound\(\)](#), and [vrna\\_fold\\_compound\\_free\(\)](#) for an alternative

## 15.44.2.9 void update\_co\_pf\_params ( int length )

Recalculate energy parameters.

This function recalculates all energy parameters given the current model settings.

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

## Parameters

<i>length</i>	Length of the current RNA sequence
---------------	------------------------------------

## 15.44.2.10 void update\_co\_pf\_params\_par ( int length, vrna\_exp\_param\_t \* parameters )

Recalculate energy parameters.

This function recalculates all energy parameters given the current model settings. It's second argument can either be NULL or a data structure containing the precomputed Boltzmann factors. In the first scenario, the necessary data structure will be created automatically according to the current global model settings, i.e. this mode might not be threadsafe. However, if the provided data structure is not NULL, threadsafety for the model parameters [dangles](#), [pf\\_scale](#) and [temperature](#) is regained, since their values are taken from this data structure during subsequent calculations.

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

## Parameters

<i>length</i>	Length of the current RNA sequence
<i>parameters</i>	data structure containing the precomputed Boltzmann factors

## 15.45 ViennaRNA/part\_func\_up.h File Reference

Partition Function Cofolding as stepwise process.

```

graph TD
    Root[Vennet/Kern_utils.R] --> DataStruct[Vennet/KData_structures.R]
    DataStruct --> DataTypes[Vennet/KDataTypes.R]
    DataStruct --> Addresses[Vennet/KAddresses.R]
    DataStruct --> Connections[Vennet/KConnections.R]
    DataTypes --> Core[Vennet/KConnections_core.R]
    Addresses --> Nodes[Vennet/KNodes.R]
    Connections --> Graph[Vennet/KConnections_graph.R]
    Connections --> Sets[Vennet/KConnections_sets.R]
    Connections --> Split[Vennet/KConnections_Split.R]
    Connections --> Subset[Vennet/KConnections_Subset.R]
    Connections --> Band[Vennet/KBand.R]
    Core --> Utils[utils.R]

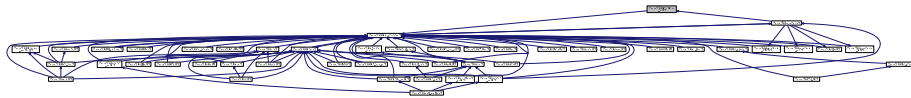
```

- `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()`.*

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.

[illegible]

This graph shows which files directly or indirectly include this file:



## Macros

- `#define VRNA_OBJECTIVE_FUNCTION_QUADRATIC 0`  
Use the sum of squared aberrations as objective function.
- `#define VRNA_OBJECTIVE_FUNCTION_ABSOLUTE 1`  
Use the sum of absolute aberrations as objective function.
- `#define VRNA_MINIMIZER_DEFAULT 0`  
Use a custom implementation of the gradient descent algorithm to minimize the objective function.
- `#define VRNA_MINIMIZER_CONJUGATE_FR 1`  
Use the GNU Scientific Library implementation of the Fletcher-Reeves conjugate gradient algorithm to minimize the objective function.
- `#define VRNA_MINIMIZER_CONJUGATE_PR 2`  
Use the GNU Scientific Library implementation of the Polak-Ribiere conjugate gradient algorithm to minimize the objective function.
- `#define VRNA_MINIMIZER_VECTOR_BFGS 3`  
Use the GNU Scientific Library implementation of the vector Broyden-Fletcher-Goldfarb-Shanno algorithm to minimize the objective function.
- `#define VRNA_MINIMIZER_VECTOR_BFGS2 4`  
Use the GNU Scientific Library implementation of the vector Broyden-Fletcher-Goldfarb-Shanno algorithm to minimize the objective function.
- `#define VRNA_MINIMIZER_STEEPEST_DESCENT 5`  
Use the GNU Scientific Library implementation of the steepest descent algorithm to minimize the objective function.

## Typedefs

- `typedef void(* progress_callback) (int iteration, double score, double *epsilon)`  
Callback for following the progress of the minimization process.

## Functions

- `void vrna_sc_minimize_perturbation (vrna_fold_compound_t *vc, const double *q_prob_unpaired, int objective_function, double sigma_squared, double tau_squared, int algorithm, int sample_size, double *epsilon, double initialStepSize, double minStepSize, double minImprovement, double minimizerTolerance, progress_callback callback)`  
Find a vector of perturbation energies that minimizes the discrepancies between predicted and observed pairing probabilities and the amount of necessary adjustments.

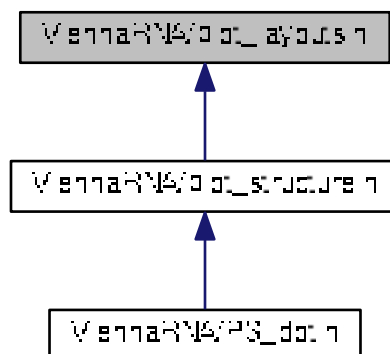
### 15.46.1 Detailed Description

Find a vector of perturbation energies that minimizes the discrepancies between predicted and observed pairing probabilities and the amount of necessary adjustments.





This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [COORDINATE](#)

*this is a workaround for the SWIG Perl Wrapper RNA plot function that returns an array of type [COORDINATE](#) [More...](#)*

## Macros

- #define [VRNA\\_PLOT\\_TYPE\\_SIMPLE](#) 0  
*Definition of Plot type simple*
- #define [VRNA\\_PLOT\\_TYPE\\_NAVIEW](#) 1  
*Definition of Plot type Naview*
- #define [VRNA\\_PLOT\\_TYPE\\_CIRCULAR](#) 2  
*Definition of Plot type Circular*

## Functions

- int [simple\\_xy\\_coordinates](#) (short \*pair\_table, float \*X, float \*Y)  
*Calculate nucleotide coordinates for secondary structure plot the Simple way*
- int [simple\\_circplot\\_coordinates](#) (short \*pair\_table, float \*x, float \*y)  
*Calculate nucleotide coordinates for Circular Plot*

## Variables

- int [rna\\_plot\\_type](#)  
*Switch for changing the secondary structure layout algorithm.*



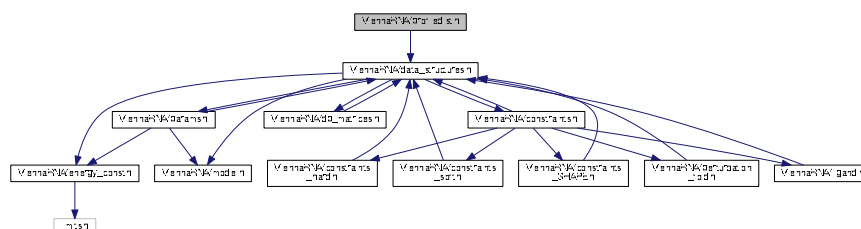
- Produce a secondary structure graph in SStructView format.*
- int [svg\\_rna\\_plot](#) (char \*string, char \*structure, char \*ssfile)  
*Produce a secondary structure plot in SVG format and write it to a file.*
- int [xrna\\_plot](#) (char \*string, char \*structure, char \*ssfile)  
*Produce a 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)*

### 15.49.1 Detailed Description

Various functions for plotting RNA secondary structures.

## 15.50 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)

### 15.50.1 Function Documentation

#### 15.50.1.1 `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

#### 15.50.1.2 `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

#### 15.50.1.3 `void free_profile ( float * T )`

free space allocated in `Make_bp_profile`

Backward compatibility only. You can just use plain `free()`

#### 15.50.1.4 `float* Make_bp_profile ( int length )`

##### Note

This function is NOT threadsafe

##### See also

[Make\\_bp\\_profile\\_bppm\(\)](#)

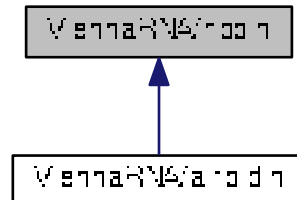
**Deprecated** This function is deprecated and will be removed soon! See [Make\\_bp\\_profile\\_bppm\(\)](#) for a replacement



## 15.53 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.*

### 15.53.1 Detailed Description

Parse RiboSum Scoring Matrices for Covariance Scoring of Alignments.

## 15.54 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](#) [STRUC]  
*contains a list of all loop sizes. `loop_size[0]` contains the number of external bases.*
- int [helix\\_size](#) [STRUC]  
*contains a list of all stack sizes.*
- int [loop\\_degree](#) [STRUC]  
*contains the corresponding list of loop degrees.*
- int [loops](#)  
*contains the number of loops ( and therefore of stacks ).*
- int [unpaired](#)  
*contains the number of unpaired bases.*
- int [pairs](#)  
*contains the number of base pairs in the last parsed structure.*

