Common Pipeline Library Reference

7.2.2

Generated by Doxygen 1.8.20

1 Deprecated List 1

2 Module Index 5

2.1 Modules . . . 5

3 Class Index 7

3.1 Class List . . . 7

4 Module Documentation 9

4.1 Arrays . . . 9

4.1.1 Detailed Description . . . 14

4.1.2 Function Documentation . . . 14

4.1.2.1 cpl_array_abs() . . . 14

4.1.2.2 cpl_array_add() . . . 15

4.1.2.3 cpl_array_add_scalar() . . . 15

4.1.2.4 cpl_array_add_scalar_complex() . . . 16

4.1.2.5 cpl_array_arg() . . . 17

4.1.2.6 cpl_array_cast() . . . 17

4.1.2.7 cpl_array_copy_data() . . . 18

4.1.2.8 cpl_array_copy_data_complex() . . . 19

4.1.2.9 cpl_array_copy_data_cplsize() . . . 19

4.1.2.10 cpl_array_copy_data_double() . . . 20

4.1.2.11 cpl_array_copy_data_double_complex() . . . 20

4.1.2.12 cpl_array_copy_data_float() . . . 21

4.1.2.13 cpl_array_copy_data_float_complex() . . . 21

4.1.2.14 cpl_array_copy_data_int() . . . 22

4.1.2.15 cpl_array_copy_data_long() . . . 22

4.1.2.16 cpl_array_copy_data_long_long() . . . 23

4.1.2.17 cpl_array_copy_data_string() . . . 23

4.1.2.18 cpl_array_count_invalid() . . . 24

4.1.2.19 cpl_array_delete() . . . 24

4.1.2.20 cpl_array_divide() . . . 24

4.1.2.21 cpl_array_divide_scalar() . . . 25

4.1.2.22 cpl_array_divide_scalar_complex() . . . 26

4.1.2.23 cpl_array_dump() . . . 26

4.1.2.24 cpl_array_dump_structure() . . . 27

4.1.2.25 cpl_array_duplicate() . . . 28

4.1.2.26 cpl_array_erase_window() . . . 28

4.1.2.27 cpl_array_exponential() . . . 29

4.1.2.28 cpl_array_extract() . . . 30

4.1.2.29 cpl_array_extract_imag() . . . 30

4.1.2.30 cpl_array_extract_real() . . . 31

4.1.2.31 cpl_array_fill_window() . . . 32

4.1.2.32 cpl_array_fill_window_complex() . . . 32
4.1.2.33 cpl_array_fill_window_cplsize() . . . 33
4.1.2.34 cpl_array_fill_window_double() . . . 33
4.1.2.35 cpl_array_fill_window_double_complex() . . . 35
4.1.2.36 cpl_array_fill_window_float() . . . 36
4.1.2.37 cpl_array_fill_window_float_complex() . . . 36
4.1.2.38 cpl_array_fill_window_int() . . . 37
4.1.2.39 cpl_array_fill_window_invalid() . . . 38
4.1.2.40 cpl_array_fill_window_long() . . . 38
4.1.2.41 cpl_array_fill_window_long_long() . . . 39
4.1.2.42 cpl_array_fill_window_string() . . . 40
4.1.2.43 cpl_array_get() . . . 40
4.1.2.44 cpl_array_get_complex() . . . 41
4.1.2.45 cpl_array_get_cplsize() . . . 41
4.1.2.46 cpl_array_get_data_cplsize() . . . 42
4.1.2.47 cpl_array_get_data_cplsize_const() . . . 43
4.1.2.48 cpl_array_get_data_double() . . . 43
4.1.2.49 cpl_array_get_data_double_complex() . . . 43
4.1.2.50 cpl_array_get_data_double_complex_const() . . . 44
4.1.2.51 cpl_array_get_data_double_const() . . . 44
4.1.2.52 cpl_array_get_data_float() . . . 45
4.1.2.53 cpl_array_get_data_float_complex() . . . 45
4.1.2.54 cpl_array_get_data_float_complex_const() . . . 46
4.1.2.55 cpl_array_get_data_float_const() . . . 46
4.1.2.56 cpl_array_get_data_int() . . . 47
4.1.2.57 cpl_array_get_data_int_const() . . . 47
4.1.2.58 cpl_array_get_data_long() . . . 48
4.1.2.59 cpl_array_get_data_long_const() . . . 48
4.1.2.60 cpl_array_get_data_long_long() . . . 49
4.1.2.61 cpl_array_get_data_long_long_const() . . . 49
4.1.2.62 cpl_array_get_data_string() . . . 50
4.1.2.63 cpl_array_get_data_string_const() . . . 50
4.1.2.64 cpl_array_get_double() . . . 51
4.1.2.65 cpl_array_get_double_complex() . . . 51
4.1.2.66 cpl_array_get_float() . . . 52
4.1.2.67 cpl_array_get_float_complex() . . . 52
4.1.2.68 cpl_array_get_int() . . . 53
4.1.2.69 cpl_array_get_long() . . . 54
4.1.2.70 cpl_array_get_long_long() . . . 54
4.1.2.71 cpl_array_get_max() . . . 55
4.1.2.72 cpl_array_get_maxpos() . . . 55
4.1.2.73 cpl_array_get_mean() . . . 56

4.1.2.74 cpl_array_get_mean_complex() . . . 57
4.1.2.75 cpl_array_get_median() . . . 57
4.1.2.76 cpl_array_get_min() . . . 58
4.1.2.77 cpl_array_get_minpos() . . . 59
4.1.2.78 cpl_array_get_size() . . . 59
4.1.2.79 cpl_array_get_stdev() . . . 60
4.1.2.80 cpl_array_get_string() . . . 60
4.1.2.81 cpl_array_get_type() . . . 61
4.1.2.82 cpl_array_has_invalid() . . . 61
4.1.2.83 cpl_array_has_valid() . . . 62
4.1.2.84 cpl_array_insert() . . . 62
4.1.2.85 cpl_array_insert_window() . . . 63
4.1.2.86 cpl_array_is_valid() . . . 64
4.1.2.87 cpl_array_logarithm() . . . 64
4.1.2.88 cpl_array_multiply() . . . 65
4.1.2.89 cpl_array_multiply_scalar() . . . 66
4.1.2.90 cpl_array_multiply_scalar_complex() . . . 66
4.1.2.91 cpl_array_new() . . . 67
4.1.2.92 cpl_array_power() . . . 68
4.1.2.93 cpl_array_power_complex() . . . 68
4.1.2.94 cpl_array_set() . . . 69
4.1.2.95 cpl_array_set_complex() . . . 70
4.1.2.96 cpl_array_set_cplsize() . . . 70
4.1.2.97 cpl_array_set_double() . . . 71
4.1.2.98 cpl_array_set_double_complex() . . . 71
4.1.2.99 cpl_array_set_float() . . . 72
4.1.2.100 cpl_array_set_float_complex() . . . 72
4.1.2.101 cpl_array_set_int() . . . 73
4.1.2.102 cpl_array_set_invalid() . . . 74
4.1.2.103 cpl_array_set_long() . . . 74
4.1.2.104 cpl_array_set_long_long() . . . 75
4.1.2.105 cpl_array_set_size() . . . 75
4.1.2.106 cpl_array_set_string() . . . 76
4.1.2.107 cpl_array_subtract() . . . 76
4.1.2.108 cpl_array_subtract_scalar() . . . 77
4.1.2.109 cpl_array_subtract_scalar_complex() . . . 78
4.1.2.110 cpl_array_unwrap() . . . 78
4.1.2.111 cpl_array_wrap_cplsize() . . . 79
4.1.2.112 cpl_array_wrap_double() . . . 80
4.1.2.113 cpl_array_wrap_double_complex() . . . 81
4.1.2.114 cpl_array_wrap_float() . . . 81
4.1.2.115 cpl_array_wrap_float_complex() . . . 82

4.1.2.116 cpl_array_wrap_int() . . . 82
4.1.2.117 cpl_array_wrap_long() . . . 83
4.1.2.118 cpl_array_wrap_long_long() . . . 84
4.1.2.119 cpl_array_wrap_string() . . . 84
4.2 Auxiliary Frame Data . . . 86
4.2.1 Detailed Description . . . 87
4.2.2 Typedef Documentation . . . 87
4.2.2.1 cpl_framedata . . . 87
4.2.3 Function Documentation . . . 87
4.2.3.1 cpl_framedata_clear() . . . 87
4.2.3.2 cpl_framedata_create() . . . 88
4.2.3.3 cpl_framedata_delete() . . . 88
4.2.3.4 cpl_framedata_duplicate() . . . 89
4.2.3.5 cpl_framedata_get_max_count() . . . 89
4.2.3.6 cpl_framedata_get_min_count() . . . 91
4.2.3.7 cpl_framedata_get_tag() . . . 91
4.2.3.8 cpl_framedata_new() . . . 92
4.2.3.9 cpl_framedata_set() . . . 92
4.2.3.10 cpl_framedata_set_max_count() . . . 93
4.2.3.11 cpl_framedata_set_min_count() . . . 94
4.2.3.12 cpl_framedata_set_tag() . . . 94
4.3 Bi-vector object . . . 96
4.3.1 Detailed Description . . . 96
4.3.2 Function Documentation . . . 97
4.3.2.1 cpl_bivector_copy() . . . 97
4.3.2.2 cpl_bivector_delete() . . . 97
4.3.2.3 cpl_bivector_dump() . . . 98
4.3.2.4 cpl_bivector_duplicate() . . . 98
4.3.2.5 cpl_bivector_get_size() . . . 99
4.3.2.6 cpl_bivector_get_x() . . . 99
4.3.2.7 cpl_bivector_get_x_const() . . . 100
4.3.2.8 cpl_bivector_get_x_data() . . . 100
4.3.2.9 cpl_bivector_get_x_data_const() . . . 101
4.3.2.10 cpl_bivector_get_y() . . . 102
4.3.2.11 cpl_bivector_get_y_const() . . . 102
4.3.2.12 cpl_bivector_get_y_data() . . . 103
4.3.2.13 cpl_bivector_get_y_data_const() . . . 103
4.3.2.14 cpl_bivector_interpolate_linear() . . . 104
4.3.2.15 cpl_bivector_new() . . . 105
4.3.2.16 cpl_bivector_read() . . . 105
4.3.2.17 cpl_bivector_sort() . . . 106
4.3.2.18 cpl_bivector_unwrap_vectors() . . . 107

4.3.2.19 cpl_bivector_wrap_vectors() . . . 107
4.4 DFS related functions . . . 109
4.4.1 Detailed Description . . . 110
4.4.2 Macro Definition Documentation . . . 110
4.4.2.1 CPL_DFS_PRO_CATG . . . 110
4.4.2.2 CPL_DFS_PRO_SCIENCE . . . 110
4.4.2.3 CPL_DFS_PRO_TECH . . . 110
4.4.2.4 CPL_DFS_PRO_TYPE . . . 111
4.4.3 Enumeration Type Documentation . . . 111
4.4.3.1 anonymous enum . . . 111
4.4.4 Function Documentation . . . 111
4.4.4.1 cpl_dfs_save_imagelist() . . . 111
4.4.4.2 cpl_dfs_save_paf() . . . 112
4.4.4.3 cpl_dfs_save_propertylist() . . . 113
4.4.4.4 cpl_dfs_save_table() . . . 114
4.4.4.5 cpl_dfs_setup_product_header() . . . 115
4.4.4.6 cpl_dfs_sign_products() . . . 116
4.4.4.7 cpl_dfs_update_product_header() . . . 117
4.5 DICB specific property functionality . . . 119
4.5.1 Detailed Description . . . 119
4.6 Error handling . . . 120
4.6.1 Detailed Description . . . 121
4.6.2 Macro Definition Documentation . . . 121
4.6.2.1 cpl_ensure . . . 122
4.6.2.2 cpl_ensure_code . . . 122
4.6.2.3 cpl_error_ensure . . . 122
4.6.2.4 CPL_ERROR_MAX_MESSAGE_LENGTH . . . 124
4.6.2.5 cpl_error_set . . . 124
4.6.2.6 cpl_error_set_message . . . 124
4.6.2.7 cpl_error_set_where . . . 125
4.6.2.8 CPL_HAVE_VA_ARGS . . . 126
4.6.3 Typedef Documentation . . . 126
4.6.3.1 cpl_error_code . . . 126
4.6.4 Enumeration Type Documentation . . . 126
4.6.4.1 _cpl_error_code_ . . . 127
4.6.5 Function Documentation . . . 127
4.6.5.1 cpl_error_get_code() . . . 128
4.6.5.2 cpl_error_get_file() . . . 128
4.6.5.3 cpl_error_get_function() . . . 128
4.6.5.4 cpl_error_get_line() . . . 129
4.6.5.5 cpl_error_get_message() . . . 129
4.6.5.6 cpl_error_get_message_default() . . . 129

4.6.5.7 cpl_error_get_where() . . . 130
4.7 FFTW wrappers . . . 131
4.7.1 Detailed Description . . . 131
4.7.2 Typedef Documentation . . . 131
4.7.2.1 cpl_fft_mode . . . 131
4.7.3 Enumeration Type Documentation . . . 131
4.7.3.1 _cpl_fft_mode_ . . . 131
4.7.4 Function Documentation . . . 132
4.7.4.1 cpl_fft_image() . . . 132
4.7.4.2 cpl_fft_imagelist() . . . 133
4.8 FITS card related basic routines . . . 134
4.8.1 Detailed Description . . . 134
4.9 FITS related basic routines . . . 135
4.9.1 Detailed Description . . . 135
4.9.2 Typedef Documentation . . . 135
4.9.2.1 cpl_fits_mode . . . 135
4.9.3 Enumeration Type Documentation . . . 136
4.9.3.1 _cpl_fits_mode_ . . . 136
4.9.4 Function Documentation . . . 136
4.9.4.1 cpl_fits_count_extensions() . . . 136
4.9.4.2 cpl_fits_find_extension() . . . 137
4.9.4.3 cpl_fits_get_extension_nb() . . . 137
4.9.4.4 cpl_fits_get_mode() . . . 138
4.9.4.5 cpl_fits_get_nb_extensions() . . . 138
4.9.4.6 cpl_fits_set_mode() . . . 139
4.10 Filters . . . 141
4.10.1 Detailed Description . . . 141
4.10.2 Typedef Documentation . . . 141
4.10.2.1 cpl_border_mode . . . 142
4.10.2.2 cpl_filter_mode . . . 142
4.10.3 Enumeration Type Documentation . . . 142
4.10.3.1 _cpl_border_mode_ . . . 142
4.10.3.2 _cpl_filter_mode_ . . . 142
4.11 Frame Set Iterators . . . 147
4.11.1 Detailed Description . . . 147
4.11.2 Typedef Documentation . . . 147
4.11.2.1 cpl_frameset_iterator . . . 148
4.11.3 Function Documentation . . . 148
4.11.3.1 cpl_frameset_iterator_advance() . . . 148
4.11.3.2 cpl_frameset_iterator_assign() . . . 149
4.11.3.3 cpl_frameset_iterator_delete() . . . 149
4.11.3.4 cpl_frameset_iterator_distance() . . . 150

4.11.3.5 cpl_frameset_iterator_duplicate() . . . 150
4.11.3.6 cpl_frameset_iterator_get() . . . 151
4.11.3.7 cpl_frameset_iterator_get_const() . . . 151
4.11.3.8 cpl_frameset_iterator_new() . . . 152
4.11.3.9 cpl_frameset_iterator_reset() . . . 153
4.12 Frame Sets . . . 154
4.12.1 Detailed Description . . . 155
4.12.2 Typedef Documentation . . . 155
4.12.2.1 cpl_frameset . . . 155
4.12.3 Function Documentation . . . 155
4.12.3.1 cpl_frameset_count_tags() . . . 155
4.12.3.2 cpl_frameset_delete() . . . 156
4.12.3.3 cpl_frameset_dump() . . . 156
4.12.3.4 cpl_frameset_duplicate() . . . 157
4.12.3.5 cpl_frameset_erase() . . . 158
4.12.3.6 cpl_frameset_erase_frame() . . . 158
4.12.3.7 cpl_frameset_extract() . . . 159
4.12.3.8 cpl_frameset_find() . . . 160
4.12.3.9 cpl_frameset_find_const() . . . 160
4.12.3.10 cpl_frameset_get_first() . . . 161
4.12.3.11 cpl_frameset_get_first_const() . . . 162
4.12.3.12 cpl_frameset_get_frame() . . . 163
4.12.3.13 cpl_frameset_get_frame_const() . . . 163
4.12.3.14 cpl_frameset_get_next() . . . 164
4.12.3.15 cpl_frameset_get_next_const() . . . 165
4.12.3.16 cpl_frameset_get_position() . . . 166
4.12.3.17 cpl_frameset_get_position_const() . . . 167
4.12.3.18 cpl_frameset_get_size() . . . 167
4.12.3.19 cpl_frameset_insert() . . . 168
4.12.3.20 cpl_frameset_is_empty() . . . 169
4.12.3.21 cpl_frameset_join() . . . 169
4.12.3.22 cpl_frameset_labelise() . . . 170
4.12.3.23 cpl_frameset_new() . . . 171
4.12.3.24 cpl_frameset_sort() . . . 171
4.13 Frame Sets IO functions . . . 173
4.13.1 Detailed Description . . . 173
4.13.2 Function Documentation . . . 173
4.13.2.1 cpl_imagelist_load_frameset() . . . 173
4.14 Frames . . . 174
4.14.1 Detailed Description . . . 175
4.14.2 Macro Definition Documentation . . . 175
4.14.2.1 CPL_FRAME_GROUP_CALIB_ID . . . 176

4.14.2.2 CPL_FRAME_GROUP_PRODUCT_ID . . . 176
4.14.2.3 CPL_FRAME_GROUP_RAW_ID . . . 176
4.14.3 Typedef Documentation . . . 176
4.14.3.1 cpl_frame . . . 176
4.14.3.2 cpl_frame_compare_func . . . 176
4.14.3.3 cpl_frame_group . . . 177
4.14.3.4 cpl_frame_level . . . 177
4.14.3.5 cpl_frame_type . . . 177
4.14.4 Enumeration Type Documentation . . . 177
4.14.4.1 _cpl_frame_group_ . . . 177
4.14.4.2 _cpl_frame_level_ . . . 177
4.14.4.3 _cpl_frame_type_ . . . 178
4.14.5 Function Documentation . . . 178
4.14.5.1 cpl_frame_delete() . . . 178
4.14.5.2 cpl_frame_dump() . . . 179
4.14.5.3 cpl_frame_duplicate() . . . 179
4.14.5.4 cpl_frame_get_filename() . . . 180
4.14.5.5 cpl_frame_get_group() . . . 181
4.14.5.6 cpl_frame_get_level() . . . 181
4.14.5.7 cpl_frame_get_nextensions() . . . 182
4.14.5.8 cpl_frame_get_tag() . . . 182
4.14.5.9 cpl_frame_get_type() . . . 183
4.14.5.10 cpl_frame_new() . . . 184
4.14.5.11 cpl_frame_set_filename() . . . 184
4.14.5.12 cpl_frame_set_group() . . . 185
4.14.5.13 cpl_frame_set_level() . . . 185
4.14.5.14 cpl_frame_set_tag() . . . 186
4.14.5.15 cpl_frame_set_type() . . . 186
4.15 Fundamental math functionality . . . 188
4.15.1 Detailed Description . . . 189
4.15.2 Macro Definition Documentation . . . 189
4.15.2.1 CPL_MATH_1_PI . . . 189
4.15.2.2 CPL_MATH_2_PI . . . 190
4.15.2.3 CPL_MATH_2_SQRTPI . . . 190
4.15.2.4 CPL_MATH_2PI . . . 190
4.15.2.5 CPL_MATH_4_PI . . . 191
4.15.2.6 CPL_MATH_DEG_RAD . . . 191
4.15.2.7 CPL_MATH_E . . . 191
4.15.2.8 CPL_MATH_FWHM_SIG . . . 192
4.15.2.9 CPL_MATH_LN10 . . . 192
4.15.2.10 CPL_MATH_LN2 . . . 192
4.15.2.11 CPL_MATH_LOG10E . . . 192

4.15.2.12 CPL_MATH_LOG2E . . . 193
4.15.2.13 CPL_MATH_PI . . . 193
4.15.2.14 CPL_MATH_PI_2 . . . 193
4.15.2.15 CPL_MATH_PI_4 . . . 194
4.15.2.16 CPL_MATH_RAD_DEG . . . 194
4.15.2.17 CPL_MATH_SIG_FWHM . . . 194
4.15.2.18 CPL_MATH_SQRT1_2 . . . 195
4.15.2.19 CPL_MATH_SQRT2 . . . 195
4.15.2.20 CPL_MATH_SQRT2PI . . . 195
4.15.2.21 CPL_MATH_SQRT3 . . . 196
4.15.2.22 CPL_MATH_STD_MAD . . . 196
4.15.2.23 CPL_MAX . . . 196
4.15.2.24 CPL_MIN . . . 197
4.16 Handling of multiple CPL errors . . . 198
4.16.1 Detailed Description . . . 198
4.16.2 Function Documentation . . . 198
4.16.2.1 cpl_errorstate_dump() . . . 198
4.16.2.2 cpl_errorstate_dump_one() . . . 199
4.16.2.3 cpl_errorstate_dump_one_debug() . . . 200
4.16.2.4 cpl_errorstate_dump_one_info() . . . 201
4.16.2.5 cpl_errorstate_dump_one_warning() . . . 201
4.16.2.6 cpl_errorstate_get() . . . 202
4.16.2.7 cpl_errorstate_is_equal() . . . 202
4.16.2.8 cpl_errorstate_set() . . . 203
4.17 High level functions for geometric transformations . . . 204
4.17.1 Detailed Description . . . 204
4.17.2 Enumeration Type Documentation . . . 204
4.17.2.1 cpl_geom_combine . . . 204
4.17.3 Function Documentation . . . 205
4.17.3.1 cpl_geom_img_offset_combine() . . . 205
4.17.3.2 cpl_geom_img_offset_fine() . . . 206
4.17.3.3 cpl_geom_img_offset_saa() . . . 207
4.18 High level functions to handle apertures . . . 210
4.18.1 Detailed Description . . . 211
4.18.2 Function Documentation . . . 211
4.18.2.1 cpl_apertures_delete() . . . 211
4.18.2.2 cpl_apertures_dump() . . . 212
4.18.2.3 cpl_apertures_extract() . . . 212
4.18.2.4 cpl_apertures_extract_mask() . . . 213
4.18.2.5 cpl_apertures_extract_sigma() . . . 214
4.18.2.6 cpl_apertures_extract_window() . . . 215
4.18.2.7 cpl_apertures_get_bottom() . . . 216

4.18.2.8 cpl_apertures_get_bottom_x() . . . 217
4.18.2.9 cpl_apertures_get_centroid_x() . . . 218
4.18.2.10 cpl_apertures_get_centroid_y() . . . 218
4.18.2.11 cpl_apertures_get_flux() . . . 219
4.18.2.12 cpl_apertures_get_fwhm() . . . 219
4.18.2.13 cpl_apertures_get_left() . . . 220
4.18.2.14 cpl_apertures_get_left_y() . . . 221
4.18.2.15 cpl_apertures_get_max() . . . 221
4.18.2.16 cpl_apertures_get_max_x() . . . 222
4.18.2.17 cpl_apertures_get_max_y() . . . 222
4.18.2.18 cpl_apertures_get_maxpos_x() . . . 223
4.18.2.19 cpl_apertures_get_maxpos_y() . . . 223
4.18.2.20 cpl_apertures_get_mean() . . . 224
4.18.2.21 cpl_apertures_get_median() . . . 225
4.18.2.22 cpl_apertures_get_min() . . . 225
4.18.2.23 cpl_apertures_get_minpos_x() . . . 226
4.18.2.24 cpl_apertures_get_minpos_y() . . . 226
4.18.2.25 cpl_apertures_get_npix() . . . 227
4.18.2.26 cpl_apertures_get_pos_x() . . . 227
4.18.2.27 cpl_apertures_get_pos_y() . . . 228
4.18.2.28 cpl_apertures_get_right() . . . 229
4.18.2.29 cpl_apertures_get_right_y() . . . 229
4.18.2.30 cpl_apertures_get_size() . . . 230
4.18.2.31 cpl_apertures_get_stdev() . . . 230
4.18.2.32 cpl_apertures_get_top() . . . 231
4.18.2.33 cpl_apertures_get_top_x() . . . 232
4.18.2.34 cpl_apertures_new_from_image() . . . 232
4.18.2.35 cpl_apertures_sort_by_flux() . . . 233
4.18.2.36 cpl_apertures_sort_by_max() . . . 234
4.18.2.37 cpl_apertures_sort_by_npix() . . . 234
4.19 High-level functions for non-linear fitting . . . 235
4.19.1 Detailed Description . . . 235
4.19.2 Function Documentation . . . 235
4.19.2.1 cpl_fit_image_gaussian() . . . 235
4.19.2.2 cpl_fit_imagelist_polynomial() . . . 238
4.19.2.3 cpl_fit_imagelist_polynomial_window() . . . 239
4.19.2.4 cpl_fit_lvmq() . . . 241
4.19.2.5 cpl_gaussian_eval_2d() . . . 242
4.20 High-level functions that are photometry related . . . 244
4.20.1 Detailed Description . . . 244
4.20.2 Function Documentation . . . 244
4.20.2.1 cpl_photom_fill_blackbody() . . . 244

4.21 High-level functions to compute detector features . . . 246
4.21.1 Detailed Description . . . 246
4.21.2 Function Documentation . . . 246
4.21.2.1 cpl_detector_interpolate_rejected() . . . 246
4.21.2.2 cpl_flux_get_bias_window() . . . 247
4.21.2.3 cpl_flux_get_noise_ring() . . . 248
4.21.2.4 cpl_flux_get_noise_window() . . . 248
4.22 I/O . . . 250
4.22.1 Detailed Description . . . 250
4.22.2 Macro Definition Documentation . . . 250
4.22.2.1 CPL_BPP_16_SIGNED . . . 250
4.22.2.2 CPL_BPP_16_UNSIGNED . . . 251
4.22.2.3 CPL_BPP_32_SIGNED . . . 251
4.22.2.4 CPL_BPP_8_UNSIGNED . . . 251
4.22.2.5 CPL_BPP_IEEE_DOUBLE . . . 251
4.22.2.6 CPL_BPP_IEEE_FLOAT . . . 251
4.22.2.7 cpl_type_bpp . . . 252
4.22.3 Typedef Documentation . . . 252
4.22.3.1 cpl_io_type . . . 252
4.22.4 Enumeration Type Documentation . . . 252
4.22.4.1 _cpl_io_type_ . . . 252
4.23 Imagelists . . . 253
4.23.1 Detailed Description . . . 254
4.23.2 Function Documentation . . . 255
4.23.2.1 cpl_image_new_from_accepted() . . . 255
4.23.2.2 cpl_imagelist_add() . . . 255
4.23.2.3 cpl_imagelist_add_image() . . . 256
4.23.2.4 cpl_imagelist_add_scalar() . . . 257
4.23.2.5 cpl_imagelist_cast() . . . 257
4.23.2.6 cpl_imagelist_collapse_create() . . . 258
4.23.2.7 cpl_imagelist_collapse_median_create() . . . 259
4.23.2.8 cpl_imagelist_collapse_minmax_create() . . . 259
4.23.2.9 cpl_imagelist_collapse_sigclip_create() . . . 260
4.23.2.10 cpl_imagelist_delete() . . . 262
4.23.2.11 cpl_imagelist_divide() . . . 262
4.23.2.12 cpl_imagelist_divide_image() . . . 263
4.23.2.13 cpl_imagelist_divide_scalar() . . . 263
4.23.2.14 cpl_imagelist_dump_structure() . . . 264
4.23.2.15 cpl_imagelist_dump_window() . . . 265
4.23.2.16 cpl_imagelist_duplicate() . . . 265
4.23.2.17 cpl_imagelist_empty() . . . 266
4.23.2.18 cpl_imagelist_erase() . . . 266

4.23.2.19 cpl_imagelist_exponential() . . . 267
4.23.2.20 cpl_imagelist_get() . . . 268
4.23.2.21 cpl_imagelist_get_const() . . . 268
4.23.2.22 cpl_imagelist_get_size() . . . 269
4.23.2.23 cpl_imagelist_is_uniform() . . . 270
4.23.2.24 cpl_imagelist_load() . . . 270
4.23.2.25 cpl_imagelist_load_window() . . . 271
4.23.2.26 cpl_imagelist_logarithm() . . . 272
4.23.2.27 cpl_imagelist_multiply() . . . 273
4.23.2.28 cpl_imagelist_multiply_image() . . . 273
4.23.2.29 cpl_imagelist_multiply_scalar() . . . 274
4.23.2.30 cpl_imagelist_new() . . . 274
4.23.2.31 cpl_imagelist_normalise() . . . 274
4.23.2.32 cpl_imagelist_power() . . . 275
4.23.2.33 cpl_imagelist_save() . . . 276
4.23.2.34 cpl_imagelist_set() . . . 277
4.23.2.35 cpl_imagelist_subtract() . . . 278
4.23.2.36 cpl_imagelist_subtract_image() . . . 278
4.23.2.37 cpl_imagelist_subtract_scalar() . . . 279
4.23.2.38 cpl_imagelist_swap_axis_create() . . . 279
4.23.2.39 cpl_imagelist_threshold() . . . 280
4.23.2.40 cpl_imagelist_unset() . . . 281
4.23.2.41 cpl_imagelist_unwrap() . . . 281
4.24 Images . . . 283
4.24.1 Detailed Description . . . 290
4.24.2 Typedef Documentation . . . 290
4.24.2.1 cpl_value . . . 290
4.24.3 Enumeration Type Documentation . . . 290
4.24.3.1 _cpl_value_ . . . 290
4.24.4 Function Documentation . . . 291
4.24.4.1 cpl_image_abs() . . . 291
4.24.4.2 cpl_image_abs_create() . . . 291
4.24.4.3 cpl_image_accept() . . . 292
4.24.4.4 cpl_image_accept_all() . . . 292
4.24.4.5 cpl_image_add() . . . 293
4.24.4.6 cpl_image_add_create() . . . 293
4.24.4.7 cpl_image_add_scalar() . . . 294
4.24.4.8 cpl_image_add_scalar_create() . . . 295
4.24.4.9 cpl_image_and() . . . 295
4.24.4.10 cpl_image_and_scalar() . . . 296
4.24.4.11 cpl_image_average_create() . . . 297
4.24.4.12 cpl_image_cast() . . . 297

4.24.4.13 cpl_image_collapse_create() . . . 298
4.24.4.14 cpl_image_collapse_median_create() . . . 299
4.24.4.15 cpl_image_collapse_window_create() . . . 300
4.24.4.16 cpl_image_conjugate() . . . 301
4.24.4.17 cpl_image_copy() . . . 301
4.24.4.18 cpl_image_count_rejected() . . . 302
4.24.4.19 cpl_image_delete() . . . 303
4.24.4.20 cpl_image_divide() . . . 303
4.24.4.21 cpl_image_divide_create() . . . 304
4.24.4.22 cpl_image_divide_scalar() . . . 305
4.24.4.23 cpl_image_divide_scalar_create() . . . 305
4.24.4.24 cpl_image_dump_structure() . . . 306
4.24.4.25 cpl_image_dump_window() . . . 307
4.24.4.26 cpl_image_duplicate() . . . 307
4.24.4.27 cpl_image_exponential() . . . 308
4.24.4.28 cpl_image_exponential_create() . . . 309
4.24.4.29 cpl_image_extract() . . . 309
4.24.4.30 cpl_image_extract_subsample() . . . 310
4.24.4.31 cpl_image_fft() . . . 311
4.24.4.32 cpl_image_fill_abs_arg() . . . 312
4.24.4.33 cpl_image_fill_gaussian() . . . 313
4.24.4.34 cpl_image_fill_jacobian() . . . 314
4.24.4.35 cpl_image_fill_jacobian_polynomial() . . . 315
4.24.4.36 cpl_image_fill_noise_uniform() . . . 316
4.24.4.37 cpl_image_fill_polynomial() . . . 317
4.24.4.38 cpl_image_fill_re_im() . . . 318
4.24.4.39 cpl_image_fill_rejected() . . . 319
4.24.4.40 cpl_image_fill_test_create() . . . 319
4.24.4.41 cpl_image_fill_window() . . . 320
4.24.4.42 cpl_image_filter() . . . 321
4.24.4.43 cpl_image_filter_linear() . . . 322
4.24.4.44 cpl_image_filter_mask() . . . 323
4.24.4.45 cpl_image_filter_median() . . . 325
4.24.4.46 cpl_image_filter_morpho() . . . 326
4.24.4.47 cpl_image_filter_stdev() . . . 327
4.24.4.48 cpl_image_fit_gaussian() . . . 327
4.24.4.49 cpl_image_flip() . . . 328
4.24.4.50 cpl_image_get() . . . 329
4.24.4.51 cpl_image_get_absflux() . . . 330
4.24.4.52 cpl_image_get_absflux_window() . . . 331
4.24.4.53 cpl_image_get_bpm() . . . 331
4.24.4.54 cpl_image_get_bpm_const() . . . 332

4.24.4.55 cpl_image_get_centroid_x() . . . 332
4.24.4.56 cpl_image_get_centroid_x_window() . . . 333
4.24.4.57 cpl_image_get_centroid_y() . . . 333
4.24.4.58 cpl_image_get_centroid_y_window() . . . 334
4.24.4.59 cpl_image_get_complex() . . . 334
4.24.4.60 cpl_image_get_data() . . . 335
4.24.4.61 cpl_image_get_data_const() . . . 336
4.24.4.62 cpl_image_get_data_double() . . . 336
4.24.4.63 cpl_image_get_data_double_complex() . . . 337
4.24.4.64 cpl_image_get_data_double_complex_const() . . . 338
4.24.4.65 cpl_image_get_data_double_const() . . . 338
4.24.4.66 cpl_image_get_data_float() . . . 338
4.24.4.67 cpl_image_get_data_float_complex() . . . 339
4.24.4.68 cpl_image_get_data_float_complex_const() . . . 339
4.24.4.69 cpl_image_get_data_float_const() . . . 340
4.24.4.70 cpl_image_get_data_int() . . . 340
4.24.4.71 cpl_image_get_data_int_const() . . . 341
4.24.4.72 cpl_image_get_flux() . . . 341
4.24.4.73 cpl_image_get_flux_window() . . . 342
4.24.4.74 cpl_image_get_fwhm() . . . 342
4.24.4.75 cpl_image_get_interpolated() . . . 343
4.24.4.76 cpl_image_get_mad() . . . 344
4.24.4.77 cpl_image_get_mad_window() . . . 345
4.24.4.78 cpl_image_get_max() . . . 346
4.24.4.79 cpl_image_get_max_window() . . . 346
4.24.4.80 cpl_image_get_maxpos() . . . 347
4.24.4.81 cpl_image_get_maxpos_window() . . . 348
4.24.4.82 cpl_image_get_mean() . . . 348
4.24.4.83 cpl_image_get_mean_window() . . . 349
4.24.4.84 cpl_image_get_median() . . . 349
4.24.4.85 cpl_image_get_median_dev() . . . 350
4.24.4.86 cpl_image_get_median_dev_window() . . . 351
4.24.4.87 cpl_image_get_median_window() . . . 351
4.24.4.88 cpl_image_get_min() . . . 352
4.24.4.89 cpl_image_get_min_window() . . . 353
4.24.4.90 cpl_image_get_minpos() . . . 354
4.24.4.91 cpl_image_get_minpos_window() . . . 354
4.24.4.92 cpl_image_get_size_x() . . . 355
4.24.4.93 cpl_image_get_size_y() . . . 356
4.24.4.94 cpl_image_get_sqflux() . . . 356
4.24.4.95 cpl_image_get_sqflux_window() . . . 357
4.24.4.96 cpl_image_get_stdev() . . . 357

4.24.4.97 cpl_image_get_stdev_window() . . . 358
4.24.4.98 cpl_image_get_type() . . . 358
4.24.4.99 cpl_image_hypot() . . . 359
4.24.4.100 cpl_image_iqe() . . . 360
4.24.4.101 cpl_image_is_rejected() . . . 361
4.24.4.102 cpl_image_labelise_mask_create() . . . 361
4.24.4.103 cpl_image_load() . . . 362
4.24.4.104 cpl_image_load_window() . . . 363
4.24.4.105 cpl_image_logarithm() . . . 364
4.24.4.106 cpl_image_logarithm_create() . . . 365
4.24.4.107 cpl_image_move() . . . 365
4.24.4.108 cpl_image_multiply() . . . 366
4.24.4.109 cpl_image_multiply_create() . . . 367
4.24.4.110 cpl_image_multiply_scalar() . . . 367
4.24.4.111 cpl_image_multiply_scalar_create() . . . 368
4.24.4.112 cpl_image_new() . . . 369
4.24.4.113 cpl_image_new_from_mask() . . . 369
4.24.4.114 cpl_image_normalise() . . . 370
4.24.4.115 cpl_image_normalise_create() . . . 371
4.24.4.116 cpl_image_not() . . . 371
4.24.4.117 cpl_image_or() . . . 372
4.24.4.118 cpl_image_or_scalar() . . . 373
4.24.4.119 cpl_image_power() . . . 373
4.24.4.120 cpl_image_power_create() . . . 374
4.24.4.121 cpl_image_rebin() . . . 375
4.24.4.122 cpl_image_reject() . . . 375
4.24.4.123 cpl_image_reject_from_mask() . . . 376
4.24.4.124 cpl_image_reject_value() . . . 377
4.24.4.125 cpl_image_save() . . . 377
4.24.4.126 cpl_image_set() . . . 378
4.24.4.127 cpl_image_set_bpm() . . . 379
4.24.4.128 cpl_image_set_complex() . . . 380
4.24.4.129 cpl_image_shift() . . . 381
4.24.4.130 cpl_image_subtract() . . . 381
4.24.4.131 cpl_image_subtract_create() . . . 382
4.24.4.132 cpl_image_subtract_scalar() . . . 382
4.24.4.133 cpl_image_subtract_scalar_create() . . . 383
4.24.4.134 cpl_image_threshold() . . . 383
4.24.4.135 cpl_image_turn() . . . 384
4.24.4.136 cpl_image_unset_bpm() . . . 385
4.24.4.137 cpl_image_unwrap() . . . 386
4.24.4.138 cpl_image_warp() . . . 387

4.24.4.139 cpl_image_warp_polynomial() . . . 388
4.24.4.140 cpl_image_wrap_double() . . . 389
4.24.4.141 cpl_image_wrap_double_complex() . . . 390
4.24.4.142 cpl_image_wrap_float() . . . 391
4.24.4.143 cpl_image_wrap_float_complex() . . . 391
4.24.4.144 cpl_image_wrap_int() . . . 392
4.24.4.145 cpl_image_xor() . . . 392
4.24.4.146 cpl_image_xor_scalar() . . . 393
4.24.4.147 cpl_vector_new_from_image_column() . . . 394
4.24.4.148 cpl_vector_new_from_image_row() . . . 394
4.25 Library Initialization . . . 396
4.25.1 Detailed Description . . . 396
4.25.2 Function Documentation . . . 396
4.25.2.1 cpl_end() . . . 396
4.25.2.2 cpl_get_description() . . . 396
4.25.2.3 cpl_init() . . . 397
4.26 Library Version Information . . . 398
4.26.1 Detailed Description . . . 398
4.26.2 Function Documentation . . . 398
4.26.2.1 cpl_version_get_binary_age() . . . 398
4.26.2.2 cpl_version_get_binary_version() . . . 399
4.26.2.3 cpl_version_get_interface_age() . . . 399
4.26.2.4 cpl_version_get_major() . . . 399
4.26.2.5 cpl_version_get_micro() . . . 399
4.26.2.6 cpl_version_get_minor() . . . 400
4.26.2.7 cpl_version_get_version() . . . 400
4.27 Masks of pixels . . . 401
4.27.1 Detailed Description . . . 402
4.27.2 Function Documentation . . . 403
4.27.2.1 cpl_mask_and() . . . 403
4.27.2.2 cpl_mask_closing() . . . 403
4.27.2.3 cpl_mask_collapse_create() . . . 404
4.27.2.4 cpl_mask_copy() . . . 405
4.27.2.5 cpl_mask_count() . . . 405
4.27.2.6 cpl_mask_count_window() . . . 406
4.27.2.7 cpl_mask_delete() . . . 406
4.27.2.8 cpl_mask_dilation() . . . 407
4.27.2.9 cpl_mask_dump_window() . . . 407
4.27.2.10 cpl_mask_duplicate() . . . 409
4.27.2.11 cpl_mask_erosion() . . . 410
4.27.2.12 cpl_mask_extract() . . . 410
4.27.2.13 cpl_mask_extract_subsample() . . . 411

4.27.2.14 cpl_mask_filter() . . . 412
4.27.2.15 cpl_mask_flip() . . . 413
4.27.2.16 cpl_mask_get() . . . 414
4.27.2.17 cpl_mask_get_data() . . . 414
4.27.2.18 cpl_mask_get_data_const() . . . 415
4.27.2.19 cpl_mask_get_size_x() . . . 416
4.27.2.20 cpl_mask_get_size_y() . . . 416
4.27.2.21 cpl_mask_is_empty() . . . 417
4.27.2.22 cpl_mask_is_empty_window() . . . 417
4.27.2.23 cpl_mask_load() . . . 418
4.27.2.24 cpl_mask_load_window() . . . 419
4.27.2.25 cpl_mask_move() . . . 419
4.27.2.26 cpl_mask_new() . . . 420
4.27.2.27 cpl_mask_not() . . . 421
4.27.2.28 cpl_mask_opening() . . . 421
4.27.2.29 cpl_mask_or() . . . 422
4.27.2.30 cpl_mask_save() . . . 423
4.27.2.31 cpl_mask_set() . . . 424
4.27.2.32 cpl_mask_shift() . . . 424
4.27.2.33 cpl_mask_threshold_image() . . . 425
4.27.2.34 cpl_mask_threshold_image_create() . . . 426
4.27.2.35 cpl_mask_turn() . . . 426
4.27.2.36 cpl_mask_unwrap() . . . 427
4.27.2.37 cpl_mask_wrap() . . . 428
4.27.2.38 cpl_mask_xor() . . . 429
4.28 Matrices . . . 430
4.28.1 Detailed Description . . . 432
4.28.2 Function Documentation . . . 433
4.28.2.1 cpl_matrix_add() . . . 433
4.28.2.2 cpl_matrix_add_scalar() . . . 433
4.28.2.3 cpl_matrix_append() . . . 434
4.28.2.4 cpl_matrix_copy() . . . 435
4.28.2.5 cpl_matrix_decomp_chol() . . . 436
4.28.2.6 cpl_matrix_delete() . . . 437
4.28.2.7 cpl_matrix_divide() . . . 438
4.28.2.8 cpl_matrix_divide_scalar() . . . 438
4.28.2.9 cpl_matrix_dump() . . . 439
4.28.2.10 cpl_matrix_duplicate() . . . 439
4.28.2.11 cpl_matrix_erase_columns() . . . 440
4.28.2.12 cpl_matrix_erase_rows() . . . 441
4.28.2.13 cpl_matrix_exponential() . . . 441
4.28.2.14 cpl_matrix_extract() . . . 442

4.28.2.15 cpl_matrix_extract_column() . . . 443
4.28.2.16 cpl_matrix_extract_diagonal() . . . 444
4.28.2.17 cpl_matrix_extract_row() . . . 444
4.28.2.18 cpl_matrix_fill() . . . 445
4.28.2.19 cpl_matrix_fill_column() . . . 445
4.28.2.20 cpl_matrix_fill_diagonal() . . . 446
4.28.2.21 cpl_matrix_fill_row() . . . 447
4.28.2.22 cpl_matrix_fill_window() . . . 447
4.28.2.23 cpl_matrix_flip_columns() . . . 448
4.28.2.24 cpl_matrix_flip_rows() . . . 449
4.28.2.25 cpl_matrix_get() . . . 449
4.28.2.26 cpl_matrix_get_data() . . . 450
4.28.2.27 cpl_matrix_get_data_const() . . . 451
4.28.2.28 cpl_matrix_get_determinant() . . . 451
4.28.2.29 cpl_matrix_get_max() . . . 452
4.28.2.30 cpl_matrix_get_maxpos() . . . 453
4.28.2.31 cpl_matrix_get_mean() . . . 453
4.28.2.32 cpl_matrix_get_median() . . . 454
4.28.2.33 cpl_matrix_get_min() . . . 454
4.28.2.34 cpl_matrix_get_minpos() . . . 455
4.28.2.35 cpl_matrix_get_ncol() . . . 456
4.28.2.36 cpl_matrix_get_ncol_() . . . 456
4.28.2.37 cpl_matrix_get_nrow() . . . 457
4.28.2.38 cpl_matrix_get_stdev() . . . 457
4.28.2.39 cpl_matrix_invert_create() . . . 458
4.28.2.40 cpl_matrix_is_diagonal() . . . 459
4.28.2.41 cpl_matrix_is_identity() . . . 459
4.28.2.42 cpl_matrix_is_zero() . . . 460
4.28.2.43 cpl_matrix_logarithm() . . . 461
4.28.2.44 cpl_matrix_multiply() . . . 461
4.28.2.45 cpl_matrix_multiply_scalar() . . . 462
4.28.2.46 cpl_matrix_new() . . . 463
4.28.2.47 cpl_matrix_power() . . . 463
4.28.2.48 cpl_matrix_product_create() . . . 464
4.28.2.49 cpl_matrix_resize() . . . 465
4.28.2.50 cpl_matrix_set_() . . . 466
4.28.2.51 cpl_matrix_set_size() . . . 467
4.28.2.52 cpl_matrix_shift() . . . 468
4.28.2.53 cpl_matrix_solve() . . . 468
4.28.2.54 cpl_matrix_solve_chol() . . . 469
4.28.2.55 cpl_matrix_solve_normal() . . . 470
4.28.2.56 cpl_matrix_solve_svd() . . . 471

4.28.2.57 cpl_matrix_solve_svd_threshold() . . . 472
4.28.2.58 cpl_matrix_sort_columns() . . . 473
4.28.2.59 cpl_matrix_sort_rows() . . . 474
4.28.2.60 cpl_matrix_subtract() . . . 474
4.28.2.61 cpl_matrix_subtract_scalar() . . . 475
4.28.2.62 cpl_matrix_swap_columns() . . . 475
4.28.2.63 cpl_matrix_swap_rowcolumn() . . . 476
4.28.2.64 cpl_matrix_swap_rows() . . . 477
4.28.2.65 cpl_matrix_threshold_small() . . . 477
4.28.2.66 cpl_matrix_transpose_create() . . . 478
4.28.2.67 cpl_matrix_unwrap() . . . 479
4.28.2.68 cpl_matrix_wrap() . . . 479
4.29 Memory Management Utilities . . . 481
4.29.1 Detailed Description . . . 481
4.29.2 Function Documentation . . . 481
4.29.2.1 cpl_calloc() . . . 481
4.29.2.2 cpl_free() . . . 482
4.29.2.3 cpl_malloc() . . . 482
4.29.2.4 cpl_memory_dump() . . . 483
4.29.2.5 cpl_memory_is_empty() . . . 483
4.29.2.6 cpl_realloc() . . . 483
4.29.2.7 cpl_sprintf() . . . 484
4.29.2.8 cpl_strdup() . . . 485
4.29.2.9 cpl_vsprintf() . . . 486
4.30 Messages . . . 487
4.30.1 Detailed Description . . . 488
4.30.2 Function Documentation . . . 489
4.30.2.1 cpl_msg_debug() . . . 489
4.30.2.2 cpl_msg_error() . . . 489
4.30.2.3 cpl_msg_get_domain() . . . 490
4.30.2.4 cpl_msg_get_level() . . . 490
4.30.2.5 cpl_msg_get_log_level() . . . 490
4.30.2.6 cpl_msg_get_log_name() . . . 491
4.30.2.7 cpl_msg_indent() . . . 491
4.30.2.8 cpl_msg_indent_less() . . . 491
4.30.2.9 cpl_msg_indent_more() . . . 492
4.30.2.10 cpl_msg_info() . . . 492
4.30.2.11 cpl_msg_info_overwritable() . . . 492
4.30.2.12 cpl_msg_init() . . . 493
4.30.2.13 cpl_msg_progress() . . . 494
4.30.2.14 cpl_msg_set_component_off() . . . 495
4.30.2.15 cpl_msg_set_component_on() . . . 495

4.30.2.16 cpl_msg_set_domain() . . . 495
4.30.2.17 cpl_msg_set_domain_off() . . . 496
4.30.2.18 cpl_msg_set_domain_on() . . . 497
4.30.2.19 cpl_msg_set_indentation() . . . 497
4.30.2.20 cpl_msg_set_level() . . . 498
4.30.2.21 cpl_msg_set_level_from_env() . . . 498
4.30.2.22 cpl_msg_set_log_level() . . . 498
4.30.2.23 cpl_msg_set_log_name() . . . 499
4.30.2.24 cpl_msg_set_threadid_off() . . . 500
4.30.2.25 cpl_msg_set_threadid_on() . . . 501
4.30.2.26 cpl_msg_set_time_off() . . . 501
4.30.2.27 cpl_msg_set_time_on() . . . 502
4.30.2.28 cpl_msg_set_width() . . . 502
4.30.2.29 cpl_msg_stop() . . . 502
4.30.2.30 cpl_msg_stop_log() . . . 503
4.30.2.31 cpl_msg_warning() . . . 503
4.31 Multi Frames . . . 505
4.31.1 Detailed Description . . . 506
4.31.2 Typedef Documentation . . . 506
4.31.2.1 cpl_multiframe . . . 506
4.31.2.2 cpl_multiframe_id_mode . . . 506
4.31.3 Enumeration Type Documentation . . . 506
4.31.3.1 _cpl_multiframe_id_mode_ . . . 506
4.31.4 Function Documentation . . . 507
4.31.4.1 cpl_multiframe_add_empty() . . . 507
4.31.4.2 cpl_multiframe_append_datagroup() . . . 507
4.31.4.3 cpl_multiframe_append_datagroup_from_position() . . . 508
4.31.4.4 cpl_multiframe_append_dataset() . . . 509
4.31.4.5 cpl_multiframe_append_dataset_from_position() . . . 510
4.31.4.6 cpl_multiframe_delete() . . . 511
4.31.4.7 cpl_multiframe_get_size() . . . 511
4.31.4.8 cpl_multiframe_new() . . . 512
4.31.4.9 cpl_multiframe_write() . . . 512
4.32 Parameter Lists . . . 514
4.32.1 Detailed Description . . . 515
4.32.2 Typedef Documentation . . . 515
4.32.2.1 cpl_parameterlist . . . 515
4.32.3 Function Documentation . . . 515
4.32.3.1 cpl_parameterlist_append() . . . 515
4.32.3.2 cpl_parameterlist_delete() . . . 516
4.32.3.3 cpl_parameterlist_dump() . . . 516
4.32.3.4 cpl_parameterlist_find() . . . 517

4.32.3.5 cpl_parameterlist_find_const() . . . 518
4.32.3.6 cpl_parameterlist_find_context() . . . 518
4.32.3.7 cpl_parameterlist_find_context_const() . . . 519
4.32.3.8 cpl_parameterlist_find_tag() . . . 520
4.32.3.9 cpl_parameterlist_find_tag_const() . . . 520
4.32.3.10 cpl_parameterlist_find_type() . . . 522
4.32.3.11 cpl_parameterlist_find_type_const() . . . 522
4.32.3.12 cpl_parameterlist_get_first() . . . 524
4.32.3.13 cpl_parameterlist_get_first_const() . . . 525
4.32.3.14 cpl_parameterlist_get_last() . . . 525
4.32.3.15 cpl_parameterlist_get_last_const() . . . 526
4.32.3.16 cpl_parameterlist_get_next() . . . 527
4.32.3.17 cpl_parameterlist_get_next_const() . . . 527
4.32.3.18 cpl_parameterlist_get_size() . . . 528
4.32.3.19 cpl_parameterlist_new() . . . 528
4.33 Parameters . . . 530
4.33.1 Detailed Description . . . 532
4.33.2 Typedef Documentation . . . 533
4.33.2.1 cpl_parameter . . . 533
4.33.2.2 cpl_parameter_class . . . 534
4.33.2.3 cpl_parameter_mode . . . 534
4.33.3 Enumeration Type Documentation . . . 534
4.33.3.1 _cpl_parameter_class_ . . . 534
4.33.3.2 _cpl_parameter_mode_ . . . 534
4.33.4 Function Documentation . . . 535
4.33.4.1 cpl_parameter_delete() . . . 535
4.33.4.2 cpl_parameter_disable() . . . 535
4.33.4.3 cpl_parameter_dump() . . . 536
4.33.4.4 cpl_parameter_duplicate() . . . 536
4.33.4.5 cpl_parameter_enable() . . . 537
4.33.4.6 cpl_parameter_get_alias() . . . 538
4.33.4.7 cpl_parameter_get_bool() . . . 538
4.33.4.8 cpl_parameter_get_class() . . . 539
4.33.4.9 cpl_parameter_get_context() . . . 540
4.33.4.10 cpl_parameter_get_default_bool() . . . 540
4.33.4.11 cpl_parameter_get_default_double() . . . 541
4.33.4.12 cpl_parameter_get_default_flag() . . . 541
4.33.4.13 cpl_parameter_get_default_int() . . . 542
4.33.4.14 cpl_parameter_get_default_string() . . . 543
4.33.4.15 cpl_parameter_get_double() . . . 543
4.33.4.16 cpl_parameter_get_enum_double() . . . 544
4.33.4.17 cpl_parameter_get_enum_int() . . . 545

4.33.4.18 cpl_parameter_get_enum_size() . . . 546
4.33.4.19 cpl_parameter_get_enum_string() . . . 546
4.33.4.20 cpl_parameter_get_help() . . . 547
4.33.4.21 cpl_parameter_get_id() . . . 548
4.33.4.22 cpl_parameter_get_int() . . . 548
4.33.4.23 cpl_parameter_get_name() . . . 549
4.33.4.24 cpl_parameter_get_range_max_double() . . . 550
4.33.4.25 cpl_parameter_get_range_max_int() . . . 550
4.33.4.26 cpl_parameter_get_range_min_double() . . . 551
4.33.4.27 cpl_parameter_get_range_min_int() . . . 552
4.33.4.28 cpl_parameter_get_string() . . . 552
4.33.4.29 cpl_parameter_get_tag() . . . 553
4.33.4.30 cpl_parameter_get_type() . . . 553
4.33.4.31 cpl_parameter_is_enabled() . . . 554
4.33.4.32 cpl_parameter_new_enum() . . . 555
4.33.4.33 cpl_parameter_new_enum_from_array() . . . 556
4.33.4.34 cpl_parameter_new_range() . . . 557
4.33.4.35 cpl_parameter_new_value() . . . 559
4.33.4.36 cpl_parameter_set_alias() . . . 560
4.33.4.37 cpl_parameter_set_bool() . . . 560
4.33.4.38 cpl_parameter_set_default_bool() . . . 561
4.33.4.39 cpl_parameter_set_default_double() . . . 562
4.33.4.40 cpl_parameter_set_default_flag() . . . 562
4.33.4.41 cpl_parameter_set_default_int() . . . 563
4.33.4.42 cpl_parameter_set_default_string() . . . 564
4.33.4.43 cpl_parameter_set_double() . . . 564
4.33.4.44 cpl_parameter_set_id() . . . 565
4.33.4.45 cpl_parameter_set_int() . . . 566
4.33.4.46 cpl_parameter_set_string() . . . 566
4.33.4.47 cpl_parameter_set_tag() . . . 567
4.34 Plotting of CPL objects . . . 568
4.34.1 Detailed Description . . . 568
4.34.2 Function Documentation . . . 569
4.34.2.1 cpl_plot_bivector() . . . 569
4.34.2.2 cpl_plot_bivectors() . . . 569
4.34.2.3 cpl_plot_column() . . . 570
4.34.2.4 cpl_plot_columns() . . . 571
4.34.2.5 cpl_plot_image() . . . 572
4.34.2.6 cpl_plot_image_col() . . . 573
4.34.2.7 cpl_plot_image_row() . . . 574
4.34.2.8 cpl_plot_mask() . . . 575
4.34.2.9 cpl_plot_vector() . . . 576

4.34.2.10 cpl_plot_vectors() . . . 577
4.35 Plugin Interface . . . 579
4.35.1 Detailed Description . . . 581
4.35.2 Macro Definition Documentation . . . 581
4.35.2.1 CPL_PLUGIN_API . . . 581
4.35.3 Typedef Documentation . . . 581
4.35.3.1 cpl_plugin . . . 581
4.35.3.2 cpl_plugin_type . . . 582
4.35.4 Enumeration Type Documentation . . . 582
4.35.4.1 _cpl_plugin_type_ . . . 582
4.35.5 Function Documentation . . . 582
4.35.5.1 cpl_plugin_copy() . . . 582
4.35.5.2 cpl_plugin_delete() . . . 583
4.35.5.3 cpl_plugin_dump() . . . 584
4.35.5.4 cpl_plugin_get_api() . . . 584
4.35.5.5 cpl_plugin_get_author() . . . 585
4.35.5.6 cpl_plugin_get_copyright() . . . 586
4.35.5.7 cpl_plugin_get_deinit() . . . 586
4.35.5.8 cpl_plugin_get_description() . . . 587
4.35.5.9 cpl_plugin_get_email() . . . 587
4.35.5.10 cpl_plugin_get_exec() . . . 588
4.35.5.11 cpl_plugin_get_info() . . . 589
4.35.5.12 cpl_plugin_get_init() . . . 589
4.35.5.13 cpl_plugin_get_name() . . . 590
4.35.5.14 cpl_plugin_get_synopsis() . . . 590
4.35.5.15 cpl_plugin_get_type() . . . 591
4.35.5.16 cpl_plugin_get_type_string() . . . 592
4.35.5.17 cpl_plugin_get_version() . . . 592
4.35.5.18 cpl_plugin_get_version_string() . . . 593
4.35.5.19 cpl_plugin_init() . . . 594
4.35.5.20 cpl_plugin_new() . . . 595
4.35.5.21 cpl_plugin_set_api() . . . 595
4.35.5.22 cpl_plugin_set_author() . . . 596
4.35.5.23 cpl_plugin_set_copyright() . . . 596
4.35.5.24 cpl_plugin_set_deinit() . . . 597
4.35.5.25 cpl_plugin_set_description() . . . 598
4.35.5.26 cpl_plugin_set_email() . . . 598
4.35.5.27 cpl_plugin_set_exec() . . . 599
4.35.5.28 cpl_plugin_set_init() . . . 600
4.35.5.29 cpl_plugin_set_name() . . . 600
4.35.5.30 cpl_plugin_set_synopsis() . . . 601
4.35.5.31 cpl_plugin_set_type() . . . 602

4.35.5.32 cpl_plugin_set_version() . . . 602
4.36 Plugin List . . . 604
4.36.1 Detailed Description . . . 604
4.36.2 Typedef Documentation . . . 604
4.36.2.1 cpl_pluginlist . . . 605
4.36.3 Function Documentation . . . 605
4.36.3.1 cpl_pluginlist_append() . . . 605
4.36.3.2 cpl_pluginlist_delete() . . . 605
4.36.3.3 cpl_pluginlist_dump() . . . 606
4.36.3.4 cpl_pluginlist_find() . . . 606
4.36.3.5 cpl_pluginlist_get_first() . . . 607
4.36.3.6 cpl_pluginlist_get_last() . . . 608
4.36.3.7 cpl_pluginlist_get_next() . . . 608
4.36.3.8 cpl_pluginlist_get_size() . . . 609
4.36.3.9 cpl_pluginlist_new() . . . 609
4.36.3.10 cpl_pluginlist_prepend() . . . 610
4.37 Point pattern matching module . . . 611
4.37.1 Detailed Description . . . 611
4.37.2 Function Documentation . . . 611
4.37.2.1 cpl_ppm_match_points() . . . 611
4.37.2.2 cpl_ppm_match_positions() . . . 614
4.38 Polynomials . . . 616
4.38.1 Detailed Description . . . 617
4.38.2 Function Documentation . . . 617
4.38.2.1 cpl_polynomial_add() . . . 617
4.38.2.2 cpl_polynomial_compare() . . . 618
4.38.2.3 cpl_polynomial_copy() . . . 619
4.38.2.4 cpl_polynomial_delete() . . . 619
4.38.2.5 cpl_polynomial_derivative() . . . 620
4.38.2.6 cpl_polynomial_dump() . . . 621
4.38.2.7 cpl_polynomial_duplicate() . . . 621
4.38.2.8 cpl_polynomial_eval() . . . 622
4.38.2.9 cpl_polynomial_eval_1d() . . . 623
4.38.2.10 cpl_polynomial_eval_1d_diff() . . . 624
4.38.2.11 cpl_polynomial_eval_2d() . . . 625
4.38.2.12 cpl_polynomial_eval_3d() . . . 625
4.38.2.13 cpl_polynomial_extract() . . . 626
4.38.2.14 cpl_polynomial_fit() . . . 627
4.38.2.15 cpl_polynomial_fit_1d_create() . . . 629
4.38.2.16 cpl_polynomial_fit_2d_create() . . . 630
4.38.2.17 cpl_polynomial_get_coeff() . . . 631
4.38.2.18 cpl_polynomial_get_degree() . . . 631

4.38.2.19 cpl_polynomial_get_dimension() . . . 632
4.38.2.20 cpl_polynomial_multiply() . . . 633
4.38.2.21 cpl_polynomial_multiply_scalar() . . . 633
4.38.2.22 cpl_polynomial_set_coeff() . . . 634
4.38.2.23 cpl_polynomial_shift_1d() . . . 635
4.38.2.24 cpl_polynomial_solve_1d() . . . 636
4.38.2.25 cpl_polynomial_subtract() . . . 636
4.38.2.26 cpl_vector_fill_polynomial() . . . 637
4.38.2.27 cpl_vector_fill_polynomial_fit_residual() . . . 638
4.39 Properties . . . 640
4.39.1 Detailed Description . . . 641
4.39.2 Typedef Documentation . . . 641
4.39.2.1 cpl_property . . . 641
4.39.3 Function Documentation . . . 641
4.39.3.1 cpl_property_delete() . . . 641
4.39.3.2 cpl_property_dump() . . . 642
4.39.3.3 cpl_property_duplicate() . . . 642
4.39.3.4 cpl_property_get_bool() . . . 643
4.39.3.5 cpl_property_get_char() . . . 644
4.39.3.6 cpl_property_get_comment() . . . 644
4.39.3.7 cpl_property_get_double() . . . 645
4.39.3.8 cpl_property_get_double_complex() . . . 646
4.39.3.9 cpl_property_get_float() . . . 646
4.39.3.10 cpl_property_get_float_complex() . . . 647
4.39.3.11 cpl_property_get_int() . . . 648
4.39.3.12 cpl_property_get_long() . . . 648
4.39.3.13 cpl_property_get_long_long() . . . 649
4.39.3.14 cpl_property_get_name() . . . 649
4.39.3.15 cpl_property_get_size() . . . 650
4.39.3.16 cpl_property_get_string() . . . 651
4.39.3.17 cpl_property_get_type() . . . 652
4.39.3.18 cpl_property_new() . . . 652
4.39.3.19 cpl_property_new_array() . . . 653
4.39.3.20 cpl_property_set_bool() . . . 654
4.39.3.21 cpl_property_set_char() . . . 655
4.39.3.22 cpl_property_set_comment() . . . 655
4.39.3.23 cpl_property_set_double() . . . 656
4.39.3.24 cpl_property_set_double_complex() . . . 657
4.39.3.25 cpl_property_set_float() . . . 657
4.39.3.26 cpl_property_set_float_complex() . . . 658
4.39.3.27 cpl_property_set_int() . . . 659
4.39.3.28 cpl_property_set_long() . . . 659

4.39.3.29 cpl_property_set_long_long() . . . 660
4.39.3.30 cpl_property_set_name() . . . 661
4.39.3.31 cpl_property_set_string() . . . 661
4.40 Property Lists . . . 663
4.40.1 Detailed Description . . . 667
4.40.2 Typedef Documentation . . . 668
4.40.2.1 cpl_propertylist . . . 668
4.40.2.2 cpl_propertylist_compare_func . . . 668
4.40.3 Function Documentation . . . 668
4.40.3.1 cpl_propertylist_append() . . . 668
4.40.3.2 cpl_propertylist_append_bool() . . . 669
4.40.3.3 cpl_propertylist_append_char() . . . 669
4.40.3.4 cpl_propertylist_append_double() . . . 670
4.40.3.5 cpl_propertylist_append_double_complex() . . . 671
4.40.3.6 cpl_propertylist_append_float() . . . 671
4.40.3.7 cpl_propertylist_append_float_complex() . . . 672
4.40.3.8 cpl_propertylist_append_int() . . . 673
4.40.3.9 cpl_propertylist_append_long() . . . 673
4.40.3.10 cpl_propertylist_append_long_long() . . . 674
4.40.3.11 cpl_propertylist_append_property() . . . 675
4.40.3.12 cpl_propertylist_append_string() . . . 675
4.40.3.13 cpl_propertylist_copy_property() . . . 676
4.40.3.14 cpl_propertylist_copy_property_regexp() . . . 677
4.40.3.15 cpl_propertylist_delete() . . . 678
4.40.3.16 cpl_propertylist_dump() . . . 678
4.40.3.17 cpl_propertylist_duplicate() . . . 679
4.40.3.18 cpl_propertylist_empty() . . . 679
4.40.3.19 cpl_propertylist_erase() . . . 680
4.40.3.20 cpl_propertylist_erase_regexp() . . . 681
4.40.3.21 cpl_propertylist_get() . . . 681
4.40.3.22 cpl_propertylist_get_bool() . . . 682
4.40.3.23 cpl_propertylist_get_char() . . . 683
4.40.3.24 cpl_propertylist_get_comment() . . . 683
4.40.3.25 cpl_propertylist_get_const() . . . 684
4.40.3.26 cpl_propertylist_get_double() . . . 685
4.40.3.27 cpl_propertylist_get_double_complex() . . . 685
4.40.3.28 cpl_propertylist_get_float() . . . 686
4.40.3.29 cpl_propertylist_get_float_complex() . . . 687
4.40.3.30 cpl_propertylist_get_int() . . . 687
4.40.3.31 cpl_propertylist_get_long() . . . 688
4.40.3.32 cpl_propertylist_get_long_long() . . . 689
4.40.3.33 cpl_propertylist_get_property() . . . 690

4.40.3.34 cpl_propertylist_get_property_const() . . . 690
4.40.3.35 cpl_propertylist_get_size() . . . 691
4.40.3.36 cpl_propertylist_get_string() . . . 691
4.40.3.37 cpl_propertylist_get_type() . . . 692
4.40.3.38 cpl_propertylist_has() . . . 693
4.40.3.39 cpl_propertylist_insert_after_bool() . . . 693
4.40.3.40 cpl_propertylist_insert_after_char() . . . 694
4.40.3.41 cpl_propertylist_insert_after_double() . . . 695
4.40.3.42 cpl_propertylist_insert_after_double_complex() . . . 695
4.40.3.43 cpl_propertylist_insert_after_float() . . . 697
4.40.3.44 cpl_propertylist_insert_after_float_complex() . . . 698
4.40.3.45 cpl_propertylist_insert_after_int() . . . 698
4.40.3.46 cpl_propertylist_insert_after_long() . . . 700
4.40.3.47 cpl_propertylist_insert_after_long_long() . . . 701
4.40.3.48 cpl_propertylist_insert_after_property() . . . 701
4.40.3.49 cpl_propertylist_insert_after_string() . . . 702
4.40.3.50 cpl_propertylist_insert_bool() . . . 703
4.40.3.51 cpl_propertylist_insert_char() . . . 703
4.40.3.52 cpl_propertylist_insert_double() . . . 705
4.40.3.53 cpl_propertylist_insert_double_complex() . . . 706
4.40.3.54 cpl_propertylist_insert_float() . . . 706
4.40.3.55 cpl_propertylist_insert_float_complex() . . . 708
4.40.3.56 cpl_propertylist_insert_int() . . . 709
4.40.3.57 cpl_propertylist_insert_long() . . . 709
4.40.3.58 cpl_propertylist_insert_long_long() . . . 711
4.40.3.59 cpl_propertylist_insert_property() . . . 712
4.40.3.60 cpl_propertylist_insert_string() . . . 712
4.40.3.61 cpl_propertylist_is_empty() . . . 713
4.40.3.62 cpl_propertylist_load() . . . 714
4.40.3.63 cpl_propertylist_load_regexp() . . . 715
4.40.3.64 cpl_propertylist_new() . . . 716
4.40.3.65 cpl_propertylist_prepend_bool() . . . 716
4.40.3.66 cpl_propertylist_prepend_char() . . . 717
4.40.3.67 cpl_propertylist_prepend_double() . . . 717
4.40.3.68 cpl_propertylist_prepend_double_complex() . . . 718
4.40.3.69 cpl_propertylist_prepend_float() . . . 719
4.40.3.70 cpl_propertylist_prepend_float_complex() . . . 719
4.40.3.71 cpl_propertylist_prepend_int() . . . 720
4.40.3.72 cpl_propertylist_prepend_long() . . . 721
4.40.3.73 cpl_propertylist_prepend_long_long() . . . 721
4.40.3.74 cpl_propertylist_prepend_property() . . . 722
4.40.3.75 cpl_propertylist_prepend_string() . . . 723

4.40.3.76 cpl_propertylist_save() . . . 723
4.40.3.77 cpl_propertylist_set_bool() . . . 724
4.40.3.78 cpl_propertylist_set_char() . . . 725
4.40.3.79 cpl_propertylist_set_comment() . . . 725
4.40.3.80 cpl_propertylist_set_double() . . . 726
4.40.3.81 cpl_propertylist_set_double_complex() . . . 727
4.40.3.82 cpl_propertylist_set_float() . . . 727
4.40.3.83 cpl_propertylist_set_float_complex() . . . 728
4.40.3.84 cpl_propertylist_set_int() . . . 729
4.40.3.85 cpl_propertylist_set_long() . . . 729
4.40.3.86 cpl_propertylist_set_long_long() . . . 730
4.40.3.87 cpl_propertylist_set_string() . . . 731
4.40.3.88 cpl_propertylist_sort() . . . 731
4.40.3.89 cpl_propertylist_update_bool() . . . 732
4.40.3.90 cpl_propertylist_update_char() . . . 733
4.40.3.91 cpl_propertylist_update_double() . . . 734
4.40.3.92 cpl_propertylist_update_double_complex() . . . 734
4.40.3.93 cpl_propertylist_update_float() . . . 735
4.40.3.94 cpl_propertylist_update_float_complex() . . . 736
4.40.3.95 cpl_propertylist_update_int() . . . 736
4.40.3.96 cpl_propertylist_update_long() . . . 737
4.40.3.97 cpl_propertylist_update_long_long() . . . 738
4.40.3.98 cpl_propertylist_update_string() . . . 738
4.41 Recipe Configurations . . . 740
4.41.1 Detailed Description . . . 740
4.41.2 Function Documentation . . . 741
4.41.2.1 cpl_recipeconfig_clear() . . . 741
4.41.2.2 cpl_recipeconfig_delete() . . . 741
4.41.2.3 cpl_recipeconfig_get_inputs() . . . 742
4.41.2.4 cpl_recipeconfig_get_max_count() . . . 742
4.41.2.5 cpl_recipeconfig_get_min_count() . . . 743
4.41.2.6 cpl_recipeconfig_get_outputs() . . . 744
4.41.2.7 cpl_recipeconfig_get_tags() . . . 745
4.41.2.8 cpl_recipeconfig_is_required() . . . 746
4.41.2.9 cpl_recipeconfig_new() . . . 747
4.41.2.10 cpl_recipeconfig_set_input() . . . 747
4.41.2.11 cpl_recipeconfig_set_inputs() . . . 748
4.41.2.12 cpl_recipeconfig_set_output() . . . 749
4.41.2.13 cpl_recipeconfig_set_outputs() . . . 750
4.41.2.14 cpl_recipeconfig_set_tag() . . . 751
4.41.2.15 cpl_recipeconfig_set_tags() . . . 752
4.42 Recipe Definition . . . 753

4.42.1 Detailed Description . . . 753
4.42.2 Macro Definition Documentation . . . 753
4.42.2.1 cpl_get_license . . . 753
4.42.2.2 cpl_recipe_define . . . 754
4.42.2.3 CPL_RECIPE_DEFINE . . . 755
4.43 Recipes . . . 756
4.43.1 Detailed Description . . . 756
4.43.2 Typedef Documentation . . . 756
4.43.2.1 cpl_recipe . . . 756
4.44 Regular Expression Filter . . . 757
4.44.1 Detailed Description . . . 757
4.44.2 Typedef Documentation . . . 757
4.44.2.1 cpl_regex . . . 757
4.44.2.2 cpl_regex_syntax_option . . . 758
4.44.3 Enumeration Type Documentation . . . 758
4.44.3.1 _cpl_regex_syntax_option_ . . . 758
4.44.4 Function Documentation . . . 758
4.44.4.1 cpl_regex_apply() . . . 758
4.44.4.2 cpl_regex_delete() . . . 759
4.44.4.3 cpl_regex_is_negated() . . . 759
4.44.4.4 cpl_regex_negate() . . . 759
4.44.4.5 cpl_regex_new() . . . 760
4.45 Statistics . . . 761
4.45.1 Detailed Description . . . 762
4.45.2 Typedef Documentation . . . 762
4.45.2.1 cpl_stats . . . 762
4.45.2.2 cpl_stats_mode . . . 762
4.45.3 Enumeration Type Documentation . . . 763
4.45.3.1 _cpl_stats_mode_ . . . 763
4.45.4 Function Documentation . . . 763
4.45.4.1 cpl_stats_delete() . . . 763
4.45.4.2 cpl_stats_dump() . . . 764
4.45.4.3 cpl_stats_get_absflux() . . . 764
4.45.4.4 cpl_stats_get_centroid_x() . . . 765
4.45.4.5 cpl_stats_get_centroid_y() . . . 765
4.45.4.6 cpl_stats_get_flux() . . . 766
4.45.4.7 cpl_stats_get_mad() . . . 766
4.45.4.8 cpl_stats_get_max() . . . 767
4.45.4.9 cpl_stats_get_max_x() . . . 767
4.45.4.10 cpl_stats_get_max_y() . . . 768
4.45.4.11 cpl_stats_get_mean() . . . 768
4.45.4.12 cpl_stats_get_median() . . . 769

4.45.4.13 cpl_stats_get_median_dev() . . . 769
4.45.4.14 cpl_stats_get_min() . . . 770
4.45.4.15 cpl_stats_get_min_x() . . . 770
4.45.4.16 cpl_stats_get_min_y() . . . 771
4.45.4.17 cpl_stats_get_npix() . . . 771
4.45.4.18 cpl_stats_get_sqflux() . . . 772
4.45.4.19 cpl_stats_get_stdev() . . . 772
4.45.4.20 cpl_stats_new_from_image() . . . 773
4.45.4.21 cpl_stats_new_from_image_window() . . . 773
4.46 Tables . . . 776
4.46.1 Detailed Description . . . 784
4.46.2 Enumeration Type Documentation . . . 784
4.46.2.1 anonymous enum . . . 784
4.46.3 Function Documentation . . . 785
4.46.3.1 cpl_table_abs_column() . . . 785
4.46.3.2 cpl_table_add_columns() . . . 785
4.46.3.3 cpl_table_add_scalar() . . . 786
4.46.3.4 cpl_table_add_scalar_complex() . . . 787
4.46.3.5 cpl_table_and_selected() . . . 787
4.46.3.6 cpl_table_and_selected_double() . . . 788
4.46.3.7 cpl_table_and_selected_double_complex() . . . 789
4.46.3.8 cpl_table_and_selected_float() . . . 789
4.46.3.9 cpl_table_and_selected_float_complex() . . . 790
4.46.3.10 cpl_table_and_selected_int() . . . 791
4.46.3.11 cpl_table_and_selected_invalid() . . . 792
4.46.3.12 cpl_table_and_selected_long() . . . 792
4.46.3.13 cpl_table_and_selected_long_long() . . . 793
4.46.3.14 cpl_table_and_selected_string() . . . 793
4.46.3.15 cpl_table_and_selected_window() . . . 794
4.46.3.16 cpl_table_arg_column() . . . 795
4.46.3.17 cpl_table_cast_column() . . . 796
4.46.3.18 cpl_table_compare_structure() . . . 797
4.46.3.19 cpl_table_conjugate_column() . . . 798
4.46.3.20 cpl_table_copy_data_double() . . . 798
4.46.3.21 cpl_table_copy_data_double_complex() . . . 799
4.46.3.22 cpl_table_copy_data_float() . . . 799
4.46.3.23 cpl_table_copy_data_float_complex() . . . 800
4.46.3.24 cpl_table_copy_data_int() . . . 801
4.46.3.25 cpl_table_copy_data_long() . . . 801
4.46.3.26 cpl_table_copy_data_long_long() . . . 802
4.46.3.27 cpl_table_copy_data_string() . . . 802
4.46.3.28 cpl_table_copy_structure() . . . 804

4.46.3.29 cpl_table_count_invalid() . . . 805
4.46.3.30 cpl_table_count_selected() . . . 805
4.46.3.31 cpl_table_delete() . . . 806
4.46.3.32 cpl_table_divide_columns() . . . 806
4.46.3.33 cpl_table_divide_scalar() . . . 807
4.46.3.34 cpl_table_divide_scalar_complex() . . . 808
4.46.3.35 cpl_table_dump() . . . 808
4.46.3.36 cpl_table_dump_structure() . . . 809
4.46.3.37 cpl_table_duplicate() . . . 810
4.46.3.38 cpl_table_duplicate_column() . . . 810
4.46.3.39 cpl_table_erase_column() . . . 811
4.46.3.40 cpl_table_erase_invalid() . . . 812
4.46.3.41 cpl_table_erase_invalid_rows() . . . 812
4.46.3.42 cpl_table_erase_selected() . . . 813
4.46.3.43 cpl_table_erase_window() . . . 814
4.46.3.44 cpl_table_exponential_column() . . . 815
4.46.3.45 cpl_table_extract() . . . 815
4.46.3.46 cpl_table_extract_selected() . . . 816
4.46.3.47 cpl_table_fill_column_window() . . . 817
4.46.3.48 cpl_table_fill_column_window_array() . . . 817
4.46.3.49 cpl_table_fill_column_window_complex() . . . 818
4.46.3.50 cpl_table_fill_column_window_double() . . . 819
4.46.3.51 cpl_table_fill_column_window_double_complex() . . . 820
4.46.3.52 cpl_table_fill_column_window_float() . . . 820
4.46.3.53 cpl_table_fill_column_window_float_complex() . . . 821
4.46.3.54 cpl_table_fill_column_window_int() . . . 822
4.46.3.55 cpl_table_fill_column_window_long() . . . 823
4.46.3.56 cpl_table_fill_column_window_long_long() . . . 824
4.46.3.57 cpl_table_fill_column_window_string() . . . 825
4.46.3.58 cpl_table_fill_invalid_double() . . . 825
4.46.3.59 cpl_table_fill_invalid_double_complex() . . . 826
4.46.3.60 cpl_table_fill_invalid_float() . . . 827
4.46.3.61 cpl_table_fill_invalid_float_complex() . . . 828
4.46.3.62 cpl_table_fill_invalid_int() . . . 829
4.46.3.63 cpl_table_fill_invalid_long() . . . 829
4.46.3.64 cpl_table_fill_invalid_long_long() . . . 830
4.46.3.65 cpl_table_get() . . . 831
4.46.3.66 cpl_table_get_array() . . . 832
4.46.3.67 cpl_table_get_column_depth() . . . 832
4.46.3.68 cpl_table_get_column_dimension() . . . 833
4.46.3.69 cpl_table_get_column_dimensions() . . . 834
4.46.3.70 cpl_table_get_column_format() . . . 834

4.46.3.71 cpl_table_get_column_max() . . . 835
4.46.3.72 cpl_table_get_column_maxpos() . . . 835
4.46.3.73 cpl_table_get_column_mean() . . . 836
4.46.3.74 cpl_table_get_column_mean_complex() . . . 837
4.46.3.75 cpl_table_get_column_median() . . . 837
4.46.3.76 cpl_table_get_column_min() . . . 838
4.46.3.77 cpl_table_get_column_minpos() . . . 838
4.46.3.78 cpl_table_get_column_name() . . . 838
4.46.3.79 cpl_table_get_column_names() . . . 839
4.46.3.80 cpl_table_get_column_stdev() . . . 840
4.46.3.81 cpl_table_get_column_type() . . . 840
4.46.3.82 cpl_table_get_column_unit() . . . 841
4.46.3.83 cpl_table_get_complex() . . . 841
4.46.3.84 cpl_table_get_data_array() . . . 842
4.46.3.85 cpl_table_get_data_array_const() . . . 843
4.46.3.86 cpl_table_get_data_double() . . . 843
4.46.3.87 cpl_table_get_data_double_complex() . . . 844
4.46.3.88 cpl_table_get_data_double_complex_const() . . . 845
4.46.3.89 cpl_table_get_data_double_const() . . . 846
4.46.3.90 cpl_table_get_data_float() . . . 846
4.46.3.91 cpl_table_get_data_float_complex() . . . 847
4.46.3.92 cpl_table_get_data_float_complex_const() . . . 848
4.46.3.93 cpl_table_get_data_float_const() . . . 849
4.46.3.94 cpl_table_get_data_int() . . . 849
4.46.3.95 cpl_table_get_data_int_const() . . . 850
4.46.3.96 cpl_table_get_data_long() . . . 851
4.46.3.97 cpl_table_get_data_long_const() . . . 851
4.46.3.98 cpl_table_get_data_long_long() . . . 852
4.46.3.99 cpl_table_get_data_long_long_const() . . . 853
4.46.3.100 cpl_table_get_data_string() . . . 853
4.46.3.101 cpl_table_get_data_string_const() . . . 854
4.46.3.102 cpl_table_get_double() . . . 855
4.46.3.103 cpl_table_get_double_complex() . . . 855
4.46.3.104 cpl_table_get_float() . . . 856
4.46.3.105 cpl_table_get_float_complex() . . . 857
4.46.3.106 cpl_table_get_int() . . . 857
4.46.3.107 cpl_table_get_long() . . . 858
4.46.3.108 cpl_table_get_long_long() . . . 859
4.46.3.109 cpl_table_get_ncol() . . . 860
4.46.3.110 cpl_table_get_nrow() . . . 860
4.46.3.111 cpl_table_get_string() . . . 861
4.46.3.112 cpl_table_has_column() . . . 862

4.46.3.113 cpl_table_has_invalid() . . . 862
4.46.3.114 cpl_table_has_valid() . . . 863
4.46.3.115 cpl_table_imag_column() . . . 864
4.46.3.116 cpl_table_insert() . . . 864
4.46.3.117 cpl_table_insert_window() . . . 865
4.46.3.118 cpl_table_is_selected() . . . 866
4.46.3.119 cpl_table_is_valid() . . . 867
4.46.3.120 cpl_table_load() . . . 867
4.46.3.121 cpl_table_load_window() . . . 868
4.46.3.122 cpl_table_logarithm_column() . . . 869
4.46.3.123 cpl_table_move_column() . . . 870
4.46.3.124 cpl_table_multiply_columns() . . . 870
4.46.3.125 cpl_table_multiply_scalar() . . . 871
4.46.3.126 cpl_table_multiply_scalar_complex() . . . 872
4.46.3.127 cpl_table_name_column() . . . 872
4.46.3.128 cpl_table_new() . . . 873
4.46.3.129 cpl_table_new_column() . . . 874
4.46.3.130 cpl_table_new_column_array() . . . 874
4.46.3.131 cpl_table_not_selected() . . . 875
4.46.3.132 cpl_table_or_selected() . . . 876
4.46.3.133 cpl_table_or_selected_double() . . . 876
4.46.3.134 cpl_table_or_selected_double_complex() . . . 877
4.46.3.135 cpl_table_or_selected_float() . . . 878
4.46.3.136 cpl_table_or_selected_float_complex() . . . 879
4.46.3.137 cpl_table_or_selected_int() . . . 879
4.46.3.138 cpl_table_or_selected_invalid() . . . 880
4.46.3.139 cpl_table_or_selected_long() . . . 880
4.46.3.140 cpl_table_or_selected_long_long() . . . 881
4.46.3.141 cpl_table_or_selected_string() . . . 882
4.46.3.142 cpl_table_or_selected_window() . . . 883
4.46.3.143 cpl_table_power_column() . . . 883
4.46.3.144 cpl_table_real_column() . . . 884
4.46.3.145 cpl_table_save() . . . 885
4.46.3.146 cpl_table_select_all() . . . 886
4.46.3.147 cpl_table_select_row() . . . 887
4.46.3.148 cpl_table_set() . . . 888
4.46.3.149 cpl_table_set_array() . . . 888
4.46.3.150 cpl_table_set_column_depth() . . . 889
4.46.3.151 cpl_table_set_column_dimensions() . . . 890
4.46.3.152 cpl_table_set_column_format() . . . 890
4.46.3.153 cpl_table_set_column_invalid() . . . 891
4.46.3.154 cpl_table_set_column_unit() . . . 892

4.46.3.155 cpl_table_set_complex() . . . 893
4.46.3.156 cpl_table_set_double() . . . 893
4.46.3.157 cpl_table_set_double_complex() . . . 894
4.46.3.158 cpl_table_set_float() . . . 895
4.46.3.159 cpl_table_set_float_complex() . . . 896
4.46.3.160 cpl_table_set_int() . . . 897
4.46.3.161 cpl_table_set_invalid() . . . 897
4.46.3.162 cpl_table_set_long() . . . 898
4.46.3.163 cpl_table_set_long_long() . . . 899
4.46.3.164 cpl_table_set_size() . . . 900
4.46.3.165 cpl_table_set_string() . . . 900
4.46.3.166 cpl_table_shift_column() . . . 901
4.46.3.167 cpl_table_sort() . . . 902
4.46.3.168 cpl_table_subtract_columns() . . . 903
4.46.3.169 cpl_table_subtract_scalar() . . . 903
4.46.3.170 cpl_table_subtract_scalar_complex() . . . 905
4.46.3.171 cpl_table_unselect_all() . . . 905
4.46.3.172 cpl_table_unselect_row() . . . 906
4.46.3.173 cpl_table_unwrap() . . . 907
4.46.3.174 cpl_table_where_selected() . . . 907
4.46.3.175 cpl_table_wrap_double() . . . 908
4.46.3.176 cpl_table_wrap_double_complex() . . . 909
4.46.3.177 cpl_table_wrap_float() . . . 909
4.46.3.178 cpl_table_wrap_float_complex() . . . 910
4.46.3.179 cpl_table_wrap_int() . . . 911
4.46.3.180 cpl_table_wrap_long() . . . 911
4.46.3.181 cpl_table_wrap_long_long() . . . 912
4.46.3.182 cpl_table_wrap_string() . . . 913
4.47 Type codes . . . 914
4.47.1 Detailed Description . . . 914
4.47.2 Macro Definition Documentation . . . 915
4.47.2.1 CPL_SIZE_FORMAT . . . 915
4.47.2.2 CPL_SIZE_MAX . . . 915
4.47.2.3 CPL_SIZE_MIN . . . 915
4.47.3 Typedef Documentation . . . 915
4.47.3.1 cpl_bitmask . . . 916
4.47.3.2 cpl_size . . . 916
4.47.3.3 cpl_type . . . 916
4.47.4 Enumeration Type Documentation . . . 916
4.47.4.1 _cpl_type_ . . . 916
4.47.5 Function Documentation . . . 917
4.47.5.1 cpl_type_get_name() . . . 917

4.47.5.2 cpl_type_get_sizeof() . . . 918
4.48 Unit testing functions . . . 919
4.48.1 Detailed Description . . . 920
4.48.2 Macro Definition Documentation . . . 920
4.48.2.1 cpl_test . . . 920
4.48.2.2 cpl_test_abs . . . 921
4.48.2.3 cpl_test_abs_complex . . . 922
4.48.2.4 cpl_test_array_abs . . . 922
4.48.2.5 cpl_test_assert . . . 923
4.48.2.6 cpl_test_eq . . . 923
4.48.2.7 cpl_test_eq_error . . . 924
4.48.2.8 cpl_test_eq_mask . . . 925
4.48.2.9 cpl_test_eq_ptr . . . 925
4.48.2.10 cpl_test_eq_string . . . 926
4.48.2.11 cpl_test_error . . . 927
4.48.2.12 cpl_test_errorstate . . . 927
4.48.2.13 cpl_test_fits . . . 928
4.48.2.14 cpl_test_image_abs . . . 929
4.48.2.15 cpl_test_image_rel . . . 929
4.48.2.16 cpl_test_imagelist_abs . . . 930
4.48.2.17 cpl_test_init . . . 930
4.48.2.18 cpl_test_leq . . . 931
4.48.2.19 cpl_test_lt . . . 932
4.48.2.20 cpl_test_matrix_abs . . . 932
4.48.2.21 cpl_test_memory_is_empty . . . 933
4.48.2.22 cpl_test_noneq . . . 933
4.48.2.23 cpl_test_noneq_ptr . . . 934
4.48.2.24 cpl_test_noneq_string . . . 934
4.48.2.25 cpl_test_nonnull . . . 935
4.48.2.26 cpl_test_null . . . 935
4.48.2.27 cpl_test_polynomial_abs . . . 936
4.48.2.28 cpl_test_rel . . . 936
4.48.2.29 cpl_test_vector_abs . . . 937
4.48.2.30 cpl_test_zero . . . 937
4.48.3 Function Documentation . . . 938
4.48.3.1 cpl_test_end() . . . 938
4.48.3.2 cpl_test_get_bytes_image() . . . 939
4.48.3.3 cpl_test_get_bytes_imagelist() . . . 939
4.48.3.4 cpl_test_get_bytes_matrix() . . . 940
4.48.3.5 cpl_test_get_bytes_vector() . . . 940
4.48.3.6 cpl_test_get_failed() . . . 941
4.48.3.7 cpl_test_get_tested() . . . 942

4.48.3.8 cpl_test_get_walltime() . . . 942
4.49 Vector . . . 943
4.49.1 Detailed Description . . . 945
4.49.2 Function Documentation . . . 945
4.49.2.1 cpl_vector_add() . . . 945
4.49.2.2 cpl_vector_add_scalar() . . . 946
4.49.2.3 cpl_vector_convolve_symmetric() . . . 946
4.49.2.4 cpl_vector_copy() . . . 947
4.49.2.5 cpl_vector_correlate() . . . 947
4.49.2.6 cpl_vector_cycle() . . . 948
4.49.2.7 cpl_vector_delete() . . . 949
4.49.2.8 cpl_vector_divide() . . . 950
4.49.2.9 cpl_vector_divide_scalar() . . . 950
4.49.2.10 cpl_vector_dump() . . . 951
4.49.2.11 cpl_vector_duplicate() . . . 952
4.49.2.12 cpl_vector_exponential() . . . 952
4.49.2.13 cpl_vector_extract() . . . 953
4.49.2.14 cpl_vector_fill() . . . 953
4.49.2.15 cpl_vector_fill_kernel_profile() . . . 954
4.49.2.16 cpl_vector_filter_lowpass_create() . . . 955
4.49.2.17 cpl_vector_filter_median_create() . . . 956
4.49.2.18 cpl_vector_find() . . . 956
4.49.2.19 cpl_vector_fit_gaussian() . . . 957
4.49.2.20 cpl_vector_get() . . . 959
4.49.2.21 cpl_vector_get_data() . . . 960
4.49.2.22 cpl_vector_get_data_const() . . . 960
4.49.2.23 cpl_vector_get_max() . . . 961
4.49.2.24 cpl_vector_get_maxpos() . . . 961
4.49.2.25 cpl_vector_get_mean() . . . 962
4.49.2.26 cpl_vector_get_median() . . . 962
4.49.2.27 cpl_vector_get_median_const() . . . 963
4.49.2.28 cpl_vector_get_min() . . . 964
4.49.2.29 cpl_vector_get_minpos() . . . 964
4.49.2.30 cpl_vector_get_size() . . . 965
4.49.2.31 cpl_vector_get_stdev() . . . 965
4.49.2.32 cpl_vector_get_sum() . . . 966
4.49.2.33 cpl_vector_load() . . . 966
4.49.2.34 cpl_vector_logarithm() . . . 967
4.49.2.35 cpl_vector_multiply() . . . 968
4.49.2.36 cpl_vector_multiply_scalar() . . . 968
4.49.2.37 cpl_vector_new() . . . 969
4.49.2.38 cpl_vector_new_lss_kernel() . . . 969

4.49.2.39 cpl_vector_power() . . . 970
4.49.2.40 cpl_vector_product() . . . 970
4.49.2.41 cpl_vector_read() . . . 971
4.49.2.42 cpl_vector_save() . . . 972
4.49.2.43 cpl_vector_set() . . . 972
4.49.2.44 cpl_vector_set_size() . . . 973
4.49.2.45 cpl_vector_sort() . . . 974
4.49.2.46 cpl_vector_sqrt() . . . 974
4.49.2.47 cpl_vector_subtract() . . . 975
4.49.2.48 cpl_vector_subtract_scalar() . . . 976
4.49.2.49 cpl_vector_unwrap() . . . 976
4.49.2.50 cpl_vector_wrap() . . . 977
4.50 Wavelength calibration . . . 978
4.50.1 Detailed Description . . . 978
4.50.2 Function Documentation . . . 978
4.50.2.1 cpl_wlcalib_fill_line_spectrum() . . . 978
4.50.2.2 cpl_wlcalib_fill_line_spectrum_fast() . . . 979
4.50.2.3 cpl_wlcalib_fill_logline_spectrum() . . . 980
4.50.2.4 cpl_wlcalib_fill_logline_spectrum_fast() . . . 981
4.50.2.5 cpl_wlcalib_find_best_1d() . . . 982
4.50.2.6 cpl_wlcalib_slitmodel_delete() . . . 983
4.50.2.7 cpl_wlcalib_slitmodel_new() . . . 983
4.50.2.8 cpl_wlcalib_slitmodel_set_catalog() . . . 984
4.50.2.9 cpl_wlcalib_slitmodel_set_threshold() . . . 985
4.50.2.10 cpl_wlcalib_slitmodel_set_wfwhm() . . . 985
4.50.2.11 cpl_wlcalib_slitmodel_set_wslit() . . . 986
4.51 World Coordinate System . . . 987
4.51.1 Detailed Description . . . 987
4.51.2 Macro Definition Documentation . . . 987
4.51.2.1 CPL_WCS_REGEXP . . . 988
4.51.3 Function Documentation . . . 988
4.51.3.1 cpl_wcs_convert() . . . 988
4.51.3.2 cpl_wcs_delete() . . . 989
4.51.3.3 cpl_wcs_get_cd() . . . 989
4.51.3.4 cpl_wcs_get_crpix() . . . 991
4.51.3.5 cpl_wcs_get_crval() . . . 992
4.51.3.6 cpl_wcs_get_ctype() . . . 992
4.51.3.7 cpl_wcs_get_cunit() . . . 993
4.51.3.8 cpl_wcs_get_image_dims() . . . 993
4.51.3.9 cpl_wcs_get_image_naxis() . . . 994
4.51.3.10 cpl_wcs_new_from_propertylist() . . . 995
4.51.3.11 cpl_wcs_platesol() . . . 996

5 Class Documentation **999**

5.1 _cpl_framedata_ Struct Reference 999

5.1.1 Detailed Description 999

5.1.2 Member Data Documentation 999

5.1.2.1 max_count 999

5.1.2.2 min_count 1000

5.1.2.3 tag 1000

5.2 _cpl_plugin_ Struct Reference 1000

5.2.1 Detailed Description 1000

5.2.2 Member Data Documentation 1001

5.2.2.1 api 1001

5.2.2.2 author 1001

5.2.2.3 copyright 1001

5.2.2.4 deinitialize 1001

5.2.2.5 description 1002

5.2.2.6 email 1002

5.2.2.7 execute 1002

5.2.2.8 initialize 1002

5.2.2.9 name 1003

5.2.2.10 synopsis 1003

5.2.2.11 type 1003

5.2.2.12 version 1003

5.3 _cpl_recipe_ Struct Reference 1003

5.3.1 Detailed Description 1004

5.3.2 Member Data Documentation 1004

5.3.2.1 frames 1004

5.3.2.2 interface 1004

5.3.2.3 parameters 1004

Index **1005**

Chapter 1

Deprecated List

Member cpl_frameset_get_first (cpl_frameset *self)

This function will be removed from CPL version 7. Code using these functions should be ported to make use of frame set iterators instead!

Member cpl_msg_progress (const char *component, int i, int iter, const char *format,...)

Use standard calls such as cpl_msg_info() instead.

Member cpl_polynomial_fit_1d_create (const cpl_vector *x_pos, const cpl_vector *values, cpl_size degree, double *pmse)

Replace this call with cpl_polynomial_fit() and optionally cpl_vector_fill_polynomial_fit_residual().

Member cpl_polynomial_fit_2d_create (const cpl_bivector *xy_pos, const cpl_vector *values, cpl_size degree, double *pmse)

Replace this call with cpl_polynomial_fit() and optionally cpl_vector_fill_polynomial_fit_residual().

Member cpl_table_get_column_name (const cpl_table *table)

This function is deprecated, because its usage could create serious problems in case it is attempted to get names from different tables simultaneously. For instance, a programmer may call cpl_table_get_column_name() in a loop, and in the same loop call a CPL function that calls as well the same function. The behaviour in this case would be unpredictable. The function cpl_table_get_column_names() should be used instead.

Member cpl_test_memory_is_empty ()

Called by cpl_test_end()

Member cpl_vector_new_lss_kernel (double slitw, double fwhm)

Unstable API, may change or disappear. Do not use in new code!

Member cpl_vector_convolve_symmetric (cpl_vector *smoothed, const cpl_vector *conv_kernel)

Unstable API, may change or disappear. Do not use in new code!

Member cpl_frameset_get_first_const (const cpl_frameset *self)

This function will be removed from CPL version 7. Code using these functions should be ported to make use of frame set iterators instead!

Member cpl_mask_dilation (cpl_mask *in, const cpl_matrix *ker)

Replace this call with cpl_mask_filter() using CPL_FILTER_DILATION and CPL_BORDER_ZERO.

Member cpl_frameset_get_next_const (const cpl_frameset *self)

This function will be removed from CPL version 7. Code using these functions should be ported to make use of frame set iterators instead!

Member cpl_frameset_get_next (cpl_frameset *self)

This function will be removed from CPL version 7. Code using these functions should be ported to make use of frame set iterators instead!

Member cpl_frameset_get_frame_const (const cpl_frameset *set, cpl_size position)

This function will be removed from CPL version 7. Code using these functions should use cpl_frameset_get_position_const() instead!

Member cpl_frameset_get_frame (cpl_frameset *set, cpl_size position)

This function will be removed from CPL version 7. Code using these functions should use cpl_frameset_get_position() instead!

Member CPL_RECIPE_DEFINE (RECIPE_NAME, RECIPE_VERSION, RECIPE_FILL_PARAMS, RECIPE_AUTHOR, RECIPE_AUTHOR_EMAIL, RECIPE_YEAR, RECIPE_SYNOPSIS, RECIPE_DESCRIPTION)

Use cpl_recipe_define()

Member cpl_apertures_get_max_x (const cpl_apertures *self, cpl_size ind)

Replace this function with cpl_apertures_get_pos_x()

Member cpl_apertures_get_max_y (const cpl_apertures *self, cpl_size ind)

Replace this function with cpl_apertures_get_pos_y()

Member cpl_apertures_get_fwhm (const cpl_image *self, const cpl_apertures *aperts)

Replace this call with a loop over cpl_image_get_fwhm()

Member CPL_BPP_16_UNSIGNED

Use CPL_TYPE_USHORT

Member cpl_fits_get_extension_nb (const char *filename, const char *extname)

Replace this call with cpl_fits_find_extension().

Member cpl_image_fit_gaussian (const cpl_image *im, cpl_size xpos, cpl_size ypos, cpl_size size, double *norm, double *xcen, double *ycen, double *sig_x, double *sig_y, double *fwhm_x, double *fwhm_y)

If you need a 2D gaussian fit please use the function *cpl_fit_image_gaussian()*. Please note that on CPL versions earlier than 5.1.0 this function was wrongly documented: the parameters *sig_x* and *sig_y* were defined as "the sigma in x (or y) of the gaussian", while actually they returned the semi-major and semi-minor axes of the gaussian distribution at 1-sigma. ***PLEASE NOTE THAT IF YOU USED THIS FUNCTION FOR DETERMINING THE SPREAD OF A DISTRIBUTION ALONG THE X DIRECTION, THIS WAS VERY LIKELY OVERESTIMATED*** (because *sig_x* was always assigned the semi-major axis of the distribution ignoring the rotation), ***WHILE THE SPREAD ALONG THE Y DIRECTION WOULD BE UNDERESTIMATED***. In addition to that, even with circular distributions this function may lead to an underestimation of *sig_x* and *sig_y* (up to 25% underestimation in the case of noiseless data with a box 4 times the sigma, 1% underestimation in the case of noiseless data with a box 7 times the sigma). This latter problem is related to the function *cpl_image_iqe()*.

Member cpl_image_filter_linear (const cpl_image *in, const cpl_matrix *ker)

Replace this call with cpl_image_filter() using CPL_FILTER_LINEAR and CPL_BORDER_FILTER.

Member cpl_image_filter_morpho (const cpl_image *in, const cpl_matrix *ker)

Replace this call with cpl_image_filter() using CPL_FILTER_MORPHO and CPL_BORDER_FILTER.

Member cpl_image_filter_median (const cpl_image *in, const cpl_matrix *ker)

Replace this call with cpl_image_filter_mask() using CPL_FILTER_MEDIAN and CPL_BORDER_FILTER.

Member cpl_image_filter_stdev (const cpl_image *in, const cpl_matrix *ker)

Replace this call with cpl_image_filter_mask() using CPL_FILTER_STDEV and CPL_BORDER_FILTER.

Member CPL_BPP_8_UNSIGNED

Use CPL_TYPE_UCHAR

Member CPL_BPP_16_SIGNED

Use CPL_TYPE_SHORT

Member cpl_fits_get_nb_extensions (const char *filename)

Replace this call with cpl_fits_count_extensions().

Member CPL_BPP_32_SIGNED

Use CPL_TYPE_INT

Member CPL_BPP_IEEE_FLOAT

Use CPL_TYPE_FLOAT

Member CPL_BPP_IEEE_DOUBLE

Use CPL_TYPE_DOUBLE

Member cpl_type_bpp

Use cpl_type

Member cpl_mask_opening (cpl_mask *in, const cpl_matrix *ker)

Replace this call with cpl_mask_filter() using CPL_FILTER_OPENING and CPL_BORDER_ZERO.

Member cpl_mask_closing (cpl_mask *in, const cpl_matrix *ker)

Replace this call with cpl_mask_filter() using CPL_FILTER_CLOSING and CPL_BORDER_ZERO.

Member cpl_mask_erosion (cpl_mask *in, const cpl_matrix *ker)

Replace this call with cpl_mask_filter() using CPL_FILTER_EROSION and CPL_BORDER_ZERO.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Arrays . . . 9
Auxiliary Frame Data . . . 86
Bi-vector object . . . 96
DFS related functions . . . 109
DICB specific property functionality . . . 119
Error handling . . . 120
FFTW wrappers . . . 131
FITS card related basic routines . . . 134
FITS related basic routines . . . 135
Filters . . . 141
Frame Sets . . . 154
    Frame Set Iterators . . . 147
Frame Sets IO functions . . . 173
Frames . . . 174
Fundamental math functionality . . . 188
Handling of multiple CPL errors . . . 198
High level functions for geometric transformations . . . 204
High level functions to handle apertures . . . 210
High-level functions for non-linear fitting . . . 235
High-level functions that are photometry related . . . 244
High-level functions to compute detector features . . . 246
I/O . . . 250
Imagelists . . . 253
Images . . . 283
Library Initialization . . . 396
Library Version Information . . . 398
Masks of pixels . . . 401
Matrices . . . 430
Memory Management Utilities . . . 481
Messages . . . 487
Multi Frames . . . 505
    Regular Expression Filter . . . 757
Parameter Lists . . . 514
Parameters . . . 530
Plotting of CPL objects . . . 568

Plugin Interface 579
Plugin List 604
Point pattern matching module 611
Polynomials 616
Properties 640
Property Lists 663
Recipe Configurations 740
Recipe Definition 753
Recipes 756
Statistics 761
Tables 776
Type codes 914
Unit testing functions 919
Vector 943
Wavelength calibration 978
World Coordinate System 987

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

_cpl_framedata_
: The public frame data object 999

_cpl_plugin_
: The type representation of the generic plugin interface 1000

_cpl_recipe_
: The type representation of the recipe plugin interface 1003

Chapter 4

Module Documentation

4.1 Arrays

Functions

- cpl_error_code cpl_array_abs (cpl_array *array)

  Compute the absolute value of array elements.
- cpl_error_code cpl_array_add (cpl_array *to_array, const cpl_array *from_array)

  Add the values of two numeric or complex arrays.
- cpl_error_code cpl_array_add_scalar (cpl_array *array, double value)

  Add a constant value to a numerical array.
- cpl_error_code cpl_array_add_scalar_complex (cpl_array *array, double complex value)

  Add a constant complex value to a complex array.
- cpl_error_code cpl_array_arg (cpl_array *array)

  Compute the phase angle value of array elements.
- cpl_array * cpl_array_cast (const cpl_array *array, cpl_type type)

  Cast a numeric array to a new numeric type array.
- cpl_error_code cpl_array_copy_data (cpl_array *array, const double *data)

  Copy buffer of numerical data to a numerical array.
- cpl_error_code cpl_array_copy_data_complex (cpl_array *array, const double complex *data)

  Copy buffer of complex data to a complex array.
- cpl_error_code cpl_array_copy_data_cplsize (cpl_array *array, const cpl_size *data)

  Copy existing data to a cpl_size array.
- cpl_error_code cpl_array_copy_data_double (cpl_array *array, const double *data)

  Copy existing data to a double array.
- cpl_error_code cpl_array_copy_data_double_complex (cpl_array *array, const double complex *data)

  Copy existing data to a double complex array.
- cpl_error_code cpl_array_copy_data_float (cpl_array *array, const float *data)

  Copy existing data to a float array.
- cpl_error_code cpl_array_copy_data_float_complex (cpl_array *array, const float complex *data)

  Copy existing data to a float complex array.
- cpl_error_code cpl_array_copy_data_int (cpl_array *array, const int *data)

  Copy existing data to an integer array.
- cpl_error_code cpl_array_copy_data_long (cpl_array *array, const long *data)

  Copy existing data to a long integer array.
- cpl_error_code cpl_array_copy_data_long_long (cpl_array *array, const long long *data)

Copy existing data to a long long integer array.

- cpl_error_code cpl_array_copy_data_string (cpl_array *array, const char **data)

  Copy existing data to a string array.
- cpl_size cpl_array_count_invalid (const cpl_array *array)

  Count number of invalid elements in an array.
- void cpl_array_delete (cpl_array *array)

  Delete an array.
- cpl_error_code cpl_array_divide (cpl_array *to_array, const cpl_array *from_array)

  Divide the values of two numeric or complex arrays.
- cpl_error_code cpl_array_divide_scalar (cpl_array *array, double value)

  Divide a numerical array by a constant value.
- cpl_error_code cpl_array_divide_scalar_complex (cpl_array *array, double complex value)

  Divide a complex array by a constant complex value.
- void cpl_array_dump (const cpl_array *array, cpl_size start, cpl_size count, FILE *stream)

  Print an array.
- void cpl_array_dump_structure (const cpl_array *array, FILE *stream)

  Describe the structure and the contents of an array.
- cpl_array * cpl_array_duplicate (const cpl_array *array)

  Make a copy of an array.
- cpl_error_code cpl_array_erase_window (cpl_array *array, cpl_size start, cpl_size count)

  Delete a segment of an array.
- cpl_error_code cpl_array_exponential (cpl_array *array, double base)

  Compute the exponential of array elements.
- cpl_array * cpl_array_extract (const cpl_array *array, cpl_size start, cpl_size count)

  Create an array from a section of another array.
- cpl_array * cpl_array_extract_imag (const cpl_array *array)

  Extract the imaginary value of array elements.
- cpl_array * cpl_array_extract_real (const cpl_array *array)

  Extract the real value of array elements.
- cpl_error_code cpl_array_fill_window (cpl_array *array, cpl_size start, cpl_size count, double value)

  Write the same value within a numerical array segment.
- cpl_error_code cpl_array_fill_window_complex (cpl_array *array, cpl_size start, cpl_size count, double complex value)

  Write the same value within a complex array segment.
- cpl_error_code cpl_array_fill_window_cplsize (cpl_array *array, cpl_size start, cpl_size count, cpl_size value)

  Write the same value within a cpl_size array segment.
- cpl_error_code cpl_array_fill_window_double (cpl_array *array, cpl_size start, cpl_size count, double value)

  Write the same value within a double array segment.
- cpl_error_code cpl_array_fill_window_double_complex (cpl_array *array, cpl_size start, cpl_size count, double complex value)

  Write the same value within a double complex array segment.
- cpl_error_code cpl_array_fill_window_float (cpl_array *array, cpl_size start, cpl_size count, float value)

  Write the same value within a float array segment.
- cpl_error_code cpl_array_fill_window_float_complex (cpl_array *array, cpl_size start, cpl_size count, float complex value)

  Write the same value within a float complex array segment.
- cpl_error_code cpl_array_fill_window_int (cpl_array *array, cpl_size start, cpl_size count, int value)

  Write the same value within an integer array segment.
- cpl_error_code cpl_array_fill_window_invalid (cpl_array *array, cpl_size start, cpl_size count)

  Set an array segment to NULL.
- cpl_error_code cpl_array_fill_window_long (cpl_array *array, cpl_size start, cpl_size count, long value)

Write the same value within a long integer array segment.

- cpl_error_code cpl_array_fill_window_long_long (cpl_array *array, cpl_size start, cpl_size count, long long value)

  Write the same value within a long long integer array segment.
- cpl_error_code cpl_array_fill_window_string (cpl_array *array, cpl_size start, cpl_size count, const char *value)

  Write a string to a string array segment.
- double cpl_array_get (const cpl_array *array, cpl_size indx, int *null)

  Read a value from a numerical array.
- double complex cpl_array_get_complex (const cpl_array *array, cpl_size indx, int *null)

  Read a value from a complex array.
- cpl_size cpl_array_get_cplsize (const cpl_array *array, cpl_size indx, int *null)

  Read a value from a cpl_size array.
- cpl_size * cpl_array_get_data_cplsize (cpl_array *array)

  Get a pointer to `cpl_size` array data.
- const cpl_size * cpl_array_get_data_cplsize_const (const cpl_array *array)

  Get a pointer to constant `cpl_size` array data.
- double * cpl_array_get_data_double (cpl_array *array)

  Get a pointer to `double` array data.
- double complex * cpl_array_get_data_double_complex (cpl_array *array)

  Get a pointer to `double` complex array data.
- const double complex * cpl_array_get_data_double_complex_const (const cpl_array *array)

  Get a pointer to constant double complex array data.
- const double * cpl_array_get_data_double_const (const cpl_array *array)

  Get a pointer to constant double array data.
- float * cpl_array_get_data_float (cpl_array *array)

  Get a pointer to `float` array data.
- float complex * cpl_array_get_data_float_complex (cpl_array *array)

  Get a pointer to `float` complex array data.
- const float complex * cpl_array_get_data_float_complex_const (const cpl_array *array)

  Get a pointer to constant float complex array data.
- const float * cpl_array_get_data_float_const (const cpl_array *array)

  Get a pointer to constant float array data.
- int * cpl_array_get_data_int (cpl_array *array)

  Get a pointer to `integer` array data.
- const int * cpl_array_get_data_int_const (const cpl_array *array)

  Get a pointer to constant `integer` array data.
- long * cpl_array_get_data_long (cpl_array *array)

  Get a pointer to `long integer` array data.
- const long * cpl_array_get_data_long_const (const cpl_array *array)

  Get a pointer to constant `long integer` array data.
- long long * cpl_array_get_data_long_long (cpl_array *array)

  Get a pointer to `long long integer` array data.
- const long long * cpl_array_get_data_long_long_const (const cpl_array *array)

  Get a pointer to constant `long long integer` array data.
- char ** cpl_array_get_data_string (cpl_array *array)

  Get a pointer to string array data.
- const char ** cpl_array_get_data_string_const (const cpl_array *array)

  Get a pointer to constant string array data.
- double cpl_array_get_double (const cpl_array *array, cpl_size indx, int *null)

  Read a value from a double array.

- double complex cpl_array_get_double_complex (const cpl_array *array, cpl_size indx, int *null)

  Read a value from a double complex array.
- float cpl_array_get_float (const cpl_array *array, cpl_size indx, int *null)

  Read a value from a float array.
- float complex cpl_array_get_float_complex (const cpl_array *array, cpl_size indx, int *null)

  Read a value from a float complex array.
- int cpl_array_get_int (const cpl_array *array, cpl_size indx, int *null)

  Read a value from an integer array.
- long cpl_array_get_long (const cpl_array *array, cpl_size indx, int *null)

  Read a value from a long integer array.
- long long cpl_array_get_long_long (const cpl_array *array, cpl_size indx, int *null)

  Read a value from a long long integer array.
- double cpl_array_get_max (const cpl_array *array)

  Get maximum value in a numerical array.
- cpl_error_code cpl_array_get_maxpos (const cpl_array *array, cpl_size *indx)

  Get position of maximum in a numerical array.
- double cpl_array_get_mean (const cpl_array *array)

  Compute the mean value of a numeric array.
- double complex cpl_array_get_mean_complex (const cpl_array *array)

  Compute the mean value of a complex array.
- double cpl_array_get_median (const cpl_array *array)

  Compute the median of a numeric array.
- double cpl_array_get_min (const cpl_array *array)

  Get minimum value in a numerical array.
- cpl_error_code cpl_array_get_minpos (const cpl_array *array, cpl_size *indx)

  Get position of minimum in a numerical array.
- cpl_size cpl_array_get_size (const cpl_array *array)

  Get the length of an array.
- double cpl_array_get_stdev (const cpl_array *array)

  Compute the standard deviation of a numeric array.
- const char * cpl_array_get_string (const cpl_array *array, cpl_size indx)

  Read a value from a string array.
- cpl_type cpl_array_get_type (const cpl_array *array)

  Get the type of an array.
- int cpl_array_has_invalid (const cpl_array *array)

  Check if an array contains at least one invalid element.
- int cpl_array_has_valid (const cpl_array *array)

  Check if an array contains at least one valid value.
- cpl_error_code cpl_array_insert (cpl_array *target_array, const cpl_array *insert_array, cpl_size start)

  Merge two arrays.
- cpl_error_code cpl_array_insert_window (cpl_array *array, cpl_size start, cpl_size count)

  Insert a segment of new elements into array.
- int cpl_array_is_valid (const cpl_array *array, cpl_size indx)

  Check if an array element is valid.
- cpl_error_code cpl_array_logarithm (cpl_array *array, double base)

  Compute the logarithm of array elements.
- cpl_error_code cpl_array_multiply (cpl_array *to_array, const cpl_array *from_array)

  Multiply the values of two numeric or complex arrays.
- cpl_error_code cpl_array_multiply_scalar (cpl_array *array, double value)

  Multiply a numerical array by a constant value.
- cpl_error_code cpl_array_multiply_scalar_complex (cpl_array *array, double complex value)

Multiply a complex array by a constant complex value.

- cpl_array * cpl_array_new (cpl_size length, cpl_type type)

  Create a new array of given type.
- cpl_error_code cpl_array_power (cpl_array *array, double exponent)

  Compute the power of numerical array elements.
- cpl_error_code cpl_array_power_complex (cpl_array *array, double complex exponent)

  Compute the complex power of complex, numerical array elements.
- cpl_error_code cpl_array_set (cpl_array *array, cpl_size indx, double value)

  Write a value to a numerical array element.
- cpl_error_code cpl_array_set_complex (cpl_array *array, cpl_size indx, double complex value)

  Write a value to a complex array element.
- cpl_error_code cpl_array_set_cplsize (cpl_array *array, cpl_size indx, cpl_size value)

  Write a value to a cpl_size array element.
- cpl_error_code cpl_array_set_double (cpl_array *array, cpl_size indx, double value)

  Write a value to a double array element.
- cpl_error_code cpl_array_set_double_complex (cpl_array *array, cpl_size indx, double complex value)

  Write a value to a double complex array element.
- cpl_error_code cpl_array_set_float (cpl_array *array, cpl_size indx, float value)

  Write a value to a float array element.
- cpl_error_code cpl_array_set_float_complex (cpl_array *array, cpl_size indx, float complex value)

  Write a value to a float complex array element.
- cpl_error_code cpl_array_set_int (cpl_array *array, cpl_size indx, int value)

  Write a value to an integer array element.
- cpl_error_code cpl_array_set_invalid (cpl_array *array, cpl_size indx)

  Invalidate an array element.
- cpl_error_code cpl_array_set_long (cpl_array *array, cpl_size indx, long value)

  Write a value to a long integer array element.
- cpl_error_code cpl_array_set_long_long (cpl_array *array, cpl_size indx, long long value)

  Write a value to a long long integer array element.
- cpl_error_code cpl_array_set_size (cpl_array *array, cpl_size new_length)

  Resize an array.
- cpl_error_code cpl_array_set_string (cpl_array *array, cpl_size indx, const char *string)

  Write a character string to a string array element.
- cpl_error_code cpl_array_subtract (cpl_array *to_array, const cpl_array *from_array)

  Subtract the values of two numeric or complex arrays.
- cpl_error_code cpl_array_subtract_scalar (cpl_array *array, double value)

  Subtract a constant value from a numerical array.
- cpl_error_code cpl_array_subtract_scalar_complex (cpl_array *array, double complex value)

  Subtract a constant complex value from a complex array.
- void * cpl_array_unwrap (cpl_array *array)

  Delete an array, without losing the data buffer.
- cpl_array * cpl_array_wrap_cplsize (cpl_size *data, cpl_size length)

  Create a new cpl_size array from existing data.
- cpl_array * cpl_array_wrap_double (double *data, cpl_size length)

  Create a new double array from existing data.
- cpl_array * cpl_array_wrap_double_complex (double complex *data, cpl_size length)

  Create a new double complex array from existing data.
- cpl_array * cpl_array_wrap_float (float *data, cpl_size length)

  Create a new float array from existing data.
- cpl_array * cpl_array_wrap_float_complex (float complex *data, cpl_size length)

  Create a new float complex array from existing data.

- cpl_array * cpl_array_wrap_int (int *data, cpl_size length)

  Create a new integer array from existing data.
- cpl_array * cpl_array_wrap_long (long *data, cpl_size length)

  Create a new long integer array from existing data.
- cpl_array * cpl_array_wrap_long_long (long long *data, cpl_size length)

  Create a new long long integer array from existing data.
- cpl_array * cpl_array_wrap_string (char **data, cpl_size length)

  Create a new character string array from existing data.

4.1.1 Detailed Description

This module provides functions to create, destroy and use a *cpl_array*.

Synopsis:

```
#include <cpl_array.h>
```

4.1.2 Function Documentation

4.1.2.1 cpl_array_abs()

```
cpl_error_code cpl_array_abs (
            cpl_array * array )
```

Compute the absolute value of array elements.

Parameters

array	Pointer to array.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is `NULL` pointer.
CPL_ERROR_INVALID_TYPE	The specified *array* is not numerical.

Each array element is replaced by its absolute value. Invalid elements are not modified by this operation. If the array is complex, its type will be turned to real (CPL_TYPE_FLOAT_COMPLEX will be changed into CPL_TYPE_FLOAT, and CPL_TYPE_DOUBLE_COMPLEX will be changed into CPL_TYPE_DOUBLE), and any pointer retrieved by

calling cpl_array_get_data_float(), cpl_array_get_data_double_complex(), etc., should be discarded.

References cpl_array_get_type(), CPL_ERROR_NULL_INPUT, and CPL_TYPE_COMPLEX.

4.1.2.2 cpl_array_add()

```
cpl_error_code cpl_array_add (
            cpl_array * to_array,
            const cpl_array * from_array )
```

Add the values of two numeric or complex arrays.

Parameters

to_array	Target array.
from_array	Source array.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any input *array* is a NULL pointer.
CPL_ERROR_INCOMPATIBLE_INPUT	The input arrays have different sizes.
CPL_ERROR_INVALID_TYPE	Any specified array is not numerical.

The arrays are summed element by element, and the result of the sum is stored in the target array. The arrays' types may differ, and in that case the operation would be performed using the standard C upcasting rules, with a final cast of the result to the target array type. Invalid elements are propagated consistently: if either or both members of the sum are invalid, the result will be invalid too. Underflows and overflows are ignored.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.3 cpl_array_add_scalar()

```
cpl_error_code cpl_array_add_scalar (
            cpl_array * array,
            double value )
```

Add a constant value to a numerical array.

Parameters

array	Target array
value	Value to add.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is a `NULL` pointer.
CPL_ERROR_INVALID_TYPE	The input array is not numerical.

The operation is always performed in double precision, with a final cast of the result to the target array type. Invalid elements are are not modified by this operation.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.4 cpl_array_add_scalar_complex()

```
cpl_error_code cpl_array_add_scalar_complex (
            cpl_array * array,
            double complex value )
```

Add a constant complex value to a complex array.

Parameters

array	Target array
value	Value to add.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is a `NULL` pointer.
CPL_ERROR_INVALID_TYPE	The input array is not complex.

The operation is always performed in double precision, with a final cast of the result to the target array type. Invalid elements are are not modified by this operation.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.5 cpl_array_arg()

```
cpl_error_code cpl_array_arg (
            cpl_array * array )
```

Compute the phase angle value of array elements.

Parameters

array	Pointer to array.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is NULL pointer.
CPL_ERROR_INVALID_TYPE	The specified *array* is not numerical.

Each array element is replaced by its phase angle value. The phase angle will be in the range of [-pi,pi]. Invalid elements are not modified by this operation. If the array is complex, its type will be turned to real (CPL_TY↩ PE_FLOAT_COMPLEX will be changed into CPL_TYPE_FLOAT, and CPL_TYPE_DOUBLE_COMPLEX will be changed into CPL_TYPE_DOUBLE), and any pointer retrieved by calling cpl_array_get_data_float(), cpl_array_get_data_double_complex(), etc., should be discarded.

References cpl_array_get_type(), CPL_ERROR_NULL_INPUT, and CPL_TYPE_COMPLEX.

4.1.2.6 cpl_array_cast()

```
cpl_array* cpl_array_cast (
            const cpl_array * array,
            cpl_type type )
```

Cast a numeric array to a new numeric type array.

Parameters

array	Pointer to array.
type	Type of new array.

Returns

New array.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is a `NULL` pointer.
CPL_ERROR_INVALID_TYPE	The specified column is not numerical.
CPL_ERROR_ILLEGAL_INPUT	The specified *type* is not numerical.

A new array of the specified type is created, and the content of the input numeric array is cast to the new type. If the input array type is identical to the specified type the array is duplicated as is done by the function `cpl_array_duplicate()`.

References cpl_calloc(), CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_free(), CPL_TYPE_↩ DOUBLE, CPL_TYPE_DOUBLE_COMPLEX, CPL_TYPE_FLOAT, CPL_TYPE_FLOAT_COMPLEX, CPL_TYPE↩ _INT, CPL_TYPE_LONG, CPL_TYPE_LONG_LONG, and CPL_TYPE_SIZE.

4.1.2.7 cpl_array_copy_data()

```
cpl_error_code cpl_array_copy_data (
            cpl_array * array,
            const double * data )
```

Copy buffer of numerical data to a numerical array.

Parameters

array	Existing array.
data	Existing data buffer.

Returns

`CPL_ERROR_NONE` on success. If the array is not numerical, a `CPL_ERROR_INVALID_TYPE` is returned. At any `NULL` input pointer a `CPL_ERROR_NULL_INPUT` would be returned.

The input data are copied into the specified array. If the type of the accessed array is not *CPL_TYPE_DO↩ UBLE*, the data values will be truncated according to C casting rules. The size of the input data buffer is not checked in any way, and the values are all considered valid: invalid values should be marked using the functions `cpl_array_set_invalid()`. If *N* is the length of the array, the first *N* values of the input data buffer would be copied to the column buffer. If the array had length zero, no values would be copied.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.8 cpl_array_copy_data_complex()

```
cpl_error_code cpl_array_copy_data_complex (
            cpl_array * array,
            const double complex * data )
```

Copy buffer of complex data to a complex array.

Parameters

array	Existing array.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success. If the array is not complex, a CPL_ERROR_INVALID_TYPE is returned. At any NULL input pointer a CPL_ERROR_NULL_INPUT would be returned.

The input data are copied into the specified array. If the type of the accessed array is not *CPL_TYPE_DOUBLE*, the data values will be truncated according to C casting rules. The size of the input data buffer is not checked in any way, and the values are all considered valid: invalid values should be marked using the functions cpl_array_set_invalid(). If *N* is the length of the array, the first *N* values of the input data buffer would be copied to the column buffer. If the array had length zero, no values would be copied.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.9 cpl_array_copy_data_cplsize()

```
cpl_error_code cpl_array_copy_data_cplsize (
            cpl_array * array,
            const cpl_size * data )
```

Copy existing data to a *cpl_size* array.

Parameters

array	Existing array.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success. If the input array is not of type CPL_TYPE_SIZE, a CPL_ERROR_TYPE_MISMATCH is returned. At any NULL input pointer a CPL_ERROR_NULL_INPUT would be returned.

The input data are copied into the specified array. The size of the input data buffer is not checked in any way, and the data values are all considered valid: invalid values should be marked using the functions cpl_array_set_invalid(). If *N* is the length of the array, the first *N* values of the input data buffer would be copied to the array buffer. If the array had length zero, no values would be copied.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.10 cpl_array_copy_data_double()

```
cpl_error_code cpl_array_copy_data_double (
            cpl_array * array,
            const double * data )
```

Copy existing data to a *double* array.

Parameters

array	Existing array.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success. If the input array is not of type CPL_TYPE_DOUBLE, a CPL_ERROR_TYPE_MISMATCH is returned. At any NULL input pointer a CPL_ERROR_NULL_INPUT would be returned.

See documentation of function cpl_array_copy_data_int().

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.11 cpl_array_copy_data_double_complex()

```
cpl_error_code cpl_array_copy_data_double_complex (
            cpl_array * array,
            const double complex * data )
```

Copy existing data to a *double* complex array.

Parameters

array	Existing array.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success. If the input array is not of type CPL_TYPE_DOUBLE_COMPLEX, a CPL_ERROR_TYPE_MISMATCH is returned. At any NULL input pointer a CPL_ERROR_NULL_INPUT would be returned.

See documentation of function cpl_array_copy_data_int().

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.12 cpl_array_copy_data_float()

```
cpl_error_code cpl_array_copy_data_float (
            cpl_array * array,
            const float * data )
```

Copy existing data to a *float* array.

Parameters

array	Existing array.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success. If the input array is not of type CPL_TYPE_FLOAT, a CPL_ERROR_TYPE_MISMATCH is returned. At any NULL input pointer a CPL_ERROR_NULL_INPUT would be returned.

See documentation of function cpl_array_copy_data_int().

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.13 cpl_array_copy_data_float_complex()

```
cpl_error_code cpl_array_copy_data_float_complex (
            cpl_array * array,
            const float complex * data )
```

Copy existing data to a *float* complex array.

Parameters

array	Existing array.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success. If the input array is not of type CPL_TYPE_FLOAT_COMPLEX, a CPL_ERROR_TYPE_MISMATCH is returned. At any NULL input pointer a CPL_ERROR_NULL_INPUT would be returned.

See documentation of function cpl_array_copy_data_int().

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.14 cpl_array_copy_data_int()

```
cpl_error_code cpl_array_copy_data_int (
            cpl_array * array,
            const int * data )
```

Copy existing data to an *integer* array.

Parameters

array	Existing array.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success. If the input array is not of type CPL_TYPE_INT, a CPL_ERROR_TYPE_MISMATCH is returned. At any NULL input pointer a CPL_ERROR_NULL_INPUT would be returned.

The input data are copied into the specified array. The size of the input data buffer is not checked in any way, and the data values are all considered valid: invalid values should be marked using the functions cpl_array_set_invalid(). If *N* is the length of the array, the first *N* values of the input data buffer would be copied to the array buffer. If the array had length zero, no values would be copied.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.15 cpl_array_copy_data_long()

```
cpl_error_code cpl_array_copy_data_long (
            cpl_array * array,
            const long * data )
```

Copy existing data to a *long integer* array.

Parameters

array	Existing array.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success. If the input array is not of type CPL_TYPE_LONG, a CPL_ERROR_TYPE_MISMATCH is returned. At any NULL input pointer a CPL_ERROR_NULL_INPUT would be returned.

The input data are copied into the specified array. The size of the input data buffer is not checked in any way, and the data values are all considered valid: invalid values should be marked using the functions cpl_array_set_invalid(). If *N* is the length of the array, the first *N* values of the input data buffer would be copied to the array buffer. If the array had length zero, no values would be copied.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.16 cpl_array_copy_data_long_long()

```
cpl_error_code cpl_array_copy_data_long_long (
            cpl_array * array,
            const long long * data )
```

Copy existing data to a *long long integer* array.

Parameters

array	Existing array.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success. If the input array is not of type CPL_TYPE_LONG_LONG, a CPL_ERROR_TYPE_MISMATCH is returned. At any NULL input pointer a CPL_ERROR_NULL_INPUT would be returned.

The input data are copied into the specified array. The size of the input data buffer is not checked in any way, and the data values are all considered valid: invalid values should be marked using the functions cpl_array_set_invalid(). If *N* is the length of the array, the first *N* values of the input data buffer would be copied to the array buffer. If the array had length zero, no values would be copied.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.17 cpl_array_copy_data_string()

```
cpl_error_code cpl_array_copy_data_string (
            cpl_array * array,
            const char ** data )
```

Copy existing data to a *string* array.

Parameters

array	Existing array.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success. If the input array is not of type CPL_TYPE_STRING, a CPL_ERROR_TYPE_MISMATCH is returned. At any NULL input pointer a CPL_ERROR_NULL_INPUT would be returned.

See documentation of function cpl_array_copy_data_int().

The input data are copied into the specified array. The size of the input buffer is not checked in any way. The strings pointed by the input buffer are all duplicated, while the strings contained in the array are released before being overwritten.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.18 cpl_array_count_invalid()

```
cpl_size cpl_array_count_invalid (
            const cpl_array * array )
```

Count number of invalid elements in an array.

Parameters

array	Array to inquire.

Returns

Number of invalid elements in an array. -1 is always returned in case of error.

Count number of invalid elements in an array. If the array itself is a `NULL` pointer, an error `CPL_ERROR_NULL_INPUT` is set.

References CPL_ERROR_NULL_INPUT.

4.1.2.19 cpl_array_delete()

```
void cpl_array_delete (
            cpl_array * array )
```

Delete an array.

Parameters

array	Array to be deleted.