### 15.54.1 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)
*
```





- 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\\_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.*

### 15.55.1 Detailed Description

General utility- and helper-functions for RNA sequence and structure strings used throughout the ViennaRNA Package.

### 15.55.2 Function Documentation

#### 15.55.2.1 void str\_uppercase ( char \* sequence )

Convert an input sequence to uppercase.

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

#### 15.55.2.2 void str\_DNA2RNA ( char \* sequence )

Convert a DNA input sequence to RNA alphabet.

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

#### 15.55.2.3 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!

15.55.2.4 `int hamming ( const char * s1, const char * s2 )`

Calculate hamming distance between two sequences.

**Deprecated** Use `vrna_hamming_distance()` instead!

15.55.2.5 `int hamming_bound ( const char * s1, const char * s2, int n )`

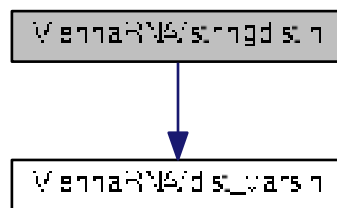
Calculate hamming distance between two sequences up to a specified length.

**Deprecated** Use `vrna_hamming_distance_bound()` instead!

## 15.56 ViennaRNA/stringdist.h File Reference

Functions for String Alignment.

Include dependency graph for stringdist.h:



### Functions

- `swString * Make_swString (char *string)`  
Convert a structure into a format suitable for `string_edit_distance()`.
- `float string_edit_distance (swString *T1, swString *T2)`  
Calculate the string edit distance of T1 and T2.

### 15.56.1 Detailed Description

Functions for String Alignment.

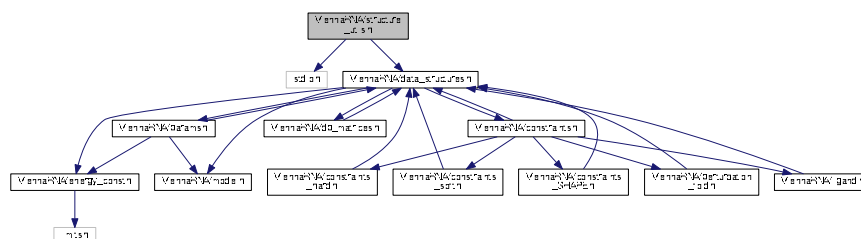
### 15.56.2 Function Documentation

15.56.2.1 `swString* Make_swString ( char * string )`

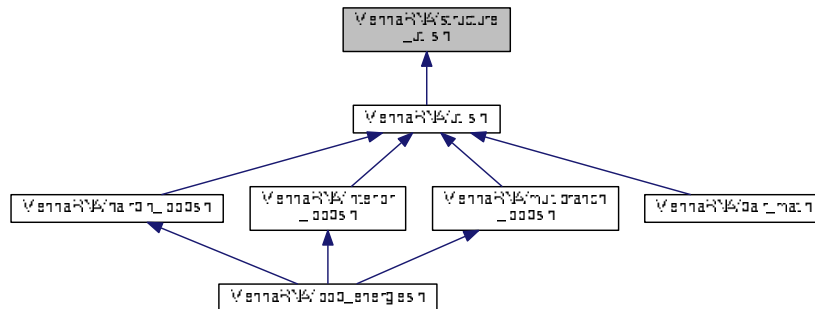
Convert a structure into a format suitable for `string_edit_distance()`.

---

*string*

$$\frac{T1}{T2}$$


This graph shows which files directly or indirectly include this file:



## Data Structures

- struct [vrna\\_hx\\_s](#)

## Functions

- char \* [vrna\\_db\\_pack](#) (const char \*struc)  
*Pack 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\(\)](#)*
- short \* [vrna\\_ptable](#) (const char \*structure)  
*Create a pair table of a secondary structure.*
- 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)*
- int \* [vrna\\_loopidx\\_from\\_ptable](#) (const short \*pt)  
*Get a loop index representation of a structure.*
- char \* [vrna\\_db\\_from\\_ptable](#) (short \*pt)  
*Convert a pair table into dot-parenthesis notation.*
- 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.*
- vrna\_plist\_t \* [vrna\\_plist](#) (const char \*struc, float pr)  
*Create a [vrna\\_plist\\_t](#) from a dot-bracket string.*
- vrna\_plist\_t \* [vrna\\_plist\\_from\\_probs](#) (vrna\_fold\_compound\_t \*vc, double cut\_off)  
*Create a [vrna\\_plist\\_t](#) from base pair probability matrix.*
- char \* [vrna\\_db\\_from\\_plist](#) (vrna\_plist\_t \*pairs, unsigned int n)  
*Convert a list of base pairs into dot-bracket notation.*
- void [assign\\_plist\\_from\\_db](#) (vrna\_plist\_t \*\*pl, const char \*struc, float pr)  
*Create a [vrna\\_plist\\_t](#) from a dot-bracket string.*
- char \* [pack\\_structure](#) (const char \*struc)  
*Pack 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\_plist\_t \*\*pl, FLT\_OR\_DBL \*probs, int length, double cutoff)  
*Create a [vrna\\_plist\\_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.*

### 15.57.1 Detailed Description

Various utility- and helper-functions for secondary structure parsing, converting, etc.



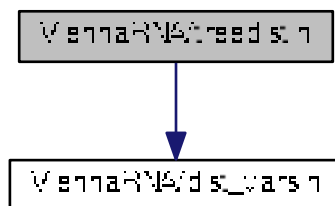
### 15.58.1 Detailed Description

RNASubopt and density of states declarations.

## 15.59 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.

### 15.59.1 Detailed Description

Functions for [Tree](#) Edit Distances.

### 15.59.2 Function Documentation

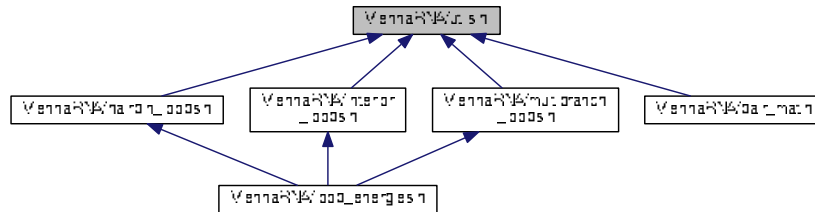
#### 15.59.2.1 [Tree](#)\* [make\\_tree](#) ( char \* *struc* )

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





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\\_message\\_error](#) (const char message[])  
*Die with an error message.*
- void [vrna\\_message\\_warning](#) (const char message[])  
*Print a warning message.*
- 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.*
- void [vrna\\_file\\_copy](#) (FILE \*from, FILE \*to)  
*Inefficient 'cp'.*
- char \* [vrna\\_time\\_stamp](#) (void)  
*Get a timestamp.*
- char \* [get\\_line](#) (FILE \*fp)  
*Read a line of arbitrary length from a stream.*
- unsigned int [get\\_input\\_line](#) (char \*\*string, unsigned int options)
- 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.*
- 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.*
- 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.*

### 15.60.1 Detailed Description

General utility- and helper-functions used throughout the *ViennaRNA Package*.

### 15.60.2 Function Documentation

#### 15.60.2.1 `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!

#### 15.60.2.2 `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!

#### 15.60.2.3 `void warn_user ( const char message[] )`

Print a warning message.

Print a warning message to *stderr*

**Deprecated** Use `vrna_message_warning()` instead!

#### 15.60.2.4 `void nrerror ( const char message[] )`

Die with an error message.

**Deprecated** Use `vrna_message_error()` instead!

#### 15.60.2.5 void\* space ( unsigned size )

Allocate space safely.

**Deprecated** Use `vrna_alloc()` instead!

#### 15.60.2.6 void\* xrealloc ( void \* p, unsigned size )

Reallocate space safely.

**Deprecated** Use `vrna_realloc()` instead!

#### 15.60.2.7 void init\_rand ( void )

Make random number seeds.

**Deprecated** Use `vrna_init_rand()` instead!

#### 15.60.2.8 double urn ( void )

get a random number from [0..1]

**Deprecated** Use `vrna_urn()` instead!

#### 15.60.2.9 int int\_urn ( int from, int to )

Generates a pseudo random integer in a specified range.

**Deprecated** Use `vrna_int_urn()` instead!

#### 15.60.2.10 void filecopy ( FILE \* from, FILE \* to )

Inefficient `cp`

**Deprecated** Use `vrna_file_copy()` instead!

#### 15.60.2.11 char\* time\_stamp ( void )

Get a timestamp.

**Deprecated** Use `vrna_time_stamp()` instead!

# 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. [192](#)
- [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. [185](#)
- [3] Katherine E. Deigan, Tian W. Li, David H. Mathews, and Kevin M. Weeks. Accurate SHAPE-directed RNA structure determination. *PNAS*, 106:97–102, 2009. [256](#)
- [4] 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. [288](#), [289](#)
- [5] 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. [3](#)
- [6] 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. [192](#)
- [7] 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. [2](#)
- [8] I.L. Hofacker and P.F. Stadler. Memory efficient folding algorithms for circular RNA secondary structures. *Bioinformatics*, 22(10):1172–1176, 2006. [145](#), [150](#), [173](#), [198](#), [202](#)
- [9] 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. [2](#)
- [10] 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. [223](#), [225](#)
- [11] D.H. Mathews, M.D. Disney, J.L. Childs, S.J. Schroeder, M. Zuker, and D.H. Turner. Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of RNA secondary structure. *Proceedings of the National Academy of Sciences of the United States of America*, 101(19):7287, 2004. [215](#)
- [12] J.S. McCaskill. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. *Biopolymers*, 29(6-7):1105–1119, 1990. [148](#)
- [13] B.A. Shapiro. An algorithm for comparing multiple RNA secondary structures. *Computer applications in the biosciences: CABIOS*, 4(3):387–393, 1988. [3](#)
- [14] 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. [5](#)
- [15] D.H. Turner and D.H. Mathews. NNDB: The nearest neighbor parameter database for predicting stability of nucleic acid secondary structure. *Nucleic Acids Research*, 38(suppl 1):D280–D282, 2010. [215](#)
- [16] 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. [264](#)

- [17] 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. [165](#)
- [18] Kourosh Zarringhalam, Michelle M. Meyer, Ivan Dotu, Jeffrey H. Chuang, and Peter Clote. Integrating chemical footprinting data into RNA secondary structure prediction. *PLOS ONE*, 7(10), 2012. [257](#)
- [19] M. Zuker. On finding all suboptimal foldings of an RNA molecule. *Science*, 244(4900):48–52, April 1989. [163](#)
- [20] 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. [145](#), [172](#)

# Index

- [\\_struct\\_en](#), [327](#)
- [2Dpfold.h](#)
  - [destroy\\_TwoDpfold\\_variables](#), [334](#)
  - [get\\_TwoDpfold\\_variables](#), [334](#)
  - [TwoDpfold\\_pbacktrack](#), [335](#)
  - [TwoDpfold\\_pbacktrack5](#), [336](#)
  - [TwoDpfoldList](#), [334](#)
- [add\\_root](#)
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [298](#)
- [aliLfold](#)
  - Local MFE consensus structures for Sequence Alignments, [214](#)
- [aliPS\\_color\\_aln](#)
  - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [321](#)
- [alifold](#)
  - MFE Consensus Structures for Sequence Alignment(s), [198](#)
- [alimake\\_pair\\_table](#)
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [307](#)
- [alipbacktrack](#)
  - Stochastic Backtracking of Consensus Structures from Sequence Alignment(s), [205](#)
- [alipf\\_circ\\_fold](#)
  - Partition Function and Base Pair Probabilities for Sequence Alignment(s), [203](#)
- [alipf\\_fold](#)
  - Partition Function and Base Pair Probabilities for Sequence Alignment(s), [203](#)
- [alipf\\_fold\\_par](#)
  - Partition Function and Base Pair Probabilities for Sequence Alignment(s), [203](#)
- [alloc\\_sequence\\_arrays](#)
  - Predicting Consensus Structures from Alignment(s), [195](#)
- [alpha](#)
  - [vrna\\_exp\\_param\\_s](#), [78](#)
- [alphabet.h](#)
  - [vrna\\_nucleotide\\_decode](#), [341](#)
  - [vrna\\_nucleotide\\_encode](#), [340](#)
  - [vrna\\_ptypes](#), [340](#)
- [assign\\_plist\\_from\\_db](#)
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [305](#)
- [assign\\_plist\\_from\\_pr](#)
  - Computing Partition Functions and Pair Probabilities, [158](#)
- [auxdata](#)
  - [vrna\\_fc\\_s](#), [269](#)
- [b2HIT](#)
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [297](#)
- [b2Shapiro](#)
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [298](#)
- [b2C](#)
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [298](#)
- [BONUS](#)
  - [energy\\_const.h](#), [360](#)
- [backtrack\\_GQuad\\_IntLoop](#)
  - Processing and Evaluating Decomposed Loops, [68](#)
- [backtrack\\_GQuad\\_IntLoop\\_L](#)
  - Processing and Evaluating Decomposed Loops, [68](#)
- [backtrack\\_type](#)
  - Manipulation of the Prediction Models, [117](#)
- [base\\_pair](#)
  - [fold\\_vars.h](#), [368](#)
- [bondT](#)
  - Data Structures and Preprocessor Macros, [136](#)
- [bp\\_distance](#)
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [307](#)
- [bppm\\_symbol](#)
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [308](#)
- [bppm\\_to\\_structure](#)
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [308](#)
- [bt](#)
  - [vrna\\_sc\\_s](#), [249](#)
- [COORDINATE](#), [320](#)
- [Calculate Partition Functions of a Distance Based Partitioning](#), [231](#)
  - [vrna\\_pf\\_TwoD](#), [232](#)
  - [vrna\\_sol\\_TwoD\\_pf\\_t](#), [232](#)
- [Calculate Secondary Structures of two RNAs upon Dimerization](#), [171](#)
- [Calculating MFE representatives of a Distance Based Partitioning](#), [224](#)
  - [destroy\\_TwoDfold\\_variables](#), [229](#)
  - [get\\_TwoDfold\\_variables](#), [228](#)
  - [TwoDfold\\_backtrack\\_f5](#), [230](#)
  - [TwoDfold\\_vars](#), [227](#)
  - [TwoDfoldList](#), [229](#)

- vrna\_backtrack5\_TwoD, 228
- vrna\_mfe\_TwoD, 227
- vrna\_sol\_TwoD\_t, 227
- canonicalBPonly
  - Manipulation of the Prediction Models, 116
- centroid
  - part\_func.h, 387
- centroid.h
  - get\_centroid\_struct\_pl, 344
  - get\_centroid\_struct\_pr, 344
- circularfold
  - MFE Consensus Structures for Sequence Alignment(s), 199
- circfold
  - MFE Structures of single Nucleic Acid Sequences, 175
- Classified Dynamic Programming, 222
- co\_pf\_fold
  - part\_func\_co.h, 390
- co\_pf\_fold\_par
  - part\_func\_co.h, 390
- cofold
  - MFE Structures of two hybridized Sequences, 179
- cofold\_par
  - MFE Structures of two hybridized Sequences, 180
- Compute the centroid structure, 160
  - vrna\_centroid, 160
  - vrna\_centroid\_from\_plist, 160
  - vrna\_centroid\_from\_probs, 161
- Compute the Density of States, 236
  - density\_of\_states, 236
- Compute the structure with maximum expected accuracy (MEA), 159
- compute\_BPdifferences
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 308
- compute\_probabilities
  - part\_func\_co.h, 391
- Computing Minimum Free Energy (MFE) Structures, 145
  - vrna\_mfe, 146
- Computing Partition Functions and Pair Probabilities, 147
  - assign\_plist\_from\_pr, 158
  - export\_bppm, 155
  - free\_pf\_arrays, 154
  - get\_pf\_arrays, 156
  - mean\_bp\_distance, 157
  - mean\_bp\_distance\_pr, 157
  - pf\_circ\_fold, 154
  - pf\_fold, 153
  - pf\_fold\_par, 152
  - update\_pf\_params, 155
  - update\_pf\_params\_par, 155
  - vrna\_mean\_bp\_distance, 151
  - vrna\_mean\_bp\_distance\_pr, 150
  - vrna\_pf, 149
  - vrna\_pf\_circfold, 150
  - vrna\_pf\_fold, 149
  - vrna\_plist\_from\_probs, 157
  - vrna\_stack\_prob, 151
- cons\_seq
  - vrna\_fc\_s, 271
- constrain, 135
- constrain\_ptypes
  - constraints\_hard.h, 350
- Constraining the Secondary Structure Predictions and Evaluations, 118
  - VRNA\_CONSTRAINT\_FILE, 122
  - VRNA\_CONSTRAINT\_SOFT\_MFE, 122
  - VRNA\_CONSTRAINT\_SOFT\_PF, 122
  - VRNA\_DECOMP\_EXT\_EXT\_EXT, 128
  - VRNA\_DECOMP\_EXT\_EXT\_STEM1, 129
  - VRNA\_DECOMP\_EXT\_EXT\_STEM, 128
  - VRNA\_DECOMP\_EXT\_EXT, 127
  - VRNA\_DECOMP\_EXT\_STEM\_EXT, 128
  - VRNA\_DECOMP\_EXT\_STEM, 127
  - VRNA\_DECOMP\_EXT\_UP, 127
  - VRNA\_DECOMP\_ML\_COAXIAL, 126
  - VRNA\_DECOMP\_ML\_ML\_ML, 124
  - VRNA\_DECOMP\_ML\_ML\_STEM, 126
  - VRNA\_DECOMP\_ML\_ML, 125
  - VRNA\_DECOMP\_ML\_STEM, 124
  - VRNA\_DECOMP\_ML\_UP, 125
  - VRNA\_DECOMP\_PAIR\_HP, 122
  - VRNA\_DECOMP\_PAIR\_IL, 123
  - VRNA\_DECOMP\_PAIR\_ML, 123
  - vrna\_constraints\_add, 129
  - vrna\_message\_constraint\_options, 130
  - vrna\_message\_constraint\_options\_all, 131
- constraints\_SHAPE.h
  - vrna\_sc\_SHAPE\_parse\_method, 351
- constraints\_hard.h
  - constrain\_ptypes, 350
  - print\_tty\_constraint, 350
  - print\_tty\_constraint\_full, 350
  - VRNA\_CONSTRAINT\_DB\_ANG\_BRACK, 349
  - VRNA\_CONSTRAINT\_NO\_HEADER, 349
- convert\_parameter\_file
  - Converting Energy Parameter Files, 221
- Converting Energy Parameter Files, 217
  - convert\_parameter\_file, 221
  - VRNA\_CONVERT\_OUTPUT\_ALL, 218
  - VRNA\_CONVERT\_OUTPUT\_BULGE, 219
  - VRNA\_CONVERT\_OUTPUT\_DANGLE3, 219
  - VRNA\_CONVERT\_OUTPUT\_DANGLE5, 219
  - VRNA\_CONVERT\_OUTPUT\_DUMP, 220
  - VRNA\_CONVERT\_OUTPUT\_HP, 218
  - VRNA\_CONVERT\_OUTPUT\_INT\_11, 219
  - VRNA\_CONVERT\_OUTPUT\_INT\_21, 219
  - VRNA\_CONVERT\_OUTPUT\_INT\_22, 219
  - VRNA\_CONVERT\_OUTPUT\_INT, 220
  - VRNA\_CONVERT\_OUTPUT\_MISC, 220
  - VRNA\_CONVERT\_OUTPUT\_MM\_EXT, 219
  - VRNA\_CONVERT\_OUTPUT\_MM\_HP, 218
  - VRNA\_CONVERT\_OUTPUT\_MM\_INT\_1N, 218
  - VRNA\_CONVERT\_OUTPUT\_MM\_INT\_23, 218



- VRNA\_CONVERT\_OUTPUT\_MM\_INT, 218
- VRNA\_CONVERT\_OUTPUT\_MM\_MULTI, 219
- VRNA\_CONVERT\_OUTPUT\_ML, 220
- VRNA\_CONVERT\_OUTPUT\_NINIO, 220
- VRNA\_CONVERT\_OUTPUT\_SPECIAL\_HP, 220
- VRNA\_CONVERT\_OUTPUT\_STACK, 218
- VRNA\_CONVERT\_OUTPUT\_VANILLA, 220
- copy\_pair\_table
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 306
- cost\_matrix
  - dist\_vars.h, 357
- cpair
  - Data Structures and Preprocessor Macros, 136
- cut\_point
  - fold\_vars.h, 368
- cv\_fact
  - Predicting Consensus Structures from Alignment(s), 196
- dangles
  - Manipulation of the Prediction Models, 115
  - vrna\_md\_s, 92
- Data Structures and Preprocessor Macros, 132
  - bondT, 136
  - cpair, 136
  - PAIR, 135
  - plist, 135
  - sect, 136
- density\_of\_states
  - Compute the Density of States, 236
- destroy\_TwoDfold\_variables
  - Calculating MFE representatives of a Distance Based Partitioning, 229
- destroy\_TwoDpfold\_variables
  - 2Dpfold.h, 334
- Direct refolding paths between two secondary structures, 287
  - find\_saddle, 289
  - free\_path, 290
  - get\_path, 290
  - path\_t, 288
  - vrna\_path\_findpath, 289
  - vrna\_path\_findpath\_saddle, 288
- dist\_vars.h
  - cost\_matrix, 357
  - edit\_backtrack, 357
- Distance based partitioning of the Secondary Structure Space, 223
- do\_backtrack
  - Manipulation of the Prediction Models, 116
- dupVar, 135
- duplexT, 135
- E\_ExtLoop
  - Processing and Evaluating Decomposed Loops, 65
- E\_Hairpin
  - Processing and Evaluating Decomposed Loops, 69
- E\_IntLoop
  - Processing and Evaluating Decomposed Loops, 71
- E\_Stem
  - Processing and Evaluating Decomposed Loops, 66
- E\_mb\_loop\_stack
  - Processing and Evaluating Decomposed Loops, 73
- edit\_backtrack
  - dist\_vars.h, 357
- encode\_ali\_sequence
  - Predicting Consensus Structures from Alignment(s), 195
- Energy Parameter Sets and Boltzmann Factors, 75
  - get\_boltzmann\_factor\_copy, 84
  - get\_boltzmann\_factors, 83
  - get\_boltzmann\_factors\_ali, 84
  - get\_scaled\_alipf\_parameters, 84
  - get\_scaled\_parameters, 85
  - get\_scaled\_pf\_parameters, 83
  - paramT, 78
  - pf\_paramT, 78
  - scale\_parameters, 85
  - vrna\_exp\_params, 79
  - vrna\_exp\_params\_comparative, 80
  - vrna\_exp\_params\_copy, 80
  - vrna\_exp\_params\_rescale, 81
  - vrna\_exp\_params\_reset, 83
  - vrna\_exp\_params\_subst, 81
  - vrna\_params, 78
  - vrna\_params\_copy, 79
  - vrna\_params\_reset, 82
  - vrna\_params\_subst, 81
- energy\_const.h
  - BONUS, 360
  - FORBIDDEN, 360
  - GASCONST, 359
  - INF, 359
  - K0, 359
  - MAXLOOP, 360
  - NBPAIRS, 360
  - TURN, 360
- energy\_of\_alistruct
  - Predicting Consensus Structures from Alignment(s), 192
- energy\_of\_circ\_struct
  - Free Energy Evaluation for given Sequence / Structure Pairs, 62
- energy\_of\_circ\_struct\_par
  - Free Energy Evaluation for given Sequence / Structure Pairs, 57
- energy\_of\_circ\_structure
  - Free Energy Evaluation for given Sequence / Structure Pairs, 56
- energy\_of\_move
  - Free Energy Evaluation for given Sequence / Structure Pairs, 59
- energy\_of\_move\_pt
  - Free Energy Evaluation for given Sequence / Structure Pairs, 59
- energy\_of\_struct

- Free Energy Evaluation for given Sequence / Structure Pairs, [60](#)
- energy\_of\_struct\_par
  - Free Energy Evaluation for given Sequence / Structure Pairs, [56](#)
- energy\_of\_struct\_pt
  - Free Energy Evaluation for given Sequence / Structure Pairs, [61](#)
- energy\_of\_struct\_pt\_par
  - Free Energy Evaluation for given Sequence / Structure Pairs, [58](#)
- energy\_of\_structure
  - Free Energy Evaluation for given Sequence / Structure Pairs, [55](#)
- energy\_of\_structure\_pt
  - Free Energy Evaluation for given Sequence / Structure Pairs, [58](#)
- energy\_set
  - Manipulation of the Prediction Models, [116](#)
- Enumerating Suboptimal Structures, [162](#)
- exp\_E\_ExtLoop
  - Processing and Evaluating Decomposed Loops, [65](#)
- exp\_E\_Hairpin
  - Processing and Evaluating Decomposed Loops, [70](#)
- exp\_E\_IntLoop
  - Processing and Evaluating Decomposed Loops, [73](#)
- exp\_E\_Stem
  - Processing and Evaluating Decomposed Loops, [67](#)
- exp\_f
  - vrna\_sc\_s, [249](#)
- expHairpinEnergy
  - part\_func.h, [388](#)
- expLoopEnergy
  - part\_func.h, [388](#)
- expand\_Full
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [299](#)
- expand\_Shapiro
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [298](#)
- export\_alibppm
  - Partition Function and Base Pair Probabilities for Sequence Alignment(s), [204](#)
- export\_bppm
  - Computing Partition Functions and Pair Probabilities, [155](#)
- export\_circfold\_arrays
  - MFE Structures of single Nucleic Acid Sequences, [177](#)
- export\_circfold\_arrays\_par
  - MFE Structures of single Nucleic Acid Sequences, [177](#)
- export\_co\_bppm
  - part\_func\_co.h, [392](#)
- export\_cofold\_arrays
  - MFE Structures of two hybridized Sequences, [181](#)
- export\_cofold\_arrays\_gq
  - MFE Structures of two hybridized Sequences, [180](#)
- export\_fold\_arrays
  - MFE Structures of single Nucleic Acid Sequences, [176](#)
- export\_fold\_arrays\_par
  - MFE Structures of single Nucleic Acid Sequences, [176](#)
- f
  - vrna\_sc\_s, [249](#)
- FILENAME\_ID\_LENGTH
  - Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings, [292](#)
- FILENAME\_MAX\_LENGTH
  - Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings, [292](#)
- FORBIDDEN
  - energy\_const.h, [360](#)
- filecopy
  - utils.h, [416](#)
- final\_cost
  - Inverse Secondary Structure Prediction, [46](#)
- find\_saddle
  - Direct refolding paths between two secondary structures, [289](#)
- fold
  - MFE Structures of single Nucleic Acid Sequences, [175](#)
- fold\_par
  - MFE Structures of single Nucleic Acid Sequences, [174](#)
- fold\_vars.h
  - base\_pair, [368](#)
  - cut\_point, [368](#)
  - iindx, [368](#)
  - james\_rule, [367](#)
  - logML, [367](#)
  - pr, [368](#)
  - RibosumFile, [367](#)
- Free Energy Evaluation for given Sequence / Structure Pairs, [48](#)
  - energy\_of\_circ\_struct, [62](#)
  - energy\_of\_circ\_struct\_par, [57](#)
  - energy\_of\_circ\_structure, [56](#)
  - energy\_of\_move, [59](#)
  - energy\_of\_move\_pt, [59](#)
  - energy\_of\_struct, [60](#)
  - energy\_of\_struct\_par, [56](#)
  - energy\_of\_struct\_pt, [61](#)
  - energy\_of\_struct\_pt\_par, [58](#)