Returns

Nothing.

This function deletes an array. If the input array is `NULL`, nothing is done, and no error is set.

References cpl_free().

Referenced by cpl_array_extract(), cpl_fit_image_gaussian(), cpl_table_compare_structure(), and cpl_wcs_delete().

4.1.2.20 cpl_array_divide()

```
cpl_error_code cpl_array_divide (
            cpl_array * to_array,
            const cpl_array * from_array )
```

Divide the values of two numeric or complex arrays.

Parameters

to_array	Target array.
from_array	Source array.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any input *array* is a NULL pointer.
CPL_ERROR_INCOMPATIBLE_INPUT	The input arrays have different sizes.
CPL_ERROR_INVALID_TYPE	Any specified array is not numerical.

The arrays are divided element by element, and the result is stored in the target array. See the documentation of the function cpl_array_add() for further details.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.21 cpl_array_divide_scalar()

```
cpl_error_code cpl_array_divide_scalar (
            cpl_array * array,
            double value )
```

Divide a numerical array by a constant value.

Parameters

array	Target array
value	Divisor.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is a NULL pointer.
CPL_ERROR_INVALID_TYPE	The input *array* is not numerical.
CPL_ERROR_DIVISION_BY_ZERO	The input *value* is zero.

The operation is always performed in double precision, with a final cast of the result to the target array type. Invalid elements are not modified by this operation.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.22 cpl_array_divide_scalar_complex()

```
cpl_error_code cpl_array_divide_scalar_complex (
            cpl_array * array,
            double complex value )
```

Divide a complex array by a constant complex value.

Parameters

array	Target array
value	Divisor.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is a NULL pointer.
CPL_ERROR_INVALID_TYPE	The input *array* is not complex.
CPL_ERROR_DIVISION_BY_ZERO	The input *value* is zero.

The operation is always performed in double precision, with a final cast of the result to the target array type. Invalid elements are not modified by this operation.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.23 cpl_array_dump()

```
void cpl_array_dump (
            const cpl_array * array,
            cpl_size start,
            cpl_size count,
            FILE * stream )
```

Print an array.

Parameters

array	Pointer to array
start	First element to print
count	Number of elements to print
stream	The output stream

Returns

Nothing.

This function is mainly intended for debug purposes. Array elements are counted from 0, and their sequence number is printed at the left of each element. Invalid elements are represented as a sequence of "-" as wide as the field occupied by the array. Specifying a *start* beyond the array boundaries, or a non-positive *count*, would generate a warning message, but no error would be set. The specified number of elements to print may exceed the array end, and in that case the array would be printed up to its last element. If the specified stream is `NULL`, it is set to *stdout*. The function used for printing is the standard C `fprintf()`.

References cpl_calloc(), cpl_free(), cpl_malloc(), CPL_SIZE_FORMAT, cpl_sprintf(), CPL_TYPE_DOUBLE, CPL↩_TYPE_DOUBLE_COMPLEX, CPL_TYPE_FLOAT, CPL_TYPE_FLOAT_COMPLEX, CPL_TYPE_INT, CPL_TY↩PE_LONG, CPL_TYPE_LONG_LONG, CPL_TYPE_SIZE, and CPL_TYPE_STRING.

4.1.2.24 cpl_array_dump_structure()

```
void cpl_array_dump_structure (
            const cpl_array * array,
            FILE * stream )
```

Describe the structure and the contents of an array.

Parameters

array	Pointer to array.
stream	The output stream

Returns

Nothing.

This function is mainly intended for debug purposes. Some information about the structure of an array and its contents is printed to terminal:

- Data type of the array
- Number of elements
- Number of invalid elements

If the specified stream is NULL, it is set to *stdout*. The function used for printing is the standard C fprintf().

References CPL_SIZE_FORMAT, CPL_TYPE_DOUBLE, CPL_TYPE_DOUBLE_COMPLEX, CPL_TYPE_FLOAT, CPL_TYPE_FLOAT_COMPLEX, CPL_TYPE_INT, CPL_TYPE_LONG, CPL_TYPE_LONG_LONG, CPL_TYPE↩_SIZE, and CPL_TYPE_STRING.

4.1.2.25 cpl_array_duplicate()

```
cpl_array* cpl_array_duplicate (
            const cpl_array * array )
```

Make a copy of an array.

Parameters

array	Array to be duplicated.

Returns

Pointer to the new array, or NULL in case of error.

If the input *array* is a NULL pointer, a CPL_ERROR_NULL_INPUT is returned. Copy is "in depth": in the case of a *string* array, also the string elements are duplicated.

References cpl_array_get_size(), cpl_array_get_type(), cpl_array_new(), and CPL_ERROR_NULL_INPUT.

4.1.2.26 cpl_array_erase_window()

```
cpl_error_code cpl_array_erase_window (
            cpl_array * array,
            cpl_size start,
            cpl_size count )
```

Delete a segment of an array.

Parameters

array	Input array
start	First element to delete.
count	Number of elements to delete.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	*array* is a NULL pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input array has length zero, or *start* is outside the table range.
CPL_ERROR_ILLEGAL_INPUT	*count* is negative.

A portion of the array data is physically removed. The pointers to data may change, therefore pointers previously retrieved by calling cpl_array_get_data_int(), cpl_array_get_data_string(), etc., should be discarded. The specified segment can extend beyond the end of the array, and in that case elements will be removed up to the end of the array.

References cpl_array_get_size(), CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.27 cpl_array_exponential()

```
cpl_error_code cpl_array_exponential (
            cpl_array * array,
            double base )
```

Compute the exponential of array elements.

Parameters

array	Pointer to array.
base	Exponential base.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is NULL pointer.
CPL_ERROR_INVALID_TYPE	The specified *array* is not numerical or complex.
CPL_ERROR_ILLEGAL_INPUT	The input *base* is not positive.

Each column element is replaced by its exponential in the specified base. The operation is always performed in double precision, with a final cast of the result to the array type. Invalid elements are not modified by this operation.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.28 cpl_array_extract()

```
cpl_array* cpl_array_extract (
            const cpl_array * array,
            cpl_size start,
            cpl_size count )
```

Create an array from a section of another array.

Parameters

array	Input array
start	First element to be copied to new array.
count	Number of elements to be copied.

Returns

Pointer to the new array, or NULL in case or error.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is a NULL pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *array* has zero length, or *start* is outside the array boundaries.
CPL_ERROR_ILLEGAL_INPUT	*count* is negative.

A number of consecutive elements are copied from an input array to a newly created array. If the sum of *start* and *count* goes beyond the end of the input array, elements are copied up to the end.

References cpl_array_delete(), cpl_array_get_size(), cpl_array_get_type(), cpl_array_new(), and CPL_ERROR_↩ NULL_INPUT.

4.1.2.29 cpl_array_extract_imag()

```
cpl_array* cpl_array_extract_imag (
            const cpl_array * array )
```

Extract the imaginary value of array elements.

Parameters

array	Pointer to array.

Returns

New array with imaginary part of input array elements.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is `NULL` pointer.
CPL_ERROR_INVALID_TYPE	The specified *array* is not numerical.

A new array is created with the imaginary part of all input array elements. If the input array is complex, the output type will be CPL_TYPE_FLOAT if input is CPL_TYPE_FLOAT_COMPLEX, and CPL_TYPE_DOUBLE if input is CPL_TYPE_DOUBLE_COMPLEX).

References cpl_array_get_type(), cpl_array_new(), CPL_ERROR_NULL_INPUT, CPL_TYPE_COMPLEX, and C↩ PL_TYPE_FLOAT.

4.1.2.30 cpl_array_extract_real()

```
cpl_array* cpl_array_extract_real (
            const cpl_array * array )
```

Extract the real value of array elements.

Parameters

array	Pointer to array.

Returns

New array with real part of input array elements.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is `NULL` pointer.
CPL_ERROR_INVALID_TYPE	The specified *array* is not numerical.

A new array is created with the real part of all input array elements. If the input array is complex, the output type will be CPL_TYPE_FLOAT if input is CPL_TYPE_FLOAT_COMPLEX, and CPL_TYPE_DOUBLE if input is CPL_TY↩ PE_DOUBLE_COMPLEX).

References cpl_array_get_type(), cpl_array_new(), CPL_ERROR_NULL_INPUT, CPL_TYPE_COMPLEX, and C↩ PL_TYPE_FLOAT.

4.1.2.31 cpl_array_fill_window()

```
cpl_error_code cpl_array_fill_window (
            cpl_array * array,
            cpl_size start,
            cpl_size count,
            double value )
```

Write the same value within a numerical array segment.

Parameters

array	Array to be accessed.
start	Position where to begin write value.
count	Number of values to write.
value	Value to write.

Returns

`CPL_ERROR_NONE` on success. If *array* is a `NULL` pointer a `CPL_ERROR_NULL_INPUT` is returned. If the array is not of numerical type, a `CPL_ERROR_INVALID_TYPE` is returned. If *start* is outside the array range, a `CPL_ERROR_ACCESS_OUT_OF_RANGE` is returned. If the input array has length zero, the `CPL_ERROR_ACCESS_OUT_OF_RANGE` is always returned.

Write the same value to a numerical array segment. The value is cast to the accessed array type. The written values are automatically flagged as valid. To invalidate an array interval use `cpl_array_fill_window_invalid()`.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_table_where_selected().

4.1.2.32 cpl_array_fill_window_complex()

```
cpl_error_code cpl_array_fill_window_complex (
            cpl_array * array,
            cpl_size start,
            cpl_size count,
            double complex value )
```

Write the same value within a complex array segment.

Parameters

array	Array to be accessed.
start	Position where to begin write value.
count	Number of values to write.
value	Value to write.

Returns

CPL_ERROR_NONE on success. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is returned. If the array is not of numerical type, a CPL_ERROR_INVALID_TYPE is returned. If *start* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is returned. If the input array has length zero, the CPL_ERROR_ACCESS_OUT_OF_RANGE is always returned.

Write the same value to a complex array segment. The value is cast to the accessed array type. The written values are automatically flagged as valid. To invalidate an array interval use cpl_array_fill_window_invalid().

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.33 cpl_array_fill_window_cplsize()

```
cpl_error_code cpl_array_fill_window_cplsize (
            cpl_array * array,
            cpl_size start,
            cpl_size count,
            cpl_size value )
```

Write the same value within a *cpl_size* array segment.

Parameters

array	Array to be accessed.
start	Position where to begin write value.
count	Number of values to write.
value	Value to write.

Returns

CPL_ERROR_NONE on success. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is returned. If the array is not of the expected type, a CPL_ERROR_TYPE_MISMATCH is returned. If *start* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is returned. If the input array has length zero, the error CPL_ERROR_ACCESS_OUT_OF_RANGE is always returned. If *count* is negative, a CPL_ERROR_↩ ILLEGAL_INPUT is returned.

Write the same value to a *cpl_size* array segment. The written values are automatically flagged as valid. To invalidate an array interval use cpl_array_fill_window_invalid(). The *count* argument can go beyond the array end, and in that case the specified *value* will be written just up to the end of the array. If *count* is zero, the array is not modified and no error is set.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.34 cpl_array_fill_window_double()

```
cpl_error_code cpl_array_fill_window_double (
            cpl_array * array,
```

```
                cpl_size start,
                cpl_size count,
                double value )
```

Write the same value within a *double* array segment.

Parameters

array	Array to be accessed.
start	Position where to begin write value.
count	Number of values to write.
value	Value to write.

Returns

CPL_ERROR_NONE on success. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is returned. If the array is not of the expected type, a CPL_ERROR_TYPE_MISMATCH is returned. If *start* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is returned. If the input array has length zero, the error CPL_ERROR_ACCESS_OUT_OF_RANGE is always returned. If *count* is negative, a CPL_ERROR_↩ ILLEGAL_INPUT is returned.

Write the same value to a *double* array segment. The written values are automatically flagged as valid. To invalidate an array interval use cpl_array_fill_window_invalid(). The *count* argument can go beyond the array end, and in that case the specified *value* will be written just up to the end of the array. If *count* is zero, the array is not modified and no error is set.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.35 cpl_array_fill_window_double_complex()

```
cpl_error_code cpl_array_fill_window_double_complex (
            cpl_array * array,
            cpl_size start,
            cpl_size count,
            double complex value )
```

Write the same value within a *double* complex array segment.

Parameters

array	Array to be accessed.
start	Position where to begin write value.
count	Number of values to write.
value	Value to write.

Returns

CPL_ERROR_NONE on success. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is returned. If the array is not of the expected type, a CPL_ERROR_TYPE_MISMATCH is returned. If *start* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is returned. If the input array has length zero, the error CPL_ERROR_ACCESS_OUT_OF_RANGE is always returned. If *count* is negative, a CPL_ERROR_↩ ILLEGAL_INPUT is returned.

Write the same value to a *double* complex array segment. The written values are automatically flagged as valid. To invalidate an array interval use cpl_array_fill_window_invalid(). The *count* argument can go beyond

the array end, and in that case the specified *value* will be written just up to the end of the array. If *count* is zero, the array is not modified and no error is set.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.36 cpl_array_fill_window_float()

```
cpl_error_code cpl_array_fill_window_float (
            cpl_array * array,
            cpl_size start,
            cpl_size count,
            float value )
```

Write the same value within a *float* array segment.

Parameters

array	Array to be accessed.
start	Position where to begin write value.
count	Number of values to write.
value	Value to write.

Returns

`CPL_ERROR_NONE` on success. If *array* is a `NULL` pointer a `CPL_ERROR_NULL_INPUT` is returned. If the array is not of the expected type, a `CPL_ERROR_TYPE_MISMATCH` is returned. If *start* is outside the array range, a `CPL_ERROR_ACCESS_OUT_OF_RANGE` is returned. If the input array has length zero, the error `CPL_ERROR_ACCESS_OUT_OF_RANGE` is always returned. If *count* is negative, a `CPL_ERROR_ILLEGAL_INPUT` is returned.

Write the same value to a *float* array segment. The written values are automatically flagged as valid. To invalidate an array interval use `cpl_array_fill_window_invalid()`. The *count* argument can go beyond the array end, and in that case the specified *value* will be written just up to the end of the array. If *count* is zero, the array is not modified and no error is set.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.37 cpl_array_fill_window_float_complex()

```
cpl_error_code cpl_array_fill_window_float_complex (
            cpl_array * array,
            cpl_size start,
            cpl_size count,
            float complex value )
```

Write the same value within a *float* complex array segment.

Parameters

array	Array to be accessed.
start	Position where to begin write value.
count	Number of values to write.
value	Value to write.

Returns

CPL_ERROR_NONE on success. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is returned. If the array is not of the expected type, a CPL_ERROR_TYPE_MISMATCH is returned. If *start* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is returned. If the input array has length zero, the error CPL_ERROR_ACCESS_OUT_OF_RANGE is always returned. If *count* is negative, a CPL_ERROR_↩ ILLEGAL_INPUT is returned.

Write the same value to a *float* complex array segment. The written values are automatically flagged as valid. To invalidate an array interval use cpl_array_fill_window_invalid(). The *count* argument can go beyond the array end, and in that case the specified *value* will be written just up to the end of the array. If *count* is zero, the array is not modified and no error is set.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.38 cpl_array_fill_window_int()

```
cpl_error_code cpl_array_fill_window_int (
            cpl_array * array,
            cpl_size start,
            cpl_size count,
            int value )
```

Write the same value within an *integer* array segment.

Parameters

array	Array to be accessed.
start	Position where to begin write value.
count	Number of values to write.
value	Value to write.

Returns

CPL_ERROR_NONE on success. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is returned. If the array is not of the expected type, a CPL_ERROR_TYPE_MISMATCH is returned. If *start* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is returned. If the input array has length zero, the error CPL_ERROR_ACCESS_OUT_OF_RANGE is always returned. If *count* is negative, a CPL_ERROR_↩ ILLEGAL_INPUT is returned.

Write the same value to an *integer* array segment. The written values are automatically flagged as valid. To invalidate an array interval use cpl_array_fill_window_invalid(). The *count* argument can go beyond

the array end, and in that case the specified *value* will be written just up to the end of the array. If *count* is zero, the array is not modified and no error is set.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.39 cpl_array_fill_window_invalid()

```
cpl_error_code cpl_array_fill_window_invalid (
            cpl_array * array,
            cpl_size start,
            cpl_size count )
```

Set an array segment to NULL.

Parameters

array	Array to be accessed.
start	Position where to start writing NULLs.
count	Number of column elements to set to NULL.

Returns

`CPL_ERROR_NONE` on success. If *array* is a `NULL` pointer a `CPL_ERROR_NULL_INPUT` is returned. If *start* is outside the array range, a `CPL_ERROR_ACCESS_OUT_OF_RANGE` is returned. If the input array has length zero, the error `CPL_ERROR_ACCESS_OUT_OF_RANGE` is always returned. If *count* is negative, a `CPL_ERROR_ILLEGAL_INPUT` is returned.

Invalidate values contained in an array segment. The *count* argument can go beyond the array end, and in that case the values will be invalidated up to the end of the array. If *count* is zero, the array is not modified and no error is set. In the case of a *string* array, the invalidated strings are set free and their pointers are set to `NULL`; for other data types, the corresponding elements are flagged as invalid.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.40 cpl_array_fill_window_long()

```
cpl_error_code cpl_array_fill_window_long (
            cpl_array * array,
            cpl_size start,
            cpl_size count,
            long value )
```

Write the same value within a *long integer* array segment.

Parameters

array	Array to be accessed.
start	Position where to begin write value.
count	Number of values to write.
value	Value to write.

Returns

CPL_ERROR_NONE on success. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is returned. If the array is not of the expected type, a CPL_ERROR_TYPE_MISMATCH is returned. If *start* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is returned. If the input array has length zero, the error CPL_ERROR_ACCESS_OUT_OF_RANGE is always returned. If *count* is negative, a CPL_ERROR_↩ ILLEGAL_INPUT is returned.

Write the same value to a *long integer* array segment. The written values are automatically flagged as valid. To invalidate an array interval use cpl_array_fill_window_invalid(). The *count* argument can go beyond the array end, and in that case the specified *value* will be written just up to the end of the array. If *count* is zero, the array is not modified and no error is set.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.41 cpl_array_fill_window_long_long()

```
cpl_error_code cpl_array_fill_window_long_long (
            cpl_array * array,
            cpl_size start,
            cpl_size count,
            long long value )
```

Write the same value within a *long long integer* array segment.

Parameters

array	Array to be accessed.
start	Position where to begin write value.
count	Number of values to write.
value	Value to write.

Returns

CPL_ERROR_NONE on success. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is returned. If the array is not of the expected type, a CPL_ERROR_TYPE_MISMATCH is returned. If *start* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is returned. If the input array has length zero, the error CPL_ERROR_ACCESS_OUT_OF_RANGE is always returned. If *count* is negative, a CPL_ERROR_↩ ILLEGAL_INPUT is returned.

Write the same value to a *long long integer* array segment. The written values are automatically flagged as valid. To invalidate an array interval use cpl_array_fill_window_invalid(). The *count* argument can go beyond the array end, and in that case the specified *value* will be written just up to the end of the array. If *count* is zero, the array is not modified and no error is set.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.42 cpl_array_fill_window_string()

```
cpl_error_code cpl_array_fill_window_string (
            cpl_array * array,
            cpl_size start,
            cpl_size count,
            const char * value )
```

Write a string to a string array segment.

Parameters

array	Array to be accessed.
start	Position where to begin write value.
count	Number of values to write.
value	Value to write.

Returns

CPL_ERROR_NONE on success. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is returned. If the array is not of the expected type, a CPL_ERROR_TYPE_MISMATCH is returned. If *start* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is returned. If the input array has length zero, the error CPL_ERROR_ACCESS_OUT_OF_RANGE is always returned. If *count* is negative, a CPL_ERROR_ILLEGAL_INPUT is returned.

Copy the same string to a string array segment. If the input string is not a NULL pointer, it is duplicated for each accessed array element. If the input string is NULL, this call is equivalent to cpl_array_fill_window_invalid(). The *count* argument can go beyond the array end, and in that case the specified *value* will be copied just up to the end of the array. If *count* is zero, the array is not modified and no error is set.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.43 cpl_array_get()

```
double cpl_array_get (
            const cpl_array * array,
            cpl_size indx,
            int * null )
```

Read a value from a numerical array.

Parameters

array	Array to be accessed.
indx	Position of element to be read.
null	Flag indicating null values, or error condition.

Returns

Value read. In case of an invalid array element, or in case of error, 0.0 is returned.

Read a value from a numerical array. A CPL_ERROR_NULL_INPUT is set in case *array* is a NULL pointer. A CPL_ERROR_INVALID_TYPE is set in case a non-numerical array is accessed. CPL_ERROR_ACCESS_OUT_OF_RANGE is set if the *indx* is outside the array range. Indexes are counted starting from 0. If the input array has length zero, CPL_ERROR_ACCESS_OUT_OF_RANGE is always set. The *null* flag is used to indicate whether the accessed array element is valid (0) or invalid (1). The null flag also signals an error condition (-1). The *null* argument can be left to NULL.

References CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), and cpl_errorstate_is_equal().

4.1.2.44 cpl_array_get_complex()

```
double complex cpl_array_get_complex (
            const cpl_array * array,
            cpl_size indx,
            int * null )
```

Read a value from a complex array.

Parameters

array	Array to be accessed.
indx	Position of element to be read.
null	Flag indicating null values, or error condition.

Returns

Value read. In case of an invalid array element, or in case of error, 0.0 is returned.

Read a value from a complex array. A CPL_ERROR_NULL_INPUT is set in case *array* is a NULL pointer. A CPL_ERROR_INVALID_TYPE is set in case a non-complex array is accessed. CPL_ERROR_ACCESS_OUT_OF_RANGE is set if the *indx* is outside the array range. Indexes are counted starting from 0. If the input array has length zero, CPL_ERROR_ACCESS_OUT_OF_RANGE is always set. The *null* flag is used to indicate whether the accessed array element is valid (0) or invalid (1). The null flag also signals an error condition (-1). The *null* argument can be left to NULL.

References CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), and cpl_errorstate_is_equal().

4.1.2.45 cpl_array_get_cplsize()

```
cpl_size cpl_array_get_cplsize (
            const cpl_array * array,
            cpl_size indx,
            int * null )
```

Read a value from a *cpl_size* array.

Parameters

array	Array to be accessed.
indx	Position of element to be read.
null	Flag indicating null values, or error condition.

Returns

The cpl_size value read. In case of an invalid array element, or in case of error, 0 is returned.

Read a value from an array of type `CPL_TYPE_SIZE`. If *array* is a `NULL` pointer a `CPL_ERROR_NULL_INPUT` is set. If the array is not of the expected type, a `CPL_ERROR_TYPE_MISMATCH` is set. If *indx* is outside the array range, a `CPL_ERROR_ACCESS_OUT_OF_RANGE` is set. Indexes are counted starting from 0. If the input array has length zero, the `CPL_ERROR_ACCESS_OUT_OF_RANGE` is always set. If the *null* flag is a valid pointer, it is used to indicate whether the accessed array element is valid (0) or invalid (1). The null flag also signals an error condition (-1).

References CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), and cpl_errorstate_is_equal().

4.1.2.46 cpl_array_get_data_cplsize()

```
cpl_size* cpl_array_get_data_cplsize (
            cpl_array * array )
```

Get a pointer to `cpl_size` array data.

Parameters

array	Array to get the data from.

Returns

Pointer to `cpl_size` array data. If *array* contains no data (zero length), a `NULL` is returned. If *array* is a `NULL`, a `NULL` is returned, and an error is set.

If the array is not of type `CPL_TYPE_SIZE`, a `CPL_ERROR_TYPE_MISMATCH` is set.

Note

Use at your own risk: direct manipulation of array data rules out any check performed by the array object interface, and may introduce inconsistencies between the array information maintained internally and the actual array data.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_table_where_selected().

4.1.2.47 cpl_array_get_data_cplsize_const()

```
const cpl_size* cpl_array_get_data_cplsize_const (
            const cpl_array * array )
```

Get a pointer to constant `cpl_size` array data.

Parameters

array	Constant array to get the data from.

Returns

Pointer to constant `cpl_size` array data. If *array* contains no data (zero length), a `NULL` is returned. If *array* is a `NULL`, a `NULL` is returned, and an error is set.

If the array is not of type `CPL_TYPE_SIZE`, a `CPL_ERROR_TYPE_MISMATCH` is set.

References CPL_ERROR_NULL_INPUT.

4.1.2.48 cpl_array_get_data_double()

```
double* cpl_array_get_data_double (
            cpl_array * array )
```

Get a pointer to `double` array data.

Parameters

array	Array to get the data from.

Returns

Pointer to `double` array data. If *array* contains no data (zero length), a `NULL` is returned. If *array* is a `NULL`, a `NULL` is returned, and an error is set.

If the array is not of type `CPL_TYPE_DOUBLE`, a `CPL_ERROR_TYPE_MISMATCH` is set.

See documentation of function cpl_array_get_data_int().

References CPL_ERROR_NULL_INPUT.

4.1.2.49 cpl_array_get_data_double_complex()

```
double complex* cpl_array_get_data_double_complex (
            cpl_array * array )
```

Get a pointer to `double` complex array data.

Parameters

array	Array to get the data from.

Returns

Pointer to `double` complex array data. If *array* contains no data (zero length), a `NULL` is returned. If *array* is a `NULL`, a `NULL` is returned, and an error is set.

If the array is not of type `CPL_TYPE_DOUBLE_COMPLEX`, a `CPL_ERROR_TYPE_MISMATCH` is set.

See documentation of function cpl_array_get_data_int().

References CPL_ERROR_NULL_INPUT.

4.1.2.50 cpl_array_get_data_double_complex_const()

```
const double complex* cpl_array_get_data_double_complex_const (
            const cpl_array * array )
```

Get a pointer to constant *double* complex array data.

Parameters

array	Constant array to get the data from.

Returns

Pointer to constant *double* complex array data. If *array* contains no data (zero length), a `NULL` is returned. If *array* is a `NULL`, a `NULL` is returned, and an error is set.

If the array is not of type `CPL_TYPE_DOUBLE_COMPLEX`, a `CPL_ERROR_TYPE_MISMATCH` is set.

See documentation of function cpl_array_get_data_int_const().

References CPL_ERROR_NULL_INPUT.

4.1.2.51 cpl_array_get_data_double_const()

```
const double* cpl_array_get_data_double_const (
            const cpl_array * array )
```

Get a pointer to constant *double* array data.

Parameters

array	Constant array to get the data from.

Returns

Pointer to constant *double* array data. If *array* contains no data (zero length), a `NULL` is returned. If *array* is a `NULL`, a `NULL` is returned, and an error is set.

If the array is not of type `CPL_TYPE_DOUBLE`, a `CPL_ERROR_TYPE_MISMATCH` is set.

See documentation of function cpl_array_get_data_int_const().

References CPL_ERROR_NULL_INPUT.

4.1.2.52 cpl_array_get_data_float()

```
float* cpl_array_get_data_float (
            cpl_array * array )
```

Get a pointer to `float` array data.

Parameters

array	Array to get the data from.

Returns

Pointer to `float` array data. If *array* contains no data (zero length), a `NULL` is returned. If *array* is a `NULL`, a `NULL` is returned, and an error is set.

If the array is not of type `CPL_TYPE_FLOAT`, a `CPL_ERROR_TYPE_MISMATCH` is set.

See documentation of function cpl_array_get_data_int().

References CPL_ERROR_NULL_INPUT.

4.1.2.53 cpl_array_get_data_float_complex()

```
float complex* cpl_array_get_data_float_complex (
            cpl_array * array )
```

Get a pointer to `float` complex array data.

Parameters

array	Array to get the data from.

Returns

Pointer to `float` complex array data. If *array* contains no data (zero length), a `NULL` is returned. If *array* is a `NULL`, a `NULL` is returned, and an error is set.

If the array is not of type `CPL_TYPE_FLOAT_COMPLEX`, a `CPL_ERROR_TYPE_MISMATCH` is set.

See documentation of function cpl_array_get_data_int().

References CPL_ERROR_NULL_INPUT.

4.1.2.54 cpl_array_get_data_float_complex_const()

```
const float complex* cpl_array_get_data_float_complex_const (
            const cpl_array * array )
```

Get a pointer to constant *float* complex array data.

Parameters

array	Constant array to get the data from.

Returns

Pointer to constant *float* complex array data. If *array* contains no data (zero length), a `NULL` is returned. If *array* is a `NULL`, a `NULL` is returned, and an error is set.

If the array is not of type `CPL_TYPE_FLOAT_COMPLEX`, a `CPL_ERROR_TYPE_MISMATCH` is set.

See documentation of function cpl_array_get_data_int_const().

References CPL_ERROR_NULL_INPUT.

4.1.2.55 cpl_array_get_data_float_const()

```
const float* cpl_array_get_data_float_const (
            const cpl_array * array )
```

Get a pointer to constant *float* array data.

Parameters

array	Constant array to get the data from.

Returns

Pointer to constant *float* array data. If *array* contains no data (zero length), a `NULL` is returned. If *array* is a `NULL`, a `NULL` is returned, and an error is set.

If the array is not of type `CPL_TYPE_FLOAT`, a `CPL_ERROR_TYPE_MISMATCH` is set.

See documentation of function cpl_array_get_data_int_const().

References CPL_ERROR_NULL_INPUT.

4.1.2.56 cpl_array_get_data_int()

```
int* cpl_array_get_data_int (
            cpl_array * array )
```

Get a pointer to `integer` array data.

Parameters

array	Array to get the data from.

Returns

Pointer to `integer` array data. If *array* contains no data (zero length), a `NULL` is returned. If *array* is a `NULL`, a `NULL` is returned, and an error is set.

If the array is not of type `CPL_TYPE_INT`, a `CPL_ERROR_TYPE_MISMATCH` is set.

Note

Use at your own risk: direct manipulation of array data rules out any check performed by the array object interface, and may introduce inconsistencies between the array information maintained internally and the actual array data.

References CPL_ERROR_NULL_INPUT.

4.1.2.57 cpl_array_get_data_int_const()

```
const int* cpl_array_get_data_int_const (
            const cpl_array * array )
```

Get a pointer to constant `integer` array data.

Parameters

array	Constant array to get the data from.

Returns

Pointer to constant `integer` array data. If *array* contains no data (zero length), a `NULL` is returned. If *array* is a `NULL`, a `NULL` is returned, and an error is set.

If the array is not of type `CPL_TYPE_INT`, a `CPL_ERROR_TYPE_MISMATCH` is set.

References CPL_ERROR_NULL_INPUT.

4.1.2.58 cpl_array_get_data_long()

```
long* cpl_array_get_data_long (
            cpl_array * array )
```

Get a pointer to `long integer` array data.

Parameters

array	Array to get the data from.

Returns

Pointer to `long integer` array data. If *array* contains no data (zero length), a `NULL` is returned. If *array* is a `NULL`, a `NULL` is returned, and an error is set.

If the array is not of type `CPL_TYPE_LONG`, a `CPL_ERROR_TYPE_MISMATCH` is set.

Note

Use at your own risk: direct manipulation of array data rules out any check performed by the array object interface, and may introduce inconsistencies between the array information maintained internally and the actual array data.

References CPL_ERROR_NULL_INPUT.

4.1.2.59 cpl_array_get_data_long_const()

```
const long* cpl_array_get_data_long_const (
            const cpl_array * array )
```

Get a pointer to constant `long integer` array data.

Parameters

array	Constant array to get the data from.

Returns

Pointer to constant `long integer` array data. If *array* contains no data (zero length), a `NULL` is returned. If *array* is a `NULL`, a `NULL` is returned, and an error is set.

If the array is not of type `CPL_TYPE_LONG`, a `CPL_ERROR_TYPE_MISMATCH` is set.

References CPL_ERROR_NULL_INPUT.

4.1.2.60 cpl_array_get_data_long_long()

```
long long* cpl_array_get_data_long_long (
            cpl_array * array )
```

Get a pointer to `long long integer` array data.

Parameters

array	Array to get the data from.

Returns

Pointer to `long long integer` array data. If *array* contains no data (zero length), a `NULL` is returned. If *array* is a `NULL`, a `NULL` is returned, and an error is set.

If the array is not of type `CPL_TYPE_LONG_LONG`, a `CPL_ERROR_TYPE_MISMATCH` is set.

Note

Use at your own risk: direct manipulation of array data rules out any check performed by the array object interface, and may introduce inconsistencies between the array information maintained internally and the actual array data.

References CPL_ERROR_NULL_INPUT.

4.1.2.61 cpl_array_get_data_long_long_const()

```
const long long* cpl_array_get_data_long_long_const (
            const cpl_array * array )
```

Get a pointer to constant `long long integer` array data.

Parameters

array	Constant array to get the data from.

Returns

Pointer to constant `long long integer` array data. If *array* contains no data (zero length), a `NULL` is returned. If *array* is a `NULL`, a `NULL` is returned, and an error is set.

If the array is not of type `CPL_TYPE_LONG_LONG`, a `CPL_ERROR_TYPE_MISMATCH` is set.

References CPL_ERROR_NULL_INPUT.

4.1.2.62 cpl_array_get_data_string()

```
char** cpl_array_get_data_string (
            cpl_array * array )
```

Get a pointer to *string* array data.

Parameters

array	Array to get the data from.

Returns

Pointer to *string* array data. If *array* contains no data (zero length), a `NULL` is returned. If *array* is a `NULL`, a `NULL` is returned, and an error is set.

If the array is not of type `CPL_TYPE_STRING`, a `CPL_ERROR_TYPE_MISMATCH` is set.

See documentation of function cpl_array_get_data_int().

References CPL_ERROR_NULL_INPUT.

4.1.2.63 cpl_array_get_data_string_const()

```
const char** cpl_array_get_data_string_const (
            const cpl_array * array )
```

Get a pointer to constant *string* array data.

Parameters

array	Constant array to get the data from.

Returns

Pointer to constant *string* array data. If *array* contains no data (zero length), a NULL is returned. If *array* is a NULL, a NULL is returned, and an error is set.

If the array is not of type CPL_TYPE_STRING, a CPL_ERROR_TYPE_MISMATCH is set.

See documentation of function cpl_array_get_data_int().

References CPL_ERROR_NULL_INPUT.

4.1.2.64 cpl_array_get_double()

```
double cpl_array_get_double (
            const cpl_array * array,
            cpl_size indx,
            int * null )
```

Read a value from a *double* array.

Parameters

array	Array to be accessed.
indx	Position of element to be read.
null	Flag indicating null values, or error condition.

Returns

Array value read. In case of an invalid array element, or in case of error, 0.0 is returned.

Read a value from an array of type CPL_TYPE_DOUBLE. See the documentation of the function cpl_array_get_int().

References CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), and cpl_errorstate_is_equal().

Referenced by cpl_fit_image_gaussian(), and cpl_gaussian_eval_2d().

4.1.2.65 cpl_array_get_double_complex()

```
double complex cpl_array_get_double_complex (
            const cpl_array * array,
            cpl_size indx,
            int * null )
```

Read a value from a *double* complex array.

Parameters

array	Array to be accessed.
indx	Position of element to be read.
null	Flag indicating null values, or error condition.

Returns

Array value read. In case of an invalid array element, or in case of error, 0.0 is returned.

Read a value from an array of type `CPL_TYPE_DOUBLE_COMPLEX`. See the documentation of the function cpl_array_get_int().

References CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), and cpl_errorstate_is_equal().

4.1.2.66 cpl_array_get_float()

```
float cpl_array_get_float (
            const cpl_array * array,
            cpl_size indx,
            int * null )
```

Read a value from a *float* array.

Parameters

array	Array to be accessed.
indx	Position of element to be read.
null	Flag indicating null values, or error condition.

Returns

Array value read. In case of an invalid array element, or in case of error, 0.0 is returned.

Read a value from an array of type `CPL_TYPE_FLOAT`. See the documentation of the function cpl_array_get_int().

References CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), and cpl_errorstate_is_equal().

4.1.2.67 cpl_array_get_float_complex()

```
float complex cpl_array_get_float_complex (
            const cpl_array * array,
            cpl_size indx,
            int * null )
```

Read a value from a *float* complex array.

Parameters

array	Array to be accessed.
indx	Position of element to be read.
null	Flag indicating null values, or error condition.

Returns

Array value read. In case of an invalid array element, or in case of error, 0.0 is returned.

Read a value from an array of type `CPL_TYPE_FLOAT_COMPLEX`. See the documentation of the function cpl_array_get_int().

References CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), and cpl_errorstate_is_equal().

4.1.2.68 cpl_array_get_int()

```
int cpl_array_get_int (
            const cpl_array * array,
            cpl_size indx,
            int * null )
```

Read a value from an *integer* array.

Parameters

array	Array to be accessed.
indx	Position of element to be read.
null	Flag indicating null values, or error condition.

Returns

Integer value read. In case of an invalid array element, or in case of error, 0 is returned.

Read a value from an array of type `CPL_TYPE_INT`. If *array* is a `NULL` pointer a `CPL_ERROR_NULL_INPUT` is set. If the array is not of the expected type, a `CPL_ERROR_TYPE_MISMATCH` is set. If *indx* is outside the array range, a `CPL_ERROR_ACCESS_OUT_OF_RANGE` is set. Indexes are counted starting from 0. If the input array has length zero, the `CPL_ERROR_ACCESS_OUT_OF_RANGE` is always set. If the *null* flag is a valid pointer, it is used to indicate whether the accessed array element is valid (0) or invalid (1). The null flag also signals an error condition (-1).

References CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), and cpl_errorstate_is_equal().

Referenced by cpl_fit_image_gaussian().

4.1.2.69 cpl_array_get_long()

```
long cpl_array_get_long (
            const cpl_array * array,
            cpl_size indx,
            int * null )
```

Read a value from a *long integer* array.

Parameters

array	Array to be accessed.
indx	Position of element to be read.
null	Flag indicating null values, or error condition.

Returns

Long integer value read. In case of an invalid array element, or in case of error, 0 is returned.

Read a value from an array of type CPL_TYPE_LONG. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is set. If the array is not of the expected type, a CPL_ERROR_TYPE_MISMATCH is set. If *indx* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is set. Indexes are counted starting from 0. If the input array has length zero, the CPL_ERROR_ACCESS_OUT_OF_RANGE is always set. If the *null* flag is a valid pointer, it is used to indicate whether the accessed array element is valid (0) or invalid (1). The null flag also signals an error condition (-1).

References CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), and cpl_errorstate_is_equal().

4.1.2.70 cpl_array_get_long_long()

```
long long cpl_array_get_long_long (
            const cpl_array * array,
            cpl_size indx,
            int * null )
```

Read a value from a *long long integer* array.

Parameters

array	Array to be accessed.
indx	Position of element to be read.
null	Flag indicating null values, or error condition.

Returns

Long long integer value read. In case of an invalid array element, or in case of error, 0 is returned.

Read a value from an array of type CPL_TYPE_LONG_LONG. If *array* is a NULL pointer a CPL_ERROR_NULL↩_INPUT is set. If the array is not of the expected type, a CPL_ERROR_TYPE_MISMATCH is set. If *indx* is outside

the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is set. Indexes are counted starting from 0. If the input array has length zero, the CPL_ERROR_ACCESS_OUT_OF_RANGE is always set. If the *null* flag is a valid pointer, it is used to indicate whether the accessed array element is valid (0) or invalid (1). The null flag also signals an error condition (-1).

References CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), and cpl_errorstate_is_equal().

4.1.2.71 cpl_array_get_max()

```
double cpl_array_get_max (
            const cpl_array * array )
```

Get maximum value in a numerical array.

Parameters

array	Input array.

Returns

Maximum value. In case of error, this is set to 0.0.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is a NULL pointer.
CPL_ERROR_INVALID_TYPE	The specified *array* is not numerical.
CPL_ERROR_DATA_NOT_FOUND	The specified *array* has either size zero, or all its elements are invalid.

Array elements marked as invalid are excluded from the search. The array must contain at least one valid value. Arrays of strings or complex are not allowed.

References CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), and cpl_errorstate_is_equal().

4.1.2.72 cpl_array_get_maxpos()

```
cpl_error_code cpl_array_get_maxpos (
            const cpl_array * array,
            cpl_size * indx )
```

Get position of maximum in a numerical array.

Parameters

array	Pointer to array.
indx	Returned position of maximum value.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *array* or *indx* is `NULL` pointer.
CPL_ERROR_INVALID_TYPE	The specified *array* is not numerical.
CPL_ERROR_DATA_NOT_FOUND	The specified *array* has either size zero, or all its elements are invalid.

Array values marked as invalid are excluded from the search. The *indx* argument will be assigned the position of the maximum value. Indexes are counted starting from 0. If more than one array element correspond to the max value, the position with the lowest indx is returned. In case of error, *indx* is set to zero. Arrays of strings or complex are not allowed.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.73 cpl_array_get_mean()

```
double cpl_array_get_mean (
            const cpl_array * array )
```

Compute the mean value of a numeric array.

Parameters

array	Input array.

Returns

Mean value. In case of error, this is set to 0.0.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is a `NULL` pointer.
CPL_ERROR_INVALID_TYPE	The specified *array* is not numerical.
CPL_ERROR_DATA_NOT_FOUND	The specified *array* has either size zero, or all its elements are invalid.

Array elements marked as invalid are excluded from the computation. The array must contain at least one valid value. Arrays of strings or complex are not allowed.

References CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), and cpl_errorstate_is_equal().

4.1.2.74 cpl_array_get_mean_complex()

```
double complex cpl_array_get_mean_complex (
            const cpl_array * array )
```

Compute the mean value of a complex array.

Parameters

array	Input array.

Returns

Mean value. In case of error, this is set to 0.0.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is a `NULL` pointer.
CPL_ERROR_INVALID_TYPE	The specified *array* is not complex.
CPL_ERROR_DATA_NOT_FOUND	The specified *array* has either size zero, or all its elements are invalid.

Array elements marked as invalid are excluded from the computation. The array must contain at least one valid value. Arrays of strings or numerical are not allowed.

References CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), and cpl_errorstate_is_equal().

4.1.2.75 cpl_array_get_median()

```
double cpl_array_get_median (
            const cpl_array * array )
```

Compute the median of a numeric array.

Parameters

array	Input array.

Returns

Median. In case of error, this is set to 0.0.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is a `NULL` pointer.
CPL_ERROR_INVALID_TYPE	The specified *array* is not numerical.
CPL_ERROR_DATA_NOT_FOUND	The specified *array* has either size zero, or all its elements are invalid.

Array elements marked as invalid are excluded from the computation. The array must contain at least one valid value. Arrays of strings or complex are not allowed.

References cpl_errorstate_get().

4.1.2.76 cpl_array_get_min()

```
double cpl_array_get_min (
            const cpl_array * array )
```

Get minimum value in a numerical array.

Parameters

array	Input array.

Returns

Minimum value. In case of error, this is set to 0.0.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is a `NULL` pointer.
CPL_ERROR_INVALID_TYPE	The specified *array* is not numerical.
CPL_ERROR_DATA_NOT_FOUND	The specified *array* has either size zero, or all its elements are invalid.

Array elements marked as invalid are excluded from the search. The array must contain at least one valid value. Arrays of strings or complex are not allowed.

References CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), and cpl_errorstate_is_equal().

4.1.2.77 cpl_array_get_minpos()

```
cpl_error_code cpl_array_get_minpos (
            const cpl_array * array,
            cpl_size * indx )
```

Get position of minimum in a numerical array.

Parameters

array	Pointer to array.
indx	Returned position of minimum value.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *array* or *indx* is `NULL` pointer.
CPL_ERROR_INVALID_TYPE	The specified *array* is not numerical.
CPL_ERROR_DATA_NOT_FOUND	The specified *array* has either size zero, or all its elements are invalid.

Array values marked as invalid are excluded from the search. The *indx* argument will be assigned the position of the minimum value. Indexes are counted starting from 0. If more than one array element correspond to the min value, the position with the lowest indx is returned. In case of error, *indx* is set to zero. Arrays of strings or complex are not allowed.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.78 cpl_array_get_size()

```
cpl_size cpl_array_get_size (
            const cpl_array * array )
```

Get the length of an array.

Parameters

array	Input array.

Returns

Length of array, or zero. The latter case can occur either with an array having zero length, or if a `NULL` array is passed to the function, but in the latter case a `CPL_ERROR_NULL_INPUT` is set.

If the array is *NULL*, zero is returned.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_array_duplicate(), cpl_array_erase_window(), cpl_array_extract(), and cpl_gaussian_eval_2d().

4.1.2.79 cpl_array_get_stdev()

```
double cpl_array_get_stdev (
            const cpl_array * array )
```

Compute the standard deviation of a numeric array.

Parameters

array	Input array.

Returns

Standard deviation. In case of error, this is set to 0.0.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is a `NULL` pointer.
CPL_ERROR_INVALID_TYPE	The specified *array* is not numerical.
CPL_ERROR_DATA_NOT_FOUND	The specified *array* has either size zero, or all its elements are invalid.

Array elements marked as invalid are excluded from the computation. The array must contain at least one valid value. Arrays of strings or complex are not allowed.

References CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), and cpl_errorstate_is_equal().

4.1.2.80 cpl_array_get_string()

```
const char* cpl_array_get_string (
            const cpl_array * array,
            cpl_size indx )
```

Read a value from a string array.

Parameters

array	Array to be accessed.
indx	Position of element to be read.

Returns

Character string read. In case of an invalid array element, or in case of error, a NULL pointer is returned.

Read a value from an array of type CPL_TYPE_STRING. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is set. If the array is not of the expected type, a CPL_ERROR_TYPE_MISMATCH is set. If *indx* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is set. Indexes are counted starting from 0. If the input array has length zero, the CPL_ERROR_ACCESS_OUT_OF_RANGE is always set.

Note

The returned string is a pointer to an array element, not its copy. Its manipulation will directly affect that array element, while changing that array element using cpl_array_set_string() will turn it into garbage. Therefore, if a real copy of a string array element is required, this function should be called as an argument of the function strdup().

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_table_compare_structure().

4.1.2.81 cpl_array_get_type()

```
cpl_type cpl_array_get_type (
            const cpl_array * array )
```

Get the type of an array.

Parameters

array	Input array

Returns

Type of array, or CPL_TYPE_INVALID if a NULL array is passed to the function.

If the array is NULL, CPL_ERROR_NULL_INPUT is set.

References CPL_ERROR_NULL_INPUT, cpl_error_set, and CPL_TYPE_INVALID.

Referenced by cpl_array_abs(), cpl_array_arg(), cpl_array_duplicate(), cpl_array_extract(), cpl_array_extract_imag(), cpl_array_extract_real(), and cpl_fit_image_gaussian().

4.1.2.82 cpl_array_has_invalid()

```
int cpl_array_has_invalid (
            const cpl_array * array )
```

Check if an array contains at least one invalid element.

Parameters

array	Array to inquire.

Returns

1 if the array contains at least one invalid element, 0 if not, -1 in case of error.

Check if there are invalid elements in an array. If the input array is a `NULL` pointer, a `CPL_ERROR_NULL_INPUT` is set.

References CPL_ERROR_NULL_INPUT.

4.1.2.83 cpl_array_has_valid()

```
int cpl_array_has_valid (
            const cpl_array * array )
```

Check if an array contains at least one valid value.

Parameters

array	Array to inquire.

Returns

1 if the array contains at least one valid value, 0 if not -1 in case of error.

Check if there are valid values in an array. If the input array is a `NULL` pointer, a `CPL_ERROR_NULL_INPUT` is set.

References CPL_ERROR_NULL_INPUT.

4.1.2.84 cpl_array_insert()

```
cpl_error_code cpl_array_insert (
            cpl_array * target_array,
            const cpl_array * insert_array,
            cpl_size start )
```

Merge two arrays.

Parameters

target_array	Target array.
insert_array	Array to be inserted in the target array.
start	Element where to insert the insert array.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any input array is a NULL pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	*start* is negative.
CPL_ERROR_TYPE_MISMATCH	The input arrays do not have the same type.

The input arrays must have the same type. Data from the *insert_array* are duplicated and inserted at the specified position of the *target_array*. If the specified *start* is not less than the target array length, the second array will be appended to the target array. The pointers to array data in the target array may change, therefore pointers previously retrieved by calling cpl_array_get_data_int(), cpl_array_get_data_string(), etc., should be discarded.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.85 cpl_array_insert_window()

```
cpl_error_code cpl_array_insert_window (
            cpl_array * array,
            cpl_size start,
            cpl_size count )
```

Insert a segment of new elements into array.

Parameters

array	Input array
start	Element where to insert the segment.
count	Length of the segment.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	*array* is a NULL pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	*start* is negative.
CPL_ERROR_ILLEGAL_INPUT	*count* is negative.

Insert a segment of empty (invalid) elements. Setting *start* to a number greater than the array length is legal,

and has the effect of appending extra elements at the end of the array: this is equivalent to expanding the array using cpl_array_set_size(). The input *array* may also have zero length. The pointers to array data values may change, therefore pointers previously retrieved by calling cpl_array_get_data_int(), cpl_array_get_data_string(), etc., should be discarded.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.86 cpl_array_is_valid()

```
int cpl_array_is_valid (
            const cpl_array * array,
            cpl_size indx )
```

Check if an array element is valid.

Parameters

array	Pointer to array.
indx	Array element to examine.

Returns

1 if the array element is valid, 0 if invalid, -1 in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is a `NULL` pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *array* has zero length, or *indx* is outside the array boundaries.

Check if an array element is valid.

References CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), and cpl_errorstate_is_equal().

4.1.2.87 cpl_array_logarithm()

```
cpl_error_code cpl_array_logarithm (
            cpl_array * array,
            double base )
```

Compute the logarithm of array elements.

Parameters

array	Pointer to array.
base	Logarithm base.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is `NULL` pointer.
CPL_ERROR_INVALID_TYPE	The specified *array* is not numerical or complex.
CPL_ERROR_ILLEGAL_INPUT	The input *base* is not positive.

Each array element is replaced by its logarithm in the specified base. The operation is always performed in double precision, with a final cast of the result to the array type. Invalid elements are not modified by this operation, but zero or negative elements are invalidated by this operation. In case of complex numbers, values very close to the origin may cause an overflow.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.88 cpl_array_multiply()

```
cpl_error_code cpl_array_multiply (
            cpl_array * to_array,
            const cpl_array * from_array )
```

Multiply the values of two numeric or complex arrays.

Parameters

to_array	Target array.
from_array	Source array.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Any input *array* is a `NULL` pointer.
CPL_ERROR_INCOMPATIBLE_INPUT	The input arrays have different sizes.
CPL_ERROR_INVALID_TYPE	Any specified array is not numerical.

The arrays are multiplied element by element, and the result is stored in the target array. See the documentation of the function cpl_array_add() for further details.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.89 cpl_array_multiply_scalar()

```
cpl_error_code cpl_array_multiply_scalar (
            cpl_array * array,
            double value )
```

Multiply a numerical array by a constant value.

Parameters

array	Target array
value	Factor.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is a NULL pointer.
CPL_ERROR_INVALID_TYPE	The input array is not numerical.

The operation is always performed in double precision, with a final cast of the result to the target array type. Invalid elements are are not modified by this operation.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.90 cpl_array_multiply_scalar_complex()

```
cpl_error_code cpl_array_multiply_scalar_complex (
            cpl_array * array,
            double complex value )
```

Multiply a complex array by a constant complex value.

Parameters

array	Target array
value	Factor.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is a NULL pointer.
CPL_ERROR_INVALID_TYPE	The input array is not complex.

The operation is always performed in double precision, with a final cast of the result to the target array type. Invalid elements are are not modified by this operation.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.91 cpl_array_new()

```
cpl_array* cpl_array_new (
             cpl_size length,
             cpl_type type )
```

Create a new array of given type.

Parameters

length	Number of elements in array.
type	Type of array

Returns

Pointer to the new array, or NULL in case of error.

This function allocates memory for an array, its type is assigned, and its number of elements is allocated. Only arrays of types CPL_TYPE_INT, CPL_TYPE_FLOAT, CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT_COMPLEX, CPL_TYPE_DOUBLE_COMPLEX and CPL_TYPE_STRING, are supported. An error CPL_ERROR_INVALI↩D_TYPE is set in case other types are specified. All array elements are initially flagged as invalid. If a negative length is specified, an error CPL_ERROR_ILLEGAL_INPUT is set. Zero length arrays are allowed.

References cpl_calloc(), CPL_ERROR_INVALID_TYPE, CPL_TYPE_DOUBLE, CPL_TYPE_DOUBLE_COMPL↩EX, CPL_TYPE_FLOAT, CPL_TYPE_FLOAT_COMPLEX, cpl_type_get_name(), CPL_TYPE_INT, CPL_TYPE_↩LONG, CPL_TYPE_LONG_LONG, CPL_TYPE_SIZE, and CPL_TYPE_STRING.

Referenced by cpl_array_duplicate(), cpl_array_extract(), cpl_array_extract_imag(), cpl_array_extract_real(), cpl↩_fit_image_gaussian(), cpl_table_get_column_names(), cpl_table_where_selected(), and cpl_wcs_convert().

4.1.2.92 cpl_array_power()

```
cpl_error_code cpl_array_power (
            cpl_array * array,
            double exponent )
```

Compute the power of numerical array elements.

Parameters

array	Pointer to numerical array.
exponent	Constant exponent.

Returns

`CPL_ERROR_NONE` on success.

See also

pow(), cpow()

Errors

CPL_ERROR_NULL_INPUT	Input *array* is `NULL`.
CPL_ERROR_INVALID_TYPE	The *array* is not numerical.

Each array element is replaced by its power to the specified exponent. Each column element is replaced by its power to the specified exponent. For float and float complex the operation is performed in single precision, otherwise it is performed in double precision and then rounded if the column is of an integer type. Results that would or do cause domain errors or overflow are marked as invalid.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.93 cpl_array_power_complex()

```
cpl_error_code cpl_array_power_complex (
            cpl_array * array,
            double complex exponent )
```

Compute the complex power of complex, numerical array elements.

Parameters

array	Pointer to numerical array.
exponent	Constant exponent.

Returns

CPL_ERROR_NONE on success.

See also

cpl_array_power()

Errors

CPL_ERROR_NULL_INPUT	Input *array* is NULL.
CPL_ERROR_INVALID_TYPE	The *array* is not numerical.

Each array element is replaced by its power to the specified exponent. Each column element is replaced by its power to the specified exponent. For float and float complex the operation is performed in single precision, otherwise it is performed in double precision and then rounded if the column is of an integer type. Results that would or do cause domain errors or overflow are marked as invalid.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.94 cpl_array_set()

```
cpl_error_code cpl_array_set (
            cpl_array * array,
            cpl_size indx,
            double value )
```

Write a value to a numerical array element.

Parameters

array	Array to be accessed.
indx	Position where to write value.
value	Value to write.

Returns

CPL_ERROR_NONE on success. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is returned. If the array is not of numerical type, a CPL_ERROR_INVALID_TYPE is returned. If *indx* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is returned. If the input array has length zero, the CPL_ERROR_ACCESS_OUT_OF_RANGE is always returned.

Write a value to a numerical array element. The value is cast to the accessed array type. The written value is automatically flagged as valid. To invalidate an array value use cpl_array_set_invalid(). Array elements are counted starting from 0.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.95 cpl_array_set_complex()

```
cpl_error_code cpl_array_set_complex (
            cpl_array * array,
            cpl_size indx,
            double complex value )
```

Write a value to a complex array element.

Parameters

array	Array to be accessed.
indx	Position where to write value.
value	Value to write.

Returns

CPL_ERROR_NONE on success. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is returned. If the array is not of numerical type, a CPL_ERROR_INVALID_TYPE is returned. If *indx* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is returned. If the input array has length zero, the CPL_ERROR_ACCESS_OUT_OF_RANGE is always returned.

Write a value to a numerical array element. The value is cast to the accessed array type. The written value is automatically flagged as valid. To invalidate an array value use cpl_array_set_invalid(). Array elements are counted starting from 0.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.96 cpl_array_set_cplsize()

```
cpl_error_code cpl_array_set_cplsize (
            cpl_array * array,
            cpl_size indx,
            cpl_size value )
```

Write a value to a *cpl_size* array element.

Parameters

array	Array to be accessed.
indx	Position where to write value.
value	Value to write.

Returns

CPL_ERROR_NONE on success. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is returned. If the array is not of the expected type, a CPL_ERROR_TYPE_MISMATCH is set. If *indx* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is returned. If the input array has length 0, the CPL_↩ ERROR_ACCESS_OUT_OF_RANGE is always returned.

Write a value to a *cpl_size* array element. The written value is automatically flagged as valid. To invalidate an array value use cpl_array_set_invalid(). Array elements are counted starting from 0.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.97 cpl_array_set_double()

```
cpl_error_code cpl_array_set_double (
            cpl_array * array,
            cpl_size indx,
            double value )
```

Write a value to a *double* array element.

Parameters

array	Array to be accessed.
indx	Position where to write value.
value	Value to write.

Returns

CPL_ERROR_NONE on success. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is returned. If the array is not of the expected type, a CPL_ERROR_TYPE_MISMATCH is set. If *indx* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is returned. If the input array has length 0, the CPL_↩ ERROR_ACCESS_OUT_OF_RANGE is always returned.

Write a value to a *double* array element. The written value is automatically flagged as valid. To invalidate an array value use cpl_array_set_invalid(). Array elements are counted starting from 0.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_fit_image_gaussian().

4.1.2.98 cpl_array_set_double_complex()

```
cpl_error_code cpl_array_set_double_complex (
            cpl_array * array,
            cpl_size indx,
            double complex value )
```

Write a value to a *double* complex array element.

Parameters

array	Array to be accessed.
indx	Position where to write value.
value	Value to write.

Returns

CPL_ERROR_NONE on success. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is returned. If the array is not of the expected type, a CPL_ERROR_TYPE_MISMATCH is set. If *indx* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is returned. If the input array has length 0, the CPL_↩ ERROR_ACCESS_OUT_OF_RANGE is always returned.

Write a value to a *double* array element. The written value is automatically flagged as valid. To invalidate an array value use cpl_array_set_invalid(). Array elements are counted starting from 0.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.99 cpl_array_set_float()

```
cpl_error_code cpl_array_set_float (
            cpl_array * array,
            cpl_size indx,
            float value )
```

Write a value to a *float* array element.

Parameters

array	Array to be accessed.
indx	Position where to write value.
value	Value to write.

Returns

CPL_ERROR_NONE on success. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is returned. If the array is not of the expected type, a CPL_ERROR_TYPE_MISMATCH is set. If *indx* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is returned. If the input array has length 0, the CPL_↩ ERROR_ACCESS_OUT_OF_RANGE is always returned.

Write a value to a *float* array element. The written value is automatically flagged as valid. To invalidate an array value use cpl_array_set_invalid(). Array elements are counted starting from 0.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.100 cpl_array_set_float_complex()

```
cpl_error_code cpl_array_set_float_complex (
            cpl_array * array,
            cpl_size indx,
            float complex value )
```

Write a value to a *float* complex array element.

Parameters

array	Array to be accessed.
indx	Position where to write value.
value	Value to write.

Returns

CPL_ERROR_NONE on success. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is returned. If the array is not of the expected type, a CPL_ERROR_TYPE_MISMATCH is set. If *indx* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is returned. If the input array has length 0, the CPL_↩ ERROR_ACCESS_OUT_OF_RANGE is always returned.

Write a value to a *float* complex array element. The written value is automatically flagged as valid. To invalidate an array value use cpl_array_set_invalid(). Array elements are counted starting from 0.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.101 cpl_array_set_int()

```
cpl_error_code cpl_array_set_int (
            cpl_array * array,
            cpl_size indx,
            int value )
```

Write a value to an *integer* array element.

Parameters

array	Array to be accessed.
indx	Position where to write value.
value	Value to write.

Returns

CPL_ERROR_NONE on success. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is returned. If the array is not of the expected type, a CPL_ERROR_TYPE_MISMATCH is set. If *indx* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is returned. If the input array has length 0, the CPL_↩ ERROR_ACCESS_OUT_OF_RANGE is always returned.

Write a value to an *integer* array element. The written value is automatically flagged as valid. To invalidate an array value use cpl_array_set_invalid(). Array elements are counted starting from 0.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.102 cpl_array_set_invalid()

```
cpl_error_code cpl_array_set_invalid (
            cpl_array * array,
            cpl_size indx )
```

Invalidate an array element.

Parameters

array	Array to be accessed
indx	Position of element to invalidate

Returns

CPL_ERROR_NONE on success. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is returned. If *indx* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is set. If the input array has length 0, the CPL_ERROR_ACCESS_OUT_OF_RANGE is always set.

In the case of a string array, the string is set free and its pointer is set to NULL; for other data types, the corresponding element of the null flags buffer is flagged. Array elements are counted starting from zero.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_fit_image_gaussian().

4.1.2.103 cpl_array_set_long()

```
cpl_error_code cpl_array_set_long (
            cpl_array * array,
            cpl_size indx,
            long value )
```

Write a value to a *long integer* array element.

Parameters

array	Array to be accessed.
indx	Position where to write value.
value	Value to write.

Returns

CPL_ERROR_NONE on success. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is returned. If the array is not of the expected type, a CPL_ERROR_TYPE_MISMATCH is set. If *indx* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is returned. If the input array has length 0, the CPL_↩ ERROR_ACCESS_OUT_OF_RANGE is always returned.

Write a value to a *long integer* array element. The written value is automatically flagged as valid. To invalidate an array value use cpl_array_set_invalid(). Array elements are counted starting from 0.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.104 cpl_array_set_long_long()

```
cpl_error_code cpl_array_set_long_long (
            cpl_array * array,
            cpl_size indx,
            long long value )
```

Write a value to a *long long integer* array element.

Parameters

array	Array to be accessed.
indx	Position where to write value.
value	Value to write.

Returns

CPL_ERROR_NONE on success. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is returned. If the array is not of the expected type, a CPL_ERROR_TYPE_MISMATCH is set. If *indx* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is returned. If the input array has length 0, the CPL_↩ ERROR_ACCESS_OUT_OF_RANGE is always returned.

Write a value to a *long long integer* array element. The written value is automatically flagged as valid. To invalidate an array value use cpl_array_set_invalid(). Array elements are counted starting from 0.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.105 cpl_array_set_size()

```
cpl_error_code cpl_array_set_size (
            cpl_array * array,
            cpl_size new_length )
```

Resize an array.

Parameters

array	Input array.
new_length	New number of elements in array.

Returns

CPL_ERROR_NONE on success. The new array size must not be negative, or a CPL_ERROR_ILLEG↩ AL_INPUT is returned. The input array pointer should not be NULL, or a CPL_ERROR_NULL_INPUT is returned.

Reallocate an array to a new number of elements. The contents of the array data buffer will be unchanged up to the lesser of the new and old sizes. If the array size is increased, the new array elements are flagged as invalid. The pointer to data may change, therefore pointers previously retrieved by calling cpl_array_get_data_int(), cpl_array_get_data_string(), etc. should be discarded). Resizing to zero is allowed, and would produce a zero-length array. In case of failure, the old data buffer is left intact.

If the array is *NULL*, zero is returned.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.106 cpl_array_set_string()

```
cpl_error_code cpl_array_set_string (
            cpl_array * array,
            cpl_size indx,
            const char * string )
```

Write a character string to a string array element.

Parameters

array	Array to be accessed.
indx	Position where to write character string.
string	Character string to write.

Returns

CPL_ERROR_NONE on success. If *array* is a NULL pointer a CPL_ERROR_NULL_INPUT is returned. If the array is not of the expected type, a CPL_ERROR_TYPE_MISMATCH is returned. If *indx* is outside the array range, a CPL_ERROR_ACCESS_OUT_OF_RANGE is returned. If the input array has length 0, the CPL_ERROR_ACCESS_OUT_OF_RANGE is always returned.

Copy a character string to a *string* array element. The written value can also be a NULL pointer. Note that the input character string is copied, therefore the original can be modified without affecting the column content. To "plug" a character string directly into an array element, use the function cpl_array_get_data_string(). Array elements are counted starting from zero.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_table_get_column_names().

4.1.2.107 cpl_array_subtract()

```
cpl_error_code cpl_array_subtract (
            cpl_array * to_array,
            const cpl_array * from_array )
```

Subtract the values of two numeric or complex arrays.

Parameters

to_array	Target array.
from_array	Source array.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any input *array* is a NULL pointer.
CPL_ERROR_INCOMPATIBLE_INPUT	The input arrays have different sizes.
CPL_ERROR_INVALID_TYPE	Any specified array is not numerical.

The arrays are subtracted element by element, and the result is stored in the target array. See the documentation of the function cpl_array_add() for further details.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.108 cpl_array_subtract_scalar()

```
cpl_error_code cpl_array_subtract_scalar (
            cpl_array * array,
            double value )
```

Subtract a constant value from a numerical array.

Parameters

array	Target array
value	Value to subtract.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is a NULL pointer.
CPL_ERROR_INVALID_TYPE	The input array is not numerical.

The operation is always performed in double precision, with a final cast of the result to the target array type. Invalid elements are are not modified by this operation.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.109 cpl_array_subtract_scalar_complex()

```
cpl_error_code cpl_array_subtract_scalar_complex (
            cpl_array * array,
            double complex value )
```

Subtract a constant complex value from a complex array.

Parameters

array	Target array
value	Value to subtract.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *array* is a `NULL` pointer.
CPL_ERROR_INVALID_TYPE	The input array is not complex.

The operation is always performed in double precision, with a final cast of the result to the target array type. Invalid elements are are not modified by this operation.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.1.2.110 cpl_array_unwrap()

```
void* cpl_array_unwrap (
            cpl_array * array )
```

Delete an array, without losing the data buffer.

Parameters

array	Array to be deleted.

Returns

Pointer to the internal data buffer.

This function deletes an array, but its data buffer is not destroyed. Supposedly, the developer knows that the data are static, or the developer holds the pointer to the data obtained with the functions cpl_array_get_data_int(), cpl_array_get_data_float(), etc. If the input array is NULL, nothing is done, and no error is set.