  - energy\_of\_structure, [55](#)
  - energy\_of\_structure\_pt, [58](#)
  - loop\_energy, [60](#)
  - vrna\_eval\_covar\_structure, [50](#)
  - vrna\_eval\_hp\_loop, [62](#)
  - vrna\_eval\_loop\_pt, [54](#)
  - vrna\_eval\_move, [54](#)
  - vrna\_eval\_move\_pt, [55](#)

- vrna\_eval\_structure, 49
- vrna\_eval\_structure\_pt, 52
- vrna\_eval\_structure\_pt\_simple, 52
- vrna\_eval\_structure\_pt\_simple\_verbose, 53
- vrna\_eval\_structure\_pt\_verbose, 53
- vrna\_eval\_structure\_simple, 50
- vrna\_eval\_structure\_simple\_verbose, 51
- vrna\_eval\_structure\_verbose, 51
- free\_alifold\_arrays
  - MFE Consensus Structures for Sequence Alignment(s), 199
- free\_alipf\_arrays
  - Partition Function and Base Pair Probabilities for Sequence Alignment(s), 204
- free\_arrays
  - MFE Structures of single Nucleic Acid Sequences, 176
- free\_auxdata
  - vrna\_fc\_s, 269
- free\_co\_arrays
  - MFE Structures of two hybridized Sequences, 180
- free\_co\_pf\_arrays
  - part\_func\_co.h, 392
- free\_data
  - vrna\_hc\_s, 240
- free\_path
  - Direct refolding paths between two secondary structures, 290
- free\_pf\_arrays
  - Computing Partition Functions and Pair Probabilities, 154
- free\_profile
  - profiledist.h, 400
- free\_sequence\_arrays
  - Predicting Consensus Structures from Alignment(s), 195
- free\_tree
  - treedist.h, 412
- Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 319
  - aliPS\_color\_aln, 321
  - gmlRNA, 323
  - PS\_dot\_plot, 326
  - PS\_dot\_plot\_list, 325
  - PS\_rna\_plot, 325
  - PS\_rna\_plot\_a, 325
  - PS\_rna\_plot\_a\_gquad, 325
  - rna\_plot\_type, 326
  - simple\_circplot\_coordinates, 322
  - simple\_xy\_coordinates, 321
  - ssv\_rna\_plot, 324
  - svg\_rna\_plot, 324
  - VRNA\_PLOT\_TYPE\_CIRCULAR, 321
  - VRNA\_PLOT\_TYPE\_NAVIEW, 321
  - VRNA\_PLOT\_TYPE\_SIMPLE, 321
  - vrna\_file\_PS\_rnaplot, 322
  - vrna\_file\_PS\_rnaplot\_a, 323
  - xrna\_plot, 324
- Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, 312
  - read\_record, 318
  - VRNA\_CONSTRAINT\_MULTILINE, 313
  - VRNA\_OPTION\_MULTILINE, 313
  - vrna\_extract\_record\_rest\_constraint, 317
  - vrna\_extract\_record\_rest\_structure, 316
  - vrna\_file\_SHAPE\_read, 317
  - vrna\_file\_bpseq, 314
  - vrna\_file\_connect, 314
  - vrna\_file\_constraints\_read, 317
  - vrna\_file\_fasta\_read\_record, 315
  - vrna\_file\_helixlist, 313
  - vrna\_file\_json, 314
- GASCONST
  - energy\_const.h, 359
- Generate soft constraints from data, 261
  - progress\_callback, 263
  - VRNA\_MINIMIZER\_CONJUGATE\_FR, 262
  - VRNA\_MINIMIZER\_CONJUGATE\_PR, 262
  - VRNA\_MINIMIZER\_STEEPEST\_DESCENT, 263
  - VRNA\_MINIMIZER\_VECTOR\_BFGS2, 263
  - VRNA\_MINIMIZER\_VECTOR\_BFGS, 262
  - VRNA\_OBJECTIVE\_FUNCTION\_ABSOLUTE, 262
  - VRNA\_OBJECTIVE\_FUNCTION\_QUADRATIC, 262
  - vrna\_sc\_minimize\_perturbation, 263
- get\_TwoDfold\_variables
  - Calculating MFE representatives of a Distance Based Partitioning, 228
- get\_TwoDpfold\_variables
  - 2Dpfold.h, 334
- get\_alipf\_arrays
  - Predicting Consensus Structures from Alignment(s), 193
- get\_boltzmann\_factor\_copy
  - Energy Parameter Sets and Boltzmann Factors, 84
- get\_boltzmann\_factors
  - Energy Parameter Sets and Boltzmann Factors, 83
- get\_boltzmann\_factors\_ali
  - Energy Parameter Sets and Boltzmann Factors, 84
- get\_centroid\_struct\_gquad\_pr
  - part\_func.h, 387
- get\_centroid\_struct\_pl
  - centroid.h, 344
- get\_centroid\_struct\_pr
  - centroid.h, 344
- get\_concentrations
  - part\_func\_co.h, 391
- get\_gquad\_matrix
  - Processing and Evaluating Decomposed Loops, 67
- get\_input\_line
  - Utilities, 142
- get\_line
  - Utilities, 142
- get\_monomere\_mfes
  - MFE Structures of two hybridized Sequences, 182

- get\_mpi
  - Predicting Consensus Structures from Alignment(s), [194](#)
- get\_path
  - Direct refolding paths between two secondary structures, [290](#)
- get\_pf\_arrays
  - Computing Partition Functions and Pair Probabilities, [156](#)
- get\_plist
  - part\_func\_co.h, [391](#)
- get\_scaled\_alipf\_parameters
  - Energy Parameter Sets and Boltzmann Factors, [84](#)
- get\_scaled\_parameters
  - Energy Parameter Sets and Boltzmann Factors, [85](#)
- get\_scaled\_pf\_parameters
  - Energy Parameter Sets and Boltzmann Factors, [83](#)
- give\_up
  - Inverse Secondary Structure Prediction, [46](#)
- gmIRNA
  - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [323](#)
- HairpinE
  - MFE Structures of single Nucleic Acid Sequences, [177](#)
- hamming
  - string\_utils.h, [405](#)
- hamming\_bound
  - string\_utils.h, [406](#)
- Hard Constraints, [237](#)
  - VRNA\_CONSTRAINT\_DB\_DEFAULT, [242](#)
  - VRNA\_CONSTRAINT\_DB\_DOT, [241](#)
  - VRNA\_CONSTRAINT\_DB\_ENFORCE\_BP, [240](#)
  - VRNA\_CONSTRAINT\_DB\_GQUAD, [242](#)
  - VRNA\_CONSTRAINT\_DB\_INTERMOL, [242](#)
  - VRNA\_CONSTRAINT\_DB\_INTRAMOL, [241](#)
  - VRNA\_CONSTRAINT\_DB\_PIPE, [240](#)
  - VRNA\_CONSTRAINT\_DB\_RND\_BRACK, [241](#)
  - VRNA\_CONSTRAINT\_DB\_X, [241](#)
  - VRNA\_CONSTRAINT\_DB, [240](#)
  - vrna\_callback\_hc\_evaluate, [243](#)
  - vrna\_hc\_add\_bp, [245](#)
  - vrna\_hc\_add\_bp\_nonspecific, [245](#)
  - vrna\_hc\_add\_from\_db, [246](#)
  - vrna\_hc\_add\_up, [244](#)
  - vrna\_hc\_add\_up\_batch, [244](#)
  - vrna\_hc\_free, [245](#)
  - vrna\_hc\_init, [244](#)
- INF
  - energy\_const.h, [359](#)
- id
  - vrna\_exp\_param\_s, [78](#)
- iindx
  - fold\_vars.h, [368](#)
- Incorporating ligands binding to specific sequence/structure motifs, [259](#)
  - vrna\_sc\_add\_hi\_motif, [259](#)
- Incorporating SHAPE reactivity data, [255](#)
  - vrna\_sc\_SHAPE\_to\_pr, [257](#)
  - vrna\_sc\_add\_SHAPE\_deigan, [256](#)
  - vrna\_sc\_add\_SHAPE\_deigan\_ali, [256](#)
  - vrna\_sc\_add\_SHAPE\_zarringhalam, [257](#)
- init\_co\_pf\_fold
  - part\_func\_co.h, [392](#)
- init\_pf\_fold
  - part\_func.h, [387](#)
- init\_pf\_foldLP
  - LPfold.h, [375](#)
- init\_rand
  - utils.h, [416](#)
- initialize\_cofold
  - MFE Structures of two hybridized Sequences, [182](#)
- initialize\_fold
  - MFE Structures of single Nucleic Acid Sequences, [177](#)
- int\_urn
  - utils.h, [416](#)
- interact, [134](#)
- inv\_verbose
  - Inverse Secondary Structure Prediction, [46](#)
- Inverse Secondary Structure Prediction, [45](#)
  - final\_cost, [46](#)
  - give\_up, [46](#)
  - inv\_verbose, [46](#)
  - inverse\_fold, [45](#)
  - inverse\_pf\_fold, [46](#)
- inverse\_fold
  - Inverse Secondary Structure Prediction, [45](#)
- inverse\_pf\_fold
  - Inverse Secondary Structure Prediction, [46](#)
- james\_rule
  - fold\_vars.h, [367](#)
- K0
  - energy\_const.h, [359](#)
- LIST, [327](#)
- LPfold.h
  - init\_pf\_foldLP, [375](#)
- LST\_BUCKET, [328](#)
- Lfold
  - Local MFE structure Prediction and Z-scores, [208](#)
- Lfoldz
  - Local MFE structure Prediction and Z-scores, [208](#)
- Local MFE consensus structures for Sequence Alignments, [214](#)
  - aliLfold, [214](#)
- Local MFE structure Prediction and Z-scores, [207](#)
  - Lfold, [208](#)
  - Lfoldz, [208](#)
  - vrna\_Lfold, [207](#)
  - vrna\_Lfoldz, [208](#)
  - vrna\_mfe\_window, [209](#)
  - vrna\_mfe\_window\_zscore, [209](#)
- logML

- fold\_vars.h, 367
- loop\_energy
  - Free Energy Evaluation for given Sequence / Structure Pairs, 60
- LoopEnergy
  - MFE Structures of single Nucleic Acid Sequences, 177
- MAXLOOP
  - energy\_const.h, 360
- MEA.h
  - MEA, 375
- MEA
  - MEA.h, 375
- MFE Consensus Structures for Sequence Alignment(s), 197
  - alifold, 198
  - circularifold, 199
  - free\_alifold\_arrays, 199
  - vrna\_alifold, 197
  - vrna\_circularifold, 198
- MFE Structures of single Nucleic Acid Sequences, 172
  - circfold, 175
  - export\_circfold\_arrays, 177
  - export\_circfold\_arrays\_par, 177
  - export\_fold\_arrays, 176
  - export\_fold\_arrays\_par, 176
  - fold, 175
  - fold\_par, 174
  - free\_arrays, 176
  - HairpinE, 177
  - initialize\_fold, 177
  - LoopEnergy, 177
  - update\_fold\_params, 176
  - update\_fold\_params\_par, 176
  - vrna\_circfold, 173
  - vrna\_fold, 173
- MFE Structures of two hybridized Sequences, 178
  - cofold, 179
  - cofold\_par, 180
  - export\_cofold\_arrays, 181
  - export\_cofold\_arrays\_gq, 180
  - free\_co\_arrays, 180
  - get\_monomere\_mfes, 182
  - initialize\_cofold, 182
  - update\_cofold\_params, 180
  - update\_cofold\_params\_par, 180
  - vrna\_cofold, 179
  - vrna\_mfe\_dimer, 182
- Make\_bp\_profile
  - profiledist.h, 400
- Make\_bp\_profile\_bppm
  - profiledist.h, 400
- make\_pair\_table
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 306
- make\_pair\_table\_snoop
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 307
- make\_referenceBP\_array
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 307
- Make\_swString
  - stringdist.h, 406
- make\_tree
  - treedist.h, 411
- Manipulation of the Prediction Models, 87
  - backtrack\_type, 117
  - canonicalBPonly, 116
  - dangles, 115
  - do\_backtrack, 116
  - energy\_set, 116
  - max\_bp\_span, 117
  - noLonelyPairs, 116
  - nonstandards, 117
  - pf\_scale, 115
  - set\_model\_details, 115
  - temperature, 115
  - tetra\_loop, 116
  - VRNA\_MODEL\_DEFAULT\_ALI\_CV\_FACT, 97
  - VRNA\_MODEL\_DEFAULT\_ALI\_NC\_FACT, 97
  - VRNA\_MODEL\_DEFAULT\_ALI\_OLD\_EN, 96
  - VRNA\_MODEL\_DEFAULT\_ALI\_RIBO, 96
  - VRNA\_MODEL\_DEFAULT\_BACKTRACK\_TYPE, 95
  - VRNA\_MODEL\_DEFAULT\_BACKTRACK, 95
  - VRNA\_MODEL\_DEFAULT\_BETA\_SCALE, 93
  - VRNA\_MODEL\_DEFAULT\_CIRC, 94
  - VRNA\_MODEL\_DEFAULT\_COMPUTE\_BPP, 95
  - VRNA\_MODEL\_DEFAULT\_DANGLES, 93
  - VRNA\_MODEL\_DEFAULT\_ENERGY\_SET, 95
  - VRNA\_MODEL\_DEFAULT\_GQUAD, 94
  - VRNA\_MODEL\_DEFAULT\_LOG\_ML, 96
  - VRNA\_MODEL\_DEFAULT\_MAX\_BP\_SPAN, 96
  - VRNA\_MODEL\_DEFAULT\_NO\_GU\_CLOSURE, 94
  - VRNA\_MODEL\_DEFAULT\_NO\_GU, 94
  - VRNA\_MODEL\_DEFAULT\_NO\_LP, 94
  - VRNA\_MODEL\_DEFAULT\_PF\_SCALE, 93
  - VRNA\_MODEL\_DEFAULT\_SPECIAL\_HP, 93
  - VRNA\_MODEL\_DEFAULT\_TEMPERATURE, 93
  - VRNA\_MODEL\_DEFAULT\_UNIQ\_ML, 95
  - VRNA\_MODEL\_DEFAULT\_WINDOW\_SIZE, 96
  - vrna\_md\_defaults\_backtrack, 108
  - vrna\_md\_defaults\_backtrack\_get, 108
  - vrna\_md\_defaults\_backtrack\_type, 108
  - vrna\_md\_defaults\_backtrack\_type\_get, 109
  - vrna\_md\_defaults\_betaScale, 99
  - vrna\_md\_defaults\_betaScale\_get, 99
  - vrna\_md\_defaults\_circ, 104
  - vrna\_md\_defaults\_circ\_get, 106
  - vrna\_md\_defaults\_compute\_bpp, 109
  - vrna\_md\_defaults\_compute\_bpp\_get, 109
  - vrna\_md\_defaults\_cv\_fact, 113
  - vrna\_md\_defaults\_cv\_fact\_get, 113
  - vrna\_md\_defaults\_dangles, 100
  - vrna\_md\_defaults\_dangles\_get, 100

- vrna\_md\_defaults\_energy\_set, [107](#)
- vrna\_md\_defaults\_energy\_set\_get, [107](#)
- vrna\_md\_defaults\_gquad, [106](#)
- vrna\_md\_defaults\_gquad\_get, [106](#)
- vrna\_md\_defaults\_logML\_get, [104](#)
- vrna\_md\_defaults\_logML, [104](#)
- vrna\_md\_defaults\_max\_bp\_span, [110](#)
- vrna\_md\_defaults\_max\_bp\_span\_get, [110](#)
- vrna\_md\_defaults\_min\_loop\_size, [110](#)
- vrna\_md\_defaults\_min\_loop\_size\_get, [111](#)
- vrna\_md\_defaults\_nc\_fact, [113](#)
- vrna\_md\_defaults\_nc\_fact\_get, [114](#)
- vrna\_md\_defaults\_noGU\_get, [103](#)
- vrna\_md\_defaults\_noGUclosure, [103](#)
- vrna\_md\_defaults\_noGUclosure\_get, [103](#)
- vrna\_md\_defaults\_noGU, [102](#)
- vrna\_md\_defaults\_noLP\_get, [102](#)
- vrna\_md\_defaults\_noLP, [102](#)
- vrna\_md\_defaults\_oldAliEn, [112](#)
- vrna\_md\_defaults\_oldAliEn\_get, [112](#)
- vrna\_md\_defaults\_reset, [98](#)
- vrna\_md\_defaults\_ribo, [112](#)
- vrna\_md\_defaults\_ribo\_get, [113](#)
- vrna\_md\_defaults\_sfact, [114](#)
- vrna\_md\_defaults\_sfact\_get, [114](#)
- vrna\_md\_defaults\_special\_hp, [100](#)
- vrna\_md\_defaults\_special\_hp\_get, [102](#)
- vrna\_md\_defaults\_temperature, [99](#)
- vrna\_md\_defaults\_temperature\_get, [99](#)
- vrna\_md\_defaults\_uniq\_ML\_get, [107](#)
- vrna\_md\_defaults\_uniq\_ML, [106](#)
- vrna\_md\_defaults\_window\_size, [111](#)
- vrna\_md\_defaults\_window\_size\_get, [111](#)
- vrna\_md\_option\_string, [98](#)
- vrna\_md\_set\_default, [97](#)
- vrna\_md\_update, [97](#)
- max\_bp\_span
  - Manipulation of the Prediction Models, [117](#)
- mean\_bp\_dist
  - part\_func.h, [387](#)
- mean\_bp\_distance
  - Computing Partition Functions and Pair Probabilities, [157](#)
- mean\_bp\_distance\_pr
  - Computing Partition Functions and Pair Probabilities, [157](#)
- min\_loop\_size
  - vrna\_md\_s, [92](#)
- n\_seq
  - vrna\_fc\_s, [271](#)
- NBPAIRS
  - energy\_const.h, [360](#)
- nc\_fact
  - Predicting Consensus Structures from Alignment(s), [196](#)
- noLonelyPairs
  - Manipulation of the Prediction Models, [116](#)
- node, [135](#)
- nonstandards
  - Manipulation of the Prediction Models, [117](#)
- nrrror
  - utils.h, [415](#)
- PAIR
  - Data Structures and Preprocessor Macros, [135](#)
- PS\_dot\_plot
  - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [326](#)
- PS\_dot\_plot\_list
  - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [325](#)
- PS\_rna\_plot
  - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [325](#)
- PS\_rna\_plot\_a
  - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [325](#)
- PS\_rna\_plot\_a\_gquad
  - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [325](#)
- pack\_structure
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [305](#)
- pair\_info
  - Various utilities for Sequence Alignments, and Comparative Structure Prediction, [310](#)
- paramT
  - Energy Parameter Sets and Boltzmann Factors, [78](#)
- parenthesis\_structure
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [308](#)
- parenthesis\_zuker
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [308](#)
- parse\_gquad
  - Processing and Evaluating Decomposed Loops, [67](#)
- parse\_structure
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [300](#)
- Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [295](#)
  - add\_root, [298](#)
  - alimake\_pair\_table, [307](#)
  - assign\_plist\_from\_db, [305](#)
  - b2HIT, [297](#)
  - b2Shapiro, [298](#)
  - b2C, [298](#)
  - bp\_distance, [307](#)
  - bppm\_symbol, [308](#)
  - bppm\_to\_structure, [308](#)
  - compute\_BPdifferences, [308](#)
  - copy\_pair\_table, [306](#)
  - expand\_Full, [299](#)
  - expand\_Shapiro, [298](#)
  - make\_pair\_table, [306](#)
  - make\_pair\_table\_snoop, [307](#)
  - make\_referenceBP\_array, [307](#)



- pack\_structure, 305
- parenthesis\_structure, 308
- parenthesis\_zucker, 308
- parse\_structure, 300
- unexpand\_Full, 299
- unexpand\_aligned\_F, 300
- unpack\_structure, 305
- unweight, 299
- vrna\_bp\_distance, 303
- vrna\_db\_from\_bp\_stack, 303
- vrna\_db\_from\_plist, 304
- vrna\_db\_from\_ptable, 302
- vrna\_db\_pack, 300
- vrna\_db\_unpack, 301
- vrna\_plist, 304
- vrna\_pt\_pk\_get, 301
- vrna\_pt\_snoop\_get, 302
- vrna\_ptable, 301
- vrna\_ptable\_copy, 302
- vrna\_refBPcnt\_matrix, 303
- vrna\_refBPdist\_matrix, 303
- Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings, 291
  - FILENAME\_ID\_LENGTH, 292
  - FILENAME\_MAX\_LENGTH, 292
  - vrna\_cut\_point\_insert, 293
  - vrna\_cut\_point\_remove, 294
  - vrna\_hamming\_distance, 292
  - vrna\_hamming\_distance\_bound, 293
  - vrna\_random\_string, 292
  - vrna\_seq\_toRNA, 293
  - vrna\_seq\_toupper, 293
- part\_func.h
  - centroid, 387
  - expHairpinEnergy, 388
  - expLoopEnergy, 388
  - get\_centroid\_struct\_gquad\_pr, 387
  - init\_pf\_fold, 387
  - mean\_bp\_dist, 387
  - stackProb, 387
  - vrna\_pf\_float\_precision, 387
- part\_func\_co.h
  - co\_pf\_fold, 390
  - co\_pf\_fold\_par, 390
  - compute\_probabilities, 391
  - export\_co\_bppm, 392
  - free\_co\_pf\_arrays, 392
  - get\_concentrations, 391
  - get\_plist, 391
  - init\_co\_pf\_fold, 392
  - update\_co\_pf\_params, 393
  - update\_co\_pf\_params\_par, 393
- Partition Function and Base Pair Probabilities for Sequence Alignment(s), 201
  - alipf\_circ\_fold, 203
  - alipf\_fold, 203
  - alipf\_fold\_par, 203
  - export\_alipf\_bppm, 204
  - free\_alipf\_arrays, 204
  - vrna\_pf\_alifold, 201
  - vrna\_pf\_circalifold, 202
- Partition Function for two hybridized Sequences, 184
  - vrna\_pf\_dimer, 186
  - vrna\_pf\_dimer\_concentrations, 186
  - vrna\_pf\_dimer\_probs, 186
- Partition Function for two hybridized Sequences as a stepwise Process, 188
  - pf\_interact, 189
  - pf\_unstru, 189
- Partition functions for locally stable secondary structures, 211
  - pfl\_fold, 212
  - putoutpU\_prob, 212
  - putoutpU\_prob\_bin, 213
  - update\_pf\_paramsLP, 211
- path\_t
  - Direct refolding paths between two secondary structures, 288
- pbacktrack
  - Stochastic backtracking in the Ensemble, 169
- pbacktrack\_circ
  - Stochastic backtracking in the Ensemble, 170
- pf\_circ\_fold
  - Computing Partition Functions and Pair Probabilities, 154
- pf\_fold
  - Computing Partition Functions and Pair Probabilities, 153
- pf\_fold\_par
  - Computing Partition Functions and Pair Probabilities, 152
- pf\_interact
  - Partition Function for two hybridized Sequences as a stepwise Process, 189
- pf\_paramT
  - Energy Parameter Sets and Boltzmann Factors, 78
- pf\_scale
  - Manipulation of the Prediction Models, 115
- pf\_unstru
  - Partition Function for two hybridized Sequences as a stepwise Process, 189
- pfl\_fold
  - Partition functions for locally stable secondary structures, 212
- plist
  - Data Structures and Preprocessor Macros, 135
- Postorder\_list, 328
- pr
  - fold\_vars.h, 368
- Predicting Consensus Structures from Alignment(s), 191
  - alloc\_sequence\_arrays, 195
  - cv\_fact, 196
  - encode\_alipf\_sequence, 195
  - energy\_of\_alistruct, 192
  - free\_sequence\_arrays, 195
  - get\_alipf\_arrays, 193

- get\_mpi, [194](#)
  - nc\_fact, [196](#)
  - update\_alifold\_params, [193](#)
  - vrna\_aln\_mpi, [194](#)
- Predicting Locally stable structures of large sequences, [206](#)
- print\_tty\_constraint
  - constraints\_hard.h, [350](#)
- print\_tty\_constraint\_full
  - constraints\_hard.h, [350](#)
- print\_tty\_input\_seq
  - utils.h, [415](#)
- print\_tty\_input\_seq\_str
  - utils.h, [415](#)
- Processing and Evaluating Decomposed Loops, [64](#)
  - backtrack\_GQuad\_IntLoop, [68](#)
  - backtrack\_GQuad\_IntLoop\_L, [68](#)
  - E\_ExtLoop, [65](#)
  - E\_Hairpin, [69](#)
  - E\_IntLoop, [71](#)
  - E\_Stem, [66](#)
  - E\_mb\_loop\_stack, [73](#)
  - exp\_E\_ExtLoop, [65](#)
  - exp\_E\_Hairpin, [70](#)
  - exp\_E\_IntLoop, [73](#)
  - exp\_E\_Stem, [67](#)
  - get\_gquad\_matrix, [67](#)
  - parse\_gquad, [67](#)
  - vrna\_BT\_hp\_loop, [71](#)
  - vrna\_BT\_mb\_loop, [73](#)
  - vrna\_E\_ext\_hp\_loop, [70](#)
  - vrna\_E\_hp\_loop, [70](#)
  - vrna\_exp\_E\_hp\_loop, [71](#)
- profile\_edit\_distance
  - profiledist.h, [400](#)
- profiledist.h
  - free\_profile, [400](#)
  - Make\_bp\_profile, [400](#)
  - Make\_bp\_profile\_bppm, [400](#)
  - profile\_edit\_distance, [400](#)
- progress\_callback
  - Generate soft constraints from data, [263](#)
- pscore
  - vrna\_fc\_s, [272](#)
- pscore\_pf\_compat
  - vrna\_fc\_s, [272](#)
- pctype
  - vrna\_fc\_s, [270](#)
- pctype\_pf\_compat
  - vrna\_fc\_s, [270](#)
- pu\_contrib, [134](#)
- pu\_out, [134](#)
- putoutpU\_prob
  - Partition functions for locally stable secondary structures, [212](#)
- putoutpU\_prob\_bin
  - Partition functions for locally stable secondary structures, [213](#)
- RNA Secondary Structure Prediction, [43](#)
- random\_string
  - string\_utils.h, [405](#)
- read\_parameter\_file
  - Reading/Writing Energy Parameter Sets from/to File, [215](#)
- read\_record
  - Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, [318](#)
- Reading/Writing Energy Parameter Sets from/to File, [215](#)
  - read\_parameter\_file, [215](#)
  - write\_parameter\_file, [216](#)
- Refolding paths between secondary structures, [47](#)
- RibosumFile
  - fold\_vars.h, [367](#)
- vrna\_plot\_type
  - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [326](#)
- S
  - vrna\_fc\_s, [271](#)
- S3
  - vrna\_fc\_s, [272](#)
- S5
  - vrna\_fc\_s, [272](#)
- S\_cons
  - vrna\_fc\_s, [271](#)
- sc
  - vrna\_fc\_s, [270](#)
- scale\_parameters
  - Energy Parameter Sets and Boltzmann Factors, [85](#)
- scs
  - vrna\_fc\_s, [272](#)
- sect
  - Data Structures and Preprocessor Macros, [136](#)
- sequence
  - vrna\_fc\_s, [269](#)
- sequence\_encoding
  - vrna\_fc\_s, [270](#)
- sequences
  - vrna\_fc\_s, [271](#)
- set\_model\_details
  - Manipulation of the Prediction Models, [115](#)
- simple\_circplot\_coordinates
  - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [322](#)
- simple\_xy\_coordinates
  - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [321](#)
- snoopT, [135](#)
- Soft Constraints, [247](#)
  - vrna\_callback\_sc\_backtrack, [250](#)
  - vrna\_callback\_sc\_energy, [249](#)
  - vrna\_callback\_sc\_exp\_energy, [250](#)
  - vrna\_sc\_add\_bp, [251](#)
  - vrna\_sc\_add\_bt, [253](#)
  - vrna\_sc\_add\_data, [253](#)



- vrna\_sc\_add\_exp\_f, 254
- vrna\_sc\_add\_f, 253
- vrna\_sc\_add\_up, 252
- vrna\_sc\_free, 252
- vrna\_sc\_init, 251
- vrna\_sc\_remove, 252
- space
  - utils.h, 415
- ssv\_rna\_plot
  - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 324
- st\_back
  - Stochastic backtracking in the Ensemble, 170
- stackProb
  - part\_func.h, 387
- stat\_cb
  - vrna\_fc\_s, 269
- Stochastic backtracking in the Ensemble, 168
  - pbacktrack, 169
  - pbacktrack\_circ, 170
  - st\_back, 170
  - vrna\_pbacktrack, 169
  - vrna\_pbacktrack5, 168
- Stochastic Backtracking of Consensus Structures from Sequence Alignment(s), 205
  - alipbacktrack, 205
- Stochastic Backtracking of Structures from Distance Based Partitioning, 234
  - vrna\_pbacktrack5\_TwoD, 235
  - vrna\_pbacktrack\_TwoD, 234
- str\_DNA2RNA
  - string\_utils.h, 405
- str\_uppercase
  - string\_utils.h, 405
- string\_edit\_distance