References cpl_free().

Referenced by cpl_wcs_delete().

4.1.2.111 cpl_array_wrap_cplsize()

```
cpl_array* cpl_array_wrap_cplsize (
            cpl_size * data,
            cpl_size length )
```

Create a new *cpl_size* array from existing data.

Parameters

data	Existing data buffer.
length	Number of elements in array.

Returns

Pointer to the new array, or NULL in case of error.

This function creates a new *cpl_size* array that will encapsulate the given data. Note that the size of the data buffer is not checked in any way, and that the data values are all considered valid: invalid values should be marked using the functions cpl_array_set_invalid(). The data array is not copied, so it should never be deallocated: to deallocate it, the function cpl_array_delete() should be called instead. Alternatively, the function cpl_array_unwrap() might be used, and the data array deallocated afterwards. A zero or negative length is illegal, and would cause an error CPL_ERROR_ILLEGAL_INPUT to be set. An input NULL pointer would set an error CPL_ERROR_NULL_INPUT.

Note

Functions that handle arrays assume that an array data buffer is dynamically allocated: with a statically allocated data buffer any function implying memory handling (cpl_array_set_size(), cpl_array_delete(), etc.) would crash the program. This means that a static data buffer should never be passed to this function if memory handling is planned. In case of a static data buffer, only the cpl_array_unwrap() destructor can be used.

References cpl_calloc().

4.1.2.112 cpl_array_wrap_double()

```
cpl_array* cpl_array_wrap_double (
             double * data,
             cpl_size length )
```

Create a new *double* array from existing data.

Parameters

data	Existing data buffer.
length	Number of elements in array.

Returns

Pointer to the new array, or NULL in case of error.

See documentation of function cpl_array_wrap_int().

References cpl_calloc().

4.1.2.113 cpl_array_wrap_double_complex()

```
cpl_array* cpl_array_wrap_double_complex (
             double complex * data,
             cpl_size length )
```

Create a new *double* complex array from existing data.

Parameters

data	Existing data buffer.
length	Number of elements in array.

Returns

Pointer to the new array, or NULL in case of error.

See documentation of function cpl_array_wrap_int().

References cpl_calloc().

4.1.2.114 cpl_array_wrap_float()

```
cpl_array* cpl_array_wrap_float (
             float * data,
             cpl_size length )
```

Create a new *float* array from existing data.

Parameters

data	Existing data buffer.
length	Number of elements in array.

Returns

Pointer to the new array, or NULL in case of error.

See documentation of function cpl_array_wrap_int().

References cpl_calloc().

4.1.2.115 cpl_array_wrap_float_complex()

```
cpl_array* cpl_array_wrap_float_complex (
             float complex * data,
             cpl_size length )
```

Create a new *float* complex array from existing data.

Parameters

data	Existing data buffer.
length	Number of elements in array.

Returns

Pointer to the new array, or NULL in case of error.

See documentation of function cpl_array_wrap_int().

References cpl_calloc().

4.1.2.116 cpl_array_wrap_int()

```
cpl_array* cpl_array_wrap_int (
             int * data,
             cpl_size length )
```

Create a new *integer* array from existing data.

Parameters

data	Existing data buffer.
length	Number of elements in array.

Returns

Pointer to the new array, or NULL in case of error.

This function creates a new *integer* array that will encapsulate the given data. Note that the size of the data buffer is not checked in any way, and that the data values are all considered valid: invalid values should be marked using the functions cpl_array_set_invalid() The data array is not copied, so it should never be deallocated: to deallocate it, the function cpl_array_delete() should be called instead. Alternatively, the function cpl_array_unwrap() might be used, and the data array deallocated afterwards. A zero or negative length is illegal, and would cause an error CPL_ERROR_ILLEGAL_INPUT to be set. An input NULL pointer would set an error CPL_ERROR_NULL_INPUT.

Note

Functions that handle arrays assume that an array data buffer is dynamically allocated: with a statically allocated data buffer any function implying memory handling (cpl_array_set_size(), cpl_array_delete(), etc.) would crash the program. This means that a static data buffer should never be passed to this function if memory handling is planned. In case of a static data buffer, only the cpl_array_unwrap() destructor can be used.

References cpl_calloc().

Referenced by cpl_matrix_get_determinant(), cpl_matrix_invert_create(), and cpl_matrix_solve().

4.1.2.117 cpl_array_wrap_long()

```
cpl_array* cpl_array_wrap_long (
             long * data,
             cpl_size length )
```

Create a new *long integer* array from existing data.

Parameters

data	Existing data buffer.
length	Number of elements in array.

Returns

Pointer to the new array, or NULL in case of error.

This function creates a new *long integer* array that will encapsulate the given data. Note that the size of the data buffer is not checked in any way, and that the data values are all considered valid: invalid values should be marked using the functions cpl_array_set_invalid() The data array is not copied, so it should never be deallocated: to deallocate it, the function cpl_array_delete() should be called instead. Alternatively, the function cpl_array_unwrap() might be used, and the data array deallocated afterwards. A zero or negative length is illegal, and would cause an error CPL_ERROR_ILLEGAL_INPUT to be set. An input NULL pointer would set an error CPL_ERROR_NULL_INPUT.

Note

Functions that handle arrays assume that an array data buffer is dynamically allocated: with a statically allocated data buffer any function implying memory handling (cpl_array_set_size(), cpl_array_delete(), etc.) would crash the program. This means that a static data buffer should never be passed to this function if memory handling is planned. In case of a static data buffer, only the cpl_array_unwrap() destructor can be used.

References cpl_calloc().

4.1.2.118 cpl_array_wrap_long_long()

```
cpl_array* cpl_array_wrap_long_long (
             long long * data,
             cpl_size length )
```

Create a new *long long integer* array from existing data.

Parameters

data	Existing data buffer.
length	Number of elements in array.

Returns

Pointer to the new array, or NULL in case of error.

This function creates a new *long long integer* array that will encapsulate the given data. Note that the size of the data buffer is not checked in any way, and that the data values are all considered valid: invalid values should be marked using the functions cpl_array_set_invalid() The data array is not copied, so it should never be deallocated: to deallocate it, the function cpl_array_delete() should be called instead. Alternatively, the function cpl_array_unwrap() might be used, and the data array deallocated afterwards. A zero or negative length is illegal, and would cause an error CPL_ERROR_ILLEGAL_INPUT to be set. An input NULL pointer would set an error CPL_ERROR_NULL_INPUT.

Note

Functions that handle arrays assume that an array data buffer is dynamically allocated: with a statically allocated data buffer any function implying memory handling (cpl_array_set_size(), cpl_array_delete(), etc.) would crash the program. This means that a static data buffer should never be passed to this function if memory handling is planned. In case of a static data buffer, only the cpl_array_unwrap() destructor can be used.

References cpl_calloc().

4.1.2.119 cpl_array_wrap_string()

```
cpl_array* cpl_array_wrap_string (
             char ** data,
             cpl_size length )
```

Create a new character string array from existing data.

Parameters

data	Existing data buffer.
length	Number of elements in array.

Returns

Pointer to the new array, or `NULL` in case of error.

See documentation of function `cpl_array_wrap_int()`.

References cpl_calloc().

4.2 Auxiliary Frame Data

Auxiliary frame data for recipe configurations.

Classes

- struct _cpl_framedata_

  The public frame data object.

Typedefs

- typedef struct _cpl_framedata_ cpl_framedata

  The frame data object type.

Functions

- void cpl_framedata_clear (cpl_framedata *self)

  Clear a frame data object.
- cpl_framedata * cpl_framedata_create (const char *tag, cpl_size min_count, cpl_size max_count)

  Create a new frame data object and initialize it with the given values.
- void cpl_framedata_delete (cpl_framedata *self)

  Delete a frame data object.
- cpl_framedata * cpl_framedata_duplicate (const cpl_framedata *other)

  Create a duplicate of another frame data object.
- cpl_size cpl_framedata_get_max_count (const cpl_framedata *self)

  Get the maximum number of frames.
- cpl_size cpl_framedata_get_min_count (const cpl_framedata *self)

  Get the minimum number of frames.
- const char * cpl_framedata_get_tag (const cpl_framedata *self)

  Get the frame tag.
- cpl_framedata * cpl_framedata_new (void)

  Create an new frame data object.
- cpl_error_code cpl_framedata_set (cpl_framedata *self, const char *tag, cpl_size min_count, cpl_size max_count)

  Assign new values to a frame data object.
- cpl_error_code cpl_framedata_set_max_count (cpl_framedata *self, cpl_size max_count)

  Set the maximum number of frames.
- cpl_error_code cpl_framedata_set_min_count (cpl_framedata *self, cpl_size min_count)

  Set the minimum number of frames.
- cpl_error_code cpl_framedata_set_tag (cpl_framedata *self, const char *tag)

  Set the frame tag to the given value.

4.2.1 Detailed Description

Auxiliary frame data for recipe configurations.

This module implements a frame data object, which stores auxiliary information of input and output frames of recipes. This information is used to store the frame configurations of recipes which can be queried by an application which is going to invoke the recipe.

The objects stores a frame tag, a unique identifier for a certain kind of frame, the minimum and maximum number of frames needed.

A frame is required if the data member *min_count* is set to a value greater than `0`. The minimum and maximum number of frames is unspecified if the respective member, *min_count* or *max_count*, is set to `-1`.

The data members of this structure are public to allow for a static initialization. Any other access of the data members should still be done using the member functions.

Synopsis:

```
#include <cpl_framedata.h>
```

4.2.2 Typedef Documentation

4.2.2.1 cpl_framedata

```
typedef struct _cpl_framedata_ cpl_framedata
```

The frame data object type.

4.2.3 Function Documentation

4.2.3.1 cpl_framedata_clear()

```
void cpl_framedata_clear (
            cpl_framedata * self )
```

Clear a frame data object.

Parameters

self	The frame data object to clear.

Returns

Nothing.

The function clears the contents of the frame data object *self*, i.e. resets the data members to their default values. If *self* is NULL, nothing is done and no error is set.

Referenced by cpl_framedata_delete().

4.2.3.2 cpl_framedata_create()

```
cpl_framedata* cpl_framedata_create (
            const char * tag,
            cpl_size min_count,
            cpl_size max_count )
```

Create a new frame data object and initialize it with the given values.

Parameters

tag	The frame tag initializer.
min_count	The initial value for the minimum number of frames.
max_count	The initial value for the maximum number of frames.

Returns

On success the function returns a pointer to the newly created object, or NULL otherwise.

The function allocates the memory for a frame data object, and initializes its data members with the given values of the arguments *tag*, *min_count*, and *max_count*.

4.2.3.3 cpl_framedata_delete()

```
void cpl_framedata_delete (
            cpl_framedata * self )
```

Delete a frame data object.

Parameters

self	The frame data object to delete.

Returns

Nothing.

The function destroys the frame data object *self*. All resources used by *self* are released. If *self* is NULL, nothing is done and no error is set.

References cpl_framedata_clear().

4.2.3.4 cpl_framedata_duplicate()

```
cpl_framedata* cpl_framedata_duplicate (
            const cpl_framedata * other )
```

Create a duplicate of another frame data object.

Parameters

other	The frame data object to clone.

Returns

On success the function returns a pointer to the newly created copy of the frame data object, or NULL otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *other* is a NULL pointer.

The function creates a clone of the given frame data object *other*. The created copy does not share any resources with the original object.

References CPL_ERROR_NULL_INPUT.

4.2.3.5 cpl_framedata_get_max_count()

```
cpl_size cpl_framedata_get_max_count (
            const cpl_framedata * self )
```

Get the maximum number of frames.

Parameters

self	The frame data object.

Returns

The function returns the maximum number of frames on success. In case an error occurred, the return value is −2 and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function returns the maximum number of frames value stored in *self*. If the returned value is `-1`, the maximum number of frames is undefined, i.e. any number may be used.

References CPL_ERROR_NULL_INPUT.

4.2.3.6 cpl_framedata_get_min_count()

```
cpl_size cpl_framedata_get_min_count (
            const cpl_framedata * self )
```

Get the minimum number of frames.

Parameters

self	The frame data object.

Returns

The function returns the minimum number of frames on success. In case an error occurred, the return value is `-2` and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function returns the minimum number of frames value stored in *self*. If the returned value is `-1`, the minimum number of frames is undefined, i.e. any number may be used.

References CPL_ERROR_NULL_INPUT.

4.2.3.7 cpl_framedata_get_tag()

```
const char* cpl_framedata_get_tag (
            const cpl_framedata * self )
```

Get the frame tag.

Parameters

self	The frame data object.

Returns

The function returns a pointer to the frame tag on success, or `NULL` if an error occurred.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function returns a handle to the frame tag stored in the frame data object *self*.

References CPL_ERROR_NULL_INPUT.

4.2.3.8 cpl_framedata_new()

```
cpl_framedata* cpl_framedata_new (
            void  )
```

Create an new frame data object.

Returns

On success the function returns a pointer to the newly created object, or `NULL` otherwise.

The function allocates the memory for a frame data object, and initializes the data members to their default values, i.e. *tag* is set to `NULL`, and both, *min_count* and *max_count* are set to `-1`.

4.2.3.9 cpl_framedata_set()

```
cpl_error_code cpl_framedata_set (
            cpl_framedata * self,
            const char * tag,
            cpl_size min_count,
            cpl_size max_count )
```

Assign new values to a frame data object.

Parameters

self	The frame data object.
tag	The tag to assign.
min_count	The value to set as minimum number of frames.
max_count	The value to set as maximum number of frames.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	The parameter *tag* is a `NULL` pointer or the empty string.

The function updates the frame data object *self* with the given values for *tag*, *min_count*, and *max_count*. All previous values stored in *self* are replaced. The string *tag* is assigned by copying its contents.

See also

cpl_framedata_set_tag()

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.2.3.10 cpl_framedata_set_max_count()

```
cpl_error_code cpl_framedata_set_max_count (
            cpl_framedata * self,
            cpl_size max_count )
```

Set the maximum number of frames.

Parameters

self	The frame data object.
max_count	The value to set as maximum number of frames.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function sets the maximum number of frames value of *self* to *max_count*. If *max_count* is `-1` the maximum number of frames is unspecified.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.2.3.11 cpl_framedata_set_min_count()

```
cpl_error_code cpl_framedata_set_min_count (
            cpl_framedata * self,
            cpl_size min_count )
```

Set the minimum number of frames.

Parameters

self	The frame data object.
min_count	The value to set as minimum number of frames.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function sets the minimum number of frames value of *self* to *min_count*. If *min_count* is `-1` the minimum number of frames is unspecified.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.2.3.12 cpl_framedata_set_tag()

```
cpl_error_code cpl_framedata_set_tag (
            cpl_framedata * self,
            const char * tag )
```

Set the frame tag to the given value.

Parameters

self	The frame data object.
tag	The tag to assign.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	The parameter *tag* is a `NULL` pointer or the empty string.

The function assigns the string *tag* to the corresponding data member of the frame data object *self* by copying its contents. Any previous tag stored in *self* is replaced.

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.3 Bi-vector object

Functions

- cpl_error_code cpl_bivector_copy (cpl_bivector *self, const cpl_bivector *other)
  Copy contents of a bivector into another bivector.
- void cpl_bivector_delete (cpl_bivector *f)
  Delete a cpl_bivector.
- void cpl_bivector_dump (const cpl_bivector *f, FILE *stream)
  Dump a cpl_bivector as ASCII to a stream.
- cpl_bivector * cpl_bivector_duplicate (const cpl_bivector *in)
  Duplicate a cpl_bivector.
- cpl_size cpl_bivector_get_size (const cpl_bivector *in)
  Get the size of the cpl_bivector.
- cpl_vector * cpl_bivector_get_x (cpl_bivector *in)
  Get a pointer to the x vector of the cpl_bivector.
- const cpl_vector * cpl_bivector_get_x_const (const cpl_bivector *in)
  Get a pointer to the x vector of the cpl_bivector.
- double * cpl_bivector_get_x_data (cpl_bivector *in)
  Get a pointer to the x data part of the cpl_bivector.
- const double * cpl_bivector_get_x_data_const (const cpl_bivector *in)
  Get a pointer to the x data part of the cpl_bivector.
- cpl_vector * cpl_bivector_get_y (cpl_bivector *in)
  Get a pointer to the y vector of the cpl_bivector.
- const cpl_vector * cpl_bivector_get_y_const (const cpl_bivector *in)
  Get a pointer to the y vector of the cpl_bivector.
- double * cpl_bivector_get_y_data (cpl_bivector *in)
  Get a pointer to the y data part of the cpl_bivector.
- const double * cpl_bivector_get_y_data_const (const cpl_bivector *in)
  Get a pointer to the y data part of the cpl_bivector.
- cpl_error_code cpl_bivector_interpolate_linear (cpl_bivector *fout, const cpl_bivector *fref)
  Linear interpolation of a 1d-function.
- cpl_bivector * cpl_bivector_new (cpl_size n)
  Create a new cpl_bivector.
- cpl_bivector * cpl_bivector_read (const char *filename)
  Read a list of values from an ASCII file and create a cpl_bivector.
- cpl_error_code cpl_bivector_sort (cpl_bivector *self, const cpl_bivector *other, cpl_sort_direction dir, cpl_sort_mode mode)
  Sort a cpl_bivector.
- void cpl_bivector_unwrap_vectors (cpl_bivector *f)
  Free memory associated to a cpl_bivector, excluding the two vectors.
- cpl_bivector * cpl_bivector_wrap_vectors (cpl_vector *x, cpl_vector *y)
  Create a new cpl_bivector from two cpl_vectors.

4.3.1 Detailed Description

This module provides functions to handle *cpl_bivector*.

A *cpl_bivector* is composed of two vectors of the same size. It can be used to store 1d functions, with the x and y positions of the samples, offsets in x and y or simply positions in an image.

This module provides among other things functions for interpolation and for sorting one vector according to another.

Synopsis:

```
#include "cpl_bivector.h"
```

4.3.2 Function Documentation

4.3.2.1 cpl_bivector_copy()

```
cpl_error_code cpl_bivector_copy (
            cpl_bivector * self,
            const cpl_bivector * other )
```

Copy contents of a bivector into another bivector.

Parameters

self	destination cpl_vector
other	source cpl_vector

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_vector_set_size() if source and destination have different sizes.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_bivector_get_x(), cpl_bivector_get_x_const(), cpl_bivector_get_y(), cpl_bivector_get_y_const(), CPL_ERROR_NONE, and cpl_vector_copy().

4.3.2.2 cpl_bivector_delete()

```
void cpl_bivector_delete (
            cpl_bivector * f )
```

Delete a cpl_bivector.

Parameters

f	cpl_bivector to delete

Returns

void

This function deletes a bivector. If the input bivector *f* is `NULL`, nothing is done, and no error is set.

References cpl_free(), and cpl_vector_delete().

Referenced by cpl_apertures_get_fwhm(), cpl_geom_img_offset_combine(), cpl_image_fit_gaussian(), cpl_↩ wlcalib_slitmodel_delete(), and cpl_wlcalib_slitmodel_set_catalog().

4.3.2.3 cpl_bivector_dump()

```
void cpl_bivector_dump (
            const cpl_bivector * f,
            FILE * stream )
```

Dump a cpl_bivector as ASCII to a stream.

Parameters

f	Input cpl_bivector to dump or NULL
stream	Output stream, accepts `stdout` or `stderr` or NULL

Returns

void

Comment lines start with the hash character.

stream may be NULL in which case `stdout` is used.

Note

In principle a cpl_bivector can be saved using cpl_bivector_dump() and re-read using cpl_bivector_read(). This will however introduce significant precision loss due to the limited accuracy of the ASCII representation.

References cpl_bivector_get_size(), cpl_vector_get_data_const(), and cpl_vector_get_size().

4.3.2.4 cpl_bivector_duplicate()

```
cpl_bivector* cpl_bivector_duplicate (
            const cpl_bivector * in )
```

Duplicate a cpl_bivector.

Parameters

in	cpl_bivector to duplicate

Returns

1 newly allocated cpl_bivector or NULL on error

The returned object must be deallocated using cpl_bivector_delete()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the input bivector contains vectors of different sizes

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_malloc(), cpl_vector↩ _duplicate(), and cpl_vector_get_size().

4.3.2.5 cpl_bivector_get_size()

```
cpl_size cpl_bivector_get_size (
            const cpl_bivector * in )
```

Get the size of the cpl_bivector.

Parameters

in	the input bivector

Returns

The size or a negative number on error

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the input bivector contains vectors of different sizes

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, and cpl_vector_get_size().

Referenced by cpl_bivector_dump(), cpl_bivector_interpolate_linear(), cpl_bivector_sort(), cpl_geom_img_offset↩ _combine(), cpl_geom_img_offset_fine(), cpl_geom_img_offset_saa(), and cpl_plot_bivector().

4.3.2.6 cpl_bivector_get_x()

```
cpl_vector* cpl_bivector_get_x (
            cpl_bivector * in )
```

Get a pointer to the x vector of the cpl_bivector.

Parameters

in	a cpl_bivector

Returns

Pointer to the x vector or NULL on error

The returned pointer refers to an already created cpl_vector.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_bivector_copy(), and cpl_geom_img_offset_combine().

4.3.2.7 cpl_bivector_get_x_const()

```
const cpl_vector* cpl_bivector_get_x_const (
            const cpl_bivector * in )
```

Get a pointer to the x vector of the cpl_bivector.

Parameters

in	a cpl_bivector

Returns

Pointer to the x vector or NULL on error

See also

cpl_bivector_get_x

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_bivector_copy(), cpl_bivector_interpolate_linear(), and cpl_geom_img_offset_saa().

4.3.2.8 cpl_bivector_get_x_data()

```
double* cpl_bivector_get_x_data (
            cpl_bivector * in )
```

Get a pointer to the x data part of the cpl_bivector.

Parameters

in	a cpl_bivector

Returns

Pointer to the double x array or NULL on error

See also

cpl_vector_get_data The returned pointer refers to already allocated data.

Note

Use at your own risk: direct manipulation of vector data rules out any check performed by the vector object interface, and may introduce inconsistencies between the information maintained internally, and the actual vector data and structure.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, CPL_ERROR_NULL_INPUT, and cpl_vector_get_data().

Referenced by cpl_apertures_get_fwhm(), cpl_bivector_interpolate_linear(), cpl_bivector_sort(), cpl_geom_img_offset_combine(), cpl_geom_img_offset_fine(), cpl_image_fit_gaussian(), and cpl_image_iqe().

4.3.2.9 cpl_bivector_get_x_data_const()

```
const double* cpl_bivector_get_x_data_const (
            const cpl_bivector * in )
```

Get a pointer to the x data part of the cpl_bivector.

Parameters

in	a cpl_bivector

Returns

Pointer to the double x array or NULL on error

See also

cpl_bivector_get_x_data

References cpl_ensure, CPL_ERROR_NULL_INPUT, and cpl_vector_get_data_const().

Referenced by cpl_bivector_interpolate_linear(), cpl_bivector_sort(), cpl_geom_img_offset_fine(), and cpl_plot_↩ bivector().

4.3.2.10 cpl_bivector_get_y()

```
cpl_vector* cpl_bivector_get_y (
            cpl_bivector * in )
```

Get a pointer to the y vector of the cpl_bivector.

Parameters

in	a cpl_bivector

Returns

Pointer to the y vector or NULL on error

The returned pointer refers to an already created cpl_vector.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_bivector_copy(), and cpl_geom_img_offset_combine().

4.3.2.11 cpl_bivector_get_y_const()

```
const cpl_vector* cpl_bivector_get_y_const (
            const cpl_bivector * in )
```

Get a pointer to the y vector of the cpl_bivector.

Parameters

in	a cpl_bivector

Returns

Pointer to the y vector or NULL on error

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_bivector_copy(), and cpl_geom_img_offset_saa().

4.3.2.12 cpl_bivector_get_y_data()

```
double* cpl_bivector_get_y_data (
            cpl_bivector * in )
```

Get a pointer to the y data part of the cpl_bivector.

Parameters

in	a cpl_bivector

Returns

Pointer to the double y array or NULL on error

See also

cpl_vector_get_x_data

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, CPL_ERROR_NULL_INPUT, and cpl_vector_get_data().

Referenced by cpl_apertures_get_fwhm(), cpl_bivector_interpolate_linear(), cpl_bivector_sort(), cpl_geom_img_↩ offset_combine(), cpl_geom_img_offset_fine(), and cpl_image_iqe().

4.3.2.13 cpl_bivector_get_y_data_const()

```
const double* cpl_bivector_get_y_data_const (
            const cpl_bivector * in )
```

Get a pointer to the y data part of the cpl_bivector.

Parameters

in	a cpl_bivector

Returns

Pointer to the double y array or NULL on error

See also

cpl_bivector_get_y_data

References cpl_ensure, CPL_ERROR_NULL_INPUT, and cpl_vector_get_data_const().

Referenced by cpl_bivector_interpolate_linear(), cpl_bivector_sort(), cpl_geom_img_offset_fine(), and cpl_plot_↩ bivector().

4.3.2.14 cpl_bivector_interpolate_linear()

```
cpl_error_code cpl_bivector_interpolate_linear (
            cpl_bivector * fout,
            const cpl_bivector * fref )
```

Linear interpolation of a 1d-function.

Parameters

fout	Preallocated with X-vector set, to hold interpolation in Y
fref	Reference 1d-function

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_

fref must have both its abscissa and ordinate defined. fout must have its abscissa defined and its ordinate allocated.

The linear interpolation will be done from the values in fref to the abscissa points in fout.

For each abscissa point in fout, fref must either have two neigboring abscissa points such that xref_i $<$ xout_j $<$ xref{i+1}, or a single identical abscissa point, such that xref_i == xout_j.

This is ensured by monotonely growing abscissa points in both fout and fref (and by min(xref) $<=$ min(xout) and max(xout) $<$ max(xref)).

However, for efficiency reasons (since fref can be very long) the monotonicity is only verified to the extent necessary to actually perform the interpolation.

This input requirement implies that extrapolation is not allowed.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_DATA_NOT_FOUND if fout has an endpoint which is out of range
- CPL_ERROR_ILLEGAL_INPUT if the monotonicity requirement on the 2 input abscissa vectors is not met.

References cpl_bivector_get_size(), cpl_bivector_get_x_const(), cpl_bivector_get_x_data(), cpl_bivector_get_x_↩ data_const(), cpl_bivector_get_y_data(), cpl_bivector_get_y_data_const(), cpl_ensure_code, CPL_ERROR_DA↩ TA_NOT_FOUND, cpl_error_get_code(), CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, and cpl_vector↩ _find().

4.3.2.15 cpl_bivector_new()

```
cpl_bivector* cpl_bivector_new (
            cpl_size n )
```

Create a new cpl_bivector.

Parameters

n	Positive number of points

Returns

1 newly allocated cpl_bivector or NULL on error

The returned object must be deallocated using cpl_bivector_delete() or cpl_bivector_unwrap_vectors(), provided the two cpl_vectors are deallocated separately.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_ILLEGAL_INPUT if n is $<$ 1.

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, cpl_malloc(), and cpl_vector_new().

Referenced by cpl_apertures_get_fwhm(), cpl_geom_img_offset_combine(), cpl_geom_img_offset_fine(), and cpl_image_iqe().

4.3.2.16 cpl_bivector_read()

```
cpl_bivector* cpl_bivector_read (
            const char * filename )
```

Read a list of values from an ASCII file and create a cpl_bivector.

Parameters

filename	Name of the input ASCII file

Returns

1 newly allocated cpl_bivector or NULL on error

See also

cpl_vector_load

The input ASCII file must contain two values per line.

The returned object must be deallocated using cpl_bivector_delete() Two columns of numbers are expected in the input file.

In addition to normal files, FIFO (see man mknod) are also supported.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_FILE_IO if the file cannot be read
- CPL_ERROR_BAD_FILE_FORMAT if the file contains no valid lines

References cpl_bivector_wrap_vectors(), cpl_ensure, CPL_ERROR_BAD_FILE_FORMAT, CPL_ERROR_FILE↩_IO, CPL_ERROR_NULL_INPUT, cpl_vector_delete(), cpl_vector_new(), cpl_vector_set(), and cpl_vector_set_↩size().

4.3.2.17 cpl_bivector_sort()

```
cpl_error_code cpl_bivector_sort (
             cpl_bivector * self,
             const cpl_bivector * other,
             cpl_sort_direction dir,
             cpl_sort_mode mode )
```

Sort a cpl_bivector.

Parameters

self	cpl_bivector to hold sorted result
other	Input cpl_bivector to sort, may equal self
dir	CPL_SORT_ASCENDING or CPL_SORT_DESCENDING
mode	CPL_SORT_BY_X or CPL_SORT_BY_Y

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

The values in the input are sorted according to direction and mode, and the result is placed self which must be of the same size as other.

As for qsort(): If two members compare as equal, their order in the sorted array is undefined.

In place sorting is supported.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if self and other have different sizes

- CPL_ERROR_ILLEGAL_INPUT if dir is neither CPL_SORT_DESCENDING nor CPL_SORT_ASCENDING.
- CPL_ERROR_UNSUPPORTED_MODE if self and other are the same or point to the same underlying arrays, or if mode is neither CPL_SORT_BY_X nor CPL_SORT_BY_Y

References cpl_bivector_get_size(), cpl_bivector_get_x_data(), cpl_bivector_get_x_data_const(), cpl_bivector_get_y_data(), cpl_bivector_get_y_data_const(), cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and CPL_ERROR_UNSUPPORTED_MODE.

4.3.2.18 cpl_bivector_unwrap_vectors()

```
void cpl_bivector_unwrap_vectors (
            cpl_bivector * f )
```

Free memory associated to a cpl_bivector, excluding the two vectors.

Parameters

f	cpl_bivector to delete

Returns

void

See also

cpl_bivector_wrap_vectors

References cpl_free().

Referenced by cpl_plot_column().

4.3.2.19 cpl_bivector_wrap_vectors()

```
cpl_bivector* cpl_bivector_wrap_vectors (
            cpl_vector * x,
            cpl_vector * y )
```

Create a new cpl_bivector from two cpl_vectors.

Parameters

x	the x cpl_vector
y	the y cpl_vector

Returns

1 cpl_bivector or NULL on error

Note

The input cpl_vectors must have identical sizes. Afterwards one of those two vectors may be resized, which will corrupt the bivector. Such a corrupted bivector should not be used any more, but rather deallocated, using cpl_bivector_unwrap_vectors() or cpl_bivector_delete().

The returned object must be deallocated using cpl_bivector_delete() or with cpl_bivector_unwrap_vectors(), provided the two cpl_vectors are deallocated separately.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the input vectors have different sizes

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_malloc(), and cpl_↩ vector_get_size().

Referenced by cpl_bivector_read(), cpl_plot_column(), and cpl_ppm_match_positions().

4.4 DFS related functions

Macros

- #define CPL_DFS_PRO_CATG "ESO PRO CATG"

  The name of the Product Category key.
- #define CPL_DFS_PRO_SCIENCE "ESO PRO SCIENCE"

  The name of the Product Science key.
- #define CPL_DFS_PRO_TECH "ESO PRO TECH"

  The name of the Product Tech key.
- #define CPL_DFS_PRO_TYPE "ESO PRO TYPE"

  The name of the Product Type key.

Enumerations

- enum { ,
  CPL_DFS_SIGNATURE_NONE,
  CPL_DFS_SIGNATURE_DATAMD5,
  CPL_DFS_SIGNATURE_CHECKSUM }

  Pipeline products digital signature flags.

Functions

- cpl_error_code cpl_dfs_save_imagelist (cpl_frameset *allframes, cpl_propertylist *header, const cpl_parameterlist *parlist, const cpl_frameset *usedframes, const cpl_frame *inherit, const cpl_imagelist *imagelist, cpl_type type, const char *recipe, const cpl_propertylist *applist, const char *remregexp, const char *pipe_id, const char *filename)

  Save an imagelist as a DFS-compliant pipeline product.
- cpl_error_code cpl_dfs_save_paf (const char *instrume, const char *recipe, const cpl_propertylist *paflist, const char *filename)

  Create a new PAF file.
- cpl_error_code cpl_dfs_save_propertylist (cpl_frameset *allframes, cpl_propertylist *header, const cpl_parameterlist *parlist, const cpl_frameset *usedframes, const cpl_frame *inherit, const char *recipe, const cpl_propertylist *applist, const char *remregexp, const char *pipe_id, const char *filename)

  Save a propertylist as a DFS-compliant pipeline product.
- cpl_error_code cpl_dfs_save_table (cpl_frameset *allframes, cpl_propertylist *header, const cpl_parameterlist *parlist, const cpl_frameset *usedframes, const cpl_frame *inherit, const cpl_table *table, const cpl_propertylist *tablelist, const char *recipe, const cpl_propertylist *applist, const char *remregexp, const char *pipe_id, const char *filename)

  Save a table as a DFS-compliant pipeline product.
- cpl_error_code cpl_dfs_setup_product_header (cpl_propertylist *header, const cpl_frame *product_frame, const cpl_frameset *framelist, const cpl_parameterlist *parlist, const char *recid, const char *pipeline_id, const char *dictionary_id, const cpl_frame *inherit_frame)

  Add product keywords to a pipeline product property list.
- cpl_error_code cpl_dfs_sign_products (const cpl_frameset *set, unsigned int flags)

  Update DFS and DICB required header information of product frames.
- cpl_error_code cpl_dfs_update_product_header (cpl_frameset *self)

  Perform any DFS-compliancy required actions (DATAMD5/PIPEFILE update)

4.4.1 Detailed Description

4.4.2 Macro Definition Documentation

4.4.2.1 CPL_DFS_PRO_CATG

```
#define CPL_DFS_PRO_CATG "ESO PRO CATG"
```

The name of the Product Category key.

See also

cpl_dfs_save_image()

Note

A pipeline product must contain a string property with this name

4.4.2.2 CPL_DFS_PRO_SCIENCE

```
#define CPL_DFS_PRO_SCIENCE "ESO PRO SCIENCE"
```

The name of the Product Science key.

See also

cpl_dfs_save_image()

Note

A pipeline product should contain a boolean property with this name

4.4.2.3 CPL_DFS_PRO_TECH

```
#define CPL_DFS_PRO_TECH "ESO PRO TECH"
```

The name of the Product Tech key.

See also

cpl_dfs_save_image()

Note

A pipeline product should contain a string property with this name

4.4.2.4 CPL_DFS_PRO_TYPE

```
#define CPL_DFS_PRO_TYPE "ESO PRO TYPE"
```

The name of the Product Type key.

See also

cpl_dfs_save_image()

Note

A pipeline product should contain a string property with this name

4.4.3 Enumeration Type Documentation

4.4.3.1 anonymous enum

```
anonymous enum
```

Pipeline products digital signature flags.

Flags to select the different digital signatures to compute for pipeline product files. The values may be combined using bitwise or.

Enumerator

CPL_DFS_SIGNATURE_NONE	Do not compute any signatures
CPL_DFS_SIGNATURE_DATAMD5	Compute the DATAMD5 data hash
CPL_DFS_SIGNATURE_CHECKSUM	Compute FITS standard CHECKSUM and DATASUM

4.4.4 Function Documentation

4.4.4.1 cpl_dfs_save_imagelist()

```
cpl_error_code cpl_dfs_save_imagelist (
            cpl_frameset * allframes,
            cpl_propertylist * header,
            const cpl_parameterlist * parlist,
            const cpl_frameset * usedframes,
            const cpl_frame * inherit,
            const cpl_imagelist * imagelist,
```

```
            cpl_type type,
            const char * recipe,
            const cpl_propertylist * applist,
            const char * remregexp,
            const char * pipe_id,
            const char * filename )
```

Save an imagelist as a DFS-compliant pipeline product.

Parameters

allframes	The list of input frames for the recipe
header	NULL, or filled with properties written to product header
parlist	The list of input parameters
usedframes	The list of raw/calibration frames used for this product
inherit	NULL or product frames inherit their header from this frame
imagelist	The imagelist to be saved
type	The type used to represent the data in the file
recipe	The recipe name
applist	Propertylist to append to primary header, w. PRO.CATG
remregexp	Optional regexp of properties not to put in main header
pipe_id	PACKAGE "/" PACKAGE_VERSION
filename	Filename of created product

Note

remregexp may be NULL

Returns

CPL_ERROR_NONE or the relevant CPL error code on error

See also

cpl_dfs_save_image(), cpl_imagelist_save().

References cpl_ensure_code, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.4.4.2 cpl_dfs_save_paf()

```
cpl_error_code cpl_dfs_save_paf (
            const char * instrume,
            const char * recipe,
            const cpl_propertylist * paflist,
            const char * filename )
```

Create a new PAF file.

Parameters

instrume	Name of instrument in capitals (NACO, VISIR, etc.)
recipe	Name of recipe
paflist	Propertylist to save
filename	Filename of created PArameter File

Returns

CPL_ERROR_NONE or the relevant CPL error code on error

See also

cpl_dfs_save_image().

The example below shows how to create a PAF from some FITS cards from the file ref_file and QC parameters in a propertylist qclist. Please note that qclist can be used also in calls to cpl_dfs_save_image() and cpl_dfs_save_table(). Error handling is omitted for brevity:

```
const char pafcopy[] = "^(DATE-OBS|ARCFILE|ESO TPL ID|ESO DET DIT|MJD-OBS)$";
cpl_propertylist * paflist = cpl_propertylist_load_regexp(ref_file, 0,
                                                          pafcopy, 0);
cpl_propertylist_append(paflist, qclist);
cpl_dfs_save_paf("IIINSTRUMENT", "rrrecipe", paflist, "rrrecipe.paf");
cpl_propertylist_delete(paflist);
```

References cpl_ensure_code, CPL_ERROR_FILE_IO, cpl_error_get_code(), CPL_ERROR_NONE, and CPL_E↩RROR_NULL_INPUT.

4.4.4.3 cpl_dfs_save_propertylist()

```
cpl_error_code cpl_dfs_save_propertylist (
             cpl_frameset * allframes,
             cpl_propertylist * header,
             const cpl_parameterlist * parlist,
             const cpl_frameset * usedframes,
             const cpl_frame * inherit,
             const char * recipe,
             const cpl_propertylist * applist,
             const char * remregexp,
             const char * pipe_id,
             const char * filename )
```

Save a propertylist as a DFS-compliant pipeline product.

Parameters

allframes	The list of input frames for the recipe
header	NULL, or filled with properties written to product header
parlist	The list of input parameters
usedframes	The list of raw/calibration frames used for this product
inherit	NULL or product frames inherit their header from this frame
recipe	The recipe name
applist	Propertylist to append to primary header, w. PRO.CATG
remregexp	Optional regexp of properties not to put in main header
pipe_id	PACKAGE "/" PACKAGE_VERSION
filename	Filename of created product

Note

remregexp may be NULL

Returns

CPL_ERROR_NONE or the relevant CPL error code on error

See also

cpl_dfs_save_image(), cpl_propertylist_save().

The FITS header of the created product is created from the provided applist and the cards copied by cpl_dfs_setup_product_header(), with exception of the cards whose keys match the provided remregexp.

The FITS data unit will be empty.

References CPL_ERROR_NONE, and CPL_TYPE_INVALID.

4.4.4.4 cpl_dfs_save_table()

```
cpl_error_code cpl_dfs_save_table (
             cpl_frameset * allframes,
             cpl_propertylist * header,
             const cpl_parameterlist * parlist,
             const cpl_frameset * usedframes,
             const cpl_frame * inherit,
             const cpl_table * table,
             const cpl_propertylist * tablelist,
             const char * recipe,
             const cpl_propertylist * applist,
             const char * remregexp,
             const char * pipe_id,
             const char * filename )
```

Save a table as a DFS-compliant pipeline product.

Parameters

allframes	The list of input frames for the recipe
header	NULL, or filled with properties written to product header
parlist	The list of input parameters
usedframes	The list of raw/calibration frames used for this product
inherit	NULL or product frames inherit their header from this frame
table	The table to be saved
tablelist	Optional propertylist to use in table extension or NULL
recipe	The recipe name
applist	Propertylist to append to primary header, w. PRO.CATG
remregexp	Optional regexp of properties not to put in main header
pipe_id	PACKAGE "/" PACKAGE_VERSION
filename	Filename of created product

Returns

CPL_ERROR_NONE or the relevant CPL error code on error

See also

cpl_dfs_save_image(), cpl_table_save().

References cpl_ensure_code, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and CPL_TYPE_INVALID.

4.4.4.5 cpl_dfs_setup_product_header()

```
cpl_error_code cpl_dfs_setup_product_header (
             cpl_propertylist * header,
             const cpl_frame * product_frame,
             const cpl_frameset * framelist,
             const cpl_parameterlist * parlist,
             const char * recid,
             const char * pipeline_id,
             const char * dictionary_id,
             const cpl_frame * inherit_frame )
```

Add product keywords to a pipeline product property list.

Parameters

header	Property list where keywords must be written
product_frame	Frame describing the product
framelist	List of frames including all input frames
parlist	Recipe parameter list
recid	Recipe name
pipeline_id	Pipeline unique identifier
dictionary_id	PRO dictionary identifier
inherit_frame	Frame from which header information is inherited

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	An input pointer is NULL.
CPL_ERROR_DATA_NOT_FOUND	The input *framelist* contains no input frames or a frame in the input *framelist* does not specify a file. In the former case the string "↩ Empty set-of-frames" is appended to the error message returned by cpl_error_get_message().
CPL_ERROR_ILLEGAL_INPUT	The product frame is not tagged or not grouped as CPL_FRA↩ ME_GROUP_PRODUCT. A specified *inherit_frame* doesn't belong to the input frame list, or it is not in FITS format.
CPL_ERROR_FILE_NOT_FOUND	A frame in the input *framelist* specifies a non-existing file.
CPL_ERROR_BAD_FILE_FORMAT	A frame in the input *framelist* specifies an invalid file.

This function checks the *header* associated to a pipeline product, to ensure that it is DICB compliant. In particular, this function does the following:

1. Selects a reference frame from which the primary and secondary keyword information is inherited. The primary information is contained in the FITS keywords ORIGIN, TELESCOPE, INSTRUME, OBJECT, RA, DEC, EPOCH, EQUINOX, RADECSYS, DATE-OBS, MJD-OBS, UTC, LST, PI-COI, OBSERVER, while the secondary information is contained in all the other keywords. If the *inherit_frame* is just a NULL pointer, both primary and secondary information is inherited from the first frame in the input framelist with group CPL_FRAME_GROUP_RAW, or if no such frames are present the first frame with group CPL_FRAME↩ _GROUP_CALIB. If *inherit_frame* is non-NULL, the secondary information is inherited from *inherit_frame* instead.
2. Copy to *header*, if they are present, the following primary FITS keywords from the first input frame in the *framelist:* ORIGIN, TELESCOPE, INSTRUME, OBJECT, RA, DEC, EPOCH, EQUINOX, RADESYS, DA↩ TE-OBS, MJD-OBS, UTC, LST, PI-COI, OBSERVER. If those keywords are already present in the *header* property list, they are overwritten only in case they have the same type. If any of these keywords are present with an unexpected type, a warning is issued, but the keywords are copied anyway (provided that the above conditions are fulfilled), and no error is set.
3. Copy all the HIERARCH.ESO._ keywords from the primary FITS header of the *inherit_frame* in *framelist*, with the exception of the HIERARCH.ESO.DPR._, and of the .PRO._ and .DRS._ keywords if the *inherit_frame* is a calibration. If those keywords are already present in *header*, they are overwritten.
4. If found, remove the HIERARCH.ESO.DPR._ keywords from *header*.
5. If found, remove the ARCFILE and ORIGFILE keywords from *header*.
6. Add to *header* the following mandatory keywords from the PRO dictionary: PIPEFILE, PRO.DID, PRO.R↩ EC1.ID, PRO.REC1.DRS.ID, PRO.REC1.PIPE.ID, and PRO.CATG. If those keywords are already present in *header*, they are overwritten. The keyword PRO.CATG is always set identical to the tag in *product_frame*.
7. Only if missing, add to *header* the following mandatory keywords from the PRO dictionary: PRO.TYPE, PR↩ O.TECH, and PRO.SCIENCE. The keyword PRO.TYPE will be set to "REDUCED". If the keyword DPR.TECH is found in the header of the first frame, PRO.TECH is given its value, alternatively if the keyword PRO.TECH is found it is copied instead, and if all fails the value "UNDEFINED" is set. Finally, if the keyword DPR.CATG is found in the header of the first frame and is set to "SCIENCE", the boolean keyword PRO.SCIENCE will be set to "true", otherwise it will be copied from an existing PRO.SCIENCE keyword, while it will be set to "false" in all other cases.
8. Check the existence of the keyword PRO.DATANCOM in *header*. If this keyword is missing, one is added, with the value of the total number of raw input frames.
9. Add to *header* the keywords PRO.REC1.RAW1.NAME, PRO.REC1.RAW1.CATG, PRO.REC1.CAL1.NAME, PRO.REC1.CAL1.CATG, to describe the content of the input set-of-frames.

See the DICB PRO dictionary to have details on the mentioned PRO keywords.

Note

Non-FITS files are handled as files with an empty FITS header.

References cpl_ensure_code, CPL_ERROR_NULL_INPUT, CPL_TYPE_DOUBLE, and CPL_TYPE_STRING.

4.4.4.6 cpl_dfs_sign_products()

```
cpl_error_code cpl_dfs_sign_products (
            const cpl_frameset * set,
            unsigned int flags )
```

Update DFS and DICB required header information of product frames.

Parameters

set	The frameset from which the product frames are taken.
flags	Bit mask for selecting the digital signatures to be written.

Returns

The function returns `CPL_ERROR_NONE` on success, or an appropriate CPL error code otherwise.

Note

Each product frame must correspond to a FITS file created with a CPL FITS saving function.

The function takes all frames marked as products from the input frameset *set*. For each product the header information `PIPEFILE` is updated unconditionally. In addition, depending on the bit mask *flags*, the `DATAMD5` data hash and/or the standard FITS checksums are computed and written to the product header. If a digital signature is not selected by *flags* when the function is called, its corresponding header keyword(s) are removed from the product frame.

Errors

CPL_ERROR_NULL_INPUT	An input pointer is `NULL`.
CPL_ERROR_DATA_NOT_FOUND	The input *framelist* contains a frame of type product with a missing filename.
CPL_ERROR_BAD_FILE_FORMAT	The input *framelist* contains a frame of type product without a FITS card with key 'DATAMD5'.
CPL_ERROR_FILE_IO	The input *framelist* contains a frame of type product for which the FITS card with key 'DATAMD5' could not be updated.

References CPL_DFS_SIGNATURE_NONE, cpl_ensure_code, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_frame_get_group(), CPL_FRAME_GROUP_PRODUCT, cpl_frameset_is_empty(), cpl_frameset_iterator_get_const(), and cpl_frameset_iterator_new().

4.4.4.7 cpl_dfs_update_product_header()

```
cpl_error_code cpl_dfs_update_product_header (
            cpl_frameset * self )
```

Perform any DFS-compliancy required actions (DATAMD5/PIPEFILE update)

Parameters

self	The list of frames with FITS products created by the recipe

Returns

CPL_ERROR_NONE or the relevant CPL error code on error

Note

Each product frame must correspond to a FITS file created with a CPL FITS saving function.

Errors

CPL_ERROR_NULL_INPUT	An input pointer is `NULL`.
CPL_ERROR_DATA_NOT_FOUND	The input *framelist* contains a frame of type product with a missing filename.
CPL_ERROR_BAD_FILE_FORMAT	The input *framelist* contains a frame of type product without a FITS card with key 'DATAMD5'.
CPL_ERROR_FILE_IO	The input *framelist* contains a frame of type product for which the FITS card with key 'DATAMD5' could not be updated.

4.5 DICB specific property functionality

4.5.1 Detailed Description

Synopsis:

```
#include "cpl_property_dicb.h"
```

4.6 Error handling

Macros

- #define cpl_ensure(BOOL, ERRCODE, RETURN)

  Set an error code and return iff a boolean expression is false.
- #define cpl_ensure_code(BOOL, ERRCODE)

  Set an error code and return it iff a boolean expression is false.
- #define cpl_error_ensure(CONDITION, CODE, ACTION, ...)

  Generic error handling macro.
- #define CPL_ERROR_MAX_MESSAGE_LENGTH 256

  The maximum length of a CPL error message.
- #define cpl_error_set(function, code)

  Set CPL error code, function name, source file and line number where an error occurred.
- #define cpl_error_set_message(function, code, ...)

  Set CPL error code, function name, source file and line number where an error occurred along with a text message.
- #define cpl_error_set_where(function)

  Propagate a CPL-error to the current location.
- #define CPL_HAVE_VA_ARGS

  Flag to indicate support for variadic macros.

Typedefs

- typedef enum _cpl_error_code_ cpl_error_code

  The cpl_error_code type definition.

Enumerations

- enum _cpl_error_code_ {
  CPL_ERROR_NONE = 0,
  CPL_ERROR_UNSPECIFIED = 1,
  CPL_ERROR_HISTORY_LOST,
  CPL_ERROR_DUPLICATING_STREAM,
  CPL_ERROR_ASSIGNING_STREAM,
  CPL_ERROR_FILE_IO,
  CPL_ERROR_BAD_FILE_FORMAT,
  CPL_ERROR_FILE_ALREADY_OPEN,
  CPL_ERROR_FILE_NOT_CREATED,
  CPL_ERROR_FILE_NOT_FOUND,
  CPL_ERROR_DATA_NOT_FOUND,
  CPL_ERROR_ACCESS_OUT_OF_RANGE,
  CPL_ERROR_NULL_INPUT,
  CPL_ERROR_INCOMPATIBLE_INPUT,
  CPL_ERROR_ILLEGAL_INPUT,
  CPL_ERROR_ILLEGAL_OUTPUT,
  CPL_ERROR_UNSUPPORTED_MODE,
  CPL_ERROR_SINGULAR_MATRIX,
  CPL_ERROR_DIVISION_BY_ZERO,
  CPL_ERROR_TYPE_MISMATCH,
  CPL_ERROR_INVALID_TYPE,
  CPL_ERROR_CONTINUE,
  CPL_ERROR_NO_WCS,
  CPL_ERROR_EOL }

  Available error codes.

Functions

- cpl_error_code cpl_error_get_code (void)

  Get the last cpl_error_code set.
- const char * cpl_error_get_file (void)

  Get the source code file name where the last CPL error occurred.
- const char * cpl_error_get_function (void)

  Get the function name where the last CPL error occurred.
- unsigned cpl_error_get_line (void)

  Get the line number where the last CPL error occurred.
- const char * cpl_error_get_message (void)

  Get the text message of the current CPL error.
- const char * cpl_error_get_message_default (cpl_error_code code)

  Return the standard CPL error message of the current CPL error.
- const char * cpl_error_get_where (void)

  Get function name, source file and line number where the last CPL error occurred.

4.6.1 Detailed Description

This module provides functions to maintain the *cpl_error_code* set by any CPL function, similarly to what is done with the *errno* variable of the standard C library. The following guidelines are respected:

- If no error occurs in a CPL function the *cpl_error_code* will remain unchanged.
- If an error occurs in a CPL function, a new CPL error is set, causing the *cpl_error_code* to be modified to the new error.

A *cpl_error_code* equal to the enumeration constant `CPL_ERROR_NONE` would indicate no error condition. Note, however, that the *cpl_error_code* is only set when an error occurs, and it is not reset by successful function calls. For this reason it may be appropriate in some cases to reset the *cpl_error_code* using the function `cpl_error_reset()`. The *cpl_error_code* set by a CPL function can be obtained by calling the function `cpl_error_get_code()`, but functions of type *cpl_error_code* would not only return this code directly, but would also return `CPL_ERROR_NONE` in case of success. Other CPL functions return zero on success, or a non-zero value to indicate a change of the *cpl_error_code*, while CPL functions returning a pointer would flag an error by returning a `NULL`.

To each *cpl_error_code* is associated a standard error message, that can be obtained by calling the function `cpl_error_get_message()`. Conventionally, no CPL function will ever display any error message, leaving to the caller the decision of how to handle a given error condition. A call to the function `cpl_error_get_function()` would return the name of the function where the error occurred, and the functions `cpl_error_get_file()` and `cpl_error_get_line()` would also return the name of the source file containing the function code, and the line number where the error occurred. The function `cpl_error_get_where()` would gather all this items together, in a colon-separated string.

Synopsis:

```
#include <cpl_error.h>
```

4.6.2 Macro Definition Documentation

4.6.2.1 cpl_ensure

```
#define cpl_ensure(
               BOOL,
               ERRCODE,
               RETURN )
```

Set an error code and return iff a boolean expression is false.

Parameters

BOOL	The boolean to evaluate
ERRCODE	The error code to set
RETURN	The value to return

Note

This macro will cause a return from its calling function. If ERRCODE equals CPL_ERROR_NONE, the error-code is set to CPL_ERROR_UNSPECIFIED. The boolean is always evaluated (unlike assert()). This macro may not be used in inline'd functions.

See also

cpl_error_set()

4.6.2.2 cpl_ensure_code

```
#define cpl_ensure_code(
               BOOL,
               ERRCODE )
```

Set an error code and return it iff a boolean expression is false.

Parameters

BOOL	The boolean to evaluate
ERRCODE	The error code to set and return

See also

cpl_ensure()

4.6.2.3 cpl_error_ensure

```
#define cpl_error_ensure(
               CONDITION,
```

```
CODE,
ACTION,
...  )
```

Generic error handling macro.

Parameters

CONDITION	The condition to check
CODE	The CPL error code to set if *CONDTION* is non-zero
ACTION	A statement that is executed iff the *CONDITION* evaluates to non-zero.
...	A printf-style message for cpl_error_set_message().

See also

cpl_error_set_message()

Note

This macro should not be used directly. It is defined only to support user-defined error handling macros. If CODE equals CPL_ERROR_NONE, a CPL error with code CPL_ERROR_UNSPECIFIED is created. The message may be a printf-like format string supplied with arguments unless variadic macros are not supported in which case only a (non-format) string is accepted. The provided *CODE*, *ACTION* and ***VA_ARGS*** are evaluated/executed only if the *CONDITION* evaluates to false (zero). The *ACTION break* is unsupported (it currently causes the execution to continue after *cpl_error_ensure()*). Some of the below examples use a *goto* statement to jump to a single cleanup label, assumed to be located immediately before the function's unified cleanup-code and return point. This error-handling scheme is reminiscent of using exceptions in languages that support exceptions (C++, Java, ...). The use of *goto* and a single clean-up label per function "if the error-handling code is non- trivial, and if errors can occur in several places" is sanctioned by Kernigan & Richie: "The C Programming Language". For any other purpose *goto* should be avoided.

Useful definitions might include

```
#define assure(BOOL, CODE)                                              \
  cpl_error_ensure(BOOL, CODE, goto cleanup, " ")
#define  check(CMD)                                                     \
  cpl_error_ensure((CMD, cpl_error_get_code() == CPL_ERROR_NONE),       \
                   cpl_error_get_code(), goto cleanup, " ")
#define assert(BOOL)                                                    \
  cpl_error_ensure(BOOL, CPL_ERROR_UNSPECIFIED, goto cleanup,           \
                   "Internal error, please report to " PACKAGE_BUGREPORT)
```

or (same as above, but including printf-style error messages)

```
#define assure(BOOL, CODE, ...)                                         \
  cpl_error_ensure(BOOL, CODE, goto cleanup, __VA_ARGS__)
#define  check(CMD, ...)                                                \
  cpl_error_ensure((CMD, cpl_error_get_code() == CPL_ERROR_NONE),       \
                   cpl_error_get_code(), goto cleanup, __VA_ARGS__)
#define assert(BOOL, ...)                                               \
  cpl_error_ensure(BOOL, CPL_ERROR_UNSPECIFIED, goto cleanup,           \
                   "Internal error, please report to " PACKAGE_BUGREPORT    \
                   " " __VA_ARGS__)
                   / *  Assumes that PACKAGE_BUGREPORT
                        contains no formatting special characters  * /
```

or

```
#define skip_if(BOOL)                                                   \
  cpl_error_ensure(BOOL, cpl_error_get_code() != CPL_ERROR_NONE ?       \
                   cpl_error_get_code() : CPL_ERROR_UNSPECIFIED,        \
                   goto cleanup, " ")
```

The check macros in the examples above can be used to check a command which sets the cpl_error_code in case of failure (or, by use of a comma expression, a longer sequence of such commands):

```
check(
    (x = cpl_table_get_int(table, "x", 0, NULL),
     y = cpl_table_get_int(table, "y", 0, NULL),
     z = cpl_table_get_int(table, "z", 0, NULL)),
    "Error reading wavelength catalogue");
```

4.6.2.4 CPL_ERROR_MAX_MESSAGE_LENGTH

```
#define CPL_ERROR_MAX_MESSAGE_LENGTH 256
```

The maximum length of a CPL error message.

See also

cpl_error_get_message()

4.6.2.5 cpl_error_set

```
#define cpl_error_set(
              function,
              code )
```

Set CPL error code, function name, source file and line number where an error occurred.

Parameters

function	Character string with function name, cpl_func
code	Error code

Returns

The specified error code.

Note

If code is CPL_ERROR_NONE, nothing is done (and code is returned), if code is CPL_ERROR_HISTORY↩ _LOST (but avoid that) then CPL_ERROR_UNSPECIFIED is set and returned.

4.6.2.6 cpl_error_set_message

```
#define cpl_error_set_message(
              function,
              code,
              ...  )
```

Set CPL error code, function name, source file and line number where an error occurred along with a text message.

Parameters

function	Character string with function name, cpl_func
code	Error code
...	Variable argument list with message

Returns

The CPL error code

See also

cpl_error_set()

Note

The message is ignored if it is NULL, empty, or consists of a single ' ' (space). Otherwise, the user supplied message is appended to the default message using ': ' (colon+space) as delimiter. If the CPL-based application may use variadic macros, the message may be a printf-style format string supplied with matching arguments. If variadic macros are not allowed (e.g. when compiling with gcc -ansi) only a non-format string is accepted. Please be aware that the format string should always be a string literal, otherwise the code may be vulnerable to the socalled 'format string exploit'. Sufficiently recent versions of gcc supports the option -Wformat-security, which tries to warn of this issue. If variadic macros are not supported, a printf-style message can be created prior to the call to cpl_error_set_message(), as in the below example.

Examples of usage:

```
if (image == NULL) {
    return cpl_error_set_message(cpl_func, CPL_ERROR_NULL_INPUT,
                                 "Image number %d is NULL", number);
}
if (error_string != NULL) {
    return cpl_error_set_message(cpl_func, CPL_ERROR_ILLEGAL_INPUT,
                                 "%s", error_string);
}
```

Example of usage if and only if variadic macros are unavaiable (e.g. when compiling with gcc -ansi):

```
if (image == NULL) {
    char * my_txt = cpl_sprintf("Image number %d is NULL", number);
    const cpl_error_code my_code =
        cpl_error_set_message(cpl_func, CPL_ERROR_NULL_INPUT, my_txt);
    cpl_free(my_txt);
    return my_code;
}
```

4.6.2.7 cpl_error_set_where

```
#define cpl_error_set_where(
                function )
```

Propagate a CPL-error to the current location.

Parameters

function	Character string with function name, cpl_func

Returns

The preexisting CPL error code (possibly CPL_ERROR_NONE).

See also

cpl_error_set()

Note

If no error exists, nothing is done and CPL_ERROR_NONE is returned

If a CPL error already exists, this function creates a new CPL error with the preexisting CPL error code and the current location.

In a function of type cpl_error_code an error can be propagated with:

```
return cpl_error_set_where(cpl_func);
```

4.6.2.8 CPL_HAVE_VA_ARGS

```
#define CPL_HAVE_VA_ARGS
```

Flag to indicate support for variadic macros.

See also

cpl_error_set_message()

Note

Unless already defined, tries to detect whether variadic macros are supported, typically at compile-time of a CPL-based application. This check, which is hardly robust, should support

- disabling of variadic macros when compiling with *gcc -ansi*
- enabling them when compiling with a C99 compliant compiler, such as *gcc -std=c99*

4.6.3 Typedef Documentation

4.6.3.1 cpl_error_code

```
typedef enum _cpl_error_code_ cpl_error_code
```

The cpl_error_code type definition.

4.6.4 Enumeration Type Documentation

4.6.4.1 _cpl_error_code_

```
enum _cpl_error_code_
```

Available error codes.

CPL_ERROR_NONE is equal to zero and compares to less than all other error codes.

All error codes are guaranteed to be less than CPL_ERROR_EOL (End Of List). CPL_ERROR_EOL allows user defined error codes to be added in this fashion:

```
const int my_first_error_code  = 0 + CPL_ERROR_EOL;
const int my_second_error_code = 1 + CPL_ERROR_EOL;
const int my_third_error_code  = 2 + CPL_ERROR_EOL;
if (is_connected() == CPL_FALSE) {
   return cpl_error_set_message(cpl_func, my_first_error_code, "No "
                                "connection to host %s (after %u tries)",
                                hostname, max_attempts);
}
```

Extensive use of user defined error codes should be avoided. Instead a request of new CPL error codes should be emailed to cpl-help@eso.org.

Enumerator

CPL_ERROR_NONE	No error condition
CPL_ERROR_UNSPECIFIED	An unspecified error
CPL_ERROR_HISTORY_LOST	The actual CPL error has been lost. Do not use to create a CPL error
CPL_ERROR_DUPLICATING_STREAM	Could not duplicate output stream
CPL_ERROR_ASSIGNING_STREAM	Could not associate a stream with a file descriptor
CPL_ERROR_FILE_IO	Permission denied
CPL_ERROR_BAD_FILE_FORMAT	Input file had not the expected format
CPL_ERROR_FILE_ALREADY_OPEN	Attempted to open a file twice
CPL_ERROR_FILE_NOT_CREATED	Could not create a file
CPL_ERROR_FILE_NOT_FOUND	A specified file or directory was not found
CPL_ERROR_DATA_NOT_FOUND	Data searched within a valid object were not found
CPL_ERROR_ACCESS_OUT_OF_RANGE	Data were accessed beyond boundaries
CPL_ERROR_NULL_INPUT	A NULL pointer was found where a valid pointer was expected
CPL_ERROR_INCOMPATIBLE_INPUT	Data that had to be processed together did not match
CPL_ERROR_ILLEGAL_INPUT	Illegal values were detected
CPL_ERROR_ILLEGAL_OUTPUT	A given operation would have generated an illegal object
CPL_ERROR_UNSUPPORTED_MODE	The requested functionality is not supported
CPL_ERROR_SINGULAR_MATRIX	Could not invert a matrix
CPL_ERROR_DIVISION_BY_ZERO	Attempted to divide a number by zero
CPL_ERROR_TYPE_MISMATCH	Data were not of the expected type
CPL_ERROR_INVALID_TYPE	Data type was unsupported or invalid
CPL_ERROR_CONTINUE	An iterative process did not converge
CPL_ERROR_NO_WCS	The WCS functionalities are missing
CPL_ERROR_EOL	To permit extensibility of error handling. It is a coding error to use this within CPL itself!

4.6.5 Function Documentation

4.6.5.1 cpl_error_get_code()

```
cpl_error_code cpl_error_get_code (
            void  )
```

Get the last *cpl_error_code* set.

Returns

cpl_error_code of last occurred CPL error.

Get *cpl_error_code* of last occurred error.

References CPL_ERROR_NONE.

Referenced by cpl_apertures_extract_window(), cpl_bivector_interpolate_linear(), cpl_detector_interpolate_↩ rejected(), cpl_dfs_save_paf(), cpl_frameset_dump(), cpl_frameset_extract(), cpl_frameset_labelise(), cpl_image↩ _abs_create(), cpl_image_add_scalar_create(), cpl_image_collapse_create(), cpl_image_divide_scalar_create(), cpl_image_exponential_create(), cpl_image_fft(), cpl_image_get_fwhm(), cpl_image_get_interpolated(), cpl_↩ image_logarithm_create(), cpl_image_multiply_scalar_create(), cpl_image_power_create(), cpl_image_subtract↩ _scalar_create(), cpl_image_warp_polynomial(), cpl_imagelist_add_scalar(), cpl_imagelist_divide_scalar(), cpl_↩ imagelist_dump_structure(), cpl_imagelist_dump_window(), cpl_imagelist_exponential(), cpl_imagelist_logarithm(), cpl_imagelist_multiply_scalar(), cpl_imagelist_normalise(), cpl_imagelist_power(), cpl_imagelist_subtract_scalar(), cpl_imagelist_threshold(), cpl_sprintf(), cpl_vector_copy(), and cpl_vector_fill_kernel_profile().

4.6.5.2 cpl_error_get_file()

```
const char* cpl_error_get_file (
            void  )
```

Get the source code file name where the last CPL error occurred.

Returns

Name of source file name where the last CPL error occurred.

Get the source code file name where the last CPL error occurred.

4.6.5.3 cpl_error_get_function()

```
const char* cpl_error_get_function (
            void  )
```

Get the function name where the last CPL error occurred.

Returns

Identifier string of the function name where the last CPL error occurred.

Get the function name where the last CPL error occurred.

4.6.5.4 cpl_error_get_line()

```
unsigned cpl_error_get_line (
            void  )
```

Get the line number where the last CPL error occurred.

Returns

Line number of the source file where the last CPL error occurred.

Get the line number of the source file where the last CPL error occurred.

4.6.5.5 cpl_error_get_message()

```
const char* cpl_error_get_message (
            void  )
```

Get the text message of the current CPL error.

Returns

The text message of the current CPL error.

See also

cpl_error_get_message_default(), cpl_error_set_message()

If the *cpl_error_code* is equal to `CPL_ERROR_NONE`, an empty string is returned. Otherwise, the message is the default message for the current CPL error code, possibly extended with a custom message supplied when the error was set.