  - stringdist.h, 407
- string\_utils.h
  - hamming, 405
  - hamming\_bound, 406
  - random\_string, 405
  - str\_DNA2RNA, 405
  - str\_uppercase, 405
- stringdist.h
  - Make\_swString, 406
  - string\_edit\_distance, 407
- subopt
  - Suboptimal structures within an energy band around the MFE, 166
- subopt\_circ
  - Suboptimal structures within an energy band around the MFE, 166
- Suboptimal structures according to Zuker et al. 1989, 163
  - vrna\_subopt\_zuker, 163
  - zukersubopt, 164
  - zukersubopt\_par, 164
- Suboptimal structures within an energy band around the MFE, 165
  - subopt, 166
  - subopt\_circ, 166
  - vrna\_subopt, 165
- svg\_rna\_plot
  - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, 324
- swString, 328
- TURN
  - energy\_const.h, 360
- temperature
  - Manipulation of the Prediction Models, 115
- tetra\_loop
  - Manipulation of the Prediction Models, 116
- The Dynamic Programming Matrices, 280
  - VRNA\_MX\_2DFOLD, 284
  - VRNA\_MX\_DEFAULT, 284
  - VRNA\_MX\_WINDOW, 284
  - vrna\_mx\_add, 285
  - vrna\_mx\_mfe\_free, 285
  - vrna\_mx\_pf\_free, 286
  - vrna\_mx\_type\_e, 284
- The Fold Compound, 265
  - VRNA\_OPTION\_EVAL\_ONLY, 274
  - VRNA\_OPTION\_MFE, 274
  - VRNA\_OPTION\_PF, 274
  - VRNA\_STATUS\_MFE\_POST, 273
  - VRNA\_STATUS\_MFE\_PRE, 273
  - VRNA\_STATUS\_PF\_POST, 273
  - VRNA\_STATUS\_PF\_PRE, 273
  - VRNA\_VC\_TYPE\_ALIGNMENT, 275
  - VRNA\_VC\_TYPE\_SINGLE, 275
  - vrna\_callback\_free\_auxdata, 274
  - vrna\_callback\_recursion\_status, 275
  - vrna\_fc\_type\_e, 275
  - vrna\_fold\_compound, 275
  - vrna\_fold\_compound\_add\_auxdata, 278
  - vrna\_fold\_compound\_add\_callback, 278
  - vrna\_fold\_compound\_comparative, 276
  - vrna\_fold\_compound\_free, 277
- time\_stamp
  - utils.h, 416
- Tree, 328
- tree\_edit\_distance
  - treedist.h, 412
- treedist.h
  - free\_tree, 412
  - make\_tree, 411
  - tree\_edit\_distance, 412
- TwoDfold\_backtrack\_f5
  - Calculating MFE representatives of a Distance Based Partitioning, 230
- TwoDfold\_vars, 225
  - Calculating MFE representatives of a Distance Based Partitioning, 227
- TwoDfoldList
  - Calculating MFE representatives of a Distance Based Partitioning, 229
- TwoDpfold\_pbacktrack

- 2Dpfold.h, 335
- TwoDpfold\_pbacktrack5
  - 2Dpfold.h, 336
- TwoDpfold\_vars, 329
- TwoDpfoldList
  - 2Dpfold.h, 334
- type
  - vrna\_fc\_s, 269
- unexpand\_Full
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 299
- unexpand\_aligned\_F
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 300
- unpack\_structure
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 305
- unweight
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 299
- update\_alifold\_params
  - Predicting Consensus Structures from Alignment(s), 193
- update\_co\_pf\_params
  - part\_func\_co.h, 393
- update\_co\_pf\_params\_par
  - part\_func\_co.h, 393
- update\_cofold\_params
  - MFE Structures of two hybridized Sequences, 180
- update\_cofold\_params\_par
  - MFE Structures of two hybridized Sequences, 180
- update\_fold\_params
  - MFE Structures of single Nucleic Acid Sequences, 176
- update\_fold\_params\_par
  - MFE Structures of single Nucleic Acid Sequences, 176
- update\_pf\_params
  - Computing Partition Functions and Pair Probabilities, 155
- update\_pf\_params\_par
  - Computing Partition Functions and Pair Probabilities, 155
- update\_pf\_paramsLP
  - Partition functions for locally stable secondary structures, 211
- urn
  - utils.h, 416
- Utilities, 137
  - get\_input\_line, 142
  - get\_line, 142
  - VRNA\_INPUT\_CONSTRAINT, 139
  - VRNA\_INPUT\_FASTA\_HEADER, 139
  - vrna\_alloc, 139
  - vrna\_idx\_col\_wise, 144
  - vrna\_idx\_row\_wise, 143
  - vrna\_int\_urn, 141
  - vrna\_message\_error, 140
  - vrna\_message\_input\_seq, 143
  - vrna\_message\_input\_seq\_simple, 143
  - vrna\_message\_warning, 140
  - vrna\_realloc, 140
  - vrna\_time\_stamp, 141
  - vrna\_urn, 141
  - xsubi, 144
- utils.h
  - filecopy, 416
  - init\_rand, 416
  - int\_urn, 416
  - nrerror, 415
  - print\_tty\_input\_seq, 415
  - print\_tty\_input\_seq\_str, 415
  - space, 415
  - time\_stamp, 416
  - urn, 416
  - warn\_user, 415
  - xrealloc, 416
- VRNA\_CONSTRAINT\_DB\_ANG\_BRACK
  - constraints\_hard.h, 349
- VRNA\_CONSTRAINT\_DB\_DEFAULT
  - Hard Constraints, 242
- VRNA\_CONSTRAINT\_DB\_DOT
  - Hard Constraints, 241
- VRNA\_CONSTRAINT\_DB\_ENFORCE\_BP
  - Hard Constraints, 240
- VRNA\_CONSTRAINT\_DB\_GQUAD
  - Hard Constraints, 242
- VRNA\_CONSTRAINT\_DB\_INTERMOL
  - Hard Constraints, 242
- VRNA\_CONSTRAINT\_DB\_INTRAMOL
  - Hard Constraints, 241
- VRNA\_CONSTRAINT\_DB\_PIPE
  - Hard Constraints, 240
- VRNA\_CONSTRAINT\_DB\_RND\_BRACK
  - Hard Constraints, 241
- VRNA\_CONSTRAINT\_DB\_X
  - Hard Constraints, 241
- VRNA\_CONSTRAINT\_DB
  - Hard Constraints, 240
- VRNA\_CONSTRAINT\_FILE
  - Constraining the Secondary Structure Predictions and Evaluations, 122
- VRNA\_CONSTRAINT\_MULTILINE
  - Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, 313
- VRNA\_CONSTRAINT\_NO\_HEADER
  - constraints\_hard.h, 349
- VRNA\_CONSTRAINT\_SOFT\_MFE
  - Constraining the Secondary Structure Predictions and Evaluations, 122
- VRNA\_CONSTRAINT\_SOFT\_PF
  - Constraining the Secondary Structure Predictions and Evaluations, 122
- VRNA\_CONVERT\_OUTPUT\_ALL
  - Converting Energy Parameter Files, 218

- VRNA\_CONVERT\_OUTPUT\_BULGE
  - Converting Energy Parameter Files, [219](#)
- VRNA\_CONVERT\_OUTPUT\_DANGLE3
  - Converting Energy Parameter Files, [219](#)
- VRNA\_CONVERT\_OUTPUT\_DANGLE5
  - Converting Energy Parameter Files, [219](#)
- VRNA\_CONVERT\_OUTPUT\_DUMP
  - Converting Energy Parameter Files, [220](#)
- VRNA\_CONVERT\_OUTPUT\_HP
  - Converting Energy Parameter Files, [218](#)
- VRNA\_CONVERT\_OUTPUT\_INT\_11
  - Converting Energy Parameter Files, [219](#)
- VRNA\_CONVERT\_OUTPUT\_INT\_21
  - Converting Energy Parameter Files, [219](#)
- VRNA\_CONVERT\_OUTPUT\_INT\_22
  - Converting Energy Parameter Files, [219](#)
- VRNA\_CONVERT\_OUTPUT\_INT
  - Converting Energy Parameter Files, [220](#)
- VRNA\_CONVERT\_OUTPUT\_MISC
  - Converting Energy Parameter Files, [220](#)
- VRNA\_CONVERT\_OUTPUT\_MM\_EXT
  - Converting Energy Parameter Files, [219](#)
- VRNA\_CONVERT\_OUTPUT\_MM\_HP
  - Converting Energy Parameter Files, [218](#)
- VRNA\_CONVERT\_OUTPUT\_MM\_INT\_1N
  - Converting Energy Parameter Files, [218](#)
- VRNA\_CONVERT\_OUTPUT\_MM\_INT\_23
  - Converting Energy Parameter Files, [218](#)
- VRNA\_CONVERT\_OUTPUT\_MM\_INT
  - Converting Energy Parameter Files, [218](#)
- VRNA\_CONVERT\_OUTPUT\_MM\_MULTI
  - Converting Energy Parameter Files, [219](#)
- VRNA\_CONVERT\_OUTPUT\_ML
  - Converting Energy Parameter Files, [220](#)
- VRNA\_CONVERT\_OUTPUT\_NINIO
  - Converting Energy Parameter Files, [220](#)
- VRNA\_CONVERT\_OUTPUT\_SPECIAL\_HP
  - Converting Energy Parameter Files, [220](#)
- VRNA\_CONVERT\_OUTPUT\_STACK
  - Converting Energy Parameter Files, [218](#)
- VRNA\_CONVERT\_OUTPUT\_VANILLA
  - Converting Energy Parameter Files, [220](#)
- VRNA\_DECOMP\_EXT\_EXT\_EXT
  - Constraining the Secondary Structure Predictions and Evaluations, [128](#)
- VRNA\_DECOMP\_EXT\_EXT\_STEM1
  - Constraining the Secondary Structure Predictions and Evaluations, [129](#)
- VRNA\_DECOMP\_EXT\_EXT\_STEM
  - Constraining the Secondary Structure Predictions and Evaluations, [128](#)
- VRNA\_DECOMP\_EXT\_EXT
  - Constraining the Secondary Structure Predictions and Evaluations, [127](#)
- VRNA\_DECOMP\_EXT\_STEM\_EXT
  - Constraining the Secondary Structure Predictions and Evaluations, [128](#)
- VRNA\_DECOMP\_EXT\_STEM
  - Constraining the Secondary Structure Predictions and Evaluations, [127](#)
- VRNA\_DECOMP\_EXT\_UP
  - Constraining the Secondary Structure Predictions and Evaluations, [127](#)
- VRNA\_DECOMP\_ML\_COAXIAL
  - Constraining the Secondary Structure Predictions and Evaluations, [126](#)
- VRNA\_DECOMP\_ML\_ML\_ML
  - Constraining the Secondary Structure Predictions and Evaluations, [124](#)
- VRNA\_DECOMP\_ML\_ML\_STEM
  - Constraining the Secondary Structure Predictions and Evaluations, [126](#)
- VRNA\_DECOMP\_ML\_ML
  - Constraining the Secondary Structure Predictions and Evaluations, [125](#)
- VRNA\_DECOMP\_ML\_STEM
  - Constraining the Secondary Structure Predictions and Evaluations, [124](#)
- VRNA\_DECOMP\_ML\_UP
  - Constraining the Secondary Structure Predictions and Evaluations, [125](#)
- VRNA\_DECOMP\_PAIR\_HP
  - Constraining the Secondary Structure Predictions and Evaluations, [122](#)
- VRNA\_DECOMP\_PAIR\_IL
  - Constraining the Secondary Structure Predictions and Evaluations, [123](#)
- VRNA\_DECOMP\_PAIR\_ML
  - Constraining the Secondary Structure Predictions and Evaluations, [123](#)
- VRNA\_INPUT\_CONSTRAINT
  - Utilities, [139](#)
- VRNA\_INPUT\_FASTA\_HEADER
  - Utilities, [139](#)
- VRNA\_MINIMIZER\_CONJUGATE\_FR
  - Generate soft constraints from data, [262](#)
- VRNA\_MINIMIZER\_CONJUGATE\_PR
  - Generate soft constraints from data, [262](#)
- VRNA\_MINIMIZER\_STEEPEST\_DESCENT
  - Generate soft constraints from data, [263](#)
- VRNA\_MINIMIZER\_VECTOR\_BFGS2
  - Generate soft constraints from data, [263](#)
- VRNA\_MINIMIZER\_VECTOR\_BFGS
  - Generate soft constraints from data, [262](#)
- VRNA\_MODEL\_DEFAULT\_ALI\_CV\_FACT
  - Manipulation of the Prediction Models, [97](#)
- VRNA\_MODEL\_DEFAULT\_ALI\_NC\_FACT
  - Manipulation of the Prediction Models, [97](#)
- VRNA\_MODEL\_DEFAULT\_ALI\_OLD\_EN
  - Manipulation of the Prediction Models, [96](#)
- VRNA\_MODEL\_DEFAULT\_ALI\_RIBO
  - Manipulation of the Prediction Models, [96](#)
- VRNA\_MODEL\_DEFAULT\_BACKTRACK\_TYPE
  - Manipulation of the Prediction Models, [95](#)
- VRNA\_MODEL\_DEFAULT\_BACKTRACK
  - Manipulation of the Prediction Models, [95](#)

- VRNA\_MODEL\_DEFAULT\_BETA\_SCALE
  - Manipulation of the Prediction Models, [93](#)
- VRNA\_MODEL\_DEFAULT\_CIRC
  - Manipulation of the Prediction Models, [94](#)
- VRNA\_MODEL\_DEFAULT\_COMPUTE\_BPP
  - Manipulation of the Prediction Models, [95](#)
- VRNA\_MODEL\_DEFAULT\_DANGLES
  - Manipulation of the Prediction Models, [93](#)
- VRNA\_MODEL\_DEFAULT\_ENERGY\_SET
  - Manipulation of the Prediction Models, [95](#)
- VRNA\_MODEL\_DEFAULT\_GQUAD
  - Manipulation of the Prediction Models, [94](#)
- VRNA\_MODEL\_DEFAULT\_LOG\_ML
  - Manipulation of the Prediction Models, [96](#)
- VRNA\_MODEL\_DEFAULT\_MAX\_BP\_SPAN
  - Manipulation of the Prediction Models, [96](#)
- VRNA\_MODEL\_DEFAULT\_NO\_GU\_CLOSURE
  - Manipulation of the Prediction Models, [94](#)
- VRNA\_MODEL\_DEFAULT\_NO\_GU
  - Manipulation of the Prediction Models, [94](#)
- VRNA\_MODEL\_DEFAULT\_NO\_LP
  - Manipulation of the Prediction Models, [94](#)
- VRNA\_MODEL\_DEFAULT\_PF\_SCALE
  - Manipulation of the Prediction Models, [93](#)
- VRNA\_MODEL\_DEFAULT\_SPECIAL\_HP
  - Manipulation of the Prediction Models, [93](#)
- VRNA\_MODEL\_DEFAULT\_TEMPERATURE