4.6.5.6 cpl_error_get_message_default()

```
const char* cpl_error_get_message_default (
            cpl_error_code code )
```

Return the standard CPL error message of the current CPL error.

Parameters

code	The error code of the current CPL error

Returns

The standard CPL error message of the current CPL error

References CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ASSIGNING_STREAM, CPL_ERROR_↩
BAD_FILE_FORMAT, CPL_ERROR_CONTINUE, CPL_ERROR_DATA_NOT_FOUND, CPL_ERROR_DIVISIO↩

N_BY_ZERO, CPL_ERROR_DUPLICATING_STREAM, CPL_ERROR_EOL, CPL_ERROR_FILE_ALREADY_O↩
PEN, CPL_ERROR_FILE_IO, CPL_ERROR_FILE_NOT_CREATED, CPL_ERROR_FILE_NOT_FOUND, CPL_↩
ERROR_HISTORY_LOST, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_ILLEGAL_OUTPUT, CPL_ERROR_I↩
NCOMPATIBLE_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NO_WCS, CPL_ERROR_NONE, CPL_↩
ERROR_NULL_INPUT, CPL_ERROR_SINGULAR_MATRIX, CPL_ERROR_TYPE_MISMATCH, CPL_ERROR↩
_UNSPECIFIED, and CPL_ERROR_UNSUPPORTED_MODE.

4.6.5.7 cpl_error_get_where()

```
const char* cpl_error_get_where (
            void  )
```

Get function name, source file and line number where the last CPL error occurred.

Returns

String containing function name, source file and line number separated by colons (:).

Get where the last CPL error occurred in the form `function_name:source_file:line_number`

4.7 FFTW wrappers

Typedefs

- typedef enum _cpl_fft_mode_ cpl_fft_mode
  The CPL fft mode.

Enumerations

- enum _cpl_fft_mode_ {
  CPL_FFT_FORWARD,
  CPL_FFT_BACKWARD,
  CPL_FFT_NOSCALE,
  CPL_FFT_FIND_MEASURE,
  CPL_FFT_FIND_PATIENT,
  CPL_FFT_FIND_EXHAUSTIVE }
  The supported values of the CPL fft mode.

Functions

- cpl_error_code cpl_fft_image (cpl_image ∗self, const cpl_image ∗other, cpl_fft_mode mode)
  Perform a FFT operation on an image.
- cpl_error_code cpl_fft_imagelist (cpl_imagelist ∗self, const cpl_imagelist ∗other, cpl_fft_mode mode)
  Perform a FFT operation on the images in an imagelist.

4.7.1 Detailed Description

This module provides FFTW wrappers

Synopsis:

```
#include "cpl_fft.h"
```

4.7.2 Typedef Documentation

4.7.2.1 cpl_fft_mode

```
typedef enum _cpl_fft_mode_ cpl_fft_mode
```

The CPL fft mode.

4.7.3 Enumeration Type Documentation

4.7.3.1 _cpl_fft_mode_

```
enum _cpl_fft_mode_
```

The supported values of the CPL fft mode.

Enumerator

CPL_FFT_FORWARD	The forward transform
CPL_FFT_BACKWARD	The backward transform
CPL_FFT_NOSCALE	Transform without scaling (has no effect on forward transform)
CPL_FFT_FIND_MEASURE	Spend time finding the best transform
CPL_FFT_FIND_PATIENT	Spend more time finding the best transform
CPL_FFT_FIND_EXHAUSTIVE	Spend even more time finding the best transform

4.7.4 Function Documentation

4.7.4.1 cpl_fft_image()

```
cpl_error_code cpl_fft_image (
            cpl_image * self,
            const cpl_image * other,
            cpl_fft_mode mode )
```

Perform a FFT operation on an image.

Parameters

self	Pre-allocated output image to transform to
other	Input image to transform from, use self for in-place transform
mode	CPL_FFT_FORWARD or CPL_FFT_BACKWARD, optionally CPL_FFT_NOSCALE

Returns

CPL_ERROR_NONE or the corresponding _cpl_error_code_ on error

This function performs an FFT on an image, using FFTW. CPL may be configured without this library, in this case an otherwise valid call will set and return the error CPL_ERROR_UNSUPPORTED_MODE.

The input and output images must match in precision level. Integer images are not supported.

In a forward transform the input image may be non-complex. In this case a real-to-complex transform is performed. This will only compute the first nx/2 + 1 columns of the transform. In this transform it is allowed to pass an output image with nx/2 + 1 columns.

Similarly, in a backward transform the output image may be non-complex. In this case a complex-to-real transform is performed. This will only transform the first nx/2 + 1 columns of the input. In this transform it is allowed to pass an input image with nx/2 + 1 columns.

Per default the backward transform scales (divides) the result with the number of elements transformed (i.e. the number of pixels in the result image). This scaling can be turned off with CPL_FFT_NOSCALE.

If many transformations in the same direction are to be done on data of the same size and type, a reduction in the time required to perform the transformations can be achieved by adding the flag CPL_FFT_FIND_MEASURE

to the first transformation. For a larger number of transformations a further reduction may be achived with the flag CPL_FFT_FIND_PATIENT and for an even larger number of transformations a further reduction may be achived with the flag CPL_FFT_FIND_EXHAUSTIVE.

If many transformations are to be done then a reduction in the time required to perform the transformations can be achieved by using cpl_fft_imagelist().

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an image is NULL
- CPL_ERROR_ILLEGAL_INPUT if the mode is illegal
- CPL_ERROR_INCOMPATIBLE_INPUT if the image sizes do not match
- CPL_ERROR_TYPE_MISMATCH if the image types are incompatible with each other or with the transform
- CPL_ERROR_UNSUPPORTED_MODE if FFTW has not been installed

References CPL_ERROR_NONE.

4.7.4.2 cpl_fft_imagelist()

```
cpl_error_code cpl_fft_imagelist (
            cpl_imagelist * self,
            const cpl_imagelist * other,
            cpl_fft_mode mode )
```

Perform a FFT operation on the images in an imagelist.

Parameters

self	Pre-allocated output imagelist to transform to
other	Input imagelist to transform from
mode	CPL_FFT_FORWARD or CPL_FFT_BACKWARD, optionally CPL_FFT_NOSCALE

Returns

CPL_ERROR_NONE or the corresponding _cpl_error_code_ on error

See also

cpl_fft_image()

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NONE, CPL_ERROR_N↩ ULL_INPUT, cpl_imagelist_get(), cpl_imagelist_get_const(), and cpl_imagelist_get_size().

4.8 FITS card related basic routines

4.8.1 Detailed Description

Synopsis:

```
#include "cpl_fits_card.h"
```

4.9 FITS related basic routines

Typedefs

- typedef enum _cpl_fits_mode_ cpl_fits_mode

  The CPL fits mode. It is a bit field.

Enumerations

- enum _cpl_fits_mode_ {
  CPL_FITS_STOP_CACHING,
  CPL_FITS_START_CACHING,
  CPL_FITS_RESTART_CACHING,
  CPL_FITS_ONE }

  The values of the CPL fits mode. The values can be combined with bitwise or.

Functions

- cpl_size cpl_fits_count_extensions (const char ∗filename)

  Get the number of extensions contained in a FITS file.
- cpl_size cpl_fits_find_extension (const char ∗filename, const char ∗extname)

  Get the place of a given extension in a FITS file.
- int cpl_fits_get_extension_nb (const char ∗filename, const char ∗extname)

  Get the place of a given extension in a FITS file.
- cpl_fits_mode cpl_fits_get_mode (void)

  Get the FITS I/O mode.
- int cpl_fits_get_nb_extensions (const char ∗filename)

  Get the number of extensions contained in a FITS file.
- cpl_error_code cpl_fits_set_mode (cpl_fits_mode mode)

  Set the FITS I/O mode.

4.9.1 Detailed Description

This module provides functions to get basic information on FITS files

Synopsis:

```
#include "cpl_fits.h"
```

4.9.2 Typedef Documentation

4.9.2.1 cpl_fits_mode

```
typedef enum _cpl_fits_mode_ cpl_fits_mode
```

The CPL fits mode. It is a bit field.

4.9.3 Enumeration Type Documentation

4.9.3.1 _cpl_fits_mode_

```
enum _cpl_fits_mode_
```

The values of the CPL fits mode. The values can be combined with bitwise or.

Enumerator

CPL_FITS_STOP_CACHING	Stop the caching of open FITS files
CPL_FITS_START_CACHING	Start the caching of open FITS files
CPL_FITS_RESTART_CACHING	Restart the caching of open FITS files
CPL_FITS_ONE	Apply the mode change only to the current thread

4.9.4 Function Documentation

4.9.4.1 cpl_fits_count_extensions()

```
cpl_size cpl_fits_count_extensions (
            const char * filename )
```

Get the number of extensions contained in a FITS file.

Parameters

filename	The file name

Returns

The number of extensions or -1 in case of error

Note

For a valid fits file without extensions zero is returned

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if the input pointer is NULL
- CPL_ERROR_FILE_IO If the FITS file could not be opened
- CPL_ERROR_BAD_FILE_FORMAT if the input FITS file is otherwise invalid

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_fits_get_nb_extensions(), and cpl_frame_get_nextensions().

4.9.4.2 cpl_fits_find_extension()

```
cpl_size cpl_fits_find_extension (
             const char * filename,
             const char * extname )
```

Get the place of a given extension in a FITS file.

Parameters

filename	The file name
extname	The extension name

Returns

The extension number, 0 if not found or -1 on error

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_FILE_IO if the file is not FITS

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_fits_get_extension_nb(), and cpl_multiframe_append_datagroup().

4.9.4.3 cpl_fits_get_extension_nb()

```
int cpl_fits_get_extension_nb (
             const char * filename,
             const char * extname )
```

Get the place of a given extension in a FITS file.

Parameters

filename	The file name
extname	The extension name

Returns

the extension place or -1 in case of error

See also

cpl_fits_find_extension

Deprecated Replace this call with cpl_fits_find_extension().

References cpl_fits_find_extension().

4.9.4.4 cpl_fits_get_mode()

```
cpl_fits_mode cpl_fits_get_mode (
            void  )
```

Get the FITS I/O mode.

Returns

One of: CPL_FITS_STOP_CACHING, CPL_FITS_START_CACHING

See also

cpl_fits_set_mode()

References CPL_FITS_START_CACHING, and CPL_FITS_STOP_CACHING.

4.9.4.5 cpl_fits_get_nb_extensions()

```
int cpl_fits_get_nb_extensions (
            const char * filename )
```

Get the number of extensions contained in a FITS file.

Parameters

filename	The file name

Returns

the number of extensions or -1 in case of error

See also

cpl_fits_count_extensions()

Deprecated Replace this call with cpl_fits_count_extensions().

References cpl_fits_count_extensions().

4.9.4.6 cpl_fits_set_mode()

```
cpl_error_code cpl_fits_set_mode (
            cpl_fits_mode mode )
```

Set the FITS I/O mode.

Parameters

mode	The FITS I/O mode: CPL_FITS_STOP_CACHING, CPL_FITS_START_CACHING, CPL_FITS_RESTART_CACHING

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_init() to control the FITS I/O mode when CPL is started

Normally when a FITS file is processed with a CPL call the file is openened and closed during that call. However repeated calls on FITS data with many extensions causes the FITS headers to be parsed many times which can lead to a significant performance penalty. If instead this function is called with CPL_FITS_START_CACHING, CPL will use internal storage to keep the FITS files open between calls and only close them when the FITS I/O mode is changed (or cpl_end() is called).

If a CPL function that creates a FITS file is called any previously opened handles to that file are closed. If a CPL function that appends to a FITS file is called any previously opened read-only handles to that file are closed. If a CPL function that reads from a FITS file is called any previously opened read/write-handle to that file is used for the read. Any additional concurrent reads causes the file to also be opened for reading. This means that there is also a performance gain when alternating between appending to and reading from the same file. This optimized I/O mode cannot be used if the CPL application accesses a FITS file without using CPL. An incomplete list of such access is: Calls to rename(), remove(), unlink(), fopen() and access via another process started with for example system() or popen().

The caching of opened FITS files may be used in a multi-threaded environment to the extent that the underlying FITS library (CFITSIO) supports it. One implication of this is that multiple threads may only call CPL FITS saving functions on the same file using proper synchronization such as the OpenMP 'ordered' construct. CPL makes no attempt to verify that a CPL based application performs thread parallel FITS I/O correctly.

The mode CPL_FITS_STOP_CACHING causes all cached filed to be closed. Beware that this can cause an I/O error for a file that has otherwise not been accessed for some time.

Since CPL_FITS_STOP_CACHING closes all cached FITS files, the caller must ensure that this does not interfere with the concurrent use of those same files in another thread.

The mode CPL_FITS_RESTART_CACHING has the same effect as a call with CPL_FITS_STOP_CACHING followed by a call with CPL_FITS_START_CACHING.

The modes CPL_FITS_RESTART_CACHING and CPL_FITS_ONE may be combined. This causes all files cached by the calling thread to be closed. The caching remains active (for all threads), so subsequently opened files will be cached.

CPL_FITS_RESTART_CACHING can be used after an external modification of a FITS file also cached by CPL, thus allowing the caching to work together with the above mentioned external access to the same FITS files.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_ILLEGAL_INPUT if the mode is zero
- CPL_ERROR_UNSUPPORTED_MODE if the mode is not supported
- CPL_ERROR_INVALID_TYPE if mode is 1, e.g. due to a logical or (||) of the allowed options.
- CPL_ERROR_BAD_FILE_FORMAT if the I/O caused by CPL_FITS_STOP_CACHING failed

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERR↩OR_UNSUPPORTED_MODE, CPL_FITS_ONE, CPL_FITS_RESTART_CACHING, CPL_FITS_START_CACHI↩NG, and CPL_FITS_STOP_CACHING.

Referenced by cpl_end().

4.10 Filters

Typedefs

- typedef enum _cpl_border_mode_ cpl_border_mode

  The border mode type.
- typedef enum _cpl_filter_mode_ cpl_filter_mode

  The filter mode type.

Enumerations

- enum _cpl_border_mode_ {
  CPL_BORDER_FILTER,
  CPL_BORDER_ZERO,
  CPL_BORDER_CROP,
  CPL_BORDER_NOP,
  CPL_BORDER_COPY }

  These are the supported border modes. For a kernel of width 2n+1, the n left- and rightmost image/mask columns do not have elements for the whole kernel. The same holds for the top and bottom image/mask rows. The border mode defines the filtering of such border pixels.
- enum _cpl_filter_mode_ {
  CPL_FILTER_EROSION,
  CPL_FILTER_DILATION,
  CPL_FILTER_OPENING,
  CPL_FILTER_CLOSING,
  CPL_FILTER_LINEAR,
  CPL_FILTER_LINEAR_SCALE,
  CPL_FILTER_AVERAGE,
  CPL_FILTER_AVERAGE_FAST,
  CPL_FILTER_MEDIAN,
  CPL_FILTER_STDEV,
  CPL_FILTER_STDEV_FAST,
  CPL_FILTER_MORPHO,
  CPL_FILTER_MORPHO_SCALE }

  These are the supported filter modes.

4.10.1 Detailed Description

This module provides definitions for filtering of a `cpl_image` and a `cpl_mask`. The actual filtering functions are defined in the `cpl_image` and `cpl_mask` modules.

Synopsis:

```
#include "cpl_filter.h"
```

4.10.2 Typedef Documentation

4.10.2.1 cpl_border_mode

```
typedef enum _cpl_border_mode_ cpl_border_mode
```

The border mode type.

4.10.2.2 cpl_filter_mode

```
typedef enum _cpl_filter_mode_ cpl_filter_mode
```

The filter mode type.

4.10.3 Enumeration Type Documentation

4.10.3.1 _cpl_border_mode_

```
enum _cpl_border_mode_
```

These are the supported border modes. For a kernel of width 2n+1, the n left- and rightmost image/mask columns do not have elements for the whole kernel. The same holds for the top and bottom image/mask rows. The border mode defines the filtering of such border pixels.

Enumerator

CPL_BORDER_FILTER	Filter the border using the reduced number of pixels. If in median filtering the number of pixels is even choose the mean of the two central values, after the borders have been filled with a chess-like pattern of +- inf
CPL_BORDER_ZERO	Set the border of the filtered image/mask to zero.
CPL_BORDER_CROP	Crop the filtered image/mask.
CPL_BORDER_NOP	Do not modify the border of the filtered image/mask.
CPL_BORDER_COPY	Copy the border of the input image/mask. For an in-place operation this has the no effect, identical to CPL_BORDER_NOP.

4.10.3.2 _cpl_filter_mode_

```
enum _cpl_filter_mode_
```

These are the supported filter modes.

Enumerator

CPL_FILTER_EROSION	The erosion filter (for a `cpl_mask`). **See also** cpl_mask_filter()
CPL_FILTER_DILATION	The dilation filter (for a `cpl_mask`). **See also** cpl_mask_filter()
CPL_FILTER_OPENING	The opening filter (for a `cpl_mask`). **See also** cpl_mask_filter()
CPL_FILTER_CLOSING	The closing filter (for a `cpl_mask`). **See also** cpl_mask_filter()
CPL_FILTER_LINEAR	A linear filter (for a @c cpl_image). The kernel elements are normalized with the sum of their absolute values. This implies that there must be at least one non-zero element in the kernel. The normalisation makes the kernel useful for filtering where flux conservation is desired. The kernel elements are thus used as weights like this: Kernel Image ... 1 2 3 ... 1.0 2.0 3.0 ... 4 5 6 ... 4.0 5.0 6.0 ... 7 8 9 ... 7.0 8.0 9.0 ... ... The filtered value corresponding to the pixel whose value is 5.0 is: $\frac{(1*1.0+2*2.0+3*3.0+4*4.0+5*5.0+6*6.0+7*7.0+8*8.0+9*9.0)}{1+2+3+4+5+6+7+8+9}$ Filtering with `CPL_FILTER_LINEAR` and a flat kernel can be done faster with `CPL_FILTER_AVERAGE`. **See also** CPL_FILTER_LINEAR_SCALE, CPL_FILTER_AVERAGE, cpl_image_filter()

Enumerator

CPL_FILTER_LINEAR_SCALE	A linear filter (for a @c cpl_image). Unlike @c CPL_FILTER_LINEAR the kernel elements are not normalized, so the filtered image will have its flux scaled with the sum of the weights of the kernel. Examples of linear, scaling kernels are gradient operators and edge detectors. The kernel elements are thus applied like this: Kernel Image ... 1 2 3 ... 1.0 2.0 3.0 ... 4 5 6 ... 4.0 5.0 6.0 ... 7 8 9 ... 7.0 8.0 9.0 ... ... The filtered value corresponding to the pixel whose value is 5.0 is: $1*1.0+2*2.0+3*3.0+4*4.0+5*5.0+6*6.0+7*7.0+8*8.0+9*9.0$ **See also** CPL_FILTER_LINEAR, cpl_image_filter()
CPL_FILTER_AVERAGE	An average filter, i.e. the output pixel is the arithmetic average of the surrounding (1 + 2 ∗ hsizex) (1 + 2 ∗ hsizey) pixels. The cost per pixel is O(hsizex∗hsizey). The two images may have different pixel types. When the input and output pixel types are identical, the arithmetic is done with that type, e.g. int for two integer images. When the input and output pixel types differ, the arithmetic is done in double precision when one of the two images have pixel type CPL_TYPE_DOUBLE, otherwise float is used. **See also** CPL_FILTER_AVERAGE_FAST, cpl_image_filter_mask()
CPL_FILTER_AVERAGE_FAST	The same as `CPL_FILTER_AVERAGE`, except that it uses a running average, which will lead to a significant loss of precision if there are large differences in the magnitudes of the input pixels. The cost per pixel is O(1) if all elements in the kernel are used, otherwise the filtering is done as for CPL_FILTER_AVERAGE. **See also** cpl_image_filter_mask()
CPL_FILTER_MEDIAN	A median filter (for a `cpl_image`). The pixel types of the input and output images must be identical. **See also** cpl_image_filter_mask()

Enumerator

CPL_FILTER_STDEV	The filtered value is the standard deviation of the included input pixels. Kernel Image ... 1 0 1 ... 1.0 2.0 3.0 ... 0 1 0 ... 4.0 5.0 6.0 ... 1 0 1 ... 7.0 8.0 9.0 ... ... The pixel with value 5.0 will have a filtered value of: std_dev(1.0, 3.0, 5.0, 7.0, 9.0) **See also** CPL_FILTER_STDEV_FAST, cpl_image_filter_mask()
CPL_FILTER_STDEV_FAST	The same as CPL_FILTER_STDEV, except that it uses the same running method employed in CPL_FILTER_AVERAGE_FAST, which will lead to a significant loss of precision if there are large differences in the magnitudes of the input pixels. As for CPL_FILTER_AVERAGE_FAST, the cost per pixel is O(1) if all elements are used, otherwise the filtering is done as for CPL_FILTER_STDEV. **See also** cpl_image_filter_mask()
CPL_FILTER_MORPHO	A morphological filter (for a @c cpl_image). The kernel elements are used as weights on the sorted values covered by the kernel. The kernel elements are normalized with the sum of their absolute values. This implies that there must be at least one non-zero element in the kernel. The normalisation makes the kernel useful for filtering where flux conservation is desired. The kernel elements are used as weights on the sorted values covered by the kernel: Kernel Image ... 1 2 3 ... 4.0 6.0 5.0 ... 4 5 6 ... 3.0 1.0 2.0 ... 7 8 9 ... 7.0 8.0 9.0 ... ... The filtered value corresponding to the pixel whose value is 5.0 is: $\frac{(1*1.0+2*2.0+3*3.0+4*4.0+5*5.0+6*6.0+7*7.0+8*8.0+9*9.0)}{1+2+3+4+5+6+7+8+9}$ **See also** CPL_FILTER_MORPHO_SCALE, cpl_image_filter()

Enumerator

CPL_FILTER_MORPHO_SCALE	A morphological filter (for a @c cpl_image). Unlike @c CPL_FILTER_MORPHO the kernel elements are not normalized, so the filtered image will have its flux scaled with the sum of the weights of the kernel. The kernel elements are thus applied to the sorted values covered by the kernel: Kernel Image ... 1 2 3 ... 4.0 6.0 5.0 ... 4 5 6 ... 3.0 1.0 2.0 ... 7 8 9 ... 7.0 8.0 9.0 ... ... The filtered value corresponding to the pixel whose value is 5.0 is: $1*1.0+2*2.0+3*3.0+4*4.0+5*5.0+6*6.0+7*7.0+8*8.0+9*9.0$ **See also** CPL_FILTER_MORPHO, cpl_image_filter()

4.11 Frame Set Iterators

Iterator support for frame sets.

Typedefs

- typedef struct _cpl_frameset_iterator_ cpl_frameset_iterator
  The frame set iterator data type.

Functions

- cpl_error_code cpl_frameset_iterator_advance (cpl_frameset_iterator *self, int distance)
  Advance an iterator by a number of elements.
- cpl_error_code cpl_frameset_iterator_assign (cpl_frameset_iterator *self, const cpl_frameset_iterator *other)
  Assign a frame set iterator to another.
- void cpl_frameset_iterator_delete (cpl_frameset_iterator *self)
  Destroy a frame set iterator.
- int cpl_frameset_iterator_distance (const cpl_frameset_iterator *self, const cpl_frameset_iterator *other)
  Calculate the distance between two iterators.
- cpl_frameset_iterator * cpl_frameset_iterator_duplicate (const cpl_frameset_iterator *other)
  Create a frame set iterator from an existing frame set iterator.
- cpl_frame * cpl_frameset_iterator_get (cpl_frameset_iterator *self)
  Get the frame from the frame set at the current position of the iterator.
- const cpl_frame * cpl_frameset_iterator_get_const (const cpl_frameset_iterator *self)
  Get the frame from the frame set at the current position of the iterator.
- cpl_frameset_iterator * cpl_frameset_iterator_new (const cpl_frameset *parent)
  Create a new frame set iterator.
- void cpl_frameset_iterator_reset (cpl_frameset_iterator *self)
  Reset a frame set iterator to the beginning of a frame set.

4.11.1 Detailed Description

Iterator support for frame sets.

Frame set iterators allow to access the contents of a frame set sequentially, in an ordered manner. An iterator is created for a particular frame set, and it stays bound to that frame set until it is destroyed. However multiple iterators can be defined for the same frame set.

By default, the order of frames in a frame set is the order in which the individual frames have been inserted. However, a particular sorting order can be defined by sorting the frame set using a custom comparison function for frames.

Frame set iterators are supposed to be short-lived objects, and it is not recommended to use them for keeping, or passing around references to the frame set contents. In particular, changing the contents of the underlying frame set by erasing, or inserting frames may invalidate all existing iterators.

4.11.2 Typedef Documentation

4.11.2.1 cpl_frameset_iterator

```
typedef struct _cpl_frameset_iterator_ cpl_frameset_iterator
```

The frame set iterator data type.

This data type is opaque.

4.11.3 Function Documentation

4.11.3.1 cpl_frameset_iterator_advance()

```
cpl_error_code cpl_frameset_iterator_advance (
            cpl_frameset_iterator * self,
            int distance )
```

Advance an iterator by a number of elements.

Parameters

self	The frame set iterator to reposition.
distance	The of number of elements by which the iterator is moved.

Returns

The function returns `CPL_ERROR_NONE` on success, or an appropriate CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The given iterator offset would move the iterator beyond the beginning or the end of the frame set.

The functions moves the iterator by *distance* number of elements, with respect to its current position. The number of elements *distance* may be negative or positive, and the iterator position will move backward and forward respectively.

It is an error if the given *distance* would move the iterator either past the one-before-the-beginning position or the one-past-the-end position of the underlying frame set. In this case the iterator is not repositioned and an out of range error is returned.

References CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_frameset_dump(), cpl_frameset_extract(), and cpl_frameset_labelise().

4.11.3.2 cpl_frameset_iterator_assign()

```
cpl_error_code cpl_frameset_iterator_assign (
            cpl_frameset_iterator * self,
            const cpl_frameset_iterator * other )
```

Assign a frame set iterator to another.

Parameters

self	The frame set iterator to assign to.
other	The frame set iterator to be assigned.

Returns

The function returns CPL_ERROR_NONE on success, or an appropriate CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameters *self* or *other* is a NULL pointer.
CPL_ERROR_ILLEGAL_INPUT	The parameters *self* and *other* are not bound to the same frame set.

The function assigns the iterator *other* to the iterator *self*. Only iterators which are bound to the same frame set can be assigned to each other!

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.11.3.3 cpl_frameset_iterator_delete()

```
void cpl_frameset_iterator_delete (
            cpl_frameset_iterator * self )
```

Destroy a frame set iterator.

Parameters

self	The frame set to destroy.

Returns

Nothing.

The function destroys the frame set iterator object *self*. If *self* is NULL, no operation is performed.

Referenced by cpl_frameset_dump(), cpl_frameset_extract(), cpl_frameset_labelise(), and cpl_imagelist_load_↩ frameset().

4.11.3.4 cpl_frameset_iterator_distance()

```
int cpl_frameset_iterator_distance (
            const cpl_frameset_iterator * self,
            const cpl_frameset_iterator * other )
```

Calculate the distance between two iterators.

Parameters

self	First frame set iterator.
other	Second frame set iterator.

Returns

The function returns the distance between the two input iterators. If an error occurs, the function returns a distance of 0 and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *other* is a NULL pointer.
CPL_ERROR_ILLEGAL_INPUT	The two iterators iterators are not bound to the same frame set.
CPL_ERROR_DATA_NOT_FOUND	At least one input iterator is invalid.

The function calculates the distance between the iterators *self* and *other*.

To properly detect whether this function has failed or not it is recommended to reset the errors before it is called, since the returned value is not a distinctive feature.

References CPL_ERROR_DATA_NOT_FOUND, CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_IN↩PUT.

4.11.3.5 cpl_frameset_iterator_duplicate()

```
cpl_frameset_iterator* cpl_frameset_iterator_duplicate (
            const cpl_frameset_iterator * other )
```

Create a frame set iterator from an existing frame set iterator.

Parameters

other	The frame set iterator to clone.

Returns

A copy of the frame set iterator *other* on success, or NULL otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *other* is a `NULL` pointer.

The function creates a copy of the iterator object *other*. An iterator copy constructed by this function is bound to the same frame set *other* has been constructed for, and it points to the same frame that *other* points to.

References CPL_ERROR_NULL_INPUT.

4.11.3.6 cpl_frameset_iterator_get()

```
cpl_frame* cpl_frameset_iterator_get (
            cpl_frameset_iterator * self )
```

Get the frame from the frame set at the current position of the iterator.

Parameters

self	The frame set iterator to dereference.

Returns

The frame stored in the frame set at the position of the iterator, or `NULL` if an error occurs. The function also returns `NULL` if the iterator is positioned at the sentinel element (i.e. the one-before-the-beginning or one-past-the-end position).

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function dereferences the iterator *self*, i.e. it retrieves the frame at the position pointed to by *self*.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_frameset_labelise().

4.11.3.7 cpl_frameset_iterator_get_const()

```
const cpl_frame* cpl_frameset_iterator_get_const (
            const cpl_frameset_iterator * self )
```

Get the frame from the frame set at the current position of the iterator.

Parameters

self	The frame set iterator to dereference.

Returns

The frame stored in the frame set at the position of the iterator or NULL if an error occurs. The function also returns NULL if the iterator is positioned at the sentinel element (i.e. the one-before-the-beginning or one-past-the-end position).

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function dereferences the iterator *self*, i.e. it retrieves the frame at the position pointed to by *self*.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_dfs_sign_products(), cpl_frameset_dump(), cpl_frameset_extract(), and cpl_imagelist_load_↩ frameset().

4.11.3.8 cpl_frameset_iterator_new()

```
cpl_frameset_iterator* cpl_frameset_iterator_new (
            const cpl_frameset * parent )
```

Create a new frame set iterator.

Parameters

parent	The frame set for which the iterator is created.

Returns

The newly allocated frame set iterator object, or NULL if an error occurred.

Errors

CPL_ERROR_NULL_INPUT	The parameter *parent* is a NULL pointer.

The function creates a new iterator object bound to the frame set *parent*. The iterator is initialized such that it points to the beginning of *parent*.

The beginning is defined by the current ordering defined for the frame set *parent*.

See also

cpl_frameset_sort()

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_dfs_sign_products(), cpl_frameset_dump(), cpl_frameset_extract(), cpl_frameset_labelise(), and cpl_imagelist_load_frameset().

4.11.3.9 cpl_frameset_iterator_reset()

```
void cpl_frameset_iterator_reset (
            cpl_frameset_iterator * self )
```

Reset a frame set iterator to the beginning of a frame set.

Parameters

self	The iterator to reposition.

Returns

Nothing.

The function moves the frame set iterator *self* back to the beginning of its underlying frame set. The first frame in the frame set is defined by the established sorting order.

See also

cpl_frameset_sort()

4.12 Frame Sets

Modules

- Frame Set Iterators

  Iterator support for frame sets.

Typedefs

- typedef struct _cpl_frameset_ cpl_frameset

  The frame set data type.

Functions

- int cpl_frameset_count_tags (const cpl_frameset *self, const char *tag)

  Counts the frames stored in a frame set having the given tag.
- void cpl_frameset_delete (cpl_frameset *self)

  Destroy a frame set.
- void cpl_frameset_dump (const cpl_frameset *self, FILE *stream)

  Dump the frameset debugging information to the given stream.
- cpl_frameset * cpl_frameset_duplicate (const cpl_frameset *other)

  Create a copy of the given frame set.
- cpl_size cpl_frameset_erase (cpl_frameset *self, const char *tag)

  Erase all frames with the given tag from a frame set.
- cpl_error_code cpl_frameset_erase_frame (cpl_frameset *self, cpl_frame *frame)

  Erase the given frame from a frame set.
- cpl_frameset * cpl_frameset_extract (const cpl_frameset *self, const cpl_size *labels, cpl_size desired_label)

  Extract a subset of frames from a set of frames.
- cpl_frame * cpl_frameset_find (cpl_frameset *self, const char *tag)

  Find a frame with the given tag in a frame set.
- const cpl_frame * cpl_frameset_find_const (const cpl_frameset *self, const char *tag)

  Find a frame with the given tag in a frame set.
- cpl_frame * cpl_frameset_get_first (cpl_frameset *self)

  Get the first frame in the given set.
- const cpl_frame * cpl_frameset_get_first_const (const cpl_frameset *self)

  Get the first frame in the given set.
- cpl_frame * cpl_frameset_get_frame (cpl_frameset *set, cpl_size position)

  Get a frame from a frame set.
- const cpl_frame * cpl_frameset_get_frame_const (const cpl_frameset *set, cpl_size position)

  Get a frame from a frame set.
- cpl_frame * cpl_frameset_get_next (cpl_frameset *self)

  Get the next frame in the given set.
- const cpl_frame * cpl_frameset_get_next_const (const cpl_frameset *self)

  Get the next frame in the given set.
- cpl_frame * cpl_frameset_get_position (cpl_frameset *self, cpl_size position)

  Get the frame at a given position in the frame set.
- const cpl_frame * cpl_frameset_get_position_const (const cpl_frameset *self, cpl_size position)

  Get the frame at a given iterator position.
- cpl_size cpl_frameset_get_size (const cpl_frameset *self)

Get the current size of a frame set.

- cpl_error_code cpl_frameset_insert (cpl_frameset *self, cpl_frame *frame)

  Insert a frame into the given frame set.
- int cpl_frameset_is_empty (const cpl_frameset *self)

  Check whether a frame set is empty.
- cpl_error_code cpl_frameset_join (cpl_frameset *self, const cpl_frameset *other)

  Join two frame sets.
- cpl_size * cpl_frameset_labelise (const cpl_frameset *self, int(*compare)(const cpl_frame *, const cpl_frame *), cpl_size *nb_labels)

  Separate a list of frames into groups, using a comparison function.
- cpl_frameset * cpl_frameset_new (void)

  Create a new, empty frame set.
- cpl_error_code cpl_frameset_sort (cpl_frameset *self, cpl_frame_compare_func compare)

  Sort a frame set.

4.12.1 Detailed Description

The module implements a container type for frames. Frames can be stored in a frame set and retrieved, either by searching for a particular frame tag or by sequential access. Frame sets can be created, filled and saved to a so called 'set of frames' file or loaded from such a file.

Synopsis:

```
#include <cpl_frameset.h>
```

4.12.2 Typedef Documentation

4.12.2.1 cpl_frameset

```
typedef struct _cpl_frameset_ cpl_frameset
```

The frame set data type.

This data type is opaque.

4.12.3 Function Documentation

4.12.3.1 cpl_frameset_count_tags()

```
int cpl_frameset_count_tags (
            const cpl_frameset * self,
            const char * tag )
```

Counts the frames stored in a frame set having the given tag.

Parameters

self	A frame set.
tag	The frame tag.

Returns

The number of frames with tag *tag*. The function returns 0 if an error occurs and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *tag* is a `NULL` pointer.

The function scans the frame set *self* for frames with the tag *tag* and returns the number of frames found.

References CPL_ERROR_NULL_INPUT.

4.12.3.2 cpl_frameset_delete()

```
void cpl_frameset_delete (
            cpl_frameset * self )
```

Destroy a frame set.

Parameters

self	The frame set to destroy.

Returns

Nothing.

The function destroys the frame set *self* and its whole contents. If *self* is `NULL`, nothing is done and no error is set.

4.12.3.3 cpl_frameset_dump()

```
void cpl_frameset_dump (
            const cpl_frameset * self,
            FILE * stream )
```

Dump the frameset debugging information to the given stream.

Parameters

self	The frameset.
stream	The output stream to use.

Returns

Nothing.

The function dumps the contents of the frameset *self* to the output stream *stream*. If *stream* is NULL the function writes to the standard output. If *self* is NULL or the frameset is empty, the function does nothing.

References CPL_ERROR_ACCESS_OUT_OF_RANGE, cpl_error_get_code(), cpl_errorstate_get(), cpl_errorstate_set(), cpl_frame_dump(), cpl_frameset_get_size(), cpl_frameset_iterator_advance(), cpl_frameset_iterator_delete(), cpl_frameset_iterator_get_const(), and cpl_frameset_iterator_new().

4.12.3.4 cpl_frameset_duplicate()

```
cpl_frameset* cpl_frameset_duplicate (
            const cpl_frameset * other )
```

Create a copy of the given frame set.

Parameters

other	The frame set to be copied.

Returns

A handle for the created clone. The function returns NULL if an error occurs and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *other* is a NULL pointer.

The function creates a deep copy, i.e. the frame set object and its contents, of the frame set *other*. The created copy and the original set do not share any resources.

References CPL_ERROR_NULL_INPUT, cpl_frame_duplicate(), cpl_frame_get_tag(), and cpl_frameset_new().

4.12.3.5 cpl_frameset_erase()

```
cpl_size cpl_frameset_erase (
            cpl_frameset * self,
            const char * tag )
```

Erase all frames with the given tag from a frame set.

Parameters

self	A frame set.
tag	The tag used to locate the frames to remove.

Returns

The function returns the number of frames removed. If an error occurs 0 is returned and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *tag* is a `NULL` pointer.

The function searches the frame set *self* for frames having the tag *tag* and removes them from the set. The removed frames are destroyed. If no frame with the tag *tag* is found the function has no effect.

References CPL_ERROR_NULL_INPUT.

4.12.3.6 cpl_frameset_erase_frame()

```
cpl_error_code cpl_frameset_erase_frame (
            cpl_frameset * self,
            cpl_frame * frame )
```

Erase the given frame from a frame set.

Parameters

self	A frame set.
frame	The frame to remove.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *frame* is a `NULL` pointer.

The function searches the frame set *self* for the first occurrance of *frame*. If it is present, the frame is removed from the set and destroyed. If frame is not present in *self* the function has no effect.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl_frame_get_tag().

4.12.3.7 cpl_frameset_extract()

```
cpl_frameset* cpl_frameset_extract (
             const cpl_frameset * self,
             const cpl_size * labels,
             cpl_size desired_label )
```

Extract a subset of frames from a set of frames.

Parameters

self	Input frame set
labels	The array of labels associated to each input frame
desired_label	The label identifying the requested frames

Note

The array of labels must have (at least) the length of the frame set

Returns

A pointer to a newly allocated frame set or NULL on error

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *labels* is a `NULL` pointer.

The returned object must be deallocated with cpl_frameset_delete()

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, cpl_error_get_code(), CPL_ERROR_NUL↩L_INPUT, cpl_errorstate_get(), cpl_errorstate_set(), cpl_frame_duplicate(), cpl_frameset_insert(), cpl_frameset_↩iterator_advance(), cpl_frameset_iterator_delete(), cpl_frameset_iterator_get_const(), cpl_frameset_iterator_new(), and cpl_frameset_new().

4.12.3.8 cpl_frameset_find()

```
cpl_frame* cpl_frameset_find (
             cpl_frameset * self,
             const char * tag )
```

Find a frame with the given tag in a frame set.

Parameters

self	A frame set.
tag	The frame tag to search for.

Returns

The handle for a frame with tag *tag*, or NULL if no such frame was found. The function returns NULL if an error occurs and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *tag* is a NULL pointer.

The function searches the frame set *self* for the frames with the tag *tag*. If such a frame is present, a handle for it is returned. If the set contains several frames with the tag *tag* the first one is returned. The remaining frames with this tag can be accessed sequentially by using NULL as tag when calling this function repeatedly, since the most recent frame accessed is cached. This cache is reset whenever the provided tag is not NULL. If no frame with the tag *tag* is present in *self* or no more frames with this tag are found the function returns NULL.

Note

Since the most recently accessed frame is cached in the frameset this function is not re-entrant!

References cpl_error_set_where, cpl_errorstate_get(), cpl_errorstate_is_equal(), and cpl_frameset_find_const().

4.12.3.9 cpl_frameset_find_const()

```
const cpl_frame* cpl_frameset_find_const (
             const cpl_frameset * self,
             const char * tag )
```

Find a frame with the given tag in a frame set.

Parameters

self	A frame set.
tag	The frame tag to search for.

Returns

The handle for a frame with tag *tag*, or NULL if no such frame was found. The function returns NULL if an error occurs and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *tag* is a NULL pointer.

The function searches the frame set *self* for the frames with the tag *tag*. If such a frame is present, a handle for it is returned. If the set contains several frames with the tag *tag* the first one is returned. The remaining frames with this tag can be accessed sequentially by using NULL as tag when calling this function repeatedly, since the most recent frame accessed is cached. This cache is reset whenever the provided tag is not NULL. If no frame with the tag *tag* is present in *self* or no more frames with this tag are found the function returns NULL.

Note

Since the most recently accessed frame is cached in the frameset this function is not re-entrant!

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_frameset_find().

4.12.3.10 cpl_frameset_get_first()

```
cpl_frame* cpl_frameset_get_first (
             cpl_frameset * self )
```

Get the first frame in the given set.

Parameters

self	A frame set.

Returns

A handle for the first frame in the set, or NULL if the set is empty. The function returns NULL if an error occurs and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function returns the first frame in the frame set *self* if it exists. If a first frame does not exist, i.e. the frame set is

empty, NULL is returned. The function also updates the internal cache.

See also

cpl_frameset_get_next()

Deprecated This function will be removed from CPL version 7. Code using these functions should be ported to make use of frame set iterators instead!

References CPL_ERROR_NULL_INPUT.

4.12.3.11 cpl_frameset_get_first_const()

```
const cpl_frame* cpl_frameset_get_first_const (
            const cpl_frameset * self )
```

Get the first frame in the given set.

Parameters

self	A frame set.

Returns

A handle for the first frame in the set, or NULL if the set is empty. The function returns NULL if an error occurs and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function returns the first frame in the frame set *self* if it exists. If a first frame does not exist, i.e. the frame set is empty, NULL is returned. The function also updates the internal cache.

See also

cpl_frameset_get_next_const()

Deprecated This function will be removed from CPL version 7. Code using these functions should be ported to make use of frame set iterators instead!

References CPL_ERROR_NULL_INPUT.

4.12.3.12 cpl_frameset_get_frame()

```
cpl_frame* cpl_frameset_get_frame (
            cpl_frameset * set,
            cpl_size position )
```

Get a frame from a frame set.

Parameters

set	Input frame set.
position	The requested frame.

Returns

The function returns a handle to the frame at position *position* in the set, or NULL in case an error occurs.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_ILLEGAL_INPUT	The parameter *position* is out of range.

The function returns a handle to the frame at the index *position* in the set. The frame position ranges from 0 to one less than the size of the frame set.

The returned frame is still owned by the frame set *set*, i.e. the obtained frame must not be deleted through the returned handle and also its tag must not be modified.

As an alternative to using this function, the functions cpl_frameset_get_first() and cpl_frameset_get_next() should be considered, if performance is an issue.

See also

cpl_frameset_get_size(), cpl_frameset_get_first(), cpl_frameset_get_next()

Deprecated This function will be removed from CPL version 7. Code using these functions should use cpl_frameset_get_position() instead!

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_frameset_get_position(), and cpl↩_frameset_get_size().

4.12.3.13 cpl_frameset_get_frame_const()

```
const cpl_frame* cpl_frameset_get_frame_const (
            const cpl_frameset * set,
            cpl_size position )
```

Get a frame from a frame set.

Parameters

set	Input frame set.
position	The requested frame.

Returns

The function returns a handle to the frame at position *position* in the set, or `NULL` in case an error occurs.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	The parameter *position* is out of range.

The function returns a handle to the frame at the index *position* in the set. The frame position ranges from `0` to one less than the size of the frame set.

The returned frame is still owned by the frame set *set*, i.e. the obtained frame must not be deleted through the returned handle and also its tag must not be modified.

As an alternative to using this function, the functions cpl_frameset_get_first_const() and cpl_frameset_get_next_const() should be considered, if performance is an issue.

See also

cpl_frameset_get_size(), cpl_frameset_get_first_const(), cpl_frameset_get_next_const()

Deprecated This function will be removed from CPL version 7. Code using these functions should use cpl_frameset_get_position_const() instead!

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_frameset_get_position_const(), and cpl_frameset_get_size().

4.12.3.14 cpl_frameset_get_next()

```
cpl_frame* cpl_frameset_get_next (
             cpl_frameset * self )
```

Get the next frame in the given set.

Parameters

self	A frame set.

Returns

A handle for the next frame in a set. If there are no more frames in the set the function returns NULL. The function returns NULL if an error occurs and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function returns the next frame in the frame set *self* if it exists and otherwise NULL. The function uses the internal cache to determine the most recently accessed frame. This means that the function only works as expected if *self* has been initialised by a call to **cpl_frameset_get_first()**, and if no function updating the internal cache was called between two subsequent calls to this function.

See also

cpl_frameset_get_first()

Deprecated This function will be removed from CPL version 7. Code using these functions should be ported to make use of frame set iterators instead!

References CPL_ERROR_NULL_INPUT.

4.12.3.15 cpl_frameset_get_next_const()

```
const cpl_frame* cpl_frameset_get_next_const (
            const cpl_frameset * self )
```

Get the next frame in the given set.

Parameters

self	A frame set.

Returns

A handle for the next frame in a set. If there are no more frames in the set the function returns NULL. The function returns NULL if an error occurs and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function returns the next frame in the frame set *self* if it exists and otherwise NULL. The function uses the

internal cache to determine the most recently accessed frame. This means that the function only works as expected if *self* has been initialised by a call to **cpl_frameset_get_first_const()**, and if no function updating the internal cache was called between two subsequent calls to this function.

See also

cpl_frameset_get_first_const()

Deprecated This function will be removed from CPL version 7. Code using these functions should be ported to make use of frame set iterators instead!

References CPL_ERROR_NULL_INPUT.

4.12.3.16 cpl_frameset_get_position()

```
cpl_frame* cpl_frameset_get_position (
             cpl_frameset * self,
             cpl_size position )
```

Get the frame at a given position in the frame set.

Parameters

self	The frame set
position	Frame position.

Returns

The function returns the frame at the given position, or `NULL` if an error occurs.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	The parameter *position* is invalid, i.e. the given value is outside of its domain.

The function retrieves the frame stored in the frame set *self* at the index position *position*. The index positions are counted from zero, and reach up to one less than the number of frames in the frame set.

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, and cpl_frameset_get_size().

Referenced by cpl_frameset_get_frame().

4.12.3.17 cpl_frameset_get_position_const()

```
const cpl_frame* cpl_frameset_get_position_const (
            const cpl_frameset * self,
            cpl_size position )
```

Get the frame at a given iterator position.

Parameters

self	The iterator to dereference
position	Iterator offset from the beginning of the frame set.

Returns

The function returns the frame at the iterator position, or NULL if an error occurs.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_ILLEGAL_INPUT	The parameter *position* is invalid, i.e. the given value is outside of its domain.

The function retrieves the frame stored in the frame set *self* at the index position *position*. The index positions are counted from zero, and reach up to one less than the number of frames in the frame set.

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, and cpl_frameset_get_size().

Referenced by cpl_frameset_get_frame_const().

4.12.3.18 cpl_frameset_get_size()

```
cpl_size cpl_frameset_get_size (
            const cpl_frameset * self )
```

Get the current size of a frame set.

Parameters

self	A frame set.

Returns

The frame set's current size, or 0 if it is empty. The function returns 0 if an error occurs and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The reports the current number of frames stored in the frame set *self*.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_frameset_dump(), cpl_frameset_get_frame(), cpl_frameset_get_frame_const(), cpl_frameset↩ _get_position(), cpl_frameset_get_position_const(), and cpl_frameset_labelise().

4.12.3.19 cpl_frameset_insert()

```
cpl_error_code cpl_frameset_insert (
            cpl_frameset * self,
            cpl_frame * frame )
```

Insert a frame into the given frame set.

Parameters

self	A frame set.
frame	The frame to insert.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *frame* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	The parameter *frame* has an invalid tag.

The function adds the frame *frame* to the frame set *self* using the frame's tag as key.

The insertion of a frame into a frameset transfers the ownership of the frame *frame* to the frameset *self*. This means that the frame must not be deallocated through the pointer *frame*.

In addition, the frame pointer returned by any member function call returning a handle to a frameset's member frame, must not be used to insert the returned frame into another frameset without prior duplication of this frame, and, it must not be used to modify the frames tag without removing it from the frameset first and re-inserting it with the new tag afterwards.

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl_frame↩ _get_tag().

Referenced by cpl_frameset_extract(), and cpl_frameset_join().

4.12.3.20 cpl_frameset_is_empty()

```
int cpl_frameset_is_empty (
            const cpl_frameset * self )
```

Check whether a frame set is empty.

Parameters

self	A frame set.

Returns

The function returns 1 if the set is empty, and 0 otherwise. If an error occurs 0 is returned and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function checks if *self* contains any frames.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_dfs_sign_products(), and cpl_frameset_join().

4.12.3.21 cpl_frameset_join()

```
cpl_error_code cpl_frameset_join (
            cpl_frameset * self,
            const cpl_frameset * other )
```

Join two frame sets.

Parameters

self	The target frame set
other	The source frame set

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *frame* is a `NULL` pointer.

The function adds the contents of the frame set *other* to *self* by inserting copies of the elements of the frame set *other*. If the source frame set *other* is NULL, or if it is empty, the function has no effect.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_frame_delete(), cpl_frame_duplicate(), cpl_↩ frameset_insert(), and cpl_frameset_is_empty().

4.12.3.22 cpl_frameset_labelise()

```
cpl_size* cpl_frameset_labelise (
              const cpl_frameset * self,
              int(*)(const cpl_frame *, const cpl_frame *) compare,
              cpl_size * nb_labels )
```

Separate a list of frames into groups, using a comparison function.

Parameters

self	Input frame set
compare	Pointer to comparison function to use.
nb_labels	Number of different sets or undefined on error

Returns

array of labels defining the selection or NULL on error

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *compare* or *nb_labels* is a NULL pointer.

This function takes a set of frames and groups the frames that are 'identical' together. The user provided comparison function defines what being identical means for two frames. A label (non-negative int) is associated to each group of identical frames, these labels are returned in an array of length equal to the size of the frameset.

The comparison function should be commutative, must take two frames and return 1 if they are identical, 0 if they are different, and -1 on error.

The number of calls to the comparison functions is O(n∗m), where n is the number of frames in the set, and m is the number of different labels found in the set. In the worst case m equals n, and the call requires n(n-1)/2 calls to the comparison function. If all identical frames appear together in the list, the number of required calls is only n + O(m^2).

The returned array must be deallocated with cpl_free().

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, cpl_error_get_code(), CPL_ERROR_ILL↩ EGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), cpl_errorstate_set(), cpl_frameset_get_size(), cpl_frameset_iterator_advance(), cpl_frameset_iterator_delete(), cpl_frameset_iterator_get(), cpl_frameset_↩ iterator_new(), cpl_msg_debug(), and CPL_SIZE_FORMAT.

4.12.3.23 cpl_frameset_new()

```
cpl_frameset* cpl_frameset_new (
            void  )
```

Create a new, empty frame set.

Returns

The handle for the newly created frame set.

The function allocates the memory for the new frame set, initialises the set to be empty and returns a handle for it.

Referenced by cpl_frameset_duplicate(), and cpl_frameset_extract().

4.12.3.24 cpl_frameset_sort()

```
cpl_error_code cpl_frameset_sort (
            cpl_frameset * self,
            cpl_frame_compare_func compare )
```

Sort a frame set.

Parameters

self	The frame set to sort.
compare	Comparison function for frames.

Returns

The function returns `CPL_ERROR_NONE` on success, or an appropriate CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *compare* is a `NULL` pointer.

-

The function replaces the existing order of the frame set *self* by sorting its contents according to the comparison function *compare*.

By default, the order of a frame set, i.e. the order of any newly created frame set object, is defined by the order in which frames are inserted into the frame set. By calling this function, this order will be lost. If this order has to be preserved, sorting has to be done on a copy of *self*.

The function *compare* compares two frames and must return -1, 0, or 1 if the first frame is considered to be less than, equal or larger than the second frame, respectively.

See also

cpl_frame_compare_func

References CPL_ERROR_NULL_INPUT.

4.13 Frame Sets IO functions

Functions

- cpl_imagelist * cpl_imagelist_load_frameset (const cpl_frameset *fset, cpl_type im_type, cpl_size pnum, cpl_size xtnum)

  Load an imagelist from a frameset.

4.13.1 Detailed Description

Synopsis:

```
#include <cpl_frameset_io.h>
```

4.13.2 Function Documentation

4.13.2.1 cpl_imagelist_load_frameset()

```
cpl_imagelist* cpl_imagelist_load_frameset (
            const cpl_frameset * fset,
            cpl_type im_type,
            cpl_size pnum,
            cpl_size xtnum )
```

Load an imagelist from a frameset.

Parameters

fset	The frames set
im_type	The required image type
pnum	The plane number, 1 for first, 0 for all planes
xtnum	The extension number, 0 for primary, n for nth, -1 for all

Returns

the loaded list of images or NULL on error.

Note

The returned cpl_imagelist must be deallocated using cpl_imagelist_delete()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if fset is NULL
- CPL_ERROR_ILLEGAL_INPUT if pnum is negative, xtnum is lower than -1 or one image cannot be loaded as specified

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_errorstate_get(), cpl_frame_get_filename(), cpl_frameset_iterator_delete(), cpl_frameset_iterator_get_const(), cpl_frameset_iterator_new(), and cpl_imagelist_new().

4.14 Frames

Macros

- #define CPL_FRAME_GROUP_CALIB_ID "CALIB"

  Frame group tag for calibration data.
- #define CPL_FRAME_GROUP_PRODUCT_ID "PRODUCT"

  Frame group tag for processed data.
- #define CPL_FRAME_GROUP_RAW_ID "RAW"

  Frame group tag for unprocessed data.

Typedefs

- typedef struct _cpl_frame_ cpl_frame

  The frame data type.
- typedef int(∗ cpl_frame_compare_func) (const cpl_frame ∗self, const cpl_frame ∗other)

  Frame comparison function.
- typedef enum _cpl_frame_group_ cpl_frame_group

  The frame group data type.
- typedef enum _cpl_frame_level_ cpl_frame_level

  The frame level data type.
- typedef enum _cpl_frame_type_ cpl_frame_type

  The frame type data type.

Enumerations

- enum _cpl_frame_group_ {
  CPL_FRAME_GROUP_NONE,
  CPL_FRAME_GROUP_RAW,
  CPL_FRAME_GROUP_CALIB,
  CPL_FRAME_GROUP_PRODUCT }

  Supported frame groups.
- enum _cpl_frame_level_ {
  CPL_FRAME_LEVEL_NONE,
  CPL_FRAME_LEVEL_TEMPORARY,
  CPL_FRAME_LEVEL_INTERMEDIATE,
  CPL_FRAME_LEVEL_FINAL }

  Supported frame processing levels.
- enum _cpl_frame_type_ {
  CPL_FRAME_TYPE_NONE,
  CPL_FRAME_TYPE_IMAGE,
  CPL_FRAME_TYPE_MATRIX,
  CPL_FRAME_TYPE_TABLE,
  CPL_FRAME_TYPE_PAF,
  CPL_FRAME_TYPE_ANY }

  Supported frame types.

Functions

- void cpl_frame_delete (cpl_frame *self)

  Destroy a frame.
- void cpl_frame_dump (const cpl_frame *frame, FILE *stream)

  Dump the frame debugging information to the given stream.
- cpl_frame * cpl_frame_duplicate (const cpl_frame *other)

  Create a copy of a frame.
- const char * cpl_frame_get_filename (const cpl_frame *self)

  Get the file name to which a frame refers.
- cpl_frame_group cpl_frame_get_group (const cpl_frame *self)

  Get the current group of a frame.
- cpl_frame_level cpl_frame_get_level (const cpl_frame *self)

  Get the current level of a frame.
- cpl_size cpl_frame_get_nextensions (const cpl_frame *self)

  Get the number of extensions of this frame.
- const char * cpl_frame_get_tag (const cpl_frame *self)

  Get the category tag of a frame.
- cpl_frame_type cpl_frame_get_type (const cpl_frame *self)

  Get the type of a frame.
- cpl_frame * cpl_frame_new (void)

  Create a new, empty frame.
- cpl_error_code cpl_frame_set_filename (cpl_frame *self, const char *filename)

  Set the file name to which a frame refers.
- cpl_error_code cpl_frame_set_group (cpl_frame *self, cpl_frame_group group)

  Set the group attribute of a frame.
- cpl_error_code cpl_frame_set_level (cpl_frame *self, cpl_frame_level level)

  Set the level attribute of a frame.
- cpl_error_code cpl_frame_set_tag (cpl_frame *self, const char *tag)

  Set a frame's category tag.
- cpl_error_code cpl_frame_set_type (cpl_frame *self, cpl_frame_type type)

  Set the type of a frame.

4.14.1 Detailed Description

This module implements the `cpl_frame` type. A frame is a container for descriptive attributes related to a data file. The attributes are related to a data file through the file name member of the frame type. Among the attributes which may be assigned to a data file is an attribute identifying the type of the data stored in the file (image or table data), a classification tag indicating the kind of data the file contains and an attribute denoting to which group the data file belongs (raw, processed or calibration file). For processed data a processing level indicates whether the product is an temporary, intermediate or final product.

Synopsis:

```
#include <cpl_frame.h>
```

4.14.2 Macro Definition Documentation

4.14.2.1 CPL_FRAME_GROUP_CALIB_ID

```
#define CPL_FRAME_GROUP_CALIB_ID "CALIB"
```

Frame group tag for calibration data.

4.14.2.2 CPL_FRAME_GROUP_PRODUCT_ID

```
#define CPL_FRAME_GROUP_PRODUCT_ID "PRODUCT"
```

Frame group tag for processed data.

4.14.2.3 CPL_FRAME_GROUP_RAW_ID

```
#define CPL_FRAME_GROUP_RAW_ID "RAW"
```

Frame group tag for unprocessed data.

4.14.3 Typedef Documentation

4.14.3.1 cpl_frame

```
typedef struct _cpl_frame_ cpl_frame
```

The frame data type.

4.14.3.2 cpl_frame_compare_func

```
typedef int(* cpl_frame_compare_func) (const cpl_frame *self, const cpl_frame *other)
```

Frame comparison function.

Type definition for functions which compares the frame *other* with the frame *self*.

All function of this type must return -1, 0, or 1 if *self* is considered to be less than, equal or greater than *other* respectively.

4.14.3.3 cpl_frame_group

`typedef enum _cpl_frame_group_ cpl_frame_group`

The frame group data type.

4.14.3.4 cpl_frame_level

`typedef enum _cpl_frame_level_ cpl_frame_level`

The frame level data type.

4.14.3.5 cpl_frame_type

`typedef enum _cpl_frame_type_ cpl_frame_type`

The frame type data type.

4.14.4 Enumeration Type Documentation

4.14.4.1 _cpl_frame_group_

`enum _cpl_frame_group_`

Supported frame groups.

Defines the possible values for the frame's group attribute.

Enumerator

CPL_FRAME_GROUP_NONE	The frame does not belong to any supported group.
CPL_FRAME_GROUP_RAW	The frame is associated to unprocessed data.
CPL_FRAME_GROUP_CALIB	The frame is associated to calibration data.
CPL_FRAME_GROUP_PRODUCT	The frame is associated to processed data.

4.14.4.2 _cpl_frame_level_

`enum _cpl_frame_level_`

Supported frame processing levels.

Note

The processing levels are just flags and it is left to the application to trigger the appropriate action for the different levels.

Enumerator

CPL_FRAME_LEVEL_NONE	Undefined processing level
CPL_FRAME_LEVEL_TEMPORARY	Temporary product. The corresponding file will be deleted when the processing chain is completed.
CPL_FRAME_LEVEL_INTERMEDIATE	Intermediate product. The corresponding file is only kept on request. The default is to delete these products at the end of the processing chain.
CPL_FRAME_LEVEL_FINAL	Final data product, which is always written to a file at the end of the processing chain.

4.14.4.3 _cpl_frame_type_

```
enum _cpl_frame_type_
```

Supported frame types.

Defines the possible values for the frame's type attribute.

Enumerator

CPL_FRAME_TYPE_NONE	Undefined frame type
CPL_FRAME_TYPE_IMAGE	Image frame type identifier
CPL_FRAME_TYPE_MATRIX	Matrix frame type identifier
CPL_FRAME_TYPE_TABLE	Table frame type identifier
CPL_FRAME_TYPE_PAF	paf frame type identifier
CPL_FRAME_TYPE_ANY	identifier for any other type

4.14.5 Function Documentation

4.14.5.1 cpl_frame_delete()

```
void cpl_frame_delete (
            cpl_frame * self )
```

Destroy a frame.

Parameters

self	A frame.

Returns

Nothing.

The function deallocates the memory used by the frame *self*. If *self* is `NULL`, nothing is done, and no error is set.

Referenced by cpl_frameset_join().

4.14.5.2 cpl_frame_dump()

```
void cpl_frame_dump (
            const cpl_frame * frame,
            FILE * stream )
```

Dump the frame debugging information to the given stream.

Parameters

frame	The frame.
stream	The output stream to use.

Returns

Nothing.

The function dumps the contents of the frame *frame* to the output stream *stream*. If *stream* is `NULL` the function writes to the standard output. If *frame* is `NULL` the function does nothing.

References cpl_frame_get_filename(), cpl_frame_get_group(), cpl_frame_get_level(), cpl_frame_get_tag(), cpl_frame_get_type(), CPL_FRAME_GROUP_CALIB, CPL_FRAME_GROUP_PRODUCT, CPL_FRAME_GROUP_RAW, CPL_FRAME_LEVEL_FINAL, CPL_FRAME_LEVEL_INTERMEDIATE, CPL_FRAME_LEVEL_TEMPORARY, CPL_FRAME_TYPE_ANY, CPL_FRAME_TYPE_IMAGE, CPL_FRAME_TYPE_MATRIX, CPL_FRAME_TYPE_PAF, and CPL_FRAME_TYPE_TABLE.

Referenced by cpl_frameset_dump().

4.14.5.3 cpl_frame_duplicate()

```
cpl_frame* cpl_frame_duplicate (
            const cpl_frame * other )
```

Create a copy of a frame.

Parameters

other	The frame to copy.

Returns

The function returns a handle for the created clone. If an error occurs the function returns NULL and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *other* is a NULL pointer.

The function creates a clone of the input frame *other*. All members of the input frame are copied.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_frameset_duplicate(), cpl_frameset_extract(), and cpl_frameset_join().

4.14.5.4 cpl_frame_get_filename()

```
const char* cpl_frame_get_filename (
             const cpl_frame * self )
```

Get the file name to which a frame refers.

Parameters

self	A frame.

Returns

The file name to which the frame refers, or NULL if a file name has not been set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	The frame *self* is not associated to a file.

The function returns the read-only name of a file associated to *self*. A file is associated to *self* by calling **cpl_frame_set_filename()** for *self*. If *self* is not associated to a file this function returns NULL.

References CPL_ERROR_DATA_NOT_FOUND, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_frame_dump(), cpl_imagelist_load_frameset(), cpl_multiframe_append_datagroup(), cpl_↩ multiframe_append_datagroup_from_position(), cpl_multiframe_append_dataset(), and cpl_multiframe_append↩ _dataset_from_position().

4.14.5.5 cpl_frame_get_group()

```
cpl_frame_group cpl_frame_get_group (
            const cpl_frame * self )
```

Get the current group of a frame.

Parameters

self	A frame.

Returns

The frame's current group. The function returns `CPL_FRAME_GROUP_NONE` if an error occurs and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function returns the group attribute of the frame *self*.

References CPL_ERROR_NULL_INPUT, and CPL_FRAME_GROUP_NONE.

Referenced by cpl_dfs_sign_products(), and cpl_frame_dump().

4.14.5.6 cpl_frame_get_level()

```
cpl_frame_level cpl_frame_get_level (
            const cpl_frame * self )
```

Get the current level of a frame.

Parameters

self	A frame.

Returns

The frame's current level. The function returns `CPL_FRAME_LEVEL_NONE` if an error occurs and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function returns the level attribute of the frame *self*.

References CPL_ERROR_NULL_INPUT, and CPL_FRAME_LEVEL_NONE.

Referenced by cpl_frame_dump().

4.14.5.7 cpl_frame_get_nextensions()

```
cpl_size cpl_frame_get_nextensions (
            const cpl_frame * self )
```

Get the number of extensions of this frame.

Parameters

self	A frame.

Returns

The number of extensions in the file

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The frame *self* is not associated to a file.

The function returns the number of extensions in the frame or -1 in case of error.

References CPL_ERROR_DATA_NOT_FOUND, CPL_ERROR_NULL_INPUT, cpl_error_set, and cpl_fits_count↩_extensions().

4.14.5.8 cpl_frame_get_tag()

```
const char* cpl_frame_get_tag (
            const cpl_frame * self )
```

Get the category tag of a frame.

Parameters

self	A frame.

Returns

The frame's category tag or NULL if the tag is not set. The function returns NULL if an error occurs and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function returns the read-only frame's category tag. If a tag has not yet been set a NULL pointer is returned.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_frame_dump(), cpl_frameset_duplicate(), cpl_frameset_erase_frame(), and cpl_frameset_↩ insert().

4.14.5.9 cpl_frame_get_type()

```
cpl_frame_type cpl_frame_get_type (
            const cpl_frame * self )
```

Get the type of a frame.

Parameters

self	A frame.

Returns

The frame's type. The returns CPL_FRAME_TYPE_NONE if an error occurs and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function returns the type of the data object to which it currently refers.

References CPL_ERROR_NULL_INPUT, and CPL_FRAME_TYPE_NONE.

Referenced by cpl_frame_dump().

4.14.5.10 cpl_frame_new()

```
cpl_frame* cpl_frame_new (
            void  )
```

Create a new, empty frame.

Returns

A handle for the newly created frame.

The function allocates the memory for the new frame and initializes it to an empty frame, i.e. it is created without tag and file information, and the type, group and level set to CPL_FRAME_TYPE_NONE, CPL_FRAME_GROUP↩_NONE, and CPL_FRAME_LEVEL_NONE, respectively.

References CPL_FRAME_GROUP_NONE, CPL_FRAME_LEVEL_NONE, and CPL_FRAME_TYPE_NONE.

4.14.5.11 cpl_frame_set_filename()

```
cpl_error_code cpl_frame_set_filename (
            cpl_frame * self,
            const char * filename )
```

Set the file name to which a frame refers.

Parameters

self	A frame.
filename	The new file name.

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *filename* is a NULL pointer.

The function sets the name of the file, to which the frame *self* refers. Any file name which was previously set by a call to this function is replaced. If no file name is present yet it is created and initialised to *filename*.

References CPL_ERROR_NULL_INPUT.

4.14.5.12 cpl_frame_set_group()

```
cpl_error_code cpl_frame_set_group (
            cpl_frame * self,
            cpl_frame_group group )
```

Set the group attribute of a frame.

Parameters

self	A frame.
group	New group attribute.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function sets the group attribute of the frame *self* to *group*.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.14.5.13 cpl_frame_set_level()

```
cpl_error_code cpl_frame_set_level (
            cpl_frame * self,
            cpl_frame_level level )
```

Set the level attribute of a frame.

Parameters

self	A frame.
level	New level attribute.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function sets the level attribute of the frame *self* to *level*.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.14.5.14 cpl_frame_set_tag()

```
cpl_error_code cpl_frame_set_tag (
            cpl_frame * self,
            const char * tag )
```

Set a frame's category tag.

Parameters

self	A frame.
tag	The new category tag.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *tag* is a `NULL` pointer.

The function sets the category tag of *self*, replacing any previously set tag. If the frame does not yet have a tag is is created and initialised to *tag*.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.14.5.15 cpl_frame_set_type()

```
cpl_error_code cpl_frame_set_type (
            cpl_frame * self,
            cpl_frame_type type )
```

Set the type of a frame.

Parameters

self	A frame.
type	New frame type.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function sets the type of the frame *self* to *type*.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.15 Fundamental math functionality

Macros

- #define CPL_MATH_1_PI 0.31830988618379067153776752674502872406891929148091 29
  1/pi
- #define CPL_MATH_2_PI 0.63661977236758134307553505349005744813783858296182 58
  2/pi
- #define CPL_MATH_2_SQRTPI 1.12837916709551257389615890312154517168810125865799 77
  2/sqrt(pi)
- #define CPL_MATH_2PI 6.28318530717958647692528676655900576839433879875021 16
  2 pi
- #define CPL_MATH_4_PI 1.27323954473516268615107010698011489627567716592365 16
  4/pi
- #define CPL_MATH_DEG_RAD 57.29577951308232087679815481410517033240547246656432 2
  180/pi
- #define CPL_MATH_E 2.71828182845904523536028747135266249775724709369995 95
  The base of the exponential function.
- #define CPL_MATH_FWHM_SIG 2.35482004503094938202313865291939927549477137877164 11
  FWHM per Sigma, 2.0∗sqrt(2.0∗log(2.0))
- #define CPL_MATH_LN10 2.30258509299404568401799145468436420760110148862877 30
  The natural logarithm of 10.
- #define CPL_MATH_LN2 0.69314718055994530941723212145817656807550013436025 53
  The natural logarithm of 2.
- #define CPL_MATH_LOG10E 0.43429448190325182765112891891660508229439700580366 66
  log10(e)
- #define CPL_MATH_LOG2E 1.44269504088896340735992468100189213742664595415298 59
  log2(e)
- #define CPL_MATH_PI 3.14159265358979323846264338327950288419716939937510 58
  The ratio of a circles circumference to its diameter.
- #define CPL_MATH_PI_2 1.57079632679489661923132169163975144209858469968755 29
  pi/2
- #define CPL_MATH_PI_4 0.78539816339744830961566084581987572104929234984377 65
  pi/4
- #define CPL_MATH_RAD_DEG 0.01745329251994329576923690768488612713442871888541 73
  pi/180
- #define CPL_MATH_SIG_FWHM 0.42466090014400952136075141705144480985757054689217 70
  Sigma per FWHM, 0.5/sqrt(2.0∗log(2.0))
- #define CPL_MATH_SQRT1_2 0.70710678118654752440084436210484903928483593768847 40
  sqrt(1/2)
- #define CPL_MATH_SQRT2 1.41421356237309504880168872420969807856967187537694 81
  The square root of 2.
- #define CPL_MATH_SQRT2PI 2.50662827463100050241576528481104525300698674060993 83
  sqrt(2pi)
- #define CPL_MATH_SQRT3 1.73205080756887729352744634150587236694280525381038 06
  The square root of 3.
- #define CPL_MATH_STD_MAD 1.4826
  Standard deviation per Median Absolute Deviation for Gaussian data.
- #define CPL_MAX(first, second)
  Return the maximum of two values.
- #define CPL_MIN(first, second)
  Return the minimum of two values.

4.15.1 Detailed Description

This module provides fundamental math constants.

Source: On-Line Encyclopedia of Integer Sequences (OEIS)

pi: http://www.research.att.com/~njas/sequences/A000796

e: http://www.research.att.com/~njas/sequences/A001113

ln(2): http://www.research.att.com/~njas/sequences/A002162

ln(10): http://www.research.att.com/~njas/sequences/A002392

sqrt(2): http://www.research.att.com/~njas/sequences/A002193

sqrt(3): http://www.research.att.com/~njas/sequences/A002194

The derived constants have been computed with the GNU Multiple-Precision Library v. 4.2.2.

The constants are listed with a precision that allows a one-line definition.

Synopsis:

```
#include <cpl_math_const.h>
```

4.15.2 Macro Definition Documentation

4.15.2.1 CPL_MATH_1_PI

```
#define CPL_MATH_1_PI 0.31830988618379067153776752674502872406891929148091 29
```

1/pi

See also

CPL_MATH_PI

Note

Derived from a fundamental constant

4.15.2.2 CPL_MATH_2_PI

```
#define CPL_MATH_2_PI 0.63661977236758134307553505349005744813783858296182 58
```

2/pi

See also

CPL_MATH_PI

Note

Derived from a fundamental constant

4.15.2.3 CPL_MATH_2_SQRTPI

```
#define CPL_MATH_2_SQRTPI 1.12837916709551257389615890312154517168810125865799 77
```

2/sqrt(pi)

See also

CPL_MATH_PI

Note

Derived from a fundamental constant

4.15.2.4 CPL_MATH_2PI

```
#define CPL_MATH_2PI 6.28318530717958647692528676655900576839433879875021 16
```

2 pi

See also

CPL_MATH_PI

Note

Derived from a fundamental constant

4.15.2.5 CPL_MATH_4_PI

```
#define CPL_MATH_4_PI 1.27323954473516268615107010698011489627567716592365 16
```

4/pi

See also

CPL_MATH_PI

Note

Derived from a fundamental constant

4.15.2.6 CPL_MATH_DEG_RAD

```
#define CPL_MATH_DEG_RAD 57.29577951308232087679815481410517033240547246656 4322
```

180/pi

See also

CPL_MATH_PI

Note

Derived from a fundamental constant

4.15.2.7 CPL_MATH_E

```
#define CPL_MATH_E 2.71828182845904523536028747135266249775724709369995 95
```

The base of the exponential function.

See also

On-Line Encyclopedia of Integer Sequences (OEIS), http://www.research.att.com/~njas/sequences/↩A001113

4.15.2.8 CPL_MATH_FWHM_SIG

```
#define CPL_MATH_FWHM_SIG 2.354820045030949382023138652919399275494771378771641 1
```

FWHM per Sigma, 2.0∗sqrt(2.0∗log(2.0))

See also

CPL_MATH_LN2

Note

Derived from a fundamental constant

4.15.2.9 CPL_MATH_LN10

```
#define CPL_MATH_LN10 2.302585092994045684017991454684364207601101488628773 0
```

The natural logarithm of 10.

See also

On-Line Encyclopedia of Integer Sequences (OEIS), http://www.research.att.com/~njas/sequences/↩A002392

4.15.2.10 CPL_MATH_LN2

```
#define CPL_MATH_LN2 0.693147180559945309417232121458176568075500134360255 3
```

The natural logarithm of 2.

See also

On-Line Encyclopedia of Integer Sequences (OEIS), http://www.research.att.com/~njas/sequences/↩A002162

4.15.2.11 CPL_MATH_LOG10E

```
#define CPL_MATH_LOG10E 0.434294481903251827651128918916605082294397005803666 6
```

log10(e)

See also

CPL_MATH_LN10

Note

Derived from a fundamental constant

4.15.2.12 CPL_MATH_LOG2E

```
#define CPL_MATH_LOG2E 1.44269504088896340735992468100189213742664595415298593
```

log2(e)

See also

CPL_MATH_LN2

Note

Derived from a fundamental constant

4.15.2.13 CPL_MATH_PI

```
#define CPL_MATH_PI 3.14159265358979323846264338327950288419716939937510582
```

The ratio of a circles circumference to its diameter.

See also

On-Line Encyclopedia of Integer Sequences (OEIS), http://www.research.att.com/~njas/sequences/↩
A000796

4.15.2.14 CPL_MATH_PI_2

```
#define CPL_MATH_PI_2 1.57079632679489661923132169163975144209858469968755291
```

pi/2

See also

CPL_MATH_PI

Note

Derived from a fundamental constant

4.15.2.15 CPL_MATH_PI_4

```
#define CPL_MATH_PI_4 0.78539816339744830961566084581987572104929234984377 65
```

pi/4

See also

CPL_MATH_PI

Note

Derived from a fundamental constant

4.15.2.16 CPL_MATH_RAD_DEG

```
#define CPL_MATH_RAD_DEG 0.01745329251994329576923690768488612713442871888541 73
```

pi/180

See also

CPL_MATH_PI

Note

Derived from a fundamental constant

4.15.2.17 CPL_MATH_SIG_FWHM

```
#define CPL_MATH_SIG_FWHM 0.42466090014400952136075141705144480985757054689217 70
```

Sigma per FWHM, 0.5/sqrt(2.0∗log(2.0))

See also

CPL_MATH_LN2

Note

Derived from a fundamental constant

4.15.2.18 CPL_MATH_SQRT1_2

```
#define CPL_MATH_SQRT1_2 0.70710678118654752440084436210484903928483593768847 40
```

sqrt(1/2)

See also

CPL_MATH_SQRT2

Note

Derived from a fundamental constant

4.15.2.19 CPL_MATH_SQRT2

```
#define CPL_MATH_SQRT2 1.41421356237309504880168872420969807856967187537694 81
```

The square root of 2.

See also

On-Line Encyclopedia of Integer Sequences (OEIS), http://www.research.att.com/~njas/sequences/↩A002193

4.15.2.20 CPL_MATH_SQRT2PI

```
#define CPL_MATH_SQRT2PI 2.50662827463100050241576528481104525300698674060993 83
```

sqrt(2pi)

See also

CPL_MATH_PI

Note

Derived from a fundamental constant

4.15.2.21 CPL_MATH_SQRT3

```
#define CPL_MATH_SQRT3 1.732050807568877293527446341505872366942805253810380 6
```

The square root of 3.

See also

On-Line Encyclopedia of Integer Sequences (OEIS), http://www.research.att.com/~njas/sequences/↩ A002194

4.15.2.22 CPL_MATH_STD_MAD

```
#define CPL_MATH_STD_MAD 1.4826
```

Standard deviation per Median Absolute Deviation for Gaussian data.

See also

cpl_image_get_mad_window()

For a Gaussian distribution the Median Absolute Deviation (MAD) is a robust and consistent estimate of the Standard Deviation (STD) in the sense that the STD is approximately K ∗ MAD, where K is a constant equal to approximately 1.4826.

4.15.2.23 CPL_MAX

```
#define CPL_MAX(
              first,
              second )
```

Return the maximum of two values.

Parameters

first	The first expression in the comparison
second	The second expression in the comparison

Returns

The maximum of the two values

See also

CPL_MIN()

4.15.2.24 CPL_MIN

```
#define CPL_MIN(
              first,
              second )
```

Return the minimum of two values.

Parameters

first	The first expression in the comparison
second	The second expression in the comparison

Returns

The minimum of the two values

Note

If the first argument is the smallest then it is evaluated twice otherwise the second argument is evaluated twice

4.16 Handling of multiple CPL errors

Functions

- void cpl_errorstate_dump (cpl_errorstate self, cpl_boolean reverse, void(*dump_one)(unsigned, unsigned, unsigned))

  Dump the CPL error state.
- void cpl_errorstate_dump_one (unsigned self, unsigned first, unsigned last)

  Dump a single CPL error.
- void cpl_errorstate_dump_one_debug (unsigned self, unsigned first, unsigned last)

  Dump a single CPL error using cpl_msg_debug()
- void cpl_errorstate_dump_one_info (unsigned self, unsigned first, unsigned last)

  Dump a single CPL error using cpl_msg_info()
- void cpl_errorstate_dump_one_warning (unsigned self, unsigned first, unsigned last)

  Dump a single CPL error using cpl_msg_warning()
- cpl_errorstate cpl_errorstate_get (void)

  Get the CPL errorstate.
- cpl_boolean cpl_errorstate_is_equal (cpl_errorstate self)

  Detect a change in the CPL error state.
- void cpl_errorstate_set (cpl_errorstate self)

  Set the CPL errorstate.

4.16.1 Detailed Description

This module provides functions for error detection and recovery and for producing an error traceback.

It is built on top of the `cpl_error` module.

Synopsis:

```
#include <cpl_errorstate.h>
```

4.16.2 Function Documentation

4.16.2.1 cpl_errorstate_dump()

```
void cpl_errorstate_dump (
            cpl_errorstate self,
            cpl_boolean reverse,
            void(*)(unsigned, unsigned, unsigned) dump_one )
```

Dump the CPL error state.

Parameters

self	Dump errors more recent than self
reverse	Reverse the chronological order of the dump
dump_one	Function that dumps a single CPL error, or NULL

Returns

void

Note

dump_one may be NULL, in that case cpl_errorstate_dump_one() is called. If there are no CPL errors to be dumped, (∗dump_one)() is called once with all zeros, which allows the dump-function to report that there are no errors to dump. During calls to (∗dump_one)() the CPL error system has been put into a special read-only mode that prevents any change of the CPL error state. This means that any calls from (∗dump_one)() to cpl_error_reset(), cpl_error_set(), cpl_error_set_where() and cpl_errorstate_set() have no effect. Also calls from (∗dump_one)() to cpl_errorstate_dump() have no effect.

See also

cpl_errorstate_dump_one

CPL-based code with insufficient error checking can generate a large number of CPL errors. To avoid that a subsequent call to cpl_errorstate_dump() will fill up the output device, calls to the dump-function are skipped and only counted and the count reported when dump_one is NULL and the CPL error code is CPL_ERROR_HISTO↩ RY_LOST.

Example of usage:

```
cpl_errorstate prev_state = cpl_errorstate_get();
// Call one or more CPL functions
if (cpl_errorstate_is_equal(prev_state)) {
    // No error happened
} else {
    // One or more errors happened
    // Dump the error(s) in chronological order
    cpl_errorstate_dump(prev_state, CPL_FALSE,
                        cpl_errorstate_dump_one);
    // Recover from the error(s)
    cpl_errorstate_set(prev_state);
}
```

References cpl_errorstate_dump_one().

4.16.2.2 cpl_errorstate_dump_one()

```
void cpl_errorstate_dump_one (
             unsigned self,
             unsigned first,
             unsigned last )
```

Dump a single CPL error.

Parameters

self	The number of the current error to be dumped
first	The number of the first error to be dumped
last	The number of the last error to be dumped

Returns

void

Note

This function is implemented using only exported CPL functions.

See also

cpl_errorstate_dump

This function will dump a single CPL error, using the CPL error accessor functions. The error is numbered with the value of self.

The actual output is produced by cpl_msg_error().

first and last are provided in the API to allow for special messaging in the dump of the first and last errors.

Alternative functions for use with cpl_errorstate_dump() may use all accessor functions of the CPL error module and those functions of the CPL message module that produce messages. Additionally, the indentation functions of the CPL message module may be used, provided that the indentation is set back to its original state after the last error has been processed.

Referenced by cpl_errorstate_dump().

4.16.2.3 cpl_errorstate_dump_one_debug()

```
void cpl_errorstate_dump_one_debug (
             unsigned self,
             unsigned first,
             unsigned last )
```

Dump a single CPL error using cpl_msg_debug()

Parameters

self	The number of the current error to be dumped
first	The number of the first error to be dumped
last	The number of the last error to be dumped

Returns

void

See also

cpl_errorstate_dump_one, cpl_msg_debug()

4.16.2.4 cpl_errorstate_dump_one_info()

```
void cpl_errorstate_dump_one_info (
             unsigned self,
             unsigned first,
             unsigned last )
```

Dump a single CPL error using cpl_msg_info()

Parameters

self	The number of the current error to be dumped
first	The number of the first error to be dumped
last	The number of the last error to be dumped

Returns

void

See also

cpl_errorstate_dump_one, cpl_msg_info()

4.16.2.5 cpl_errorstate_dump_one_warning()

```
void cpl_errorstate_dump_one_warning (
             unsigned self,
             unsigned first,
             unsigned last )
```

Dump a single CPL error using cpl_msg_warning()

Parameters

self	The number of the current error to be dumped
first	The number of the first error to be dumped
last	The number of the last error to be dumped

Returns

void

See also

cpl_errorstate_dump_one, cpl_msg_warning()

4.16.2.6 cpl_errorstate_get()

```
cpl_errorstate cpl_errorstate_get (
            void  )
```

Get the CPL errorstate.

Returns

The CPL errorstate

Note

The caller should not modify the returned value nor transfer it to another function/scope.

Example of usage:

```
cpl_errorstate prev_state = cpl_errorstate_get();
// (Call one or more CPL functions here)
if (cpl_errorstate_is_equal(prev_state)) {
    // No error happened
} else {
    // One or more errors happened
    // Recover from the error(s)
    cpl_errorstate_set(prev_state);
}
```

Referenced by cpl_apertures_extract(), cpl_apertures_extract_mask(), cpl_apertures_get_fwhm(), cpl_array_get(), cpl_array_get_complex(), cpl_array_get_cplsize(), cpl_array_get_double(), cpl_array_get_double_complex(), cpl_array_get_float(), cpl_array_get_float_complex(), cpl_array_get_int(), cpl_array_get_long(), cpl_array_get_long_long(), cpl_array_get_max(), cpl_array_get_mean(), cpl_array_get_mean_complex(), cpl_array_get_median(), cpl_array_get_min(), cpl_array_get_stdev(), cpl_array_is_valid(), cpl_fit_image_gaussian(), cpl_frameset_dump(), cpl_frameset_extract(), cpl_frameset_find(), cpl_frameset_labelise(), cpl_gaussian_eval_2d(), cpl_geom_img_offset_fine(), cpl_image_get_fwhm(), cpl_imagelist_load_frameset(), cpl_mask_extract(), cpl_matrix_get_determinant(), cpl_parameterlist_find(), cpl_parameterlist_find_context(), cpl_parameterlist_find_tag(), cpl_parameterlist_find_type(), cpl_parameterlist_get_first(), cpl_parameterlist_get_last(), cpl_parameterlist_get_next(), cpl_plot_bivectors(), cpl_plot_column(), cpl_plot_columns(), cpl_plot_image_col(), cpl_plot_image_row(), cpl_plot_vectors(), cpl_propertylist_get(), cpl_table_get_column_max(), cpl_table_get_column_mean(), cpl_table_get_column_mean_complex(), cpl_table_get_column_min(), cpl_table_get_column_stdev(), cpl_table_has_invalid(), cpl_table_has_valid(), cpl_table_is_valid(), and cpl_wlcalib_find_best_1d().

4.16.2.7 cpl_errorstate_is_equal()

```
cpl_boolean cpl_errorstate_is_equal (
            cpl_errorstate self )
```

Detect a change in the CPL error state.

Parameters

self	The return value of a previous call to cpl_errorstate_get()

Returns

CPL_TRUE iff the current error state is equal to self.

See also

cpl_errorstate_get

Referenced by cpl_array_get(), cpl_array_get_complex(), cpl_array_get_cplsize(), cpl_array_get_double(), cpl_array_get_double_complex(), cpl_array_get_float(), cpl_array_get_float_complex(), cpl_array_get_int(), cpl_array_get_long(), cpl_array_get_long_long(), cpl_array_get_max(), cpl_array_get_mean(), cpl_array_get_mean_complex(), cpl_array_get_min(), cpl_array_get_stdev(), cpl_array_is_valid(), cpl_frameset_find(), cpl_gaussian_eval_2d(), cpl_image_get_fwhm(), cpl_mask_extract(), cpl_parameterlist_find(), cpl_parameterlist_find_context(), cpl_parameterlist_find_tag(), cpl_parameterlist_find_type(), cpl_parameterlist_get_first(), cpl_parameterlist_get_last(), cpl_parameterlist_get_next(), cpl_plot_column(), cpl_propertylist_get(), and cpl_wlcalib_find_best_1d().

4.16.2.8 cpl_errorstate_set()

```
void cpl_errorstate_set (
             cpl_errorstate self )
```

Set the CPL errorstate.

Parameters

self	The return value of a previous call to cpl_errorstate_get()

Returns

void

See also

cpl_errorstate_get

Note

If a CPL error was created before the call to cpl_errorstate_get() that returned self and if more than CPL_ERROR_HISTORY_SIZE CPL errors was created after that, then the accessor functions of the CPL error object (cpl_error_get_code() etc.) will return wrong values. In this case cpl_error_get_code() is still guaranteed not to return CPL_ERROR_NONE. Moreover, the default value of CPL_ERROR_HISTORY_SIZE is guaranteed to be large enough to prevent this situation from arising due to a call of a CPL function.

Referenced by cpl_apertures_extract(), cpl_apertures_get_fwhm(), cpl_frameset_dump(), cpl_frameset_extract(), and cpl_frameset_labelise().

4.17 High level functions for geometric transformations

Functions

- cpl_image ** cpl_geom_img_offset_combine (const cpl_imagelist *self, const cpl_bivector *offs, int refine, const cpl_bivector *aperts, const cpl_vector *sigmas, cpl_size *pisigma, cpl_size s_hx, cpl_size s_hy, cpl_size m_hx, cpl_size m_hy, cpl_size min_rej, cpl_size max_rej, cpl_geom_combine union_flag)

  Images list recombination.
- cpl_bivector * cpl_geom_img_offset_fine (const cpl_imagelist *ilist, const cpl_bivector *estimates, const cpl_bivector *anchors, cpl_size s_hx, cpl_size s_hy, cpl_size m_hx, cpl_size m_hy, cpl_vector *correl)

  Get the offsets by correlating the images.
- cpl_image ** cpl_geom_img_offset_saa (const cpl_imagelist *ilist, const cpl_bivector *offs, cpl_kernel kernel, cpl_size rejmin, cpl_size rejmax, cpl_geom_combine union_flag, double *ppos_x, double *ppos_y)

  Shift and add an images list to a single image.

- enum cpl_geom_combine {
  CPL_GEOM_INTERSECT,
  CPL_GEOM_UNION,
  CPL_GEOM_FIRST }

  The CPL Geometry combination modes.

4.17.1 Detailed Description

This module contains functions to compute the shift-and-add operation on an image list.

Synopsis:

```
#include "cpl_geom_img.h"
```

4.17.2 Enumeration Type Documentation

4.17.2.1 cpl_geom_combine

```
enum cpl_geom_combine
```

The CPL Geometry combination modes.

Enumerator

CPL_GEOM_INTERSECT	Combine using the intersection of the images.
CPL_GEOM_UNION	Combine using the union of the images.
CPL_GEOM_FIRST	Combine using the first image to aggregate the other ones.

4.17.3 Function Documentation

4.17.3.1 cpl_geom_img_offset_combine()

```
cpl_image** cpl_geom_img_offset_combine (
             const cpl_imagelist * self,
             const cpl_bivector * offs,
             int refine,
             const cpl_bivector * aperts,
             const cpl_vector * sigmas,
             cpl_size * pisigma,
             cpl_size s_hx,
             cpl_size s_hy,
             cpl_size m_hx,
             cpl_size m_hy,
             cpl_size min_rej,
             cpl_size max_rej,
             cpl_geom_combine union_flag )
```

Images list recombination.

Parameters

self	Input imagelist
offs	List of offsets in x and y
refine	Iff non-zero, the offsets will be refined
aperts	List of correlation apertures or NULL if unknown
sigmas	Positive, decreasing sigmas to apply
pisigma	Index of the sigma that was used or undefined on error
s_hx	Search area half-width.
s_hy	Search area half-height.
m_hx	Measurement area half-width.
m_hy	Measurement area half-height.
min_rej	number of low values to reject in stacking
max_rej	number of high values to reject in stacking
union_flag	Combination mode (CPL_GEOM_UNION or CPL_GEOM_INTERSECT)

Returns

Pointer to newly allocated images array, or NULL on error.

See also

cpl_geom_img_offset_saa()

cpl_apertures_extract()

cpl_geom_img_offset_fine()

With offset refinement enabled: This function detects cross correlation points in the first image (if not provided by the user), use them to refine the provided offsets with a cross correlation method, and then apply the shift and add to recombine the images together.

The supported types are CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT.

The number of provided offsets shall be equal to the number of input images. The ith offset (offs_x, offs_y) is the offset that has to be used to shift the ith image to align it on the first one.

sigmas may be NULL if offset refinement is disabled or if aperts is non-NULL.

On success the returned image array contains 2 images:

- the combined image
- the contribution map

The returned cpl_image array must be deallocated like this:

```
if (array != NULL) {
    cpl_image_delete(array[0]);
    cpl_image_delete(array[1]);
    cpl_free(array);
}
```

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if self or offs is NULL, or if sigmas is NULL with refinement enabled and aperts NULL.
- CPL_ERROR_ILLEGAL_INPUT if self is not uniform
- CPL_ERROR_INCOMPATIBLE_INPUT if self and offs have different sizes
- CPL_ERROR_DATA_NOT_FOUND if the shift and add of the images fails

References cpl_apertures_delete(), cpl_apertures_extract(), cpl_apertures_get_pos_x(), cpl_apertures_get_pos_y(), cpl_apertures_sort_by_npix(), cpl_bivector_delete(), cpl_bivector_get_size(), cpl_bivector_get_x(), cpl_bivector_get_x_data(), cpl_bivector_get_y(), cpl_bivector_get_y_data(), cpl_bivector_new(), cpl_ensure, CPL_ERROR_DATA_NOT_FOUND, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NULL_INPUT, cpl_geom_img_offset_fine(), cpl_geom_img_offset_saa(), cpl_imagelist_delete(), cpl_imagelist_get_const(), cpl_imagelist_get_size(), cpl_imagelist_is_uniform(), cpl_imagelist_new(), cpl_imagelist_set(), cpl_imagelist_unset(), CPL_SIZE_FORMAT, cpl_vector_delete(), cpl_vector_get_data_const(), cpl_vector_new(), and cpl_vector_set_size().

4.17.3.2 cpl_geom_img_offset_fine()

```
cpl_bivector* cpl_geom_img_offset_fine (
             const cpl_imagelist * ilist,
             const cpl_bivector * estimates,
             const cpl_bivector * anchors,
             cpl_size s_hx,
             cpl_size s_hy,
             cpl_size m_hx,
             cpl_size m_hy,
             cpl_vector * correl )
```

Get the offsets by correlating the images.

Parameters

ilist	Input image list
estimates	First-guess estimation of the offsets
anchors	List of anchor points
s_hx	Half-width of search area
s_hy	Half-height of search area
m_hx	Half-width of measurement area
m_hy	Half-height of measurement area
correl	List of cross-correlation quality factors

Returns

List of offsets or NULL on error

The matching is performed using a 2d cross-correlation, using a minimal squared differences criterion. One measurement is performed per input anchor point, and the median offset is returned together with a measure of similarity for each plane.

The images in the input list must only differ from a shift. In order from the correlation to work, they must have the same level (check the average values of your input images if the correlation does not work).

The supported types are CPL_TYPE_DOUBLE and CPL_TYPE_FLOAT. The bad pixel maps are ignored by this function.

The ith offset (offsx, offsy) in the returned offsets is the one that have to be used to shift the ith image to align it on the reference image (the first one).

If not NULL, the returned cpl_bivector must be deallocated with cpl_bivector_delete().

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if (one of) the input pointer(s) is NULL
- CPL_ERROR_ILLEGAL_INPUT if ilist is not valid
- CPL_ERROR_INCOMPATIBLE_INPUT if ilist and correl have different lengths

References cpl_bivector_get_size(), cpl_bivector_get_x_data(), cpl_bivector_get_x_data_const(), cpl_bivector_get_y_data(), cpl_bivector_get_y_data_const(), cpl_bivector_new(), CPL_BORDER_FILTER, cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), CPL_FILTER_MEDIAN, cpl_image_filter_mask(), cpl_image_get_size_x(), cpl_image_get_size_y(), cpl_image_get_type(), cpl_image_new(), cpl_imagelist_get_const(), cpl_imagelist_get_size(), cpl_imagelist_is_uniform(), cpl_mask_new(), cpl_mask_not(), cpl_vector_get_data(), and cpl_vector_get_size().

Referenced by cpl_geom_img_offset_combine().

4.17.3.3 cpl_geom_img_offset_saa()

```
cpl_image** cpl_geom_img_offset_saa (
             const cpl_imagelist * ilist,
             const cpl_bivector * offs,
             cpl_kernel kernel,
             cpl_size rejmin,
             cpl_size rejmax,
             cpl_geom_combine union_flag,
             double * ppos_x,
             double * ppos_y )
```

Shift and add an images list to a single image.

Parameters

ilist	Input image list
offs	List of offsets in x and y
kernel	Interpolation kernel to use for resampling
rejmin	Number of minimum value pixels to reject in stacking
rejmax	Number of maximum value pixels to reject in stacking
union_flag	Combination mode, CPL_GEOM_UNION, CPL_GEOM_INTERSECT or CPL_GEOM_FIRST
ppos_x	If non-NULL, ∗ppos_x is the X-position of the first image in the combined image
ppos_y	If non-NULL, ∗ppos_y is the Y- position of the first image in the combined image

Returns

Pointer to newly allocated images array, or NULL on error.

The supported types are CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT.

The number of provided offsets shall be equal to the number of input images. The ith offset (offs_x, offs_y) is the offset that has to be used to shift the ith image to align it on the first one.

Provide the name of the kernel you want to generate. Supported kernel types are:

- CPL_KERNEL_DEFAULT: default kernel, currently CPL_KERNEL_TANH
- CPL_KERNEL_TANH: Hyperbolic tangent
- CPL_KERNEL_SINC: Sinus cardinal
- CPL_KERNEL_SINC2: Square sinus cardinal
- CPL_KERNEL_LANCZOS: Lanczos2 kernel
- CPL_KERNEL_HAMMING: Hamming kernel
- CPL_KERNEL_HANN: Hann kernel
- CPL_KERNEL_NEAREST: Nearest neighbor kernel (1 when dist $<$ 0.5, else 0)

If the number of input images is lower or equal to 3, the rejection parameters are ignored. If the number of input images is lower or equal to 2∗(rejmin+rejmax), the rejection parameters are ignored.

On success the returned image array contains 2 images:

- the combined image
- the contribution map

Pixels with a zero in the contribution map are flagged as bad in the combined image.

If not NULL, the returned cpl_image array arr must be deallocated like:

```
if (arr[0] != NULL) cpl_image_delete(arr[0]);
if (arr[1] != NULL) cpl_image_delete(arr[1]);
cpl_free(arr);
```

If the call is successful, (∗ppos_x, ∗ppos_y) is the pixel coordinate in the created output image-pair where the lowermost-leftmost pixel of the first input image is located. So with CPL_GEOM_FIRST this will always be (1, 1).

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if (one of) the input pointer(s) is NULL
- CPL_ERROR_ILLEGAL_INPUT if ilist is invalid or if rejmin or rejmax is negative
- CPL_ERROR_INCOMPATIBLE_INPUT if ilist and offs have different sizes
- CPL_ERROR_ILLEGAL_OUTPUT if the CPL_GEOM_INTERSECT method is used with non-overlapping images.
- CPL_ERROR_INVALID_TYPE if the passed image list type is not supported
- CPL_ERROR_UNSUPPORTED_MODE if the union_flag is not one of the supported combination modes, which are *CPL_GEOM_INTERSECT*, *CPL_GEOM_UNION*, *CPL_GEOM_FIRST*.

References cpl_bivector_get_size(), cpl_bivector_get_x_const(), cpl_bivector_get_y_const(), cpl_ensure, CPL_E↩ RROR_ILLEGAL_INPUT, CPL_ERROR_ILLEGAL_OUTPUT, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ER↩ ROR_NULL_INPUT, CPL_GEOM_FIRST, CPL_GEOM_INTERSECT, CPL_GEOM_UNION, cpl_image_get_size↩ _x(), cpl_image_get_size_y(), cpl_image_get_type(), cpl_imagelist_get_const(), cpl_imagelist_get_size(), cpl_↩ imagelist_is_uniform(), cpl_vector_delete(), cpl_vector_duplicate(), cpl_vector_get_data_const(), cpl_vector_get↩ _max(), cpl_vector_get_min(), and cpl_vector_sort().

Referenced by cpl_geom_img_offset_combine().

4.18 High level functions to handle apertures

Functions

- void cpl_apertures_delete (cpl_apertures *self)

  Destructor for cpl_apertures.
- void cpl_apertures_dump (const cpl_apertures *self, FILE *fp)

  Dump a cpl_apertures to an opened file pointer.
- cpl_apertures * cpl_apertures_extract (const cpl_image *self, const cpl_vector *sigmas, cpl_size *pisigma)

  Simple detection of apertures in an image.
- cpl_apertures * cpl_apertures_extract_mask (const cpl_image *self, const cpl_mask *selection)

  Simple apertures creation from a user supplied selection mask.
- cpl_apertures * cpl_apertures_extract_sigma (const cpl_image *self, double sigma)

  Simple apertures detection in an image using a provided sigma.
- cpl_apertures * cpl_apertures_extract_window (const cpl_image *self, const cpl_vector *sigmas, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury, cpl_size *pisigma)

  Simple detection of apertures in an image window.
- cpl_size cpl_apertures_get_bottom (const cpl_apertures *self, cpl_size ind)

  Get the bottommost y position in an aperture.
- cpl_size cpl_apertures_get_bottom_x (const cpl_apertures *self, cpl_size ind)

  Get the x position of the bottommost y position in an aperture.
- double cpl_apertures_get_centroid_x (const cpl_apertures *self, cpl_size ind)

  Get the X-centroid of an aperture.
- double cpl_apertures_get_centroid_y (const cpl_apertures *self, cpl_size ind)

  Get the Y-centroid of an aperture.
- double cpl_apertures_get_flux (const cpl_apertures *self, cpl_size ind)

  Get the flux of an aperture.
- cpl_bivector * cpl_apertures_get_fwhm (const cpl_image *self, const cpl_apertures *aperts)

  Compute FWHM values in x and y for a list of apertures.
- cpl_size cpl_apertures_get_left (const cpl_apertures *self, cpl_size ind)

  Get the leftmost x position in an aperture.
- cpl_size cpl_apertures_get_left_y (const cpl_apertures *self, cpl_size ind)

  Get the y position of the leftmost x position in an aperture.
- double cpl_apertures_get_max (const cpl_apertures *self, cpl_size ind)

  Get the maximum value of an aperture.
- double cpl_apertures_get_max_x (const cpl_apertures *self, cpl_size ind)

  Get the average X-position of an aperture.
- double cpl_apertures_get_max_y (const cpl_apertures *self, cpl_size ind)

  Get the average Y-position of an aperture.
- cpl_size cpl_apertures_get_maxpos_x (const cpl_apertures *self, cpl_size ind)

  Get the X-position of the aperture maximum value.
- cpl_size cpl_apertures_get_maxpos_y (const cpl_apertures *self, cpl_size ind)

  Get the Y-position of the aperture maximum value.
- double cpl_apertures_get_mean (const cpl_apertures *self, cpl_size ind)

  Get the mean value of an aperture.
- double cpl_apertures_get_median (const cpl_apertures *self, cpl_size ind)

  Get the median value of an aperture.
- double cpl_apertures_get_min (const cpl_apertures *self, cpl_size ind)

  Get the minimum value of an aperture.
- cpl_size cpl_apertures_get_minpos_x (const cpl_apertures *self, cpl_size ind)

Get the X-position of the aperture minimum value.

- cpl_size cpl_apertures_get_minpos_y (const cpl_apertures ∗self, cpl_size ind)

  Get the Y-position of the aperture minimum value.
- cpl_size cpl_apertures_get_npix (const cpl_apertures ∗self, cpl_size ind)

  Get the number of pixels of an aperture.
- double cpl_apertures_get_pos_x (const cpl_apertures ∗self, cpl_size ind)

  Get the average X-position of an aperture.
- double cpl_apertures_get_pos_y (const cpl_apertures ∗self, cpl_size ind)

  Get the average Y-position of an aperture.
- cpl_size cpl_apertures_get_right (const cpl_apertures ∗self, cpl_size ind)

  Get the rightmost x position in an aperture.
- cpl_size cpl_apertures_get_right_y (const cpl_apertures ∗self, cpl_size ind)

  Get the y position of the rightmost x position in an aperture.
- cpl_size cpl_apertures_get_size (const cpl_apertures ∗self)

  Get the number of apertures.
- double cpl_apertures_get_stdev (const cpl_apertures ∗self, cpl_size ind)

  Get the standard deviation of an aperture.
- cpl_size cpl_apertures_get_top (const cpl_apertures ∗self, cpl_size ind)

  Get the topmost y position in an aperture.
- cpl_size cpl_apertures_get_top_x (const cpl_apertures ∗self, cpl_size ind)

  Get the x position of the topmost y position in an aperture.
- cpl_apertures ∗ cpl_apertures_new_from_image (const cpl_image ∗self, const cpl_image ∗lab)

  Compute statistics on selected apertures.
- cpl_error_code cpl_apertures_sort_by_flux (cpl_apertures ∗self)

  Sort by decreasing aperture flux.
- cpl_error_code cpl_apertures_sort_by_max (cpl_apertures ∗self)

  Sort by decreasing aperture peak value.
- cpl_error_code cpl_apertures_sort_by_npix (cpl_apertures ∗self)

  Sort by decreasing aperture size.

4.18.1 Detailed Description

The aperture object contains a list of zones in an image. It is typically used to contain the results of an objects detection, or if one wants to work on a very specific zone in an image.

This module provides functions to handle *cpl_apertures*.

4.18.2 Function Documentation

4.18.2.1 cpl_apertures_delete()

```
void cpl_apertures_delete (
            cpl_apertures * self )
```

Destructor for cpl_apertures.

Parameters

self	The object to delete.

Returns

void

Note

If *self* is NULL, nothing is done and no error is set.

This function deallocates all memory allocated for the object.

References cpl_free().

Referenced by cpl_geom_img_offset_combine().

4.18.2.2 cpl_apertures_dump()

```
void cpl_apertures_dump (
            const cpl_apertures * self,
            FILE * fp )
```

Dump a cpl_apertures to an opened file pointer.

Parameters

self	The cpl_apertures to dump
fp	File pointer, may use *stdout* or *stderr*

Returns

void

This function dumps all information in a cpl_apertures to the passed file pointer. If the object is unallocated or contains nothing, this function does nothing.

References CPL_SIZE_FORMAT.

4.18.2.3 cpl_apertures_extract()

```
cpl_apertures* cpl_apertures_extract (
            const cpl_image * self,
            const cpl_vector * sigmas,
            cpl_size * pisigma )
```

Simple detection of apertures in an image.

Parameters

self	The image to process
sigmas	Positive, decreasing sigmas to apply
pisigma	Index of the sigma that was used or unchanged on error

Returns

The detected apertures or NULL on error

See also

cpl_apertures_extract_sigma()

pisigma may be NULL.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if self or sigmas is NULL
- CPL_ERROR_DATA_NOT_FOUND if the apertures cannot be detected

References cpl_apertures_extract_sigma(), cpl_ensure, CPL_ERROR_DATA_NOT_FOUND, CPL_ERROR_NU↩LL_INPUT, cpl_errorstate_get(), cpl_errorstate_set(), cpl_vector_get(), and cpl_vector_get_size().

Referenced by cpl_apertures_extract_window(), and cpl_geom_img_offset_combine().

4.18.2.4 cpl_apertures_extract_mask()

```
cpl_apertures* cpl_apertures_extract_mask (
            const cpl_image * self,
            const cpl_mask * selection )
```

Simple apertures creation from a user supplied selection mask.

Parameters

self	The image to process
selection	The mask of selected pixels

Returns

The list of detected apertures or NULL if nothing detected or on error.

See also

cpl_image_labelise_mask_create(), cpl_apertures_new_from_image()

The values selected for inclusion in the apertures must have the non-zero value in the selection mask, and must not be flagged as bad in the bad pixel map of the image.

The input image type can be CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT or CPL_TYPE_INT.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if self and selection have different sizes.
- CPL_ERROR_TYPE_MISMATCH if self is of a complex type
- CPL_ERROR_DATA_NOT_FOUND if the selection mask is empty

References cpl_errorstate_get(), and cpl_image_labelise_mask_create().

4.18.2.5 cpl_apertures_extract_sigma()

```
cpl_apertures* cpl_apertures_extract_sigma (
            const cpl_image * self,
            double sigma )
```

Simple apertures detection in an image using a provided sigma.

Parameters

self	The image to process
sigma	Detection level

Returns

The list of detected apertures or NULL on error

Note

In order to avoid (the potentially many) detections of small objects the mask of detected pixels is subjected to a 3x3 morphological opening filter.

See also

cpl_apertures_extract_mask(), cpl_mask_filter()

The threshold used for the detection is the median plus the average distance to the median times sigma.

The input image type can be CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT or CPL_TYPE_INT.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if sigma is non-positive
- CPL_ERROR_TYPE_MISMATCH if self is of a complex type
- CPL_ERROR_DATA_NOT_FOUND if the no apertures are found

Referenced by cpl_apertures_extract().

4.18.2.6 cpl_apertures_extract_window()

```
cpl_apertures* cpl_apertures_extract_window (
             const cpl_image * self,
             const cpl_vector * sigmas,
             cpl_size llx,
             cpl_size lly,
             cpl_size urx,
             cpl_size ury,
             cpl_size * pisigma )
```

Simple detection of apertures in an image window.

Parameters

self	The image to process
sigmas	Positive, decreasing sigmas to apply
llx	Lower left x position (FITS convention)
lly	Lower left y position (FITS convention)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)
pisigma	Index of the sigma that was used or undefined on error

Returns

The list of detected apertures or NULL on error

See also

cpl_apertures_extract()

cpl_image_extract()

References cpl_apertures_extract(), cpl_ensure, cpl_error_get_code(), CPL_ERROR_NULL_INPUT, cpl_image↩_delete(), and cpl_image_extract().

4.18.2.7 cpl_apertures_get_bottom()

```
cpl_size cpl_apertures_get_bottom (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the bottommost y position in an aperture.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

the bottommost y position in the aperture or negative on error

Note

In case of an error the _cpl_error_code_ code is set

See also

cpl_apertures_get_maxpos_x()

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩_ERROR_NULL_INPUT.

4.18.2.8 cpl_apertures_get_bottom_x()

```
cpl_size cpl_apertures_get_bottom_x (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the x position of the bottommost y position in an aperture.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

the bottommost x position of the aperture or negative on error

Note

An aperture may have multiple bottom x positions, in which case one of these is returned.

See also

cpl_apertures_get_maxpos_x()

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩_ERROR_NULL_INPUT.

4.18.2.9 cpl_apertures_get_centroid_x()

```
double cpl_apertures_get_centroid_x (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the X-centroid of an aperture.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

The X-centroid position of the aperture or negative on error

Note

In case of an error the _cpl_error_code_ code is set

For a concave aperture the centroid may not belong to the aperture.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if ind is non-positive
- CPL_ERROR_ACCESS_OUT_OF_RANGE if ind exceeds the number of apertures in self

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩_ERROR_NULL_INPUT.

4.18.2.10 cpl_apertures_get_centroid_y()

```
double cpl_apertures_get_centroid_y (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the Y-centroid of an aperture.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

The X-centroid position of the aperture or negative on error

Note

In case of an error the _cpl_error_code_ code is set

See also

cpl_apertures_get_centroid_x()

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩ _ERROR_NULL_INPUT.

4.18.2.11 cpl_apertures_get_flux()

```
double cpl_apertures_get_flux (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the flux of an aperture.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

The flux of the aperture or undefined on error

See also

cpl_apertures_get_max()

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩ _ERROR_NULL_INPUT.

4.18.2.12 cpl_apertures_get_fwhm()

```
cpl_bivector* cpl_apertures_get_fwhm (
            const cpl_image * self,
            const cpl_apertures * aperts )
```

Compute FWHM values in x and y for a list of apertures.

Parameters

self	The image to process
aperts	The list of apertures

Returns

A newly allocated object containing the fwhms in x and y or NULL

See also

cpl_image_get_fwhm()

Deprecated Replace this call with a loop over cpl_image_get_fwhm()

References cpl_apertures_get_pos_x(), cpl_apertures_get_pos_y(), cpl_apertures_get_size(), cpl_bivector_↩ delete(), cpl_bivector_get_x_data(), cpl_bivector_get_y_data(), cpl_bivector_new(), cpl_ensure, CPL_ERR↩ OR_DATA_NOT_FOUND, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), cpl_errorstate_set(), and cpl_image_get_fwhm().

4.18.2.13 cpl_apertures_get_left()

```
cpl_size cpl_apertures_get_left (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the leftmost x position in an aperture.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

the leftmost x position of the aperture or negative on error

Note

In case of an error the _cpl_error_code_ code is set

See also

cpl_apertures_get_maxpos_x()

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩ _ERROR_NULL_INPUT.

4.18.2.14 cpl_apertures_get_left_y()

```
cpl_size cpl_apertures_get_left_y (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the y position of the leftmost x position in an aperture.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

the y position of the leftmost x position or negative on error

Note

An aperture may have multiple leftmost x positions, in which case one of these is returned.

See also

cpl_apertures_get_maxpos_x()

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩_ERROR_NULL_INPUT.

4.18.2.15 cpl_apertures_get_max()

```
double cpl_apertures_get_max (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the maximum value of an aperture.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

The maximum value of the aperture or undefined on error

Note

In case of an error the _cpl_error_code_ code is set

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if ind is non-positive
- CPL_ERROR_ACCESS_OUT_OF_RANGE if ind exceeds the number of apertures in self

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩ _ERROR_NULL_INPUT.

4.18.2.16 cpl_apertures_get_max_x()

```
double cpl_apertures_get_max_x (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the average X-position of an aperture.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

The average X-position of the aperture or negative on error

Deprecated Replace this function with cpl_apertures_get_pos_x()

References cpl_apertures_get_pos_x().

4.18.2.17 cpl_apertures_get_max_y()

```
double cpl_apertures_get_max_y (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the average Y-position of an aperture.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

The average Y-position of the aperture or negative on error

Deprecated Replace this function with cpl_apertures_get_pos_y()

References cpl_apertures_get_pos_y().

4.18.2.18 cpl_apertures_get_maxpos_x()

```
cpl_size cpl_apertures_get_maxpos_x (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the X-position of the aperture maximum value.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

The X-position of the aperture maximum value or negative on error

Note

In case of an error the _cpl_error_code_ code is set

See also

cpl_apertures_get_centroid_x()

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩_ERROR_NULL_INPUT.

4.18.2.19 cpl_apertures_get_maxpos_y()

```
cpl_size cpl_apertures_get_maxpos_y (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the Y-position of the aperture maximum value.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

The Y-position of the aperture maximum value or negative on error

Note

In case of an error the _cpl_error_code_ code is set

See also

cpl_apertures_get_maxpos_x()

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩_ERROR_NULL_INPUT.

4.18.2.20 cpl_apertures_get_mean()

```
double cpl_apertures_get_mean (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the mean value of an aperture.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

The mean value of the aperture or undefined on error

See also

cpl_apertures_get_max()

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩_ERROR_NULL_INPUT.

4.18.2.21 cpl_apertures_get_median()

```
double cpl_apertures_get_median (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the median value of an aperture.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

The median value of the aperture or undefined on error

See also

cpl_apertures_get_max()

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩ _ERROR_NULL_INPUT.

4.18.2.22 cpl_apertures_get_min()

```
double cpl_apertures_get_min (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the minimum value of an aperture.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

The minimum value of the aperture or undefined on error

See also

cpl_apertures_get_max()

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩ _ERROR_NULL_INPUT.

4.18.2.23 cpl_apertures_get_minpos_x()

```
cpl_size cpl_apertures_get_minpos_x (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the X-position of the aperture minimum value.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

The X-position of the aperture minimum value or negative on error

Note

In case of an error the _cpl_error_code_ code is set

See also

cpl_apertures_get_maxpos_x()

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩
_ERROR_NULL_INPUT.

4.18.2.24 cpl_apertures_get_minpos_y()

```
cpl_size cpl_apertures_get_minpos_y (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the Y-position of the aperture minimum value.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

The Y-position of the aperture minimum value or negative on error

Note

In case of an error the _cpl_error_code_ code is set

See also

cpl_apertures_get_minpos_x()

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩ _ERROR_NULL_INPUT.

4.18.2.25 cpl_apertures_get_npix()

```
cpl_size cpl_apertures_get_npix (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the number of pixels of an aperture.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

The number of pixels of the aperture or negative on error

Note

In case of an error the _cpl_error_code_ code is set

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if ind is non-positive
- CPL_ERROR_ACCESS_OUT_OF_RANGE if ind exceeds the number of apertures in self

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩ _ERROR_NULL_INPUT.

4.18.2.26 cpl_apertures_get_pos_x()

```
double cpl_apertures_get_pos_x (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the average X-position of an aperture.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

The average X-position of the aperture or negative on error

Note

In case of an error the _cpl_error_code_ code is set

See also

cpl_apertures_get_centroid_x()

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩_ERROR_NULL_INPUT.

Referenced by cpl_apertures_get_fwhm(), cpl_apertures_get_max_x(), and cpl_geom_img_offset_combine().

4.18.2.27 cpl_apertures_get_pos_y()

```
double cpl_apertures_get_pos_y (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the average Y-position of an aperture.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

The average Y-position of the aperture or negative on error

See also

cpl_apertures_get_pos_x()

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩_ERROR_NULL_INPUT.

Referenced by cpl_apertures_get_fwhm(), cpl_apertures_get_max_y(), and cpl_geom_img_offset_combine().

4.18.2.28 cpl_apertures_get_right()

```
cpl_size cpl_apertures_get_right (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the rightmost x position in an aperture.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

the rightmost x position in an aperture or negative on error

Note

In case of an error the _cpl_error_code_ code is set

See also

cpl_apertures_get_maxpos_x()

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩_ERROR_NULL_INPUT.

4.18.2.29 cpl_apertures_get_right_y()

```
cpl_size cpl_apertures_get_right_y (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the y position of the rightmost x position in an aperture.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

the y position of the rightmost x position or negative on error

Note

An aperture may have multiple rightmost x positions, in which case one of these is returned.

See also

cpl_apertures_get_maxpos_x()

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩ _ERROR_NULL_INPUT.

4.18.2.30 cpl_apertures_get_size()

```
cpl_size cpl_apertures_get_size (
            const cpl_apertures * self )
```

Get the number of apertures.

Parameters

self	The cpl_apertures object

Returns

The number of apertures or -1 on error

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_apertures_get_fwhm().

4.18.2.31 cpl_apertures_get_stdev()

```
double cpl_apertures_get_stdev (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the standard deviation of an aperture.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

The standard deviation of the aperture or negative on error

See also

cpl_apertures_get_max()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if ind is non-positive
- CPL_ERROR_ACCESS_OUT_OF_RANGE if ind exceeds the number of apertures in self
- CPL_ERROR_DATA_NOT_FOUND if the aperture comprises less than two pixels

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_DATA_NOT_FOUND, CPL↩_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_INPUT.

4.18.2.32 cpl_apertures_get_top()

```
cpl_size cpl_apertures_get_top (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the topmost y position in an aperture.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

the topmost y position in the aperture or negative on error

Note

In case of an error the _cpl_error_code_ code is set

See also

cpl_apertures_get_maxpos_x()

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩_ERROR_NULL_INPUT.

4.18.2.33 cpl_apertures_get_top_x()

```
cpl_size cpl_apertures_get_top_x (
            const cpl_apertures * self,
            cpl_size ind )
```

Get the x position of the topmost y position in an aperture.

Parameters

self	The cpl_apertures object
ind	The aperture index (1 for the first one)

Returns

the x position of the topmost y position or negative on error

Note

An aperture may have multiple topmost x positions, in which case one of these is returned.

See also

cpl_apertures_get_maxpos_x()

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩ _ERROR_NULL_INPUT.

4.18.2.34 cpl_apertures_new_from_image()

```
cpl_apertures* cpl_apertures_new_from_image (
            const cpl_image * self,
            const cpl_image * lab )
```

Compute statistics on selected apertures.

Parameters

self	Reference image
lab	Labelized image (of type CPL_TYPE_INT)

Returns

An CPL apertures object or *NULL* on error

Note

The returned object must be deleted using cpl_apertures_delete().

See also

cpl_image_labelise_mask_create()

The labelized image must contain at least one pixel for each value from 1 to the maximum value in the image.

For the centroiding computation of an aperture, if some pixels have values lower or equal to 0, all the values of the aperture are locally shifted such as the minimum value of the aperture has a value of epsilon. The centroid is then computed on these positive values. In principle, centroid should always be computed on positive values, this is done to avoid raising an error in case the caller of the function wants to use it on negative values images without caring about the centroid results. In such cases, the centroid result would be meaningful, but slightly depend on the hardcoded value chosen for epsilon (1e-10).

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_TYPE_MISMATCH if lab is not of type CPL_TYPE_INT or if self is of a complex type
- CPL_ERROR_ILLEGAL_INPUT if lab has a negative value or zero maximum
- CPL_ERROR_INCOMPATIBLE_INPUT if lab and self have different sizes.
- CPL_ERROR_DATA_NOT_FOUND if one of the lab values is missing.

4.18.2.35 cpl_apertures_sort_by_flux()

```
cpl_error_code cpl_apertures_sort_by_flux (
            cpl_apertures * self )
```

Sort by decreasing aperture flux.

Parameters

self	Apertures to sort (MODIFIED)

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

See also

cpl_apertures_sort_by_npix()

References cpl_calloc(), cpl_ensure_code, CPL_ERROR_NULL_INPUT, cpl_free(), and cpl_malloc().

4.18.2.36 cpl_apertures_sort_by_max()

```
cpl_error_code cpl_apertures_sort_by_max (
            cpl_apertures * self )
```

Sort by decreasing aperture peak value.

Parameters

self	Apertures to sort (MODIFIED)

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

See also

cpl_apertures_sort_by_npix()

References cpl_calloc(), cpl_ensure_code, CPL_ERROR_NULL_INPUT, cpl_free(), and cpl_malloc().

4.18.2.37 cpl_apertures_sort_by_npix()

```
cpl_error_code cpl_apertures_sort_by_npix (
            cpl_apertures * self )
```

Sort by decreasing aperture size.

Parameters

self	Apertures to sort (MODIFIED)

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_calloc(), cpl_ensure_code, CPL_ERROR_NULL_INPUT, cpl_free(), and cpl_malloc().

Referenced by cpl_geom_img_offset_combine().

4.19 High-level functions for non-linear fitting

Functions

- cpl_error_code cpl_fit_image_gaussian (const cpl_image ∗im, const cpl_image ∗im_err, cpl_size xpos, cpl_size ypos, cpl_size xsize, cpl_size ysize, cpl_array ∗parameters, cpl_array ∗err_params, const cpl_↩ array ∗fit_params, double ∗rms, double ∗red_chisq, cpl_matrix ∗∗covariance, double ∗major, double ∗minor, double ∗angle, cpl_matrix ∗∗phys_cov)

  Fit a 2D gaussian to image values.
- cpl_imagelist ∗ cpl_fit_imagelist_polynomial (const cpl_vector ∗x_pos, const cpl_imagelist ∗values, cpl_size mindeg, cpl_size maxdeg, cpl_boolean is_symsamp, cpl_type pixeltype, cpl_image ∗fiterror)

  Least-squares fit a polynomial to each pixel in a list of images.
- cpl_imagelist ∗ cpl_fit_imagelist_polynomial_window (const cpl_vector ∗x_pos, const cpl_imagelist ∗values, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury, cpl_size mindeg, cpl_size maxdeg, cpl_boolean is_↩ symsamp, cpl_type pixeltype, cpl_image ∗fiterror)

  Least-squares fit a polynomial to each pixel in a list of images.
- cpl_error_code cpl_fit_lvmq (const cpl_matrix ∗x, const cpl_matrix ∗sigma_x, const cpl_vector ∗y, const cpl↩ _vector ∗sigma_y, cpl_vector ∗a, const int ia[ ], int(∗f)(const double x[ ], const double a[ ], double ∗result), int(∗dfda)(const double x[ ], const double a[ ], double result[ ]), double relative_tolerance, int tolerance_count, int max_iterations, double ∗mse, double ∗red_chisq, cpl_matrix ∗∗covariance)

  Fit a function to a set of data.
- double cpl_gaussian_eval_2d (const cpl_array ∗self, double x, double y)

  Evaluate the Gaussian in a 2D-point.

4.19.1 Detailed Description

This module provides a routine for non-linear fitting.

Synopsis:

```
#include "cpl_fit.h"
```

4.19.2 Function Documentation

4.19.2.1 cpl_fit_image_gaussian()

```
cpl_error_code cpl_fit_image_gaussian (
            const cpl_image * im,
            const cpl_image * im_err,
            cpl_size xpos,
            cpl_size ypos,
            cpl_size xsize,
            cpl_size ysize,
            cpl_array * parameters,
            cpl_array * err_params,
            const cpl_array * fit_params,
            double * rms,
            double * red_chisq,
            cpl_matrix ** covariance,
            double * major,
            double * minor,
            double * angle,
            cpl_matrix ** phys_cov )
```

Fit a 2D gaussian to image values.

Parameters

im	Input image with data values to fit.
im_err	Optional input image with statistical errors associated to data.
xpos	X position of center of fitting domain.
ypos	Y position of center of fitting domain.
xsize	X size of fitting domain. It must be at least 3 pixels.
ysize	Y size of fitting domain. It must be at least 3 pixels.
parameters	Preallocated array for returning the values of the best-fit gaussian parameters (the parametrisation of the fitted gaussian is described in the main documentation section, below). This array must be of type CPL_TYPE_DOUBLE, and it must have exactly 7 elements. Generally, when passed to this function, this array would not be initialised (all elements are "invalid"). A first-guess for the gaussian parameters is not mandatory: but it is possible to specify here a first-guess value for each parameter. First-guess values can also be specified just for a subset of parameters.
err_params	Optional preallocated array for returning the statistical error associated to each fitted parameter. This array must be of type CPL_TYPE_DOUBLE, and it must have exactly 7 elements. This makes mandatory to specify *im_err*. Note that the returned values are the square root of the diagonal elements (variances) of the *covariance* matrix (see ahead).
fit_params	Optional array, used for flagging the parameters to freeze. This array must be of type CPL_TYPE_INT, and it must have exactly 7 elements. If an array element is set to 0, the corresponding parameter will be frozen. Any other value (including an "invalid" array element) would indicate a free parameter. If a parameter is frozen, a first-guess value *must* be specified at the corresponding element of the *parameters* array. If no array is specified here (NULL pointer), all parameters are free.
rms	If not NULL, returned standard deviation of fit residuals.
red_chisq	If not NULL, returned reduced chi-squared of fit. This makes mandatory to specify *im_err*.
covariance	If not NULL, a newly allocated covariance matrix will be returned. This makes mandatory to specify *im_err*. On error it is not modified.
major	If not NULL, returned semi-major axis of ellipse at 1-sigma.
minor	If not NULL, returned semi-minor axis of ellipse at 1-sigma.
angle	If not NULL, returned angle between X axis and major axis of ellipse, counted counterclockwise (radians).
phys_cov	If not NULL, a newly allocated 3x3 covariance matrix for the derived physical parameters *major*, *minor*, and *angle*, will be returned. This makes mandatory to specify *im_err*. On error it is not modified.

Returns

CPL_ERROR_NONE on successful fit.

This function fits a 2d gaussian to pixel values within a specified region by minimizing χ^2 using a Levenberg-↩ Marquardt algorithm. The gaussian model adopted here is based on the well-known cartesian form

$$z = B + \frac{A}{2\pi\sigma_x\sigma_y\sqrt{1-\rho^2}} \exp\left(-\frac{1}{2\left(1-\rho^2\right)}\left(\left(\frac{x-\mu_x}{\sigma_x}\right)^2 - 2\rho\left(\frac{x-\mu_x}{\sigma_x}\right)\left(\frac{y-\mu_y}{\sigma_y}\right) + \left(\frac{y-\mu_y}{\sigma_y}\right)^2\right)\right)$$

where B is a background level and A the volume of the gaussian (they both can be negative!), making 7 parameters altogether. Conventionally the parameters are indexed from 0 to 6 in the elements of the arrays *parameters*, *err_↩ params*, *fit_params*, and of the 7x7 *covariance* matrix:

$$\begin{aligned}
\text{parameters}[0] &= B \\
\text{parameters}[1] &= A \\
\text{parameters}[2] &= \rho \\
\text{parameters}[3] &= \mu_x \\
\text{parameters}[4] &= \mu_y \\
\text{parameters}[5] &= \sigma_x \\
\text{parameters}[6] &= \sigma_y
\end{aligned}$$

The semi-axes a, b and the orientation θ of the ellipse at 1-sigma level are finally derived from the fitting parameters as:

$$\begin{aligned}
\theta &= \frac{1}{2}\arctan\left(2\rho\frac{\sigma_x\sigma_y}{\sigma_x^2-\sigma_y^2}\right) \\
a &= \sigma_x\sigma_y\sqrt{2(1-\rho^2)\frac{\cos 2\theta}{\left(\sigma_x^2+\sigma_y^2\right)\cos 2\theta+\sigma_y^2-\sigma_x^2}} \\
b &= \sigma_x\sigma_y\sqrt{2(1-\rho^2)\frac{\cos 2\theta}{\left(\sigma_x^2+\sigma_y^2\right)\cos 2\theta-\sigma_y^2+\sigma_x^2}}
\end{aligned}$$

Note that θ is counted counterclockwise starting from the positive direction of the x axis, ranging bewteen $-\pi/2$ and $+\pi/2$ radians.

If the correlation $\rho = 0$ and $\sigma_x \geq \sigma_y$ (within uncertainties) the ellipse is either a circle or its major axis is aligned with the x axis, so it is conventionally set

$$\begin{aligned}
\theta &= 0 \\
a &= \sigma_x \\
b &= \sigma_y
\end{aligned}$$

If the correlation $\rho = 0$ and $\sigma_x < \sigma_y$ (within uncertainties) the major axis of the ellipse is aligned with the y axis, so it is conventionally set

$$\begin{aligned}
\theta &= \frac{\pi}{2} \\
a &= \sigma_y \\
b &= \sigma_x
\end{aligned}$$

If requested, the 3x3 covariance matrix G associated to the derived physical quantities is also computed, applying the usual

$$\mathrm{G} = \mathrm{JCJ}^{\mathrm{T}}$$

where J is the Jacobian of the transformation $(B, A, \rho, \mu_x, \mu_y, \sigma_x, \sigma_y) \rightarrow (\theta, a, b)$ and C is the 7x7 matrix of the gaussian parameters.

References cpl_array_delete(), cpl_array_get_double(), cpl_array_get_int(), cpl_array_get_type(), cpl_array_↩ new(), cpl_array_set_double(), cpl_array_set_invalid(), CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERRO↩ R_DATA_NOT_FOUND, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERRO↩ R_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, cpl↩ _errorstate_get(), cpl_free(), cpl_image_delete(), cpl_image_extract(), cpl_image_get(), cpl_image_get_bpm↩ _const(), cpl_image_get_data_double_const(), cpl_image_get_maxpos(), cpl_image_get_mean(), cpl_image_↩ get_median(), cpl_image_get_minpos(), cpl_image_get_size_x(), cpl_image_get_size_y(), cpl_image_get_type(), cpl_malloc(), cpl_mask_is_empty(), CPL_MATH_2PI, CPL_MATH_FWHM_SIG, cpl_matrix_wrap(), CPL_MAX, CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT, cpl_type_get_name(), CPL_TYPE_INT, and cpl_vector_wrap().

4.19.2.2 cpl_fit_imagelist_polynomial()

```
cpl_imagelist* cpl_fit_imagelist_polynomial (
             const cpl_vector * x_pos,
             const cpl_imagelist * values,
             cpl_size mindeg,
             cpl_size maxdeg,
             cpl_boolean is_symsamp,
             cpl_type pixeltype,
             cpl_image * fiterror )
```

Least-squares fit a polynomial to each pixel in a list of images.

Parameters

x_pos	The vector of positions to fit
values	The list of images with values to fit
mindeg	The smallest degree with a non-zero coefficient
maxdeg	The polynomial degree of the fit, at least mindeg
is_symsamp	True iff the x_pos values are symmetric around their mean
pixeltype	The pixel-type of the created image list
fiterror	When non-NULL, the error of the fit

Note

values and x_pos must have the same number of elements.

The created imagelist must be deallocated with cpl_imagelist_delete().

x_pos must have at least 1 + (maxdeg - mindeg) distinct values.

Returns

The image list of the fitted polynomial coefficients or NULL on error.

See also

cpl_fit_imagelist_polynomial_window()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input const pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if mindeg is negative or maxdeg is less than mindeg.
- CPL_ERROR_INCOMPATIBLE_INPUT if x_pos and values have different lengths, or if fiterror is non-NULL with a different size than that of values, or if the input images do not all have the same dimensions and pixel type.
- CPL_ERROR_DATA_NOT_FOUND if x_pos contains less than nc values.
- CPL_ERROR_SINGULAR_MATRIX if x_pos contains less than nc distinct values.
- CPL_ERROR_UNSUPPORTED_MODE if the chosen pixel type is not one of CPL_TYPE_DOUBLE, CPL↩ _TYPE_FLOAT, CPL_TYPE_INT.

References cpl_fit_imagelist_polynomial_window(), cpl_image_get_size_x(), cpl_image_get_size_y(), and cpl_↩ imagelist_get_const().

4.19.2.3 cpl_fit_imagelist_polynomial_window()

```
cpl_imagelist* cpl_fit_imagelist_polynomial_window (
              const cpl_vector * x_pos,
              const cpl_imagelist * values,
              cpl_size llx,
              cpl_size lly,
              cpl_size urx,
              cpl_size ury,
              cpl_size mindeg,
              cpl_size maxdeg,
              cpl_boolean is_symsamp,
              cpl_type pixeltype,
              cpl_image * fiterror )
```

Least-squares fit a polynomial to each pixel in a list of images.

Parameters

x_pos	The vector of positions to fit
values	The list of images with values to fit
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)
mindeg	The smallest degree with a non-zero coefficient
maxdeg	The polynomial degree of the fit, at least mindeg
is_symsamp	True iff the x_pos values are symmetric around their mean
pixeltype	The (non-complex) pixel-type of the created image list
fiterror	When non-NULL, the error of the fit. Must be non-complex

Note

values and x_pos must have the same number of elements.

The created imagelist must be deallocated with cpl_imagelist_delete().

x_pos must have at least 1 + (maxdeg - mindeg) distinct values.

Returns

The image list of the fitted polynomial coefficients or NULL on error.

See also

cpl_polynomial_fit()

For each pixel, a polynomial representing the relation value = P(x) is computed where: P(x) = x^{mindeg} ∗ (a_0 + a_1 ∗ x + ... + a_{nc-1} ∗ x^{nc-1}), where mindeg >= 0 and maxdeg >= mindeg, and nc is the number of polynomial coefficients to determine, nc = 1 + (maxdeg - mindeg).

The returned image list thus contains nc coefficient images, a_0, a_1, ..., a_{nc-1}.

np is the number of sample points, i.e. the number of elements in x_pos and number of images in the input image list.

If mindeg is nonzero then is_symsamp is ignored, otherwise is_symsamp may to be set to CPL_TRUE if and only if the values in x_pos are known a-priori to be symmetric around their mean, e.g. (1, 2, 4, 6, 10, 14, 16, 18, 19), but not (1, 2, 4, 6, 10, 14, 16). Setting is_symsamp to CPL_TRUE while mindeg is zero eliminates certain round-off errors. For higher order fitting the fitting problem known as "Runge's phenomenon" is minimized using the socalled "Chebyshev nodes" as sampling points. For Chebyshev nodes is_symsamp can be set to CPL_TRUE.

Even though it is not an error, it is hardly useful to use an image of pixel type integer for the fitting error. An image of pixel type float should on the other hand be sufficient for most fitting errors.

The call requires the following number of FLOPs, where nz is the number of pixels in any one image in the imagelist:

2 ∗ nz ∗ nc ∗ (nc + np) + np ∗ nc^2 + nc^3/3 + O(nc ∗ (nc + np)).

If mindeg is zero an additional nz ∗ nc^2 FLOPs are required.

If fiterror is non-NULL an additional 2 ∗ nz ∗ nc ∗ np FLOPs are required.

Bad pixels in the input is suported as follows: First all pixels are fitted ignoring any bad pixel maps in the input. If this succeeds then each fit, where bad pixel(s) are involved is redone. During this second pass all input pixels flagged as bad are ignored. For each pixel to be redone, the remaining good samples are passed to cpl_polynomial_fit(). The input is_symsamp is ignored in this second pass. The reduced number of samples may reduce the number of sampling points to equal the number of coefficients to fit. In this case the fit has another meaning (any non-zero residual is due to rounding errors, not a fitting error). If for a given fit bad pixels reduces the number of sampling points to less than the number of coefficients to fit, then as many coefficients are fit as there are sampling points. The higher order coefficients are set to zero and flagged as bad. If a given pixel has no good samples, then the resulting fit will consist of zeroes, all flagged as bad.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input const pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if mindeg is negative or maxdeg is less than mindeg or if llx or lly are smaller than 1 or if urx or ury is smaller than llx and lly respectively.
- CPL_ERROR_ACCESS_OUT_OF_RANGE if urx or ury exceed the size of values.
- CPL_ERROR_INCOMPATIBLE_INPUT if x_pos and values have different lengths, or if fiterror is non-NULL with a different size than that of values, or if the input images do not all have the same dimensions and pixel type.
- CPL_ERROR_DATA_NOT_FOUND if x_pos contains less than nc values.
- CPL_ERROR_SINGULAR_MATRIX if x_pos contains less than nc distinct values.
- CPL_ERROR_UNSUPPORTED_MODE if the chosen pixel type is not one of CPL_TYPE_DOUBLE, CPL↩ _TYPE_FLOAT, CPL_TYPE_INT.

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_DATA_NOT_FOUND, CPL↩ _ERROR_ILLEGAL_INPUT, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NONE, CPL_ERROR_NU↩ LL_INPUT, CPL_ERROR_SINGULAR_MATRIX, CPL_ERROR_UNSUPPORTED_MODE, cpl_image_get_bpm↩ _const(), cpl_image_get_size_x(), cpl_image_get_size_y(), cpl_imagelist_get_const(), cpl_imagelist_get_size(), cpl_imagelist_is_uniform(), cpl_imagelist_new(), cpl_imagelist_set(), cpl_malloc(), cpl_mask_delete(), cpl_mask↩ _or(), CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT, cpl_type_get_sizeof(), CPL_TYPE_INT, and cpl_vector_get_↩ size().

Referenced by cpl_fit_imagelist_polynomial().

4.19.2.4 cpl_fit_lvmq()

```
cpl_error_code cpl_fit_lvmq (
             const cpl_matrix * x,
             const cpl_matrix * sigma_x,
             const cpl_vector * y,
             const cpl_vector * sigma_y,
             cpl_vector * a,
             const int ia[],
             int(*)(const double x[], const double a[], double *result) f,
             int(*)(const double x[], const double a[], double result[]) dfda,
             double relative_tolerance,
             int tolerance_count,
             int max_iterations,
             double * mse,
             double * red_chisq,
             cpl_matrix ** covariance )
```

Fit a function to a set of data.

Parameters

x	N x D matrix of the positions to fit. Each matrix row is a D-dimensional position.
sigma_x	Uncertainty (one sigma, gaussian errors assumed) assosiated with *x*. Taking into account the uncertainty of the independent variable is currently unsupported, and this parameter must therefore be set to NULL.
y	The N values to fit.
sigma_y	Vector of size N containing the uncertainties of the y-values. If this parameter is NULL, constant uncertainties are assumed.
a	Vector containing M fit parameters. Must contain a guess solution on input and contains the best fit parameters on output.
ia	Array of size M defining which fit parameters participate in the fit (non-zero) and which fit parameters are held constant (zero). At least one element must be non-zero. Alternatively, pass NULL to fit all parameters.
f	Function that evaluates the fit function at the position specified by the first argument (an array of size D) using the fit parameters specified by the second argument (an array of size M). The result must be output using the third parameter, and the function must return zero iff the evaluation succeded.
dfda	Function that evaluates the first order partial derivatives of the fit function with respect to the fit parameters at the position specified by the first argument (an array of size D) using the parameters specified by the second argument (an array of size M). The result must be output using the third parameter (array of size M), and the function must return zero iff the evaluation succeded.
relative_tolerance	The algorithm converges by definition if the relative decrease in chi squared is less than *tolerance tolerance_count* times in a row. Recommended default: CPL_FIT_LVMQ_TOLERANCE
tolerance_count	The algorithm converges by definition if the relative decrease in chi squared is less than *tolerance tolerance_count* times in a row. Recommended default: CPL_FIT_LVMQ_COUNT
max_iterations	If this number of iterations is reached without convergence, the algorithm diverges, by definition. Recommended default: CPL_FIT_LVMQ_MAXITER
mse	If non-NULL, the mean squared error of the best fit is computed.
red_chisq	If non-NULL, the reduced chi square of the best fit is computed. This requires *sigma_y* to be specified.

Parameters

covariance	If non-NULL, the formal covariance matrix of the best fit parameters is computed (or NULL on error). On success the diagonal terms of the covariance matrix are guaranteed to be positive. However, terms that involve a constant parameter (as defined by the input array *ia*) are always set to zero. Computation of the covariacne matrix requires *sigma_y* to be specified.

Returns

CPL_ERROR_NONE iff OK.

This function makes a minimum chi squared fit of the specified function to the specified data set using a Levenberg-Marquardt algorithm.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer other than *sigma_x*, *sigma_y*, *mse*, *red_chisq* or *covariance* is NULL.
- CPL_ERROR_ILLEGAL_INPUT if an input matrix/vector is empty, if *ia* contains only zero values, if any of *relative_tolerance*, *tolerance_count* or max_iterations *is* non-positive, if N <= M and *red_chisq* is non-NULL, if any element of *sigma_x* or *sigma_y* is non-positive, or if evaluation of the fit function or its derivative failed.
- CPL_ERROR_INCOMPATIBLE_INPUT if the dimensions of the input vectors/matrices do not match, or if chi square or covariance computation is requested and *sigma_y* is NULL.
- CPL_ERROR_ILLEGAL_OUTPUT if memory allocation failed.
- CPL_ERROR_CONTINUE if the Levenberg-Marquardt algorithm failed to converge.
- CPL_ERROR_SINGULAR_MATRIX if the covariance matrix could not be computed.

References CPL_ERROR_NONE.

4.19.2.5 cpl_gaussian_eval_2d()

```
double cpl_gaussian_eval_2d (
            const cpl_array * self,
            double x,
            double y )
```

Evaluate the Gaussian in a 2D-point.

Parameters

self	The seven Gaussian parameters
x	The X-coordinate to evaluate
y	The Y-coordinate to evaluate

Returns

The gaussian value or zero on error

See also

cpl_fit_image_gaussian()

Note

The function should not be able to fail if the parameters come from a succesful call to cpl_fit_image_gaussian()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if a pointer is NULL.
- CPL_ERROR_TYPE_MISMATCH if the array is not of type double
- CPL_ERROR_ILLEGAL_INPUT if the array has a length different from 7
- CPL_ERROR_ILLEGAL_OUTPUT if the (absolute value of the) radius exceeds 1
- CPL_ERROR_DIVISION_BY_ZERO if a sigma is 0, or the radius is 1

References cpl_array_get_double(), cpl_array_get_size(), CPL_ERROR_DIVISION_BY_ZERO, CPL_ERROR_↩ ILLEGAL_INPUT, CPL_ERROR_ILLEGAL_OUTPUT, cpl_errorstate_get(), cpl_errorstate_is_equal(), and CPL_↩ MATH_2PI.

4.20 High-level functions that are photometry related

Functions

- cpl_error_code cpl_photom_fill_blackbody (cpl_vector *spectrum, cpl_unit out_unit, const cpl_vector *evalpoints, cpl_unit in_unit, double temp)

  The Planck radiance from a black-body.

4.20.1 Detailed Description

Synopsis:

```
#include "cpl_photom.h"
```

4.20.2 Function Documentation

4.20.2.1 cpl_photom_fill_blackbody()

```
cpl_error_code cpl_photom_fill_blackbody (
            cpl_vector * spectrum,
            cpl_unit out_unit,
            const cpl_vector * evalpoints,
            cpl_unit in_unit,
            double temp )
```

The Planck radiance from a black-body.

Parameters

spectrum	Preallocated, the computed radiance
out_unit	CPL_UNIT_PHOTONRADIANCE, CPL_UNIT_ENERGYRADIANCE or CPL_UNIT_LESS
evalpoints	The evaluation points (wavelengths or frequencies)
in_unit	CPL_UNIT_LENGTH or CPL_UNIT_FREQUENCY
temp	The black body temperature [K]

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_

The Planck black-body radiance can be computed in 5 different ways: As a radiance of either energy [J*radian/s/m^3] or photons [radian/s/m^3], and in terms of either wavelength [m] or frequency [1/s]. The fifth way is as a unit-less radiance in terms of wavelength, in which case the area under the planck curve is 1.

The dimension of the spectrum (energy or photons or unit-less, CPL_UNIT_LESS) is controlled by out_unit, and the dimension of the input (length or frequency) is controlled by in_unit.

evalpoints and spectrum must be of equal, positive length.

The input wavelengths/frequencies and the temperature must be positive.

The four different radiance formulas are: Rph1(l,T) = 2pi c/l^4/(exp(hc/klT)-1) Rph2(f,T) = 2pi f^2/c^2/(exp(hf/kT)-1) Re1(l,T) = 2pi hc^2/l^5/(exp(hc/klT)-1) = Rph1(l,T) ∗ hc/l Re2(f,T) = 2pi hf^3/c^2/(exp(hf/kT)-1) = Rph2(f,T) ∗ hf R1(l,T) = 15h^5c^5/(pi^4k^5l^5T^5/(exp(hc/klT)-1) = Rph1(l,T) ∗ h^4c^3/(2pi^5k^5T^5)

where l is the wavelength, f is the frequency, T is the temperature, h is the Planck constant, k is the Boltzmann constant and c is the speed of light in vacuum.

When the radiance is computed in terms of wavelength, the radiance peaks at l_max = CPL_PHYS_Wien/temp. When the radiance is unit-less this maximum, R1(l_max,T), is approximately 3.2648. R1(l,T) integrated over l from 0 to infinity is 1.

A unit-less black-body radiance in terms of frequency may be added later, until then it is an error to combine CPL↩ _UNIT_LESS and CPL_UNIT_FREQUENCY.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if the size of evalpoints is different from the size of spectrum
- CPL_ERROR_UNSUPPORTED_MODE if in_unit and out_unit are not as requested
- CPL_ERROR_ILLEGAL_INPUT if temp or a wavelength is non-positive

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_E↩ RROR_NONE, CPL_ERROR_NULL_INPUT, CPL_ERROR_UNSUPPORTED_MODE, CPL_MATH_2PI, CPL_↩ MATH_PI, cpl_vector_get_data(), cpl_vector_get_data_const(), and cpl_vector_get_size().

4.21 High-level functions to compute detector features

Functions

- cpl_error_code cpl_detector_interpolate_rejected (cpl_image ∗self)

  Interpolate any bad pixels in an image and delete the bad pixel map.
- cpl_error_code cpl_flux_get_bias_window (const cpl_image ∗diff, const cpl_size ∗zone_def, cpl_size ron_↩ hsize, cpl_size ron_nsamp, double ∗bias, double ∗error)

  Compute the bias in a rectangle.
- cpl_error_code cpl_flux_get_noise_ring (const cpl_image ∗diff, const double ∗zone_def, cpl_size ron_hsize, cpl_size ron_nsamp, double ∗noise, double ∗error)

  Compute the readout noise in a ring.
- cpl_error_code cpl_flux_get_noise_window (const cpl_image ∗diff, const cpl_size ∗zone_def, cpl_size ron↩ _hsize, cpl_size ron_nsamp, double ∗noise, double ∗error)

  Compute the readout noise in a rectangle.

4.21.1 Detailed Description

Synopsis:

```
#include "cpl_detector.h"
```

4.21.2 Function Documentation

4.21.2.1 cpl_detector_interpolate_rejected()

```
cpl_error_code cpl_detector_interpolate_rejected (
            cpl_image * self )
```

Interpolate any bad pixels in an image and delete the bad pixel map.

Parameters

self	The image to clean

Returns

The _cpl_error_code_ or CPL_ERROR_NONE

The value of a bad pixel is interpolated from the good pixels among the 8 nearest. (If all but one of the eight neighboring pixels are bad, the interpolation becomes a nearest neighbor interpolation). For integer images the interpolation in done with floating-point and rounded to the nearest integer.

If there are pixels for which all of the eight neighboring pixels are bad, a subsequent interpolation pass is done, where the already interpolated pixels are included as source for the interpolation.

The interpolation passes are repeated until all bad pixels have been interpolated. In the worst case, all pixels will be interpolated from a single good pixel.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_DATA_NOT_FOUND if all pixels are bad

References cpl_ensure_code, cpl_error_get_code(), CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_image_accept_all(), cpl_image_get_bpm_const(), cpl_image_get_data(), cpl_image_get_size_x(), cpl_image_get_size_y(), cpl_image_get_type(), cpl_mask_get_data_const(), CPL_TYPE_DOUBLE, CPL_TYPE_DOUBLE_COMPLEX, CPL_TYPE_FLOAT, CPL_TYPE_FLOAT_COMPLEX, and CPL_TYPE_INT.

4.21.2.2 cpl_flux_get_bias_window()

```
cpl_error_code cpl_flux_get_bias_window (
            const cpl_image * diff,
            const cpl_size * zone_def,
            cpl_size ron_hsize,
            cpl_size ron_nsamp,
            double * bias,
            double * error )
```

Compute the bias in a rectangle.

Parameters

diff	Input image, usually a difference frame.
zone_def	Zone where the bias is to be computed.
ron_hsize	to specify half size of squares (<0 to use default)
ron_nsamp	to specify the nb of samples (<0 to use default)
bias	Output parameter: bias in the frame.
error	Output parameter: error on the bias.

Returns

CPL_ERROR_NONE on success or the relevant _cpl_error_code_ on error

See also

cpl_flux_get_noise_window(), rand()

Note

No calls to srand() are made from CPL

4.21.2.3 cpl_flux_get_noise_ring()

```
cpl_error_code cpl_flux_get_noise_ring (
            const cpl_image * diff,
            const double * zone_def,
            cpl_size ron_hsize,
            cpl_size ron_nsamp,
            double * noise,
            double * error )
```

Compute the readout noise in a ring.

Parameters

diff	Input image, usually a difference frame.
zone_def	Zone where the readout noise is to be computed.
ron_hsize	to specify half size of squares (<0 to use default)
ron_nsamp	to specify the nb of samples (<0 to use default)
noise	On success, ∗noise is the noise in the frame.
error	NULL, or on success, ∗error is the error in the noise

Returns

CPL_ERROR_NONE on success or the relevant _cpl_error_code_ on error

See also

cpl_flux_get_noise_window(), rand()

Note

No calls to srand() are made from CPL

The provided zone is an array of four integers specifying the zone to take into account for the computation. The integers specify a ring as x, y, r1, r2 where these coordinates are given in the FITS notation (x from 1 to nx, y from 1 to ny and bottom to top).

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if diff noise or zone_def is NULL
- CPL_ERROR_ILLEGAL_INPUT if the internal radius is bigger than the external one in zone_def
- CPL_ERROR_DATA_NOT_FOUND if an insufficient number of samples were found inside the ring

4.21.2.4 cpl_flux_get_noise_window()

```
cpl_error_code cpl_flux_get_noise_window (
            const cpl_image * diff,
            const cpl_size * zone_def,
            cpl_size ron_hsize,
            cpl_size ron_nsamp,
            double * noise,
            double * error )
```

Compute the readout noise in a rectangle.

Parameters

diff	Input image, usually a difference frame.
zone_def	Zone where the readout noise is to be computed.
ron_hsize	to specify half size of squares (<0 to use default)
ron_nsamp	to specify the nb of samples (<0 to use default)
noise	Output parameter: noise in the frame.
error	Output parameter: error on the noise.

Returns

CPL_ERROR_NONE on success or the relevant _cpl_error_code_ on error

See also

rand()

Note

No calls to srand() are made from CPL

This function is meant to compute the readout noise in a frame by means of a MonteCarlo approach. The input is a frame, usually a difference between two frames taken with the same settings for the acquisition system, although no check is done on that, it is up to the caller to feed in the right kind of frame.

The provided zone is an array of four integers specifying the zone to take into account for the computation. The integers specify ranges as xmin, xmax, ymin, ymax, where these coordinates are given in the FITS notation (x from 1 to lx, y from 1 to ly and bottom to top). Specify NULL instead of an array of four values to use the whole frame in the computation.

The algorithm will create typically 100 9x9 windows on the frame, scattered optimally using a Poisson law. In each window, the standard deviation of all pixels in the window is computed and this value is stored. The readout noise is the median of all computed standard deviations, and the error is the standard deviation of the standard deviations.

Both noise and error are returned by modifying a passed double. If you do not care about the error, pass NULL.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if diff or noise is NULL
- CPL_ERROR_ILLEGAL_INPUT if the specified window (zone_def) is invalid

4.22 I/O

Macros

- #define CPL_BPP_16_SIGNED CPL_TYPE_SHORT
- #define CPL_BPP_16_UNSIGNED CPL_TYPE_USHORT
- #define CPL_BPP_32_SIGNED CPL_TYPE_INT
- #define CPL_BPP_8_UNSIGNED CPL_TYPE_UCHAR
- #define CPL_BPP_IEEE_DOUBLE CPL_TYPE_DOUBLE
- #define CPL_BPP_IEEE_FLOAT CPL_TYPE_FLOAT
- #define cpl_type_bpp cpl_type

Typedefs

- typedef enum _cpl_io_type_ cpl_io_type

  The file I/O modes.

Enumerations

- enum _cpl_io_type_ {
  CPL_IO_CREATE,
  CPL_IO_EXTEND,
  CPL_IO_APPEND,
  CPL_IO_COMPRESS_GZIP,
  CPL_IO_COMPRESS_RICE,
  CPL_IO_COMPRESS_HCOMPRESS,
  CPL_IO_COMPRESS_PLIO,
  CPL_IO_MAX,
  CPL_IO_DEFAULT }

  These are the file I/O modes.

4.22.1 Detailed Description

This module provides definitions related to I/O. The actual I/O functions are defined in the respective CPL modules.

Synopsis:

```
#include "cpl_io.h"
```

4.22.2 Macro Definition Documentation

4.22.2.1 CPL_BPP_16_SIGNED

```
#define CPL_BPP_16_SIGNED CPL_TYPE_SHORT
```

Deprecated Use CPL_TYPE_SHORT

4.22.2.2 CPL_BPP_16_UNSIGNED

```
#define CPL_BPP_16_UNSIGNED CPL_TYPE_USHORT
```

Deprecated Use CPL_TYPE_USHORT

4.22.2.3 CPL_BPP_32_SIGNED

```
#define CPL_BPP_32_SIGNED CPL_TYPE_INT
```

Deprecated Use CPL_TYPE_INT

4.22.2.4 CPL_BPP_8_UNSIGNED

```
#define CPL_BPP_8_UNSIGNED CPL_TYPE_UCHAR
```

Deprecated Use CPL_TYPE_UCHAR

4.22.2.5 CPL_BPP_IEEE_DOUBLE

```
#define CPL_BPP_IEEE_DOUBLE CPL_TYPE_DOUBLE
```

Deprecated Use CPL_TYPE_DOUBLE

4.22.2.6 CPL_BPP_IEEE_FLOAT

```
#define CPL_BPP_IEEE_FLOAT CPL_TYPE_FLOAT
```

Deprecated Use CPL_TYPE_FLOAT

4.22.2.7 cpl_type_bpp

```
#define cpl_type_bpp cpl_type
```

Deprecated Use cpl_type

4.22.3 Typedef Documentation

4.22.3.1 cpl_io_type

```
typedef enum _cpl_io_type_ cpl_io_type
```

The file I/O modes.

4.22.4 Enumeration Type Documentation

4.22.4.1 _cpl_io_type_

```
enum _cpl_io_type_
```

These are the file I/O modes.

For the compression modes, see http://heasarc.nasa.gov/docs/software/fitsio/compression.html

Enumerator

CPL_IO_CREATE	Overwrite the file, if it already exists.
CPL_IO_EXTEND	Append a new extension to the file.
CPL_IO_APPEND	Append to the last data unit of the file.
CPL_IO_COMPRESS_GZIP	Use FITS tiled-image compression with GZIP algorithm.
CPL_IO_COMPRESS_RICE	Use FITS tiled-image compression with RICE algorithm.
CPL_IO_COMPRESS_HCOMPRESS	Use FITS tiled-image compression with HCOMPRESS algorithm.
CPL_IO_COMPRESS_PLIO	Use FITS tiled-image compression with PLIO algorithm.
CPL_IO_MAX	Reserved for internal CPL usage.
CPL_IO_DEFAULT	Deprecated, kept only for backwards compatibility

4.23 Imagelists

- cpl_error_code cpl_imagelist_add (cpl_imagelist *in1, const cpl_imagelist *in2)

  Add two image lists, the first one is replaced by the result.
- cpl_error_code cpl_imagelist_subtract (cpl_imagelist *in1, const cpl_imagelist *in2)

  Subtract two image lists, the first one is replaced by the result.
- cpl_error_code cpl_imagelist_multiply (cpl_imagelist *in1, const cpl_imagelist *in2)

  Multiply two image lists, the first one is replaced by the result.
- cpl_error_code cpl_imagelist_divide (cpl_imagelist *in1, const cpl_imagelist *in2)

  Divide two image lists, the first one is replaced by the result.
- cpl_error_code cpl_imagelist_add_image (cpl_imagelist *imlist, const cpl_image *img)

  Add an image to an image list.
- cpl_error_code cpl_imagelist_subtract_image (cpl_imagelist *imlist, const cpl_image *img)

  Subtract an image from an image list.
- cpl_error_code cpl_imagelist_multiply_image (cpl_imagelist *imlist, const cpl_image *img)

  Multiply an image list by an image.
- cpl_error_code cpl_imagelist_divide_image (cpl_imagelist *imlist, const cpl_image *img)

  Divide an image list by an image.
- cpl_error_code cpl_imagelist_add_scalar (cpl_imagelist *imlist, double addend)

  Elementwise addition of a scalar to each image in the imlist.
- cpl_error_code cpl_imagelist_subtract_scalar (cpl_imagelist *imlist, double subtrahend)

  Elementwise subtraction of a scalar from each image in the imlist.
- cpl_error_code cpl_imagelist_multiply_scalar (cpl_imagelist *imlist, double factor)

  Elementwise multiplication of the imlist with a scalar.
- cpl_error_code cpl_imagelist_divide_scalar (cpl_imagelist *imlist, double divisor)

  Elementwise division of each image in the imlist with a scalar.
- cpl_error_code cpl_imagelist_logarithm (cpl_imagelist *imlist, double base)

  Compute the elementwise logarithm of each image in the imlist.
- cpl_error_code cpl_imagelist_exponential (cpl_imagelist *imlist, double base)

  Compute the elementwise exponential of each image in the imlist.
- cpl_error_code cpl_imagelist_power (cpl_imagelist *imlist, double exponent)

  Compute the elementwise power of each image in the imlist.
- cpl_error_code cpl_imagelist_normalise (cpl_imagelist *imlist, cpl_norm mode)

  Normalize each image in the list.
- cpl_error_code cpl_imagelist_threshold (cpl_imagelist *imlist, double lo_cut, double hi_cut, double assign_lo_cut, double assign_hi_cut)

  Threshold all pixel values to an interval.
- cpl_image * cpl_image_new_from_accepted (const cpl_imagelist *imlist)

  Create a contribution map from the bad pixel maps of the images.
- cpl_image * cpl_imagelist_collapse_create (const cpl_imagelist *imlist)

  Average an imagelist to a single image.
- cpl_image * cpl_imagelist_collapse_minmax_create (const cpl_imagelist *self, cpl_size nlow, cpl_size nhigh)

  Average with rejection an imagelist to a single image.
- cpl_image * cpl_imagelist_collapse_sigclip_create (const cpl_imagelist *self, double kappalow, double kappahigh, double keepfrac, cpl_collapse_mode mode, cpl_image *contrib)

  Collapse an imagelist with kappa-sigma-clipping rejection.
- cpl_image * cpl_imagelist_collapse_median_create (const cpl_imagelist *self)

  Create a median image from the input imagelist.
- cpl_imagelist * cpl_imagelist_swap_axis_create (const cpl_imagelist *ilist, cpl_swap_axis mode)

  Swap the axis of an image list.

- cpl_imagelist * cpl_imagelist_new (void)

  Create an empty imagelist.
- cpl_imagelist * cpl_imagelist_load (const char *filename, cpl_type im_type, cpl_size xtnum)

  Load a FITS file extension into a list of images.
- cpl_imagelist * cpl_imagelist_load_window (const char *filename, cpl_type im_type, cpl_size xtnum, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury)

  Load images windows from a FITS file extension into an image list.
- cpl_size cpl_imagelist_get_size (const cpl_imagelist *imlist)

  Get the number of images in the imagelist.
- cpl_image * cpl_imagelist_get (cpl_imagelist *imlist, cpl_size inum)

  Get an image from a list of images.
- const cpl_image * cpl_imagelist_get_const (const cpl_imagelist *imlist, cpl_size inum)

  Get an image from a list of images.
- cpl_error_code cpl_imagelist_set (cpl_imagelist *self, cpl_image *im, cpl_size pos)

  Insert an image into an imagelist.
- cpl_image * cpl_imagelist_unset (cpl_imagelist *self, cpl_size pos)

  Remove an image from an imagelist.
- void cpl_imagelist_empty (cpl_imagelist *self)

  Empty an imagelist and deallocate all its images.
- void cpl_imagelist_unwrap (cpl_imagelist *self)

  Free memory used by a cpl_imagelist object, except the images.
- void cpl_imagelist_delete (cpl_imagelist *self)

  Free all memory used by a cpl_imagelist object including the images.
- cpl_error_code cpl_imagelist_cast (cpl_imagelist *self, const cpl_imagelist *other, cpl_type type)

  Cast an imagelist, optionally in-place.
- cpl_imagelist * cpl_imagelist_duplicate (const cpl_imagelist *imlist)

  Copy an image list.
- cpl_error_code cpl_imagelist_erase (cpl_imagelist *imlist, const cpl_vector *valid)

  Reject one or more images in a list according to an array of flags.
- cpl_error_code cpl_imagelist_save (const cpl_imagelist *self, const char *filename, cpl_type type, const cpl_propertylist *plist, unsigned mode)

  Save an imagelist to disk in FITS format.
- int cpl_imagelist_is_uniform (const cpl_imagelist *imlist)

  Determine if an imagelist contains images of equal size and type.
- cpl_error_code cpl_imagelist_dump_structure (const cpl_imagelist *self, FILE *stream)

  Dump structural information of images in an imagelist.
- cpl_error_code cpl_imagelist_dump_window (const cpl_imagelist *self, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury, FILE *stream)

  Dump pixel values of images in a CPL imagelist.

4.23.1 Detailed Description

This module provides functions to create, use, and destroy a *cpl_imagelist*. A *cpl_imagelist* is an ordered list of cpl_images. All images in a list must have the same pixel-type and the same dimensions. It is allowed to insert the same image into different positions in the list. Different images in the list are allowed to share the same bad pixel map.

Synopsis:

```
#include "cpl_imagelist.h"
```

4.23.2 Function Documentation

4.23.2.1 cpl_image_new_from_accepted()

```
cpl_image* cpl_image_new_from_accepted (
            const cpl_imagelist * imlist )
```

Create a contribution map from the bad pixel maps of the images.

Parameters

imlist	The imagelist

Returns

The contributions map (a CPL_TYPE_INT cpl_image) or NULL on error

See also

cpl_imagelist_is_uniform()

The returned map counts for each pixel the number of good pixels in the list. The returned map has to be deallocated with cpl_image_delete().

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the input image list is not valid

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_image_add_scalar(), cpl_image_get_bpm_const(), cpl_image_get_size_x(), cpl_image_get_size_y(), cpl_image_new(), cpl_imagelist↩_get_const(), cpl_imagelist_get_size(), cpl_imagelist_is_uniform(), and CPL_TYPE_INT.

Referenced by cpl_imagelist_collapse_create().

4.23.2.2 cpl_imagelist_add()

```
cpl_error_code cpl_imagelist_add (
            cpl_imagelist * in1,
            const cpl_imagelist * in2 )
```

Add two image lists, the first one is replaced by the result.

Parameters

in1	first input image list (modified)
in2	image list to add

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

See also

cpl_image_add()

The two input lists must have the same size, the image number n in the list in2 is added to the image number n in the list in1.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the input images have different sizes

4.23.2.3 cpl_imagelist_add_image()

```
cpl_error_code cpl_imagelist_add_image (
            cpl_imagelist * imlist,
            const cpl_image * img )
```

Add an image to an image list.

Parameters

imlist	input image list (modified)
img	image to add

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

See also

cpl_image_add()

The passed image is added to each image of the passed image list.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

4.23.2.4 cpl_imagelist_add_scalar()

```
cpl_error_code cpl_imagelist_add_scalar (
            cpl_imagelist * imlist,
            double addend )
```

Elementwise addition of a scalar to each image in the imlist.

Parameters

imlist	Imagelist to be modified in place.
addend	Number to add

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

See also

cpl_image_add_scalar()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure_code, cpl_error_get_code(), CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl↩
_image_add_scalar().

4.23.2.5 cpl_imagelist_cast()

```
cpl_error_code cpl_imagelist_cast (
            cpl_imagelist * self,
            const cpl_imagelist * other,
            cpl_type type )
```

Cast an imagelist, optionally in-place.

Parameters

self	Destination imagelist
other	Source imagelist, or NULL to cast in-place
type	If called with empty self, cast to this pixel-type

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_image_cast()

Note

If called with a non-empty self in an out-of-place cast, the input images are cast to the type already present in self and appended to the output list. In this case the parameter type is ignored.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if the destination pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if the same pointer is passed twice
- CPL_ERROR_ILLEGAL_INPUT if the passed type is invalid
- CPL_ERROR_TYPE_MISMATCH if the passed image type is complex and requested casting type is non-complex.
- CPL_ERROR_INVALID_TYPE if the passed pixel type is not supported

References CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_free(), cpl_image_cast(), cpl_image_delete(), cpl_image_get_bpm_const(), cpl_image_get_type(), cpl_image_unset_bpm(), cpl_imagelist_delete(), cpl_imagelist_get_const(), cpl_imagelist_get_size(), cpl_imagelist_new(), cpl_imagelist_set(), cpl_imagelist_unset(), and cpl_malloc().

4.23.2.6 cpl_imagelist_collapse_create()

```
cpl_image* cpl_imagelist_collapse_create (
             const cpl_imagelist * imlist )
```

Average an imagelist to a single image.

Parameters

imlist	the input images list

Returns

the average image or NULL on error case.

See also

cpl_imagelist_is_uniform()

The returned image has to be deallocated with cpl_image_delete().

The bad pixel maps of the images in the input list are taken into account, the result image pixels are flagged as rejected for those where there were no good pixel at the same position in the input image list.

For integer pixel types, the averaging is performed using integer division.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the input image list is not valid

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_image_accept_all(), cpl_image_add(), cpl_image_delete(), cpl_image_divide(), cpl_image_duplicate(), cpl_image_fill_rejected(), cpl_image_get_bpm_const(), cpl_image_new_from_accepted(), and cpl_imagelist_is_uniform().

4.23.2.7 cpl_imagelist_collapse_median_create()

```
cpl_image* cpl_imagelist_collapse_median_create (
            const cpl_imagelist * self )
```

Create a median image from the input imagelist.

Parameters

self	The input image list

Returns

The median image of the input pixel type or NULL on error

Note

The created image has to be deallocated with cpl_image_delete()

The input image list can be of type CPL_TYPE_INT, CPL_TYPE_FLOAT and CPL_TYPE_DOUBLE.

On success each pixel in the created image is the median of the values on the same pixel position in the input image list. If for a given pixel all values in the input image list are rejected, the resulting pixel is set to zero and flagged as rejected.

The median is defined here as the middle value of an odd number of sorted samples and for an even number of samples as the mean of the two central values. Note that with an even number of samples the median may not be among the input samples.

Also, note that in the case of an even number of integer data, the mean value will be computed using integer arithmetic. Cast your integer data to a floating point pixel type if that is not the desired behavior.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the input image list is not valid
- CPL_ERROR_INVALID_TYPE if the passed image list pixel type is not supported

4.23.2.8 cpl_imagelist_collapse_minmax_create()

```
cpl_image* cpl_imagelist_collapse_minmax_create (
            const cpl_imagelist * self,
            cpl_size nlow,
            cpl_size nhigh )
```

Average with rejection an imagelist to a single image.

Parameters

self	The image list to average
nlow	Number of low rejected values
nhigh	Number of high rejected values

Returns

The average image or NULL on error

Note

The returned image has to be deallocated with cpl_image_delete().

The input images are averaged, for each pixel position the nlow lowest pixels and the nhigh highest pixels are discarded for the average computation.

The input image list can be of type CPL_TYPE_INT, CPL_TYPE_FLOAT and CPL_TYPE_DOUBLE. The created image will be of the same type.

On success each pixel in the created image is the average of the non-rejected values on the pixel position in the input image list.

For a given pixel position any bad pixels (i.e. values) are handled as follows: Given n bad values on a given pixel position, n/2 of those values are assumed to be low outliers and n/2 of those values are assumed to be high outliers. Any low or high rejection will first reject up to n/2 bad values and if more values need to be rejected that rejection will take place on the good values. This rationale behind this is to allow the rejection of outliers to include bad pixels without introducing a bias. If for a given pixel all values in the input image list are rejected, the resulting pixel is set to zero and flagged as rejected.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an the input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the input image list is not valid or if the sum of the rejections is not lower than the number of images or if nlow or nhigh is negative
- CPL_ERROR_INVALID_TYPE if the passed image list type is not supported

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NULL_↩INPUT, cpl_image_get_size_x(), cpl_image_get_size_y(), cpl_image_get_type(), cpl_imagelist_get_const(), cpl_↩imagelist_get_size(), and cpl_imagelist_is_uniform().

4.23.2.9 cpl_imagelist_collapse_sigclip_create()

```
cpl_image* cpl_imagelist_collapse_sigclip_create (
             const cpl_imagelist * self,
             double kappalow,
             double kappahigh,
             double keepfrac,
             cpl_collapse_mode mode,
             cpl_image * contrib )
```

Collapse an imagelist with kappa-sigma-clipping rejection.

Parameters

self	The input imagelist
kappalow	kappa-factor for lower clipping threshold
kappahigh	kappa-factor for upper clipping threshold
keepfrac	The fraction of values to keep (0.0 $<$ keepfrac $<=$ 1.0)
mode	Clipping mode, CPL_COLLAPSE_MEAN or CPL_COLLAPSE_MEDIAN
contrib	Pre-allocated integer-image for contribution map or NULL

Returns

The collapsed image or NULL on error case.

Note

The returned image has to be deallocated with cpl_image_delete().

The collapsing is an iterative process which will stop when it converges (i.e. an iteration did not reject any values for a given pixel) or when the next iteration would reduce the fraction of values to keep to less than or equal to keepfrac.

A call with keepfrac == 1.0 will thus perform no clipping.

Supported modes: CPL_COLLAPSE_MEAN: The center value of the acceptance range will be the mean. CPL↩ _COLLAPSE_MEDIAN: The center value of the acceptance range will be the median. CPL_COLLAPSE_MED↩ IAN_MEAN: The center value of the acceptance range will be the median in the first iteration and in subsequent iterations it will be the mean.

For each pixel position the pixels whose value is higher than center + kappahigh $*$ stdev or lower than center - kappalow $*$ stdev are discarded for the subsequent center and stdev computation, where center is defined according to the clipping mode, and stdev is the standard deviation of the values at that pixel position. Since the acceptance interval must be non-empty, the sum of kappalow and kappahigh must be positive. A typical call has both kappalow and kappahigh positive.

The minimum number of values that the clipping can select is 2. This is because the clipping criterion is based on the sample standard deviation, which needs at least two values to be defined. This means that all calls with (positive) values of keepfrac less than 2/n will behave the same. To ensure that the values in (at least) i planes out of n are kept, keepfrac can be set to (i - 0.5) / n, e.g. to keep at least 50 out of 100 values, keepfrac can be set to 0.495.

The output pixel is set to the mean of the non-clipped values, also in the median mode. Regardless of the input pixel type, the mean is computed in double precision. The result is then cast to the output-pixel type, which is identical to the input pixel type.

The input parameter contrib is optional. It must be either NULL or point to a pre-allocated image of type CPL_↩ TYPE_INT and size equal to the images in the imagelist. On success, it will contain the contribution map, i.e. the number of kept (non-clipped) values after the iterative process on every pixel.

Bad pixels are ignored from the start. This means that with a sufficient number of bad pixels, the fraction of good values will be less than keepfrac. In this case no iteration is performed at all. If there is at least one good value available, then the mean will be based on the good value(s). If for a given pixel position there are no good values, then that pixel is set to zero, rejected as bad and if available the value in the contribution map is set to zero.

The input imagelist can be of type CPL_TYPE_INT, CPL_TYPE_FLOAT and CPL_TYPE_DOUBLE.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_DATA_NOT_FOUND if there are less than 2 images in the list
- CPL_ERROR_ILLEGAL_INPUT if the sum of kappalow and kappahigh is non-positive,
- CPL_ERROR_ACCESS_OUT_OF_RANGE if keepfrac is outside the required interval which is 0.0 < keepfrac <= 1.0
- CPL_ERROR_TYPE_MISMATCH if contrib is non-NULL but not of type CPL_TYPE_INT
- CPL_ERROR_INCOMPATIBLE_INPUT if contrib is non-NULL but of a size incompatible with the input imagelist
- CPL_ERROR_INVALID_TYPE if the type of the input imagelist is unsupported
- CPL_ERROR_UNSUPPORTED_MODE if the passed mode is none of the above listed

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_DATA_NOT_FOUND, CPL↩_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NULL_INPUT, CPL_ERROR_UNSU↩PPORTED_MODE, cpl_image_get_data_int(), cpl_image_get_size_x(), cpl_image_get_size_y(), cpl_image_get↩_type(), cpl_imagelist_get_const(), cpl_imagelist_get_size(), cpl_imagelist_is_uniform(), and CPL_TYPE_INT.

4.23.2.10 cpl_imagelist_delete()

```
void cpl_imagelist_delete (
            cpl_imagelist * self )
```

Free all memory used by a cpl_imagelist object including the images.

Parameters

self	The image list or NULL

Returns

Nothing

See also

cpl_imagelist_empty(), cpl_imagelist_unwrap()

References cpl_imagelist_empty(), and cpl_imagelist_unwrap().

Referenced by cpl_geom_img_offset_combine(), and cpl_imagelist_cast().

4.23.2.11 cpl_imagelist_divide()

```
cpl_error_code cpl_imagelist_divide (
            cpl_imagelist * in1,
            const cpl_imagelist * in2 )
```

Divide two image lists, the first one is replaced by the result.

Parameters

in1	first input image list (modified)
in2	image list to divide

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

See also

cpl_image_divide()
cpl_imagelist_add()

4.23.2.12 cpl_imagelist_divide_image()

```
cpl_error_code cpl_imagelist_divide_image (
            cpl_imagelist * imlist,
            const cpl_image * img )
```

Divide an image list by an image.

Parameters

imlist	input image list (modified)
img	image for division

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

See also

cpl_image_divide()
cpl_imagelist_add_image()

4.23.2.13 cpl_imagelist_divide_scalar()

```
cpl_error_code cpl_imagelist_divide_scalar (
            cpl_imagelist * imlist,
            double divisor )
```

Elementwise division of each image in the imlist with a scalar.

Parameters

imlist	Imagelist to be modified in place.
divisor	Non-zero number to divide with

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

See also

cpl_imagelist_add_scalar()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure_code, cpl_error_get_code(), CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl↩_image_divide_scalar().

4.23.2.14 cpl_imagelist_dump_structure()

```
cpl_error_code cpl_imagelist_dump_structure (
            const cpl_imagelist * self,
            FILE * stream )
```

Dump structural information of images in an imagelist.

Parameters

self	Imagelist to dump
stream	Output stream, accepts `stdout` or `stderr`

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_FILE_IO if a write operation fails

References cpl_ensure_code, CPL_ERROR_FILE_IO, cpl_error_get_code(), CPL_ERROR_NONE, CPL_ERRO↩R_NULL_INPUT, cpl_image_dump_structure(), and cpl_imagelist_get_const().

4.23.2.15 cpl_imagelist_dump_window()

```
cpl_error_code cpl_imagelist_dump_window (
            const cpl_imagelist * self,
            cpl_size llx,
            cpl_size lly,
            cpl_size urx,
            cpl_size ury,
            FILE * stream )
```

Dump pixel values of images in a CPL imagelist.

Parameters

self	Imagelist to dump
llx	Specifies the window position
lly	Specifies the window position
urx	Specifies the window position
ury	Specifies the window position
stream	Output stream, accepts `stdout` or `stderr`

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_FILE_IO if a write operation fails
- CPL_ERROR_ACCESS_OUT_OF_RANGE if the defined window is not in the image
- CPL_ERROR_ILLEGAL_INPUT if the window definition is wrong (e.g llx $>$ urx)

References cpl_ensure_code, CPL_ERROR_FILE_IO, cpl_error_get_code(), CPL_ERROR_NONE, CPL_ERRO↩R_NULL_INPUT, cpl_image_dump_window(), and cpl_imagelist_get_const().

4.23.2.16 cpl_imagelist_duplicate()

```
cpl_imagelist* cpl_imagelist_duplicate (
            const cpl_imagelist * imlist )
```

Copy an image list.

Parameters

imlist	Source image list.

Returns

1 newly allocated image list, or NULL on error.

Copy an image list into a new image list object. The returned image list must be deallocated using cpl_imagelist_delete().

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, CPL_ERROR_NULL_INPUT, cpl_image_duplicate(), cpl_imagelist_new(), and cpl_imagelist_set().

4.23.2.17 cpl_imagelist_empty()

```
void cpl_imagelist_empty (
            cpl_imagelist * self )
```

Empty an imagelist and deallocate all its images.

Parameters

self	The image list or NULL

Returns

Nothing

See also

cpl_imagelist_empty(), cpl_imagelist_delete()

Note

If *self* is `NULL` nothing is done and no error is set.

After the call the image list can be populated again. It must eventually be deallocted with a call to cpl_imagelist_delete().

References cpl_image_delete(), cpl_image_get_bpm_const(), cpl_image_unset_bpm(), and cpl_imagelist_unset().

Referenced by cpl_imagelist_delete().

4.23.2.18 cpl_imagelist_erase()

```
cpl_error_code cpl_imagelist_erase (
            cpl_imagelist * imlist,
            const cpl_vector * valid )
```

Reject one or more images in a list according to an array of flags.

Parameters

imlist	Non-empty imagelist to examine for image rejection.
valid	Vector of flags (>=-0.5: valid, <-0.5: invalid)

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

This function takes an imagelist and a vector of flags. The imagelist and vector must have equal lengths.

Images flagged as invalid are removed from the list.

The removal of image(s) will reduce the length of the list accordingly.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if the vector size and the image list size are different

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NONE, CPL_ERROR_↩ NULL_INPUT, cpl_image_delete(), cpl_image_get_bpm_const(), cpl_image_unset_bpm(), cpl_vector_get(), and cpl_vector_get_size().

4.23.2.19 cpl_imagelist_exponential()

```
cpl_error_code cpl_imagelist_exponential (
            cpl_imagelist * imlist,
            double base )
```

Compute the elementwise exponential of each image in the imlist.

Parameters

imlist	Imagelist to be modified in place.
base	Base of the exponential.

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

See also

cpl_image_exponential()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure_code, cpl_error_get_code(), CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl↩
_image_exponential().

4.23.2.20 cpl_imagelist_get()

```
cpl_image* cpl_imagelist_get (
             cpl_imagelist * imlist,
             cpl_size inum )
```

Get an image from a list of images.

Parameters

imlist	the image list
inum	the image id (from 0 to number of images-1)

Returns

A pointer to the image or NULL in error case.

The returned pointer refers to already allocated data.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ACCESS_OUT_OF_RANGE if inum is bigger thant the list size
- CPL_ERROR_ILLEGAL_INPUT if inum is negative

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩
_ERROR_NULL_INPUT.

Referenced by cpl_fft_imagelist(), and cpl_imagelist_swap_axis_create().

4.23.2.21 cpl_imagelist_get_const()

```
const cpl_image* cpl_imagelist_get_const (
             const cpl_imagelist * imlist,
             cpl_size inum )
```

Get an image from a list of images.

Parameters

imlist	the image list
inum	the image id (from 0 to number of images-1)

Returns

A pointer to the image or NULL in error case.

See also

cpl_imagelist_get

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩ _ERROR_NULL_INPUT.

Referenced by cpl_fft_imagelist(), cpl_fit_imagelist_polynomial(), cpl_fit_imagelist_polynomial_window(), cpl_↩ geom_img_offset_combine(), cpl_geom_img_offset_fine(), cpl_geom_img_offset_saa(), cpl_image_new_from↩ _accepted(), cpl_imagelist_cast(), cpl_imagelist_collapse_minmax_create(), cpl_imagelist_collapse_sigclip_↩ create(), cpl_imagelist_dump_structure(), cpl_imagelist_dump_window(), cpl_imagelist_swap_axis_create(), and cpl_test_get_bytes_imagelist().

4.23.2.22 cpl_imagelist_get_size()

```
cpl_size cpl_imagelist_get_size (
            const cpl_imagelist * imlist )
```

Get the number of images in the imagelist.

Parameters

imlist	the list of image

Returns

The number of images or -1 on error

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_fft_imagelist(), cpl_fit_imagelist_polynomial_window(), cpl_geom_img_offset_combine(), cpl↩ _geom_img_offset_fine(), cpl_geom_img_offset_saa(), cpl_image_new_from_accepted(), cpl_imagelist_cast(), cpl_imagelist_collapse_minmax_create(), cpl_imagelist_collapse_sigclip_create(), cpl_imagelist_swap_axis_↩ create(), and cpl_test_get_bytes_imagelist().

4.23.2.23 cpl_imagelist_is_uniform()

```
int cpl_imagelist_is_uniform (
            const cpl_imagelist * imlist )
```

Determine if an imagelist contains images of equal size and type.

Parameters

imlist	The imagelist to check

Returns

Zero if uniform, positive if non-uniform and negative on error.

The function returns 1 if the list is empty.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, CPL_ERROR_NULL_INPUT, cpl_image_get_size_x(), cpl_image_get_size_y(), and cpl_image_get_type().

Referenced by cpl_fit_imagelist_polynomial_window(), cpl_geom_img_offset_combine(), cpl_geom_img_offset_fine(), cpl_geom_img_offset_saa(), cpl_image_new_from_accepted(), cpl_imagelist_collapse_create(), cpl_imagelist_collapse_minmax_create(), cpl_imagelist_collapse_sigclip_create(), cpl_imagelist_save(), and cpl_imagelist_swap_axis_create().

4.23.2.24 cpl_imagelist_load()

```
cpl_imagelist* cpl_imagelist_load (
            const char * filename,
            cpl_type im_type,
            cpl_size xtnum )
```

Load a FITS file extension into a list of images.

Parameters

filename	The FITS file name
im_type	Type of the images in the created image list
xtnum	The extension number (0 for primary HDU)

Returns

The loaded list of images or NULL on error.

See also

cpl_image_load()

This function loads all the images of a specified extension (NAXIS=2 or 3) into an image list.

Type can be CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT or CPL_TYPE_INT. The loaded images have an empty bad pixel map.

The returned cpl_imagelist must be deallocated using cpl_imagelist_delete()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if xtnum is negative
- CPL_ERROR_INVALID_TYPE if the passed type is not supported
- CPL_ERROR_FILE_IO If the file cannot be opened or read, or if xtnum is bigger than the number of extensions in the FITS file
- CPL_ERROR_BAD_FILE_FORMAT if the file cannot be parsed
- CPL_ERROR_DATA_NOT_FOUND if the data cannot be read from the file

4.23.2.25 cpl_imagelist_load_window()

```
cpl_imagelist* cpl_imagelist_load_window (
             const char * filename,
             cpl_type im_type,
             cpl_size xtnum,
             cpl_size llx,
             cpl_size lly,
             cpl_size urx,
             cpl_size ury )
```

Load images windows from a FITS file extension into an image list.

Parameters

filename	The FITS file name
im_type	Type of the images in the created image list
xtnum	The extension number (0 for primary HDU)
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)

Returns

The loaded list of image windows or NULL on error.

See also

cpl_imagelist_load(), cpl_image_load_window()

Note

The returned cpl_imagelist must be deallocated using cpl_imagelist_delete()

This function loads all the image windows of a specified extension in an image list.

Type can be CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT or CPL_TYPE_INT.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if xtnum is negative
- CPL_ERROR_INVALID_TYPE if the passed type is not supported
- CPL_ERROR_ACCESS_OUT_OF_RANGE if xtnum is bigger than the number of extensions in the FITS file
- CPL_ERROR_BAD_FILE_FORMAT if the file cannot be parsed
- CPL_ERROR_DATA_NOT_FOUND if the data cannot be read from the file

4.23.2.26 cpl_imagelist_logarithm()

```
cpl_error_code cpl_imagelist_logarithm (
            cpl_imagelist * imlist,
            double base )
```

Compute the elementwise logarithm of each image in the imlist.

Parameters

imlist	Imagelist to be modified in place.
base	Base of the logarithm.

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

See also

cpl_image_logarithm()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure_code, cpl_error_get_code(), CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl↩_image_logarithm().

4.23.2.27 cpl_imagelist_multiply()

```
cpl_error_code cpl_imagelist_multiply (
            cpl_imagelist * in1,
            const cpl_imagelist * in2 )
```

Multiply two image lists, the first one is replaced by the result.

Parameters

in1	first input image list (modified)
in2	image list to multiply

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

See also

cpl_image_multiply()

cpl_imagelist_add()

4.23.2.28 cpl_imagelist_multiply_image()

```
cpl_error_code cpl_imagelist_multiply_image (
            cpl_imagelist * imlist,
            const cpl_image * img )
```

Multiply an image list by an image.

Parameters

imlist	input image list (modified)
img	image to multiply

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

See also

cpl_image_multiply()

cpl_imagelist_add_image()

4.23.2.29 cpl_imagelist_multiply_scalar()

```
cpl_error_code cpl_imagelist_multiply_scalar (
            cpl_imagelist * imlist,
            double factor )
```

Elementwise multiplication of the imlist with a scalar.

Parameters

imlist	Imagelist to be modified in place.
factor	Number to multiply with

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

See also

cpl_imagelist_add_scalar()

References cpl_ensure_code, cpl_error_get_code(), CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl↩_image_multiply_scalar().

4.23.2.30 cpl_imagelist_new()

```
cpl_imagelist* cpl_imagelist_new (
            void  )
```

Create an empty imagelist.

Returns

1 newly allocated cpl_imagelist

See also

cpl_imagelist_set()

The returned cpl_imagelist must be deallocated using cpl_imagelist_delete()

References cpl_calloc().

Referenced by cpl_fit_imagelist_polynomial_window(), cpl_geom_img_offset_combine(), cpl_imagelist_cast(), cpl_imagelist_duplicate(), cpl_imagelist_load_frameset(), and cpl_imagelist_swap_axis_create().

4.23.2.31 cpl_imagelist_normalise()

```
cpl_error_code cpl_imagelist_normalise (
            cpl_imagelist * imlist,
            cpl_norm mode )
```

Normalize each image in the list.

Parameters

imlist	Imagelist to modify.
mode	Normalization mode.

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_image_normalise()

The list may be partly modified if an error occurs.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure_code, cpl_error_get_code(), CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl↩_image_normalise().

4.23.2.32 cpl_imagelist_power()

```
cpl_error_code cpl_imagelist_power (
            cpl_imagelist * imlist,
            double exponent )
```

Compute the elementwise power of each image in the imlist.

Parameters

imlist	Imagelist to be modified in place.
exponent	Scalar exponent

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

See also

cpl_image_power()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure_code, cpl_error_get_code(), CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl↩_image_power().

4.23.2.33 cpl_imagelist_save()

```
cpl_error_code cpl_imagelist_save (
            const cpl_imagelist * self,
            const char * filename,
            cpl_type type,
            const cpl_propertylist * plist,
            unsigned mode )
```

Save an imagelist to disk in FITS format.

Parameters

self	Imagelist to save
filename	Name of the FITS file to write
type	The type used to represent the data in the file
plist	Property list for the output header or NULL
mode	The desired output options (combined with bitwise or)

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

See also

cpl_image_save()

This function saves an image list to a FITS file. If a property list is provided, it is written to the named file before the pixels are written.

Supported image lists types are CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT, CPL_TYPE_INT.

The type used in the file can be one of: CPL_TYPE_UCHAR (8 bit unsigned), CPL_TYPE_SHORT (16 bit signed), CPL_TYPE_USHORT (16 bit unsigned), CPL_TYPE_INT (32 bit signed), CPL_TYPE_FLOAT (32 bit floating point), or CPL_TYPE_DOUBLE (64 bit floating point). Additionally, the special value CPL_TYPE_UNSPECIFIED is allowed. This value means that the type used for saving is the pixel type of the input image. Using the image pixel type as saving type ensures that the saving incurs no loss of information.

Supported output modes are CPL_IO_CREATE (create a new file), CPL_IO_EXTEND (extend an existing file with a new extension) and CPL_IO_APPEND (append a list of images to the last data unit, which must already contain compatible image(s)).

For the CPL_IO_APPEND mode it is recommended to pass a NULL pointer for the output header, since updating the already existing header incurs significant overhead.

When the data written to disk are of an integer type, the output mode CPL_IO_EXTEND can be combined (via bit-wise or) with an option for tile-compression. This compression of integer data is lossless. The options are: CPL_IO_COMPRESS_GZIP, CPL_IO_COMPRESS_RICE, CPL_IO_COMPRESS_HCOMPRESS, CPL_IO_CO↩MPRESS_PLIO. With compression the type must be CPL_TYPE_UNSPECIFIED or CPL_TYPE_INT.

In extend and append mode, make sure that the file has write permissions. You may have problems if you create a file in your application and append something to it with the umask set to 222. In this case, the file created by your application would not be writable, and the append would fail.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the type or the mode is not supported
- CPL_ERROR_FILE_IO if the file cannot be written
- CPL_ERROR_INVALID_TYPE if the passed image list type is not supported

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_IN↩PUT, and cpl_imagelist_is_uniform().

4.23.2.34 cpl_imagelist_set()

```
cpl_error_code cpl_imagelist_set (
             cpl_imagelist * self,
             cpl_image * im,
             cpl_size pos )
```

Insert an image into an imagelist.

Parameters

self	The imagelist
im	The image to insert
pos	The list position (from 0 to number of images)

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

It is allowed to specify the position equal to the number of images in the list. This will increment the size of the imagelist.

No action occurs if an image is inserted more than once into the same position. It is allowed to insert the same image into two different positions in a list.

The image is inserted at the position pos in the image list. If the image already there is only present in that one location in the list, then the image is deallocated.

It is not allowed to insert images of different size into a list.

The added image is owned by the imagelist object, which deallocates it when cpl_imagelist_delete is called. Another option is to use cpl_imagelist_unset to recover ownership of the image, in which case the cpl_imagelist object is not longer responsible for deallocating it.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if pos is negative
- CPL_ERROR_TYPE_MISMATCH if im and self are of different types

- CPL_ERROR_INCOMPATIBLE_INPUT if im and self have different sizes
- CPL_ERROR_ACCESS_OUT_OF_RANGE if pos is bigger than the number of images in self

References cpl_ensure_code, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, cpl_image_delete(), cpl_image_get_bpm_const(), cpl_image_get_size_x(), cpl_image_get_size_y(), cpl_image_get_type(), cpl_image_unset_bpm(), and cpl_realloc().

Referenced by cpl_fit_imagelist_polynomial_window(), cpl_geom_img_offset_combine(), cpl_imagelist_cast(), and cpl_imagelist_duplicate().

4.23.2.35 cpl_imagelist_subtract()

```
cpl_error_code cpl_imagelist_subtract (
            cpl_imagelist * in1,
            const cpl_imagelist * in2 )
```

Subtract two image lists, the first one is replaced by the result.

Parameters

in1	first input image list (modified)
in2	image list to subtract

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

See also

cpl_image_subtract()
cpl_imagelist_add()

4.23.2.36 cpl_imagelist_subtract_image()

```
cpl_error_code cpl_imagelist_subtract_image (
            cpl_imagelist * imlist,
            const cpl_image * img )
```

Subtract an image from an image list.

Parameters

imlist	input image list (modified)
img	image to subtract

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

See also

cpl_image_subtract()

cpl_imagelist_add_image()

4.23.2.37 cpl_imagelist_subtract_scalar()

```
cpl_error_code cpl_imagelist_subtract_scalar (
            cpl_imagelist * imlist,
            double subtrahend )
```

Elementwise subtraction of a scalar from each image in the imlist.

Parameters

imlist	Imagelist to be modified in place.
subtrahend	Number to subtract

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

See also

cpl_imagelist_add_scalar()

References cpl_ensure_code, cpl_error_get_code(), CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl↩_image_subtract_scalar().

4.23.2.38 cpl_imagelist_swap_axis_create()

```
cpl_imagelist* cpl_imagelist_swap_axis_create (
            const cpl_imagelist * ilist,
            cpl_swap_axis mode )
```

Swap the axis of an image list.

Parameters

ilist	The image list to swap
mode	The swapping mode

Returns

The swapped image list or NULL in error case

This function is intended for users that want to use the cpl_imagelist object as a cube. Swapping the axis would give them access to the usual functions in the 3 dimensions. This has the cost that it duplicates the memory consumption, which can be a problem for big amounts of data.

Image list can be CPL_TYPE_INT, CPL_TYPE_FLOAT or CPL_TYPE_DOUBLE. The mode can be either CPL_↩ SWAP_AXIS_XZ or CPL_SWAP_AXIS_YZ

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if mode is not equal to one of the possible values or if the image list is not valid
- CPL_ERROR_INVALID_TYPE if the passed image list type is not supported

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NU↩ LL_INPUT, cpl_image_get_size_x(), cpl_image_get_size_y(), cpl_image_get_type(), cpl_image_reject_from↩ _mask(), cpl_imagelist_get(), cpl_imagelist_get_const(), cpl_imagelist_get_size(), cpl_imagelist_is_uniform(), cpl_imagelist_new(), cpl_mask_delete(), cpl_mask_get_data(), and cpl_mask_new().

4.23.2.39 cpl_imagelist_threshold()

```
cpl_error_code cpl_imagelist_threshold (
            cpl_imagelist * imlist,
            double lo_cut,
            double hi_cut,
            double assign_lo_cut,
            double assign_hi_cut )
```

Threshold all pixel values to an interval.

Parameters

imlist	Image list to threshold.
lo_cut	Lower bound.
hi_cut	Higher bound.
assign_lo_cut	Value to assign to pixels below low bound.
assign_hi_cut	Value to assign to pixels above high bound.

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_image_threshold()

Threshold the images of the list using cpl_image_threshold() The input image list is modified.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if lo_cut is bigger than hi_cut

References cpl_ensure_code, cpl_error_get_code(), CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl_image_threshold().

4.23.2.40 cpl_imagelist_unset()

```
cpl_image* cpl_imagelist_unset (
             cpl_imagelist * self,
             cpl_size pos )
```

Remove an image from an imagelist.

Parameters

self	The imagelist
pos	The list position (from 0 to number of images-1)

Returns

The pointer to the removed image or NULL in error case

The specified image is not deallocated, it is simply removed from the list. The pointer to the image is returned to let the user decide to deallocate it or not. Eventually, the image will have to be deallocated with cpl_image_delete().

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if pos is negative
- CPL_ERROR_ACCESS_OUT_OF_RANGE if pos is bigger than the number of images in self

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_geom_img_offset_combine(), cpl_imagelist_cast(), and cpl_imagelist_empty().

4.23.2.41 cpl_imagelist_unwrap()

```
void cpl_imagelist_unwrap (
             cpl_imagelist * self )
```

Free memory used by a cpl_imagelist object, except the images.

Parameters

self	The image list or NULL

Returns

Nothing

See also

cpl_imagelist_empty()

Note

The caller must have pointers to all images in the list and is reponsible for their deallocation. If *self* is `NULL` nothing is done and no error is set.

References cpl_free().

Referenced by cpl_imagelist_delete().

4.24 Images

Typedefs

- typedef enum _cpl_value_ cpl_value

  The CPL special value. It is a bit field.

Enumerations

- enum _cpl_value_ {
  CPL_VALUE_NAN,
  CPL_VALUE_PLUSINF,
  CPL_VALUE_MINUSINF,
  CPL_VALUE_ZERO,
  CPL_VALUE_INF,
  CPL_VALUE_NOTFINITE }

  The special values that can be rejected They are a bit-field and can be combined with bitwise or.

Functions

- cpl_error_code cpl_image_abs (cpl_image *image)

  Take the absolute value of an image.
- cpl_image * cpl_image_abs_create (const cpl_image *image_in)

  Take the absolute value of an image.
- cpl_error_code cpl_image_accept (cpl_image *im, cpl_size x, cpl_size y)

  Set a pixel as good in an image.
- cpl_error_code cpl_image_accept_all (cpl_image *self)

  Set all pixels in the image as good.
- cpl_error_code cpl_image_add (cpl_image *im1, const cpl_image *im2)

  Add two images, store the result in the first image.
- cpl_image * cpl_image_add_create (const cpl_image *image1, const cpl_image *image2)

  Add two images.
- cpl_error_code cpl_image_add_scalar (cpl_image *self, double scalar)

  Elementwise addition of a scalar to an image.
- cpl_image * cpl_image_add_scalar_create (const cpl_image *image, double addend)

  Create a new image by elementwise addition of a scalar to an image.
- cpl_error_code cpl_image_and (cpl_image *self, const cpl_image *first, const cpl_image *second)

  The bit-wise and of two images with integer pixels.
- cpl_error_code cpl_image_and_scalar (cpl_image *self, const cpl_image *first, cpl_bitmask second)

  The bit-wise and of a scalar and an image with integer pixels.
- cpl_image * cpl_image_average_create (const cpl_image *image_1, const cpl_image *image_2)

  Build the average of two images.
- cpl_image * cpl_image_cast (const cpl_image *self, cpl_type type)

  Convert a cpl_image to a given type.
- cpl_image * cpl_image_collapse_create (const cpl_image *self, int direction)

  Collapse an image along its rows or columns.
- cpl_image * cpl_image_collapse_median_create (const cpl_image *self, int direction, cpl_size drop_ll, cpl_size drop_ur)

  Collapse an image along its rows or columns, with filtering.