  - Manipulation of the Prediction Models, [93](#)
- VRNA\_MODEL\_DEFAULT\_UNIQ\_ML
  - Manipulation of the Prediction Models, [95](#)
- VRNA\_MODEL\_DEFAULT\_WINDOW\_SIZE
  - Manipulation of the Prediction Models, [96](#)
- VRNA\_MX\_2DFOLD
  - The Dynamic Programming Matrices, [284](#)
- VRNA\_MX\_DEFAULT
  - The Dynamic Programming Matrices, [284](#)
- VRNA\_MX\_WINDOW
  - The Dynamic Programming Matrices, [284](#)
- VRNA\_OBJECTIVE\_FUNCTION\_ABSOLUTE
  - Generate soft constraints from data, [262](#)
- VRNA\_OBJECTIVE\_FUNCTION\_QUADRATIC
  - Generate soft constraints from data, [262](#)
- VRNA\_OPTION\_EVAL\_ONLY
  - The Fold Compound, [274](#)
- VRNA\_OPTION\_MFE
  - The Fold Compound, [274](#)
- VRNA\_OPTION\_MULTILINE
  - Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, [313](#)
- VRNA\_OPTION\_PF
  - The Fold Compound, [274](#)
- VRNA\_PLOT\_TYPE\_CIRCULAR
  - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [321](#)
- VRNA\_PLOT\_TYPE\_NAVIEW
  - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [321](#)
- VRNA\_PLOT\_TYPE\_SIMPLE
  - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [321](#)
- VRNA\_STATUS\_MFE\_POST
  - The Fold Compound, [273](#)
- VRNA\_STATUS\_MFE\_PRE
  - The Fold Compound, [273](#)
- VRNA\_STATUS\_PF\_POST
  - The Fold Compound, [273](#)
- VRNA\_STATUS\_PF\_PRE
  - The Fold Compound, [273](#)
- VRNA\_VC\_TYPE\_ALIGNMENT
  - The Fold Compound, [275](#)
- VRNA\_VC\_TYPE\_SINGLE
  - The Fold Compound, [275](#)
- Various utilities for Sequence Alignments, and Comparative Structure Prediction, [309](#)
  - pair\_info, [310](#)
  - vrna\_aln\_pinfo, [310](#)
- ViennaRNA/1.8.4\_epars.h, [331](#)
- ViennaRNA/1.8.4\_intloops.h, [331](#)
- ViennaRNA/2Dfold.h, [332](#)
- ViennaRNA/2Dpfold.h, [333](#)
- ViennaRNA/LPfold.h, [374](#)
- ViennaRNA/Lfold.h, [372](#)
- ViennaRNA/MEA.h, [375](#)
- ViennaRNA/PS\_dot.h, [401](#)
- ViennaRNA/RNAstruct.h, [402](#)
- ViennaRNA/alifold.h, [336](#)
- ViennaRNA/aln\_util.h, [338](#)
- ViennaRNA/alphabet.h, [339](#)
- ViennaRNA/boltzmann\_sampling.h, [341](#)
- ViennaRNA/centroid.h, [343](#)
- ViennaRNA/cofold.h, [344](#)
- ViennaRNA/constraints.h, [345](#)
- ViennaRNA/constraints\_SHAPE.h, [350](#)
- ViennaRNA/constraints\_hard.h, [347](#)
- ViennaRNA/constraints\_soft.h, [352](#)
- ViennaRNA/convert\_epars.h, [353](#)
- ViennaRNA/data\_structures.h, [354](#)
- ViennaRNA/dist\_vars.h, [356](#)
- ViennaRNA/dp\_matrices.h, [357](#)
- ViennaRNA/duplex.h, [358](#)
- ViennaRNA/edit\_cost.h, [358](#)
- ViennaRNA/energy\_const.h, [359](#)
- ViennaRNA/eval.h, [360](#)
- ViennaRNA/exterior\_loops.h, [362](#)
- ViennaRNA/file\_formats.h, [363](#)
- ViennaRNA/findpath.h, [364](#)
- ViennaRNA/fold.h, [365](#)
- ViennaRNA/fold\_vars.h, [366](#)
- ViennaRNA/gquad.h, [368](#)
- ViennaRNA/hairpin\_loops.h, [369](#)
- ViennaRNA/interior\_loops.h, [370](#)
- ViennaRNA/inverse.h, [371](#)
- ViennaRNA/ligand.h, [373](#)
- ViennaRNA/loop\_energies.h, [373](#)
- ViennaRNA/mfe.h, [376](#)

- ViennaRNA/mm.h, [376](#)
- ViennaRNA/model.h, [377](#)
- ViennaRNA/multibranch\_loops.h, [381](#)
- ViennaRNA/naview.h, [383](#)
- ViennaRNA/params.h, [383](#)
- ViennaRNA/part\_func.h, [385](#)
- ViennaRNA/part\_func\_co.h, [388](#)
- ViennaRNA/part\_func\_up.h, [393](#)
- ViennaRNA/perturbation\_fold.h, [394](#)
- ViennaRNA/plot\_aln.h, [396](#)
- ViennaRNA/plot\_layouts.h, [396](#)
- ViennaRNA/plot\_structure.h, [398](#)
- ViennaRNA/profiledist.h, [399](#)
- ViennaRNA/read\_epars.h, [401](#)
- ViennaRNA/ribo.h, [402](#)
- ViennaRNA/string\_utils.h, [404](#)
- ViennaRNA/stringdist.h, [406](#)
- ViennaRNA/structure\_utils.h, [407](#)
- ViennaRNA/subopt.h, [410](#)
- ViennaRNA/treedist.h, [411](#)
- ViennaRNA/utis.h, [412](#)
- vrna\_BT\_hp\_loop
  - Processing and Evaluating Decomposed Loops, [71](#)
- vrna\_BT\_mb\_loop
  - Processing and Evaluating Decomposed Loops, [73](#)
- vrna\_E\_ext\_hp\_loop
  - Processing and Evaluating Decomposed Loops, [70](#)
- vrna\_E\_hp\_loop
  - Processing and Evaluating Decomposed Loops, [70](#)
- vrna\_Lfold
  - Local MFE structure Prediction and Z-scores, [207](#)
- vrna\_Lfoldz
  - Local MFE structure Prediction and Z-scores, [208](#)
- vrna\_alifold
  - MFE Consensus Structures for Sequence Alignment(s), [197](#)
- vrna\_alloc
  - Utilities, [139](#)
- vrna\_aln\_mpi
  - Predicting Consensus Structures from Alignment(s), [194](#)
- vrna\_aln\_pinfo
  - Various utilities for Sequence Alignments, and Comparative Structure Prediction, [310](#)
- vrna\_backtrack5\_TwoD
  - Calculating MFE representatives of a Distance Based Partitioning, [228](#)
- vrna\_basepair\_s, [133](#)
- vrna\_bp\_distance
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [303](#)
- vrna\_bp\_stack\_s, [134](#)
- vrna\_callback\_free\_auxdata
  - The Fold Compound, [274](#)
- vrna\_callback\_hc\_evaluate
  - Hard Constraints, [243](#)
- vrna\_callback\_recursion\_status
  - The Fold Compound, [275](#)
- vrna\_callback\_sc\_backtrack
  - Soft Constraints, [250](#)
- vrna\_callback\_sc\_energy
  - Soft Constraints, [249](#)
- vrna\_callback\_sc\_exp\_energy
  - Soft Constraints, [250](#)
- vrna\_centroid
  - Compute the centroid structure, [160](#)
- vrna\_centroid\_from\_plist
  - Compute the centroid structure, [160](#)
- vrna\_centroid\_from\_probs
  - Compute the centroid structure, [161](#)
- vrna\_circalifold
  - MFE Consensus Structures for Sequence Alignment(s), [198](#)
- vrna\_circfold
  - MFE Structures of single Nucleic Acid Sequences, [173](#)
- vrna\_cofold
  - MFE Structures of two hybridized Sequences, [179](#)
- vrna\_constraints\_add
  - Constraining the Secondary Structure Predictions and Evaluations, [129](#)
- vrna\_cpair\_s, [133](#)
- vrna\_cut\_point\_insert
  - Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings, [293](#)
- vrna\_cut\_point\_remove
  - Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings, [294](#)
- vrna\_db\_from\_bp\_stack
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [303](#)
- vrna\_db\_from\_plist
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [304](#)
- vrna\_db\_from\_ptable
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [302](#)
- vrna\_db\_pack
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [300](#)
- vrna\_db\_unpack
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, [301](#)
- vrna\_dimer\_conc\_s, [185](#)
- vrna\_dimer\_pf\_s, [185](#)
- vrna\_eval\_covar\_structure
  - Free Energy Evaluation for given Sequence / Structure Pairs, [50](#)
- vrna\_eval\_hp\_loop
  - Free Energy Evaluation for given Sequence / Structure Pairs, [62](#)
- vrna\_eval\_loop\_pt
  - Free Energy Evaluation for given Sequence / Structure Pairs, [54](#)

- vrna\_eval\_move
  - Free Energy Evaluation for given Sequence / Structure Pairs, [54](#)
- vrna\_eval\_move\_pt
  - Free Energy Evaluation for given Sequence / Structure Pairs, [55](#)
- vrna\_eval\_structure
  - Free Energy Evaluation for given Sequence / Structure Pairs, [49](#)
- vrna\_eval\_structure\_pt
  - Free Energy Evaluation for given Sequence / Structure Pairs, [52](#)
- vrna\_eval\_structure\_pt\_simple
  - Free Energy Evaluation for given Sequence / Structure Pairs, [52](#)
- vrna\_eval\_structure\_pt\_simple\_verbose
  - Free Energy Evaluation for given Sequence / Structure Pairs, [53](#)
- vrna\_eval\_structure\_pt\_verbose
  - Free Energy Evaluation for given Sequence / Structure Pairs, [53](#)
- vrna\_eval\_structure\_simple
  - Free Energy Evaluation for given Sequence / Structure Pairs, [50](#)
- vrna\_eval\_structure\_simple\_verbose
  - Free Energy Evaluation for given Sequence / Structure Pairs, [51](#)
- vrna\_eval\_structure\_verbose
  - Free Energy Evaluation for given Sequence / Structure Pairs, [51](#)
- vrna\_exp\_E\_hp\_loop
  - Processing and Evaluating Decomposed Loops, [71](#)
- vrna\_exp\_param\_s, [77](#)
  - alpha, [78](#)
  - id, [78](#)
- vrna\_exp\_params
  - Energy Parameter Sets and Boltzmann Factors, [79](#)
- vrna\_exp\_params\_comparative
  - Energy Parameter Sets and Boltzmann Factors, [80](#)
- vrna\_exp\_params\_copy
  - Energy Parameter Sets and Boltzmann Factors, [80](#)
- vrna\_exp\_params\_rescale
  - Energy Parameter Sets and Boltzmann Factors, [81](#)
- vrna\_exp\_params\_reset
  - Energy Parameter Sets and Boltzmann Factors, [83](#)
- vrna\_exp\_params\_subst
  - Energy Parameter Sets and Boltzmann Factors, [81](#)
- vrna\_extract\_record\_rest\_constraint
  - Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, [317](#)
- vrna\_extract\_record\_rest\_structure
  - Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, [316](#)
- vrna\_fc\_s, [266](#)
  - auxdata, [269](#)
  - cons\_seq, [271](#)
  - free\_auxdata, [269](#)
  - n\_seq, [271](#)
  - pscore, [272](#)
  - pscore\_pf\_compat, [272](#)
  - ptype, [270](#)
  - ptype\_pf\_compat, [270](#)
  - S, [271](#)
  - S3, [272](#)
  - S5, [272](#)
  - S\_cons, [271](#)
  - sc, [270](#)
  - scs, [272](#)
  - sequence, [269](#)
  - sequence\_encoding, [270](#)
  - sequences, [271](#)
  - stat\_cb, [269](#)
  - type, [269](#)
- vrna\_fc\_type\_e
  - The Fold Compound, [275](#)
- vrna\_file\_PS\_rnaplot
  - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [322](#)
- vrna\_file\_PS\_rnaplot\_a
  - Functions for Creating RNA Secondary Structures Plots, Dot-Plots, and More, [323](#)
- vrna\_file\_SHAPE\_read
  - Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, [317](#)
- vrna\_file\_bpseq
  - Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, [314](#)
- vrna\_file\_connect
  - Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, [314](#)
- vrna\_file\_constraints\_read
  - Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, [317](#)
- vrna\_file\_fasta\_read\_record
  - Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, [315](#)
- vrna\_file\_helixlist
  - Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, [313](#)
- vrna\_file\_json
  - Functions to Read/Write several File Formats for RNA Sequences, Structures, and Alignments, [314](#)
- vrna\_fold
  - MFE Structures of single Nucleic Acid Sequences, [173](#)
- vrna\_fold\_compound
  - The Fold Compound, [275](#)



- `vrna_fold_compound_add_auxdata`
  - The Fold Compound, [278](#)
- `vrna_fold_compound_add_callback`
  - The Fold Compound, [278](#)
- `vrna_fold_compound_comparative`
  - The Fold Compound, [276](#)
- `vrna_fold_compound_free`
  - The Fold Compound, [277](#)
- `vrna_hamming_distance`
  - Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings, [292](#)
- `vrna_hamming_distance_bound`
  - Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings, [293](#)
- `vrna_hc_add_bp`
  - Hard Constraints, [245](#)
- `vrna_hc_add_bp_nonspecific`
  - Hard Constraints, [245](#)
- `vrna_hc_add_from_db`
  - Hard Constraints, [246](#)
- `vrna_hc_add_up`
  - Hard Constraints, [244](#)
- `vrna_hc_add_up_batch`
  - Hard Constraints, [244](#)
- `vrna_hc_free`
  - Hard Constraints, [245](#)