- cpl_image * cpl_image_collapse_window_create (const cpl_image *self, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury, int direction)

  Collapse an image region along its rows or columns.
- cpl_error_code cpl_image_conjugate (cpl_image *self, const cpl_image *other)

  Complex conjugate the pixels in a complex image.
- cpl_error_code cpl_image_copy (cpl_image *im1, const cpl_image *im2, cpl_size xpos, cpl_size ypos)

  Copy one image into another.
- cpl_size cpl_image_count_rejected (const cpl_image *im)

  Count the number of bad pixels declared in an image.
- void cpl_image_delete (cpl_image *d)

  Free memory associated to an cpl_image object.
- cpl_error_code cpl_image_divide (cpl_image *im1, const cpl_image *im2)

  Divide two images, store the result in the first image.
- cpl_image * cpl_image_divide_create (const cpl_image *image1, const cpl_image *image2)

  Divide two images.
- cpl_error_code cpl_image_divide_scalar (cpl_image *self, double scalar)

  Elementwise division of an image with a scalar.
- cpl_image * cpl_image_divide_scalar_create (const cpl_image *image, double divisor)

  Create a new image by elementwise division of an image with a scalar.
- cpl_error_code cpl_image_dump_structure (const cpl_image *self, FILE *stream)

  Dump structural information of a CPL image.
- cpl_error_code cpl_image_dump_window (const cpl_image *self, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury, FILE *stream)

  Dump pixel values in a CPL image.
- cpl_image * cpl_image_duplicate (const cpl_image *src)

  Copy an image.
- cpl_error_code cpl_image_exponential (cpl_image *self, double base)

  Compute the elementwise exponential of the image.
- cpl_image * cpl_image_exponential_create (const cpl_image *image, double base)

  Create a new image by elementwise exponentiation of an image.
- cpl_image * cpl_image_extract (const cpl_image *in, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury)

  Extract a rectangular zone from an image into another image.
- cpl_image * cpl_image_extract_subsample (const cpl_image *image, cpl_size xstep, cpl_size ystep)

  Sub-sample an image.
- cpl_error_code cpl_image_fft (cpl_image *img_real, cpl_image *img_imag, unsigned mode)

  Fast Fourier Transform a square, power-of-two sized image.
- cpl_error_code cpl_image_fill_abs_arg (cpl_image *im_abs, cpl_image *im_arg, const cpl_image *self)

  Split a complex image into its absolute and argument part(s)
- cpl_error_code cpl_image_fill_gaussian (cpl_image *ima, double xcen, double ycen, double norm, double sig_x, double sig_y)

  Generate an image from a 2d gaussian function.
- cpl_error_code cpl_image_fill_jacobian (cpl_image *out, const cpl_image *deltax, const cpl_image *deltay)

  Compute area change ratio for a transformation map.
- cpl_error_code cpl_image_fill_jacobian_polynomial (cpl_image *out, const cpl_polynomial *poly_x, const cpl_polynomial *poly_y)

  Compute area change ratio for a 2D polynomial transformation.
- cpl_error_code cpl_image_fill_noise_uniform (cpl_image *ima, double min_pix, double max_pix)

  Generate an image with uniform random noise distribution.
- cpl_error_code cpl_image_fill_polynomial (cpl_image *ima, const cpl_polynomial *poly, double startx, double stepx, double starty, double stepy)

  Generate an image from a 2d polynomial function.

- cpl_error_code cpl_image_fill_re_im (cpl_image *im_real, cpl_image *im_imag, const cpl_image *self)

  Split a complex image into its real and/or imaginary part(s)
- cpl_error_code cpl_image_fill_rejected (cpl_image *im, double a)

  Set the bad pixels in an image to a fixed value.
- cpl_image * cpl_image_fill_test_create (cpl_size nx, cpl_size ny)

  Generate a test image with pixel type CPL_TYPE_DOUBLE.
- cpl_error_code cpl_image_fill_window (cpl_image *self, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury, double value)

  Fill an image window with a constant.
- cpl_error_code cpl_image_filter (cpl_image *self, const cpl_image *other, const cpl_matrix *kernel, cpl_filter_mode filter, cpl_border_mode border)

  Filter an image using a floating-point kernel.
- cpl_image * cpl_image_filter_linear (const cpl_image *in, const cpl_matrix *ker)

  Compute a linear filtering.
- cpl_error_code cpl_image_filter_mask (cpl_image *self, const cpl_image *other, const cpl_mask *kernel, cpl_filter_mode filter, cpl_border_mode border)

  Filter an image using a binary kernel.
- cpl_image * cpl_image_filter_median (const cpl_image *in, const cpl_matrix *ker)

  Apply a spatial median filter to an image.
- cpl_image * cpl_image_filter_morpho (const cpl_image *in, const cpl_matrix *ker)

  Filter an image in spatial domain with a morpho kernel.
- cpl_image * cpl_image_filter_stdev (const cpl_image *in, const cpl_matrix *ker)

  Standard deviation filter.
- cpl_error_code cpl_image_fit_gaussian (const cpl_image *im, cpl_size xpos, cpl_size ypos, cpl_size size, double *norm, double *xcen, double *ycen, double *sig_x, double *sig_y, double *fwhm_x, double *fwhm↩_y)

  Apply a gaussian fit on an image sub window.
- cpl_error_code cpl_image_flip (cpl_image *im, int angle)

  Flip an image on a given mirror line.
- double cpl_image_get (const cpl_image *image, cpl_size xpos, cpl_size ypos, int *pis_rejected)

  Get the value of a pixel at a given position.
- double cpl_image_get_absflux (const cpl_image *image)

  Computes the sum of absolute values over an image.
- double cpl_image_get_absflux_window (const cpl_image *image, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury)

  Computes the sum of absolute values over an image sub-window.
- cpl_mask * cpl_image_get_bpm (cpl_image *img)

  Gets the bad pixels map.
- const cpl_mask * cpl_image_get_bpm_const (const cpl_image *img)

  Gets the bad pixels map.
- double cpl_image_get_centroid_x (const cpl_image *image)

  Computes the x centroid value over the whole image.
- double cpl_image_get_centroid_x_window (const cpl_image *image, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury)

  Computes the x centroid value over an image sub-window.
- double cpl_image_get_centroid_y (const cpl_image *image)

  Computes the y centroid value over the whole image.
- double cpl_image_get_centroid_y_window (const cpl_image *image, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury)

  Computes the y centroid value over an image sub-window.
- double complex cpl_image_get_complex (const cpl_image *image, cpl_size xpos, cpl_size ypos, int *pis_↩rejected)

Get the value of a complex pixel at a given position.

- void * cpl_image_get_data (cpl_image *img)

  Gets the pixel data.
- const void * cpl_image_get_data_const (const cpl_image *img)

  Gets the pixel data.
- double * cpl_image_get_data_double (cpl_image *img)

  Get the data as a double array.
- double complex * cpl_image_get_data_double_complex (cpl_image *img)

  Get the data as a double complex array.
- const double complex * cpl_image_get_data_double_complex_const (const cpl_image *img)

  Get the data as a double complex array.
- const double * cpl_image_get_data_double_const (const cpl_image *img)

  Get the data as a double array.
- float * cpl_image_get_data_float (cpl_image *img)

  Get the data as a float array.
- float complex * cpl_image_get_data_float_complex (cpl_image *img)

  Get the data as a float complex array.
- const float complex * cpl_image_get_data_float_complex_const (const cpl_image *img)

  Get the data as a float complex array.
- const float * cpl_image_get_data_float_const (const cpl_image *img)

  Get the data as a float array.
- int * cpl_image_get_data_int (cpl_image *img)

  Get the data as a integer array.
- const int * cpl_image_get_data_int_const (const cpl_image *img)

  Get the data as a integer array.
- double cpl_image_get_flux (const cpl_image *image)

  Computes the sum of pixel values over an image.
- double cpl_image_get_flux_window (const cpl_image *image, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury)

  Computes the sum of pixel values over an image sub-window.
- cpl_error_code cpl_image_get_fwhm (const cpl_image *in, cpl_size xpos, cpl_size ypos, double *fwhm_x, double *fwhm_y)

  Compute FWHM values in x and y for an object.
- double cpl_image_get_interpolated (const cpl_image *source, double xpos, double ypos, const cpl_vector *xprofile, double xradius, const cpl_vector *yprofile, double yradius, double *pconfid)

  Interpolate a pixel.
- double cpl_image_get_mad (const cpl_image *image, double *sigma)

  Computes median and median absolute deviation (MAD) on an image.
- double cpl_image_get_mad_window (const cpl_image *image, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury, double *sigma)

  Computes median and median absolute deviation (MAD) on an image window.
- double cpl_image_get_max (const cpl_image *image)

  computes maximum pixel value over an image.
- double cpl_image_get_max_window (const cpl_image *image, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury)

  computes maximum pixel value over an image sub-window.
- cpl_error_code cpl_image_get_maxpos (const cpl_image *image, cpl_size *px, cpl_size *py)

  Computes maximum pixel value and position over an image.
- cpl_error_code cpl_image_get_maxpos_window (const cpl_image *image, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury, cpl_size *px, cpl_size *py)

  Computes maximum pixel value and position over an image sub window.

- double cpl_image_get_mean (const cpl_image *image)

  computes mean pixel value over an image.
- double cpl_image_get_mean_window (const cpl_image *image, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury)

  computes mean pixel value over an image sub-window.
- double cpl_image_get_median (const cpl_image *image)

  computes median pixel value over an image.
- double cpl_image_get_median_dev (const cpl_image *image, double *sigma)

  Computes median and mean absolute median deviation on an image window.
- double cpl_image_get_median_dev_window (const cpl_image *image, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury, double *sigma)

  Computes median and mean absolute median deviation on an image window.
- double cpl_image_get_median_window (const cpl_image *image, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury)

  computes median pixel value over an image sub-window.
- double cpl_image_get_min (const cpl_image *image)

  computes minimum pixel value over an image.
- double cpl_image_get_min_window (const cpl_image *image, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury)

  computes minimum pixel value over an image sub-window.
- cpl_error_code cpl_image_get_minpos (const cpl_image *image, cpl_size *px, cpl_size *py)

  Computes minimum pixel value and position over an image.
- cpl_error_code cpl_image_get_minpos_window (const cpl_image *image, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury, cpl_size *px, cpl_size *py)

  Computes minimum pixel value and position over an image sub window.
- cpl_size cpl_image_get_size_x (const cpl_image *img)

  Get the image x size.
- cpl_size cpl_image_get_size_y (const cpl_image *img)

  Get the image y size.
- double cpl_image_get_sqflux (const cpl_image *image)

  Computes the sum of squared values over an image.
- double cpl_image_get_sqflux_window (const cpl_image *image, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury)

  Computes the sum of squared values over an image sub-window.
- double cpl_image_get_stdev (const cpl_image *image)

  computes pixel standard deviation over an image.
- double cpl_image_get_stdev_window (const cpl_image *image, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury)

  computes pixel standard deviation over an image sub-window.
- cpl_type cpl_image_get_type (const cpl_image *img)

  Get the image type.
- cpl_error_code cpl_image_hypot (cpl_image *self, const cpl_image *first, const cpl_image *second)

  The pixel-wise Euclidean distance function of the images.
- cpl_bivector * cpl_image_iqe (const cpl_image *in, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury)

  Compute an image quality estimation for an object.
- int cpl_image_is_rejected (const cpl_image *im, cpl_size x, cpl_size y)

  Test if a pixel is good or bad.
- cpl_image * cpl_image_labelise_mask_create (const cpl_mask *in, cpl_size *nbobjs)

  Labelise a mask to differentiate different objects.
- cpl_image * cpl_image_load (const char *filename, cpl_type im_type, cpl_size pnum, cpl_size xtnum)

  Load an image from a FITS file.

- cpl_image * cpl_image_load_window (const char *filename, cpl_type im_type, cpl_size pnum, cpl_size xtnum, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury)

  Load an image from a FITS file.
- cpl_error_code cpl_image_logarithm (cpl_image *self, double base)

  Compute the elementwise logarithm of the image.
- cpl_image * cpl_image_logarithm_create (const cpl_image *image, double base)

  Create a new image by taking the elementwise logarithm of an image.
- cpl_error_code cpl_image_move (cpl_image *im, cpl_size nb_cut, const cpl_size *new_pos)

  Permute tiles in an image.
- cpl_error_code cpl_image_multiply (cpl_image *im1, const cpl_image *im2)

  Multiply two images, store the result in the first image.
- cpl_image * cpl_image_multiply_create (const cpl_image *image1, const cpl_image *image2)

  Multiply two images.
- cpl_error_code cpl_image_multiply_scalar (cpl_image *self, double scalar)

  Elementwise multiplication of an image with a scalar.
- cpl_image * cpl_image_multiply_scalar_create (const cpl_image *image, double factor)

  Create a new image by multiplication of a scalar and an image.
- cpl_image * cpl_image_new (cpl_size nx, cpl_size ny, cpl_type type)

  Allocate an image structure and pixel buffer for a image.
- cpl_image * cpl_image_new_from_mask (const cpl_mask *mask)

  Create an int image from a mask.
- cpl_error_code cpl_image_normalise (cpl_image *image, cpl_norm mode)

  Normalise pixels in an image.
- cpl_image * cpl_image_normalise_create (const cpl_image *image_in, cpl_norm mode)

  Create a new normalised image from an existing image.
- cpl_error_code cpl_image_not (cpl_image *self, const cpl_image *first)

  The bit-wise complement (not) of an image with integer pixels.
- cpl_error_code cpl_image_or (cpl_image *self, const cpl_image *first, const cpl_image *second)

  The bit-wise or of two images with integer pixels.
- cpl_error_code cpl_image_or_scalar (cpl_image *self, const cpl_image *first, cpl_bitmask second)

  The bit-wise or of a scalar and an image with integer pixels.
- cpl_error_code cpl_image_power (cpl_image *self, double exponent)

  Compute the elementwise power of the image.
- cpl_image * cpl_image_power_create (const cpl_image *image, double exponent)

  Create a new image by elementwise raising of an image to a power.
- cpl_image * cpl_image_rebin (const cpl_image *image, cpl_size xstart, cpl_size ystart, cpl_size xstep, cpl_size ystep)

  Rebin an image.
- cpl_error_code cpl_image_reject (cpl_image *im, cpl_size x, cpl_size y)

  Set a pixel as bad in an image.
- cpl_error_code cpl_image_reject_from_mask (cpl_image *im, const cpl_mask *map)

  Set the bad pixels in an image as defined in a mask.
- cpl_error_code cpl_image_reject_value (cpl_image *self, cpl_value mode)

  Reject pixels with the specified special value(s)
- cpl_error_code cpl_image_save (const cpl_image *self, const char *filename, cpl_type type, const cpl_propertylist *plist, unsigned mode)

  Save an image to a FITS file.
- cpl_error_code cpl_image_set (cpl_image *image, cpl_size xpos, cpl_size ypos, double value)

  Set the pixel at the given position to the given value.
- cpl_mask * cpl_image_set_bpm (cpl_image *self, cpl_mask *bpm)

  Replace the bad pixel map of the image.

- cpl_error_code cpl_image_set_complex (cpl_image *image, cpl_size xpos, cpl_size ypos, double complex value)

  Set the pixel at the given position to the given complex value.
- cpl_error_code cpl_image_shift (cpl_image *self, cpl_size dx, cpl_size dy)

  Shift an image by integer offsets.
- cpl_error_code cpl_image_subtract (cpl_image *im1, const cpl_image *im2)

  Subtract two images, store the result in the first image.
- cpl_image * cpl_image_subtract_create (const cpl_image *image1, const cpl_image *image2)

  Subtract two images.
- cpl_error_code cpl_image_subtract_scalar (cpl_image *self, double scalar)

  Elementwise subtraction of a scalar from an image.
- cpl_image * cpl_image_subtract_scalar_create (const cpl_image *image, double subtrahend)

  Create an image by elementwise subtraction of a scalar from an image.
- cpl_error_code cpl_image_threshold (cpl_image *image_in, double lo_cut, double hi_cut, double assign_↩ lo_cut, double assign_hi_cut)

  Threshold an image to a given interval.
- cpl_error_code cpl_image_turn (cpl_image *self, int rot)

  Rotate an image by a multiple of 90 degrees clockwise.
- cpl_mask * cpl_image_unset_bpm (cpl_image *self)

  Remove the bad pixel map from the image.
- void * cpl_image_unwrap (cpl_image *d)

  Free memory associated to an cpl_image object, but the pixel buffer.
- cpl_error_code cpl_image_warp (cpl_image *out, const cpl_image *in, const cpl_image *deltax, const cpl↩ _image *deltay, const cpl_vector *xprofile, double xradius, const cpl_vector *yprofile, double yradius)

  Warp an image.
- cpl_error_code cpl_image_warp_polynomial (cpl_image *out, const cpl_image *in, const cpl_polynomial *poly_x, const cpl_polynomial *poly_y, const cpl_vector *xprofile, double xradius, const cpl_vector *yprofile, double yradius)

  Warp an image according to a 2D polynomial transformation.
- cpl_image * cpl_image_wrap_double (cpl_size nx, cpl_size ny, double *pixels)

  Create a double image using an existing pixel buffer.
- cpl_image * cpl_image_wrap_double_complex (cpl_size nx, cpl_size ny, double complex *pixels)

  Create a double complex image using an existing pixel buffer.
- cpl_image * cpl_image_wrap_float (cpl_size nx, cpl_size ny, float *pixels)

  Create a float image using an existing pixel buffer.
- cpl_image * cpl_image_wrap_float_complex (cpl_size nx, cpl_size ny, float complex *pixels)

  Create a float complex image using an existing pixel buffer.
- cpl_image * cpl_image_wrap_int (cpl_size nx, cpl_size ny, int *pixels)

  Create an integer image using an existing pixel buffer.
- cpl_error_code cpl_image_xor (cpl_image *self, const cpl_image *first, const cpl_image *second)

  The bit-wise xor of two images with integer pixels.
- cpl_error_code cpl_image_xor_scalar (cpl_image *self, const cpl_image *first, cpl_bitmask second)

  The bit-wise xor of a scalar and an image with integer pixels.
- cpl_vector * cpl_vector_new_from_image_column (const cpl_image *image_in, cpl_size pos)

  Extract a column from an image.
- cpl_vector * cpl_vector_new_from_image_row (const cpl_image *image_in, cpl_size pos)

  Extract a row from an image.

4.24.1 Detailed Description

This module provides functions to create, use, and destroy a *cpl_image*. A *cpl_image* is a 2-dimensional data structure with a pixel type (one of `CPL_TYPE_INT`, `CPL_TYPE_FLOAT`, `CPL_TYPE_DOUBLE`, `CPL_TYPE_FLOAT_COMPLEX` or `CPL_TYPE_DOUBLE_COMPLEX`) and an optional bad pixel map.

The pixel indexing follows the FITS convention in the sense that the lower left pixel in a CPL image has index (1, 1). The pixel buffer is stored row-wise so for optimum performance any pixel-wise access should be done likewise.

Functionality include: FITS I/O Image arithmetic, casting, extraction, thresholding, filtering, resampling Bad pixel handling Image statistics Generation of test images Special functions, such as the image quality estimator

Synopsis:

```
#include "cpl_image.h"
```

Usage: define the following preprocessor symbols as needed, then include this file

4.24.2 Typedef Documentation

4.24.2.1 cpl_value

```
typedef enum _cpl_value_ cpl_value
```

The CPL special value. It is a bit field.

4.24.3 Enumeration Type Documentation

4.24.3.1 _cpl_value_

```
enum _cpl_value_
```

The special values that can be rejected They are a bit-field and can be combined with bitwise or.

Enumerator

CPL_VALUE_NAN	Not-a-Number (NaN)
CPL_VALUE_PLUSINF	Plus Infinity
CPL_VALUE_MINUSINF	Minus Infinity
CPL_VALUE_ZERO	Zero
CPL_VALUE_INF	Infinity with any sign
CPL_VALUE_NOTFINITE	NaN or infinity with any sign

4.24.4 Function Documentation

4.24.4.1 cpl_image_abs()

```
cpl_error_code cpl_image_abs (
            cpl_image * image )
```

Take the absolute value of an image.

Parameters

image	Image to be modified in place

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

Set each pixel to its absolute value.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure_code, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, and CPL_ERROR_NULL_↩ INPUT.

Referenced by cpl_image_abs_create(), cpl_image_fill_jacobian(), and cpl_image_fill_jacobian_polynomial().

4.24.4.2 cpl_image_abs_create()

```
cpl_image* cpl_image_abs_create (
            const cpl_image * image_in )
```

Take the absolute value of an image.

Parameters

image↩ *_in*	Image operand.

Returns

1 newly allocated image or NULL on error

See also

cpl_image_abs

For each pixel, out = abs(in). The returned image must be deallocated using cpl_image_delete().

References cpl_ensure, cpl_error_get_code(), cpl_image_abs(), cpl_image_delete(), and cpl_image_duplicate().

4.24.4.3 cpl_image_accept()

```
cpl_error_code cpl_image_accept (
            cpl_image * im,
            cpl_size x,
            cpl_size y )
```

Set a pixel as good in an image.

Parameters

im	the input image
x	the x pixel position in the image (first pixel is 1)
y	the y pixel position in the image (first pixel is 1)

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ACCESS_OUT_OF_RANGE if the specified position is out of the image

References cpl_ensure_code, CPL_ERROR_ACCESS_OUT_OF_RANGE, and CPL_ERROR_NULL_INPUT.

4.24.4.4 cpl_image_accept_all()

```
cpl_error_code cpl_image_accept_all (
            cpl_image * self )
```

Set all pixels in the image as good.

Parameters

self	the input image

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References CPL_ERROR_NONE, cpl_image_unset_bpm(), and cpl_mask_delete().

Referenced by cpl_detector_interpolate_rejected(), and cpl_imagelist_collapse_create().

4.24.4.5 cpl_image_add()

```
cpl_error_code cpl_image_add (
             cpl_image * im1,
             const cpl_image * im2 )
```

Add two images, store the result in the first image.

Parameters

im1	first operand.
im2	second operand.

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

The first input image is modified to contain the result of the operation.

The bad pixel map of the first image becomes the union of the bad pixel maps of the input images.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if the input images have different sizes
- CPL_ERROR_TYPE_MISMATCH if the second input image has complex type while the first one does not

Referenced by cpl_image_fill_test_create(), and cpl_imagelist_collapse_create().

4.24.4.6 cpl_image_add_create()

```
cpl_image* cpl_image_add_create (
             const cpl_image * image1,
             const cpl_image * image2 )
```

Add two images.

Parameters

image1	first operand
image2	second operand

Returns

1 newly allocated image or NULL on error

Creates a new image, being the result of the operation, and returns it to the caller. The returned image must be deallocated using cpl_image_delete(). The function supports images with different types among CPL_TYPE_INT, CPL_TYPE_FLOAT and CPL_TYPE_DOUBLE. The returned image type is the one of the first passed image.

The bad pixels map of the result is the union of the bad pixels maps of the input images.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if the input images have different sizes
- CPL_ERROR_TYPE_MISMATCH if the second input image has complex type while the first one does not

4.24.4.7 cpl_image_add_scalar()

```
cpl_error_code cpl_image_add_scalar (
            cpl_image * self,
            double scalar )
```

Elementwise addition of a scalar to an image.

Parameters

self	Image to be modified in place.
scalar	Number to add

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

Modifies the image by adding a number to each of its pixels.

The operation is always performed in double precision, with a final cast of the result to the image pixel type.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure_code, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INP↩UT, and cpl_image_get_type().

Referenced by cpl_image_add_scalar_create(), cpl_image_new_from_accepted(), and cpl_imagelist_add_scalar().

4.24.4.8 cpl_image_add_scalar_create()

```
cpl_image* cpl_image_add_scalar_create (
             const cpl_image * image,
             double addend )
```

Create a new image by elementwise addition of a scalar to an image.

Parameters

image	Image to add
addend	Number to add

Returns

1 newly allocated image or NULL in case of an error

See also

cpl_image_add_scalar

Creates a new image, being the result of the operation, and returns it to the caller. The returned image must be deallocated using cpl_image_delete(). The function supports images with different types among CPL_TYPE_INT, CPL_TYPE_FLOAT and CPL_TYPE_DOUBLE. The type of the created image is that of the passed image.

References cpl_ensure, cpl_error_get_code(), cpl_image_add_scalar(), cpl_image_delete(), and cpl_image_↩ duplicate().

4.24.4.9 cpl_image_and()

```
cpl_error_code cpl_image_and (
             cpl_image * self,
             const cpl_image * first,
             const cpl_image * second )
```

The bit-wise and of two images with integer pixels.

Parameters

self	Pre-allocated image to hold the result
first	First operand, or NULL for an in-place operation
second	Second operand

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

Note

CPL_TYPE_INT is required

See also

cpl_mask_and() for the equivalent logical operation

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if the images have different sizes
- CPL_ERROR_INVALID_TYPE if the passed image type is as required

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_INVALID_TYPE, CPL_E↩RROR_NULL_INPUT, cpl_image_get_type(), cpl_type_get_name(), and CPL_TYPE_INT.

4.24.4.10 cpl_image_and_scalar()

```
cpl_error_code cpl_image_and_scalar (
            cpl_image * self,
            const cpl_image * first,
            cpl_bitmask second )
```

The bit-wise and of a scalar and an image with integer pixels.

Parameters

self	Pre-allocated image to hold the result
first	First operand, or NULL for an in-place operation
second	Second operand (scalar)

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

Note

CPL_TYPE_INT is required

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if the images have different sizes
- CPL_ERROR_INVALID_TYPE if the passed image type is as required

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_INVALID_TYPE, CPL_E↩RROR_NONE, CPL_ERROR_NULL_INPUT, cpl_image_get_type(), cpl_type_get_name(), cpl_type_get_sizeof(), and CPL_TYPE_INT.

4.24.4.11 cpl_image_average_create()

```
cpl_image* cpl_image_average_create (
             const cpl_image * image_1,
             const cpl_image * image_2 )
```

Build the average of two images.

Parameters

image_1	First image operand.
image_2	Second image operand.

Returns

1 newly allocated image or NULL on error

Builds the average of two images and returns a newly allocated image, to be deallocated using cpl_image_delete(). The average is arithmetic, i.e. outpix=(pix1+pix2)/2 Images can be of type CPL_TYPE_INT, CPL_TYPE_FLOAT or CPL_TYPE_DOUBLE.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NULL_INPUT, cpl_mask_duplicate(), and cpl_mask_or().

4.24.4.12 cpl_image_cast()

```
cpl_image* cpl_image_cast (
             const cpl_image * self,
             cpl_type type )
```

Convert a cpl_image to a given type.

Parameters

self	The image to convert
type	The destination type

Returns

the newly allocated cpl_image or NULL on error

Casting to non-complex types is only supported for non-complex types.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the passed type is invalid
- CPL_ERROR_TYPE_MISMATCH if the passed image type is complex and requested casting type is non-complex.
- CPL_ERROR_INVALID_TYPE if the passed pixel type is not supported

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NULL_↩ INPUT, CPL_ERROR_TYPE_MISMATCH, cpl_free(), cpl_image_duplicate(), cpl_malloc(), cpl_mask_duplicate(), CPL_TYPE_DOUBLE, CPL_TYPE_DOUBLE_COMPLEX, CPL_TYPE_FLOAT, CPL_TYPE_FLOAT_COMPLEX, cpl_type_get_name(), cpl_type_get_sizeof(), CPL_TYPE_INT, and CPL_TYPE_INVALID.

Referenced by cpl_image_fit_gaussian(), and cpl_imagelist_cast().

4.24.4.13 cpl_image_collapse_create()

```
cpl_image* cpl_image_collapse_create (
            const cpl_image * self,
            int direction )
```

Collapse an image along its rows or columns.

Parameters

self	Input image to collapse.
direction	Collapsing direction.

Returns

1 newly allocated 1D image or NULL on error

On success the function returns a 1D image, created by adding up all pixels on the same row or column.

```
Collapse along y (sum of rows):

p7  p8  p9     Input image is a 3x3 image containing 9 pixels.
p4  p5  p6     The output is an image containing one row with
p1  p2  p3     3 pixels A, B, C, where:
----------

A   B   C      A = p1+p4+p7
               B = p2+p5+p8
               C = p3+p6+p9

If p7 is a bad pixel, A = (p1+p4)*3/2.
If p1, p4, p7 are bad, A is flagged as bad.
```

Provide the collapsing direction as an int. Give 0 to collapse along y (sum of rows) and get an image with a single row in output, or give 1 to collapse along x (sum of columns) to get an image with a single column in output. Only the good pixels are collapsed. Images can be of type CPL_TYPE_INT, CPL_TYPE_FLOAT or CPL_TYPE_DOUBLE. The returned image must be deallocated using cpl_image_delete().

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References cpl_ensure, cpl_error_get_code(), cpl_image_collapse_window_create(), cpl_image_get_size_x(), and cpl_image_get_size_y().

4.24.4.14 cpl_image_collapse_median_create()

```
cpl_image* cpl_image_collapse_median_create (
             const cpl_image * self,
             int direction,
             cpl_size drop_ll,
             cpl_size drop_ur )
```

Collapse an image along its rows or columns, with filtering.

Parameters

self	Input image to collapse.
direction	Collapsing direction.
drop_ll	Ignore this many lower rows/leftmost columns
drop_ur	Ignore this many upper rows/rightmost columns

Returns

1 newly allocated image having 1 row or 1 column or NULL on error

See also

cpl_image_collapse_create()

The collapsing direction is defined as for cpl_image_collapse_create(). For each output pixel, the median of the corresponding non-ignored pixels is computed. A combination of bad pixels and drop parameters can cause a median value in the output image to be undefined. Such pixels will be flagged as bad and set to zero.

If the output would contain only bad pixels an error is set.

Images can be of type CPL_TYPE_INT, CPL_TYPE_FLOAT or CPL_TYPE_DOUBLE. The returned image must be deallocated using cpl_image_delete().

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if a rejection parameter is negative, or if the sum of ignored pixels is bigger than the image size in the collapsing direction
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported
- CPL_ERROR_DATA_NOT_FOUND if the output image would have only bad pixels

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, and CPL_ERROR_N↩ ULL_INPUT.

4.24.4.15 cpl_image_collapse_window_create()

```
cpl_image* cpl_image_collapse_window_create (
             const cpl_image * self,
             cpl_size llx,
             cpl_size lly,
             cpl_size urx,
             cpl_size ury,
             int direction )
```

Collapse an image region along its rows or columns.

Parameters

self	Image to collapse.
llx	lower left x coord.
lly	lower left y coord
urx	upper right x coord
ury	upper right y coord
direction	Collapsing direction.

Returns

a newly allocated image or NULL on error

See also

cpl_image_collapse_create()

llx, lly, urx, ury are the image region coordinates in FITS convention. Those specified bounds are included in the collapsed region.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the specified window is not valid

References cpl_image_get_type(), CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT, and CPL_TYPE_INT.

Referenced by cpl_image_collapse_create().

4.24.4.16 cpl_image_conjugate()

```
cpl_error_code cpl_image_conjugate (
            cpl_image * self,
            const cpl_image * other )
```

Complex conjugate the pixels in a complex image.

Parameters

self	Pre-allocated complex image to hold the result
other	The complex image to conjugate, may equal self

Returns

CPL_ERROR_NONE or _cpl_error_code_ on error

Note

The two images must match in size and precision

Any bad pixels are also conjugated.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT If an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT If the images have different sizes
- CPL_ERROR_INVALID_TYPE If an input image is not of complex type
- CPL_ERROR_TYPE_MISMATCH If the images have different precision

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, CPL_TYPE_DOUBLE_COMPLEX, and CPL_TYPE_FLOAT_COMPLEX.

4.24.4.17 cpl_image_copy()

```
cpl_error_code cpl_image_copy (
            cpl_image * im1,
            const cpl_image * im2,
            cpl_size xpos,
            cpl_size ypos )
```

Copy one image into another.

Parameters

im1	the image in which im2 is inserted
im2	the inserted image
xpos	the x pixel position in im1 where the lower left pixel of im2 should go (from 1 to the x size of im1)
ypos	the y pixel position in im1 where the lower left pixel of im2 should go (from 1 to the y size of im1)

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error.

Note

The two pixel buffers may not overlap

(xpos, ypos) must be a valid position in im1. If im2 is bigger than the place left in im1, the part that falls outside of im1 is simply ignored, an no error is raised. The bad pixels are inherited from im2 in the concerned im1 zone.

The two input images must be of the same type, namely one of CPL_TYPE_INT, CPL_TYPE_FLOAT, CPL_TYP↩E_DOUBLE.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_TYPE_MISMATCH if the input images are of different types
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported
- CPL_ERROR_ACCESS_OUT_OF_RANGE if xpos or ypos are outside the specified range

References cpl_ensure_code, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_NONE, CPL_ERRO↩R_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, cpl_image_get_data(), cpl_image_get_size_x(), cpl_image↩_get_size_y(), cpl_image_get_type(), cpl_mask_copy(), cpl_mask_delete(), cpl_mask_new(), and cpl_type_get_↩sizeof().

Referenced by cpl_image_filter_mask().

4.24.4.18 cpl_image_count_rejected()

```
cpl_size cpl_image_count_rejected (
            const cpl_image * im )
```

Count the number of bad pixels declared in an image.

Parameters

im	the input image

Returns

the number of bad pixels or -1 if the input image is NULL

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, CPL_ERROR_NULL_INPUT, and cpl_mask_count().

4.24.4.19 cpl_image_delete()

```
void cpl_image_delete (
            cpl_image * d )
```

Free memory associated to an cpl_image object.

Parameters

d	Image to destroy.

Returns

void

Frees all memory associated with a cpl_image. If the passed image is NULL, the function returns without doing anything.

References cpl_free(), and cpl_mask_delete().

Referenced by cpl_apertures_extract_window(), cpl_fit_image_gaussian(), cpl_image_abs_create(), cpl_image_add_scalar_create(), cpl_image_divide_create(), cpl_image_divide_scalar_create(), cpl_image_exponential_create(), cpl_image_fill_jacobian(), cpl_image_fill_test_create(), cpl_image_filter_linear(), cpl_image_filter_morpho(), cpl_image_fit_gaussian(), cpl_image_iqe(), cpl_image_logarithm_create(), cpl_image_multiply_scalar_create(), cpl_image_normalise_create(), cpl_image_power_create(), cpl_image_subtract_scalar_create(), cpl_imagelist_cast(), cpl_imagelist_collapse_create(), cpl_imagelist_empty(), cpl_imagelist_erase(), and cpl_imagelist_set().

4.24.4.20 cpl_image_divide()

```
cpl_error_code cpl_image_divide (
            cpl_image * im1,
            const cpl_image * im2 )
```

Divide two images, store the result in the first image.

Parameters

im1	first operand.
im2	second operand.

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

See also

cpl_image_add()

Note

The result of division with a zero-valued pixel is marked as a bad pixel.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if the input images have different sizes
- CPL_ERROR_TYPE_MISMATCH if the second input image has complex type while the first one does not
- CPL_ERROR_DIVISION_BY_ZERO is all pixels in the divisor are zero

References cpl_ensure_code, CPL_ERROR_DIVISION_BY_ZERO, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, cpl_mask_delete(), cpl_mask_duplicate(), cpl_mask_get_data(), cpl_mask_new(), cpl_mask_or(), CPL_TYPE_DOUBLE, CPL_TYPE_DOUBLE_COMPLEX, CPL_TYPE_FLOAT, CPL_TYPE_FLOAT_COMPLEX, and CPL_TYPE_INT.

Referenced by cpl_imagelist_collapse_create().

4.24.4.21 cpl_image_divide_create()

```
cpl_image* cpl_image_divide_create (
             const cpl_image * image1,
             const cpl_image * image2 )
```

Divide two images.

Parameters

image1	first operand
image2	second operand

Returns

1 newly allocated image or NULL on error

See also

cpl_image_add_create()

cpl_image_divide() The result of division with a zero-valued pixel is marked as a bad pixel.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if the input images have different sizes

- CPL_ERROR_TYPE_MISMATCH if the second input image has complex type while the first one does not
- CPL_ERROR_DIVISION_BY_ZERO is all pixels in the divisor are zero

References cpl_ensure, CPL_ERROR_DIVISION_BY_ZERO, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_E↩RROR_INVALID_TYPE, CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, cpl_free(), cpl_image↩_delete(), cpl_image_wrap_double(), cpl_image_wrap_double_complex(), cpl_image_wrap_float(), cpl_image_↩wrap_float_complex(), cpl_image_wrap_int(), cpl_malloc(), cpl_mask_delete(), cpl_mask_duplicate(), cpl_mask_↩get_data(), cpl_mask_new(), cpl_mask_or(), CPL_TYPE_DOUBLE, CPL_TYPE_DOUBLE_COMPLEX, CPL_TY↩PE_FLOAT, CPL_TYPE_FLOAT_COMPLEX, and CPL_TYPE_INT.

4.24.4.22 cpl_image_divide_scalar()

```
cpl_error_code cpl_image_divide_scalar (
            cpl_image * self,
            double scalar )
```

Elementwise division of an image with a scalar.

Parameters

self	Image to be modified in place.
scalar	Non-zero number to divide with

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

See also

cpl_image_add_scalar()

Modifies the image by dividing each of its pixels with a number.

If the scalar is zero the image is not modified and an error is returned.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_DIVISION_BY_ZERO a division by 0 occurs

References cpl_ensure_code, CPL_ERROR_DIVISION_BY_ZERO, CPL_ERROR_INVALID_TYPE, CPL_ERR↩OR_NONE, CPL_ERROR_NULL_INPUT, and cpl_image_get_type().

Referenced by cpl_image_divide_scalar_create(), cpl_image_fft(), and cpl_imagelist_divide_scalar().

4.24.4.23 cpl_image_divide_scalar_create()

```
cpl_image* cpl_image_divide_scalar_create (
            const cpl_image * image,
            double divisor )
```

Create a new image by elementwise division of an image with a scalar.

Parameters

image	Image to divide
divisor	Non-zero number to divide with

Returns

1 newly allocated image or NULL in case of an error

See also

cpl_image_divide_scalar

cpl_image_add_scalar_create

References cpl_ensure, cpl_error_get_code(), cpl_image_delete(), cpl_image_divide_scalar(), and cpl_image_↩ duplicate().

4.24.4.24 cpl_image_dump_structure()

```
cpl_error_code cpl_image_dump_structure (
            const cpl_image * self,
            FILE * stream )
```

Dump structural information of a CPL image.

Parameters

self	Image to dump
stream	Output stream, accepts `stdout` or `stderr`

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_FILE_IO if a write operation fails

References cpl_ensure_code, CPL_ERROR_FILE_IO, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_↩ mask_count(), CPL_SIZE_FORMAT, and cpl_type_get_name().

Referenced by cpl_imagelist_dump_structure().

4.24.4.25 cpl_image_dump_window()

```
cpl_error_code cpl_image_dump_window (
            const cpl_image * self,
            cpl_size llx,
            cpl_size lly,
            cpl_size urx,
            cpl_size ury,
            FILE * stream )
```

Dump pixel values in a CPL image.

Parameters

self	Image to dump
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Specifies the window position
ury	Specifies the window position
stream	Output stream, accepts `stdout` or `stderr`

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_FILE_IO if a write operation fails
- CPL_ERROR_ACCESS_OUT_OF_RANGE if the defined window is not in the image
- CPL_ERROR_ILLEGAL_INPUT if the window definition is wrong (e.g llx $>$ urx)

References cpl_ensure_code, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_FILE_IO, CPL_ER↩ ROR_ILLEGAL_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_image_get(), cpl_image_get_↩ complex(), cpl_mask_get(), cpl_mask_is_empty(), CPL_SIZE_FORMAT, CPL_TYPE_DOUBLE_COMPLEX, CP↩ L_TYPE_FLOAT_COMPLEX, and CPL_TYPE_INT.

Referenced by cpl_imagelist_dump_window().

4.24.4.26 cpl_image_duplicate()

```
cpl_image* cpl_image_duplicate (
            const cpl_image * src )
```

Copy an image.

Parameters

src	Source image.

Returns

1 newly allocated image, or NULL on error.

Copy an image into a new image object. The pixels and the bad pixel map are also copied. The returned image must be deallocated using cpl_image_delete().

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, CPL_ERROR_NULL_INPUT, cpl_malloc(), cpl_mask_duplicate(), and cpl_type_get_sizeof().

Referenced by cpl_image_abs_create(), cpl_image_add_scalar_create(), cpl_image_cast(), cpl_image_divide_scalar_create(), cpl_image_exponential_create(), cpl_image_logarithm_create(), cpl_image_multiply_scalar_create(), cpl_image_normalise_create(), cpl_image_power_create(), cpl_image_subtract_scalar_create(), cpl_imagelist_collapse_create(), and cpl_imagelist_duplicate().

4.24.4.27 cpl_image_exponential()

```
cpl_error_code cpl_image_exponential (
            cpl_image * self,
            double base )
```

Compute the elementwise exponential of the image.

Parameters

self	Image to be modified in place.
base	Base of the exponential.

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

Modifies the image by computing the base-scalar exponential of each of its pixels.

Images can be of type CPL_TYPE_INT, CPL_TYPE_FLOAT or CPL_TYPE_DOUBLE.

Pixels for which the power of the given base is not defined are rejected and set to zero.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References cpl_ensure_code, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INP↩UT, cpl_image_get_type(), CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT, cpl_type_get_name(), and CPL_TYPE_INT.

Referenced by cpl_image_exponential_create(), and cpl_imagelist_exponential().

4.24.4.28 cpl_image_exponential_create()

```
cpl_image* cpl_image_exponential_create (
            const cpl_image * image,
            double base )
```

Create a new image by elementwise exponentiation of an image.

Parameters

image	Image to exponentiate
base	Base of the exponential

Returns

1 newly allocated image or NULL in case of an error

See also

cpl_image_logarithm

cpl_image_add_scalar_create

References cpl_ensure, cpl_error_get_code(), cpl_image_delete(), cpl_image_duplicate(), and cpl_image_↩exponential().

4.24.4.29 cpl_image_extract()

```
cpl_image* cpl_image_extract (
            const cpl_image * in,
            cpl_size llx,
            cpl_size lly,
            cpl_size urx,
            cpl_size ury )
```

Extract a rectangular zone from an image into another image.

Parameters

in	Input image
llx	Lower left X coordinate
lly	Lower left Y coordinate
urx	Upper right X coordinate
ury	Upper right Y coordinate

Returns

1 newly allocated image or NULL on error

Note

The returned image must be deallocated using cpl_image_delete()

The input coordinates define the extracted region by giving the coordinates of the lower left and upper right corners (inclusive).

Coordinates must be provided in the FITS convention: lower left corner of the image is at (1,1), x increasing from left to right, y increasing from bottom to top. Images can be of type CPL_TYPE_INT, CPL_TYPE_FLOAT or CPL↩ _TYPE_DOUBLE.

If the input image has a bad pixel map and if the extracted rectangle has bad pixel(s), then the extracted image will have a bad pixel map, otherwise it will not.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the window coordinates are not valid
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, and CPL_ERROR_N↩ ULL_INPUT.

Referenced by cpl_apertures_extract_window(), cpl_fit_image_gaussian(), and cpl_image_iqe().

4.24.4.30 cpl_image_extract_subsample()

```
cpl_image* cpl_image_extract_subsample (
             const cpl_image * image,
             cpl_size xstep,
             cpl_size ystep )
```

Sub-sample an image.

Parameters

image	The image to subsample
xstep	Take every xstep pixel in x
ystep	Take every ystep pixel in y

Returns

The newly allocated sub-sampled image or NULL in error case

See also

cpl_image_extract

step represents the sampling step in x and y: both steps = 2 will create an image with a quarter of the pixels of the input image.

image type can be CPL_TYPE_INT, CPL_TYPE_FLOAT and CPL_TYPE_DOUBLE. If the image has bad pixels, they will be resampled in the same way.

The flux of the sampled pixels will be preserved, while the flux of the pixels not sampled will be lost. Using steps = 2 in each direction on a uniform image will thus create an image with a quarter of the flux.

The returned image must be deallocated using cpl_image_delete().

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if the input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if xstep, ystep are not positive
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NULL_↩ INPUT, and cpl_mask_extract_subsample().

4.24.4.31 cpl_image_fft()

```
cpl_error_code cpl_image_fft (
            cpl_image * img_real,
            cpl_image * img_imag,
            unsigned mode )
```

Fast Fourier Transform a square, power-of-two sized image.

Parameters

img_real	The image real part to be transformed in place
img_imag	The image imaginary part to be transformed in place
mode	The desired FFT options (combined with bitwise or)

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

The input images must be of double type.

If the second passed image is NULL, the resulting imaginary part cannot be returned. This can be useful if the input is real and the output is known to also be real. But if the output has a significant imaginary part, you might want to pass a 0-valued image as the second parameter.

Any rejected pixel is used as if it were a good pixel.

The image must be square with a size that is a power of two.

These are the supported FFT modes: CPL_FFT_DEFAULT: Default, forward FFT transform CPL_FFT_INVERSE: Inverse FFT transform CPL_FFT_UNNORMALIZED: Do not normalize (with N∗N for N-by-N image) on inverse. Has no effect on forward transform. CPL_FFT_SWAP_HALVES: Swap the four quadrants of the result image.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if (one of) the input pointer(s) is NULL
- CPL_ERROR_ILLEGAL_INPUT if the image is not square or if the image size is not a power of 2.
- CPL_ERROR_INVALID_TYPE if mode is 1, e.g. due to a logical or (||) of the allowed FFT options.
- CPL_ERROR_UNSUPPORTED_MODE if mode is otherwise different from the allowed FFT options.
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References cpl_calloc(), cpl_ensure_code, cpl_error_get_code(), CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, CPL_ERROR_UNSUPPORTED_MODE, cpl_free(), cpl_image_divide_scalar(), cpl_image_move(), and CPL_TYPE_DOUBLE.

4.24.4.32 cpl_image_fill_abs_arg()

```
cpl_error_code cpl_image_fill_abs_arg (
            cpl_image * im_abs,
            cpl_image * im_arg,
            const cpl_image * self )
```

Split a complex image into its absolute and argument part(s)

Parameters

im_abs	Pre-allocated image to hold the absolute part, or NULL
im_arg	Pre-allocated image to hold the argument part, or NULL
self	Complex image to process

Returns

CPL_ERROR_NONE or _cpl_error_code_ on error

Note

At least one output image must be non-NULL. The images must match in size and precision

Any bad pixels are also processed.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT If the input image or both output images are NULL
- CPL_ERROR_INCOMPATIBLE_INPUT If the images have different sizes
- CPL_ERROR_INVALID_TYPE If the input image is not of complex type
- CPL_ERROR_TYPE_MISMATCH If the images have different precision

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_INVALID_TYPE, CPL_E↩RROR_NONE, CPL_ERROR_NULL_INPUT, CPL_TYPE_COMPLEX, CPL_TYPE_DOUBLE_COMPLEX, and C↩PL_TYPE_FLOAT_COMPLEX.

4.24.4.33 cpl_image_fill_gaussian()

```
cpl_error_code cpl_image_fill_gaussian (
            cpl_image * ima,
            double xcen,
            double ycen,
            double norm,
            double sig_x,
            double sig_y )
```

Generate an image from a 2d gaussian function.

Parameters

ima	the gaussian image to generate
xcen	x position of the center (1 for the first pixel)
ycen	y position of the center (1 for the first pixel)
norm	norm of the gaussian.
sig↩_x	Sigma in x for the gaussian distribution.
sig↩_y	Sigma in y for the gaussian distribution.

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

This function expects an already allocated image. This function generates an image of a 2d gaussian. The gaussian is defined by the position of its centre, given in pixel coordinates inside the image with the FITS convention (x from 1 to nx, y from 1 to ny), its norm and the value of sigma in x and y.

f(x, y) = (norm/(2∗pi∗sig_x∗sig_y)) ∗ exp(-(x-xcen)^2/(2∗sig_x^2)) ∗ exp(-(y-ycen)^2/(2∗sig_y^2))

The input image type can be CPL_TYPE_FLOAT or CPL_TYPE_DOUBLE.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if (one of) the input pointer(s) is NULL
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References cpl_ensure_code, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, and CPL_ERROR_NULL_↩INPUT.

Referenced by cpl_image_fill_test_create().

4.24.4.34 cpl_image_fill_jacobian()

```
cpl_error_code cpl_image_fill_jacobian (
            cpl_image * out,
            const cpl_image * deltax,
            const cpl_image * deltay )
```

Compute area change ratio for a transformation map.

Parameters

out	Pre-allocated image to hold the result
deltax	The x shifts for each pixel
deltay	The y shifts for each pixel

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_image_warp()

The shifts images *deltax* and *deltay*, describing the transformation, must be of type CPL_TYPE_DOUBLE and of the same size as *out*. For each pixel (u, v) of the *out* image, the deltax and deltay code the following transformation:

```
* u - deltax(u,v) = x
* v - deltay(u,v) = y
*
```

This function writes the density of the (u, v) coordinate system relative to the (x, y) coordinates for each (u, v) pixel of image *out*.

This is trivially obtained by computing the absolute value of the determinant of the Jacobian of the transformation for each pixel of the (u, v) image *out*.

The partial derivatives are estimated at the position (u, v) in the following way:

```
* dx/du = 1 + 1/2 ( deltax(u-1, v) - deltax(u+1, v) )
* dx/dv =     1/2 ( deltax(u, v-1) - deltax(u, v+1) )
* dy/du =     1/2 ( deltay(u-1, v) - deltay(u+1, v) )
* dy/dv = 1 + 1/2 ( deltay(u, v-1) - deltay(u, v+1) )
*
```

Typically this function would be used to determine a flux-conservation factor map for the target image specified in function `cpl_image_warp()`. For example,

```
* cpl_image_warp(out, in, deltax, deltay, xprof, xrad, yprof, yrad);
* correction_map = cpl_image_new(cpl_image_get_size_x(out),
*                                cpl_image_get_size_y(out),
*                                cpl_image_get_type(out));
* cpl_image_fill_jacobian(correction_map, deltax, deltay);
* out_flux_corrected = cpl_image_multiply_create(out, correction_map);
*
```

where *out_flux_corrected* is the resampled image *out* after correction for flux conservation.

Note

The map produced by this function is not applicable for flux conservation in case the transformation implies severe undersampling of the original signal.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if (one of) the input pointer(s) is NULL
- CPL_ERROR_ILLEGAL_INPUT if the polynomial dimensions are not 2
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERR↩ OR_NULL_INPUT, CPL_ERROR_UNSPECIFIED, cpl_image_abs(), cpl_image_delete(), cpl_image_get_data_↩ double(), cpl_image_get_data_double_const(), cpl_image_get_data_float(), cpl_image_get_size_x(), cpl_image↩ _get_size_y(), cpl_image_get_type(), cpl_image_new(), CPL_TYPE_DOUBLE, and CPL_TYPE_FLOAT.

4.24.4.35 cpl_image_fill_jacobian_polynomial()

```
cpl_error_code cpl_image_fill_jacobian_polynomial (
             cpl_image * out,
             const cpl_polynomial * poly_x,
             const cpl_polynomial * poly_y )
```

Compute area change ratio for a 2D polynomial transformation.

Parameters

out	Pre-allocated image to hold the result
poly↩ _x	Defines source x-pos corresponding to destination (u,v).
poly↩ _y	Defines source y-pos corresponding to destination (u,v).

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_image_warp_polynomial()

Given an input image with pixel coordinates (x, y) which is mapped into an output image with pixel coordinates (u, v), and the polynomial inverse transformation (u, v) to (x, y) as in cpl_image_warp_polynomial(), this function writes the density of the (u, v) coordinate system relative to the (x, y) coordinates for each (u, v) pixel of image *out*.

This is trivially obtained by computing the absolute value of the determinant of the Jacobian of the transformation for each pixel of the (u, v) image *out*.

Typically this function would be used to determine a flux-conservation factor map for the target image specified in function `cpl_image_warp_polynomial()`. For example,

```
* cpl_image_warp_polynomial(out, in, poly_x, poly_y, xprof, xrad, yprof, yrad);
* correction_map = cpl_image_new(cpl_image_get_size_x(out),
*                                cpl_image_get_size_y(out),
*                                cpl_image_get_type(out));
* cpl_image_fill_jacobian_polynomial(correction_map, poly_x, poly_y);
* out_flux_corrected = cpl_image_multiply_create(out, correction_map);
*
```

where *out_flux_corrected* is the resampled image *out* after correction for flux conservation.

Note

The map produced by this function is not applicable for flux conservation in case the transformation implies severe undersampling of the original signal.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if (one of) the input pointer(s) is NULL
- CPL_ERROR_ILLEGAL_INPUT if the polynomial dimensions are not 2
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERR↩OR_NULL_INPUT, CPL_ERROR_UNSPECIFIED, cpl_image_abs(), cpl_image_get_data_double(), cpl_image_↩get_data_float(), cpl_image_get_size_x(), cpl_image_get_size_y(), cpl_image_get_type(), cpl_polynomial_delete(), cpl_polynomial_derivative(), cpl_polynomial_duplicate(), cpl_polynomial_eval(), cpl_polynomial_get_dimension(), CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT, cpl_vector_delete(), cpl_vector_get_data(), and cpl_vector_new().

4.24.4.36 cpl_image_fill_noise_uniform()

```
cpl_error_code cpl_image_fill_noise_uniform (
            cpl_image * ima,
            double min_pix,
            double max_pix )
```

Generate an image with uniform random noise distribution.

Parameters

ima	the image to generate
min_pix	Minimum output pixel value.
max_pix	Maximum output pixel value.

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

Generate an image with a uniform random noise distribution. Pixel values are within the provided bounds. This function expects an already allocated image. The input image type can be CPL_TYPE_INT, CPL_TYPE_FLOAT or CPL_TYPE_DOUBLE.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if (one of) the input pointer(s) is NULL
- CPL_ERROR_ILLEGAL_INPUT if min_pix is bigger than max_pix
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_↩ NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_image_fill_test_create().

4.24.4.37 cpl_image_fill_polynomial()

```
cpl_error_code cpl_image_fill_polynomial (
            cpl_image * ima,
            const cpl_polynomial * poly,
            double startx,
            double stepx,
            double starty,
            double stepy )
```

Generate an image from a 2d polynomial function.

Parameters

ima	the polynomial image to generate
poly	the 2d polynomial
startx	the x value associated with the left pixels column
stepx	the x step
starty	the y value associated with the bottom pixels row
stepy	the y step

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

This function expects an already allocated image. The pixel value of the pixel (i, j) is set to poly(startx+(i-1)∗stepx, starty+(j-1)∗stepy).

The input image type can be CPL_TYPE_FLOAT or CPL_TYPE_DOUBLE.

If you want to generate an image whose pixel values are the values of the polynomial applied to the pixel positions, just call cpl_image_fill_polynomial(ima, poly, 1.0, 1.0, 1.0, 1.0);

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if (one of) the input pointer(s) is NULL
- CPL_ERROR_ILLEGAL_INPUT if the polynomial's dimension is not 2
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_↩ NONE, CPL_ERROR_NULL_INPUT, and cpl_polynomial_get_dimension().

4.24.4.38 cpl_image_fill_re_im()

```
cpl_error_code cpl_image_fill_re_im (
            cpl_image * im_real,
            cpl_image * im_imag,
            const cpl_image * self )
```

Split a complex image into its real and/or imaginary part(s)

Parameters

im_real	Pre-allocated image to hold the real part, or NULL
im_imag	Pre-allocated image to hold the imaginary part, or NULL
self	Complex image to process

Returns

CPL_ERROR_NONE or _cpl_error_code_ on error

Note

At least one output image must be non-NULL. The images must match in size and precision

Any bad pixels are also processed.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT If the input image or both output images are NULL
- CPL_ERROR_INCOMPATIBLE_INPUT If the images have different sizes
- CPL_ERROR_INVALID_TYPE If the input image is not of complex type
- CPL_ERROR_TYPE_MISMATCH If the images have different precision

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_INVALID_TYPE, CPL_E↩RROR_NONE, CPL_ERROR_NULL_INPUT, CPL_TYPE_COMPLEX, CPL_TYPE_DOUBLE_COMPLEX, and C↩PL_TYPE_FLOAT_COMPLEX.

4.24.4.39 cpl_image_fill_rejected()

```
cpl_error_code cpl_image_fill_rejected (
            cpl_image * im,
            double a )
```

Set the bad pixels in an image to a fixed value.

Parameters

im	The image to modify.
a	The fixed value

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

Images can be CPL_TYPE_FLOAT, CPL_TYPE_INT, CPL_TYPE_DOUBLE.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT
- CPL_ERROR_INVALID_TYPE if the image pixel type is not supported

References cpl_ensure_code, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INP↩UT, cpl_mask_get_data(), and cpl_mask_is_empty().

Referenced by cpl_imagelist_collapse_create().

4.24.4.40 cpl_image_fill_test_create()

```
cpl_image* cpl_image_fill_test_create (
            cpl_size nx,
            cpl_size ny )
```

Generate a test image with pixel type CPL_TYPE_DOUBLE.

Parameters

nx	x size
ny	y size

Returns

1 newly allocated image or NULL on error

Generates a reference pattern for testing purposes only. The created image has to be deallocated with cpl_image_delete().

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_ILLEGAL_INPUT if nx or ny is non-positive

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, cpl_image_add(), cpl_image_delete(), cpl_image_fill_↩ gaussian(), cpl_image_fill_noise_uniform(), cpl_image_multiply_scalar(), cpl_image_new(), and CPL_TYPE_DO↩ UBLE.

4.24.4.41 cpl_image_fill_window()

```
cpl_error_code cpl_image_fill_window (
            cpl_image * self,
            cpl_size llx,
            cpl_size lly,
            cpl_size urx,
            cpl_size ury,
            double value )
```

Fill an image window with a constant.

Parameters

self	The image to fill
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Specifies the window position
ury	Specifies the window position
value	The value to fill with

Returns

The _cpl_error_code_ or CPL_ERROR_NONE

Note

Any bad pixels are accepted

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the specified window is not valid

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_↩ NULL_INPUT, and CPL_TYPE_COMPLEX.

4.24.4.42 cpl_image_filter()

```
cpl_error_code cpl_image_filter (
             cpl_image * self,
             const cpl_image * other,
             const cpl_matrix * kernel,
             cpl_filter_mode filter,
             cpl_border_mode border )
```

Filter an image using a floating-point kernel.

Parameters

self	Pre-allocated image to hold the filtered result
other	Image to filter
kernel	Pixel weigths
filter	CPL_FILTER_LINEAR, CPL_FILTER_MORPHO
border	CPL_BORDER_FILTER

Returns

CPL_ERROR_NONE or the relevant CPL error code

See also

cpl_image_filter_mask

The two images must have equal dimensions.

The kernel must have an odd number of rows and an odd number of columns and at least one non-zero element.

For scaling filters (`CPL_FILTER_LINEAR_SCALE` and `CPL_FILTER_MORPHO_SCALE`) the flux of the filtered image will be scaled with the sum of the weights of the kernel. If for a given input pixel location the kernel covers only bad pixels, the filtered pixel value is flagged as bad and set to zero.

For flux-preserving filters (`CPL_FILTER_LINEAR` and `CPL_FILTER_MORPHO`) the filtered pixel must have at least one input pixel with a non-zero weight available. Output pixels where this is not the case are set to zero and flagged as bad.

In-place filtering is not supported.

Supported filters: CPL_FILTER_LINEAR, CPL_FILTER_MORPHO, CPL_FILTER_LINEAR_SCALE, CPL↩_FILTER_MORPHO_SCALE.

Supported borders modes: CPL_BORDER_FILTER

Example:

```
cpl_image_filter(filtered, raw, kernel, CPL_FILTER_LINEAR,
                                        CPL_BORDER_FILTER);
```

Beware that the 1st pixel - at (1,1) - in an image is the lower left, while the 1st element in a matrix - at (0,0) - is the top left. Thus to shift an image 1 pixel up and 1 pixel right with the CPL_FILTER_LINEAR and a 3 by 3 kernel, one should set to 1.0 the bottom leftmost matrix element which is at row 3, column 1, i.e.

```
cpl_matrix_set(kernel, 2, 0);
```

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL.
- CPL_ERROR_ILLEGAL_INPUT if the kernel has a side of even length.
- CPL_ERROR_DIVISION_BY_ZERO If the kernel is a zero-matrix.
- CPL_ERROR_ACCESS_OUT_OF_RANGE If the kernel has a side longer than the input image.
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported.
- CPL_ERROR_TYPE_MISMATCH if in median filtering the input and output pixel types differ.
- CPL_ERROR_INCOMPATIBLE_INPUT If the input and output images have incompatible sizes.
- CPL_ERROR_UNSUPPORTED_MODE If the output pixel buffer overlaps the input one (or the kernel), or the border/filter mode is unsupported.

References cpl_image_get_data_const(), cpl_image_get_size_x(), cpl_image_get_size_y(), cpl_image_get_type(), cpl_matrix_get_data_const(), cpl_matrix_get_ncol(), cpl_matrix_get_nrow(), and cpl_type_get_sizeof().

Referenced by cpl_image_filter_linear(), and cpl_image_filter_morpho().

4.24.4.43 cpl_image_filter_linear()

```
cpl_image* cpl_image_filter_linear (
             const cpl_image * in,
             const cpl_matrix * ker )
```

Compute a linear filtering.

Parameters

in	The image to filter
ker	The kernel

Returns

The filtered image or NULL on error

See also

cpl_image_filter()

Deprecated Replace this call with cpl_image_filter() using CPL_FILTER_LINEAR and CPL_BORDER_FILTER.

References CPL_BORDER_FILTER, cpl_ensure, CPL_ERROR_NULL_INPUT, CPL_FILTER_LINEAR, cpl_image_delete(), cpl_image_filter(), and cpl_image_new().

4.24.4.44 cpl_image_filter_mask()

```
cpl_error_code cpl_image_filter_mask (
            cpl_image * self,
            const cpl_image * other,
            const cpl_mask * kernel,
            cpl_filter_mode filter,
            cpl_border_mode border )
```

Filter an image using a binary kernel.

Parameters

self	Pre-allocated image to hold the filtered result
other	Image to filter
kernel	Pixels to use, if set to CPL_BINARY_1
filter	CPL_FILTER_MEDIAN, CPL_FILTER_AVERAGE and more, see below
border	CPL_BORDER_FILTER and more, see below

Returns

CPL_ERROR_NONE or the relevant CPL error code

The kernel must have an odd number of rows and an odd number of columns.

The two images must have equal dimensions, except for the border mode CPL_BORDER_CROP, where the input image must have 2 ∗ hx columns more and 2 ∗ hy rows more than the output image, where the kernel has size (1 + 2 ∗ hx, 1 + 2 ∗ hy).

In standard deviation filtering the kernel must have at least two elements set to CPL_BINARY_1, for others at least one element must be set to CPL_BINARY_1.

Supported pixel types are: CPL_TYPE_INT, CPL_TYPE_FLOAT and CPL_TYPE_DOUBLE.

In median filtering the two images must have the same pixel type.

In standard deviation filtering a filtered pixel must be computed from at least two input pixels, for other filters at least one input pixel must be available. Output pixels where this is not the case are set to zero and flagged as rejected.

In-place filtering is not supported.

Supported modes:

CPL_FILTER_MEDIAN: CPL_BORDER_FILTER, CPL_BORDER_COPY, CPL_BORDER_NOP, CPL_BORDER_CROP.

CPL_FILTER_AVERAGE: CPL_BORDER_FILTER

CPL_FILTER_AVERAGE_FAST: CPL_BORDER_FILTER

CPL_FILTER_STDEV: CPL_BORDER_FILTER

CPL_FILTER_STDEV_FAST: CPL_BORDER_FILTER

Example:

```
cpl_image_filter_mask(filtered, raw, kernel, CPL_FILTER_MEDIAN,
                      CPL_BORDER_FILTER);
```

To shift an image 1 pixel up and 1 pixel right with the CPL_FILTER_MEDIAN filter and a 3 by 3 kernel, one should set to CPL_BINARY_1 the bottom leftmost kernel element - at row 3, column 1, i.e.

```
cpl_mask * kernel = cpl_mask_new(3, 3);
cpl_mask_set(kernel, 1, 1);
```

The kernel required to do a 5 x 5 median filtering is created like this:

```
cpl_mask * kernel = cpl_mask_new(5, 5);
cpl_mask_not(kernel);
```

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL.
- CPL_ERROR_ILLEGAL_INPUT if the kernel has a side of even length.
- CPL_ERROR_DATA_NOT_FOUND If the kernel is empty, or in case of CPL_FILTER_STDEV if the kernel has only one element set to CPL_BINARY_1.
- CPL_ERROR_ACCESS_OUT_OF_RANGE If the kernel has a side longer than the input image.
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported.
- CPL_ERROR_TYPE_MISMATCH if in median filtering the input and output pixel types differ