- `vrna_hc_init`
  - Hard Constraints, [244](#)
- `vrna_hc_s`, [239](#)
  - `free_data`, [240](#)
- `vrna_hc_up_s`, [240](#)
- `vrna_hx_s`, [297](#)
- `vrna_idx_col_wise`
  - Utilities, [144](#)
- `vrna_idx_row_wise`
  - Utilities, [143](#)
- `vrna_int_urn`
  - Utilities, [141](#)
- `vrna_md_defaults_backtrack`
  - Manipulation of the Prediction Models, [108](#)
- `vrna_md_defaults_backtrack_get`
  - Manipulation of the Prediction Models, [108](#)
- `vrna_md_defaults_backtrack_type`
  - Manipulation of the Prediction Models, [108](#)
- `vrna_md_defaults_backtrack_type_get`
  - Manipulation of the Prediction Models, [109](#)
- `vrna_md_defaults_betaScale`
  - Manipulation of the Prediction Models, [99](#)
- `vrna_md_defaults_betaScale_get`
  - Manipulation of the Prediction Models, [99](#)
- `vrna_md_defaults_circ`
  - Manipulation of the Prediction Models, [104](#)
- `vrna_md_defaults_circ_get`
  - Manipulation of the Prediction Models, [106](#)
- `vrna_md_defaults_compute_bpp`
  - Manipulation of the Prediction Models, [109](#)
- `vrna_md_defaults_compute_bpp_get`
  - Manipulation of the Prediction Models, [109](#)
- `vrna_md_defaults_cv_fact`
  - Manipulation of the Prediction Models, [113](#)
- `vrna_md_defaults_cv_fact_get`
  - Manipulation of the Prediction Models, [113](#)
- `vrna_md_defaults_dangles`
  - Manipulation of the Prediction Models, [100](#)
- `vrna_md_defaults_dangles_get`
  - Manipulation of the Prediction Models, [100](#)
- `vrna_md_defaults_energy_set`
  - Manipulation of the Prediction Models, [107](#)
- `vrna_md_defaults_energy_set_get`
  - Manipulation of the Prediction Models, [107](#)
- `vrna_md_defaults_gquad`
  - Manipulation of the Prediction Models, [106](#)
- `vrna_md_defaults_gquad_get`
  - Manipulation of the Prediction Models, [106](#)
- `vrna_md_defaults_logML_get`
  - Manipulation of the Prediction Models, [104](#)
- `vrna_md_defaults_logML`
  - Manipulation of the Prediction Models, [104](#)
- `vrna_md_defaults_max_bp_span`
  - Manipulation of the Prediction Models, [110](#)
- `vrna_md_defaults_max_bp_span_get`
  - Manipulation of the Prediction Models, [110](#)
- `vrna_md_defaults_min_loop_size`
  - Manipulation of the Prediction Models, [110](#)
- `vrna_md_defaults_min_loop_size_get`
  - Manipulation of the Prediction Models, [111](#)
- `vrna_md_defaults_nc_fact`
  - Manipulation of the Prediction Models, [113](#)
- `vrna_md_defaults_nc_fact_get`
  - Manipulation of the Prediction Models, [114](#)
- `vrna_md_defaults_noGU_get`
  - Manipulation of the Prediction Models, [103](#)
- `vrna_md_defaults_noGUclosure`
  - Manipulation of the Prediction Models, [103](#)
- `vrna_md_defaults_noGUclosure_get`
  - Manipulation of the Prediction Models, [103](#)
- `vrna_md_defaults_noGU`
  - Manipulation of the Prediction Models, [102](#)
- `vrna_md_defaults_noLP_get`
  - Manipulation of the Prediction Models, [102](#)
- `vrna_md_defaults_noLP`
  - Manipulation of the Prediction Models, [102](#)
- `vrna_md_defaults_oldAliEn`
  - Manipulation of the Prediction Models, [112](#)
- `vrna_md_defaults_oldAliEn_get`
  - Manipulation of the Prediction Models, [112](#)
- `vrna_md_defaults_reset`
  - Manipulation of the Prediction Models, [98](#)
- `vrna_md_defaults_ribo`
  - Manipulation of the Prediction Models, [112](#)
- `vrna_md_defaults_ribo_get`
  - Manipulation of the Prediction Models, [113](#)
- `vrna_md_defaults_sfact`
  - Manipulation of the Prediction Models, [114](#)

- vrna\_md\_defaults\_sfact\_get
  - Manipulation of the Prediction Models, [114](#)
- vrna\_md\_defaults\_special\_hp
  - Manipulation of the Prediction Models, [100](#)
- vrna\_md\_defaults\_special\_hp\_get
  - Manipulation of the Prediction Models, [102](#)
- vrna\_md\_defaults\_temperature
  - Manipulation of the Prediction Models, [99](#)
- vrna\_md\_defaults\_temperature\_get
  - Manipulation of the Prediction Models, [99](#)
- vrna\_md\_defaults\_uniq\_ML\_get
  - Manipulation of the Prediction Models, [107](#)
- vrna\_md\_defaults\_uniq\_ML
  - Manipulation of the Prediction Models, [106](#)
- vrna\_md\_defaults\_window\_size
  - Manipulation of the Prediction Models, [111](#)
- vrna\_md\_defaults\_window\_size\_get
  - Manipulation of the Prediction Models, [111](#)
- vrna\_md\_option\_string
  - Manipulation of the Prediction Models, [98](#)
- vrna\_md\_s, [91](#)
  - dangles, [92](#)
  - min\_loop\_size, [92](#)
- vrna\_md\_set\_default
  - Manipulation of the Prediction Models, [97](#)
- vrna\_md\_update
  - Manipulation of the Prediction Models, [97](#)
- vrna\_mean\_bp\_distance
  - Computing Partition Functions and Pair Probabilities, [151](#)
- vrna\_mean\_bp\_distance\_pr
  - Computing Partition Functions and Pair Probabilities, [150](#)
- vrna\_message\_constraint\_options
  - Constraining the Secondary Structure Predictions and Evaluations, [130](#)
- vrna\_message\_constraint\_options\_all
  - Constraining the Secondary Structure Predictions and Evaluations, [131](#)
- vrna\_message\_error
  - Utilities, [140](#)
- vrna\_message\_input\_seq
  - Utilities, [143](#)
- vrna\_message\_input\_seq\_simple
  - Utilities, [143](#)
- vrna\_message\_warning
  - Utilities, [140](#)
- vrna\_mfe
  - Computing Minimum Free Energy (MFE) Structures, [146](#)
- vrna\_mfe\_TwoD
  - Calculating MFE representatives of a Distance Based Partitioning, [227](#)
- vrna\_mfe\_dimer
  - MFE Structures of two hybridized Sequences, [182](#)
- vrna\_mfe\_window
  - Local MFE structure Prediction and Z-scores, [209](#)
- vrna\_mfe\_window\_zscore
  - Local MFE structure Prediction and Z-scores, [209](#)
- vrna\_mx\_add
  - The Dynamic Programming Matrices, [285](#)
- vrna\_mx\_mfe\_free
  - The Dynamic Programming Matrices, [285](#)
- vrna\_mx\_mfe\_s, [281](#)
- vrna\_mx\_pf\_free
  - The Dynamic Programming Matrices, [286](#)
- vrna\_mx\_pf\_s, [282](#)
- vrna\_mx\_type\_e
  - The Dynamic Programming Matrices, [284](#)
- vrna\_nucleotide\_decode
  - alphabet.h, [341](#)
- vrna\_nucleotide\_encode
  - alphabet.h, [340](#)
- vrna\_param\_s, [77](#)
- vrna\_params
  - Energy Parameter Sets and Boltzmann Factors, [78](#)
- vrna\_params\_copy
  - Energy Parameter Sets and Boltzmann Factors, [79](#)
- vrna\_params\_reset
  - Energy Parameter Sets and Boltzmann Factors, [82](#)
- vrna\_params\_subst
  - Energy Parameter Sets and Boltzmann Factors, [81](#)
- vrna\_path\_findpath
  - Direct refolding paths between two secondary structures, [289](#)
- vrna\_path\_findpath\_saddle
  - Direct refolding paths between two secondary structures, [288](#)
- vrna\_path\_s, [288](#)
- vrna\_pbacktrack
  - Stochastic backtracking in the Ensemble, [169](#)
- vrna\_pbacktrack5
  - Stochastic backtracking in the Ensemble, [168](#)
- vrna\_pbacktrack5\_TwoD
  - Stochastic Backtracking of Structures from Distance Based Partitioning, [235](#)
- vrna\_pbacktrack\_TwoD
  - Stochastic Backtracking of Structures from Distance Based Partitioning, [234](#)
- vrna\_pf
  - Computing Partition Functions and Pair Probabilities, [149](#)
- vrna\_pf\_TwoD
  - Calculate Partition Functions of a Distance Based Partitioning, [232](#)
- vrna\_pf\_alifold
  - Partition Function and Base Pair Probabilities for Sequence Alignment(s), [201](#)
- vrna\_pf\_circalifold
  - Partition Function and Base Pair Probabilities for Sequence Alignment(s), [202](#)
- vrna\_pf\_circfold
  - Computing Partition Functions and Pair Probabilities, [150](#)
- vrna\_pf\_dimer



- Partition Function for two hybridized Sequences, 186
- vrna\_pf\_dimer\_concentrations
  - Partition Function for two hybridized Sequences, 186
- vrna\_pf\_dimer\_probs
  - Partition Function for two hybridized Sequences, 186
- vrna\_pf\_float\_precision
  - part\_func.h, 387
- vrna\_pf\_fold
  - Computing Partition Functions and Pair Probabilities, 149
- vrna\_pinfo\_s, 309
- vrna\_plist
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 304
- vrna\_plist\_from\_probs
  - Computing Partition Functions and Pair Probabilities, 157
- vrna\_plist\_s, 133
- vrna\_pt\_pk\_get
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 301
- vrna\_pt\_snoop\_get
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 302
- vrna\_ptable
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 301
- vrna\_ptable\_copy
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 302
- vrna\_ptypes
  - alphabet.h, 340
- vrna\_random\_string
  - Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings, 292
- vrna\_realloc
  - Utilities, 140
- vrna\_refBPcnt\_matrix
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 303
- vrna\_refBPdist\_matrix
  - Parsing, Converting, and Comparing - Functions to Manipulate Secondary Structures, 303
- vrna\_sc\_SHAPE\_parse\_method
  - constraints\_SHAPE.h, 351
- vrna\_sc\_SHAPE\_to\_pr
  - Incorporating SHAPE reactivity data, 257
- vrna\_sc\_add\_SHAPE\_deigan
  - Incorporating SHAPE reactivity data, 256
- vrna\_sc\_add\_SHAPE\_deigan\_al
  - Incorporating SHAPE reactivity data, 256
- vrna\_sc\_add\_SHAPE\_zarringham
  - Incorporating SHAPE reactivity data, 257
- vrna\_sc\_add\_bp
  - Soft Constraints, 251
- vrna\_sc\_add\_bt
  - Soft Constraints, 253
- vrna\_sc\_add\_data
  - Soft Constraints, 253
- vrna\_sc\_add\_exp\_f
  - Soft Constraints, 254
- vrna\_sc\_add\_f
  - Soft Constraints, 253
- vrna\_sc\_add\_hi\_motif
  - Incorporating ligands binding to specific sequence/structure motifs, 259
- vrna\_sc\_add\_up
  - Soft Constraints, 252
- vrna\_sc\_free
  - Soft Constraints, 252
- vrna\_sc\_init
  - Soft Constraints, 251
- vrna\_sc\_minimize\_perturbation
  - Generate soft constraints from data, 263
- vrna\_sc\_remove
  - Soft Constraints, 252
- vrna\_sc\_s, 248
  - bt, 249
  - exp\_f, 249
  - f, 249
- vrna\_sect\_s, 134
- vrna\_seq\_toRNA
  - Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings, 293
- vrna\_seq\_toupper
  - Parsing, Converting, and Comparing - Functions to Manipulate Sequence and Structure Strings, 293
- vrna\_sol\_TwoD\_pf\_t, 232
  - Calculate Partition Functions of a Distance Based Partitioning, 232
- vrna\_sol\_TwoD\_t, 225
  - Calculating MFE representatives of a Distance Based Partitioning, 227
- vrna\_stack\_prob
  - Computing Partition Functions and Pair Probabilities, 151
- vrna\_subopt
  - Suboptimal structures within an energy band around the MFE, 165
- vrna\_subopt\_sol\_s, 330
- vrna\_subopt\_zuker
  - Suboptimal structures according to Zuker et al. 1989, 163
- vrna\_time\_stamp
  - Utilities, 141
- vrna\_urn
  - Utilities, 141
- warn\_user
  - utils.h, 415
- write\_parameter\_file

Reading/Writing Energy Parameter Sets from/to  
File, [216](#)

xrealloc  
utils.h, [416](#)

xrna\_plot  
Functions for Creating RNA Secondary Structures  
Plots, Dot-Plots, and More, [324](#)

xsubi  
Utilities, [144](#)

zuckersubopt  
Suboptimal structures according to Zuker et al.  
1989, [164](#)

zuckersubopt\_par  
Suboptimal structures according to Zuker et al.  
1989, [164](#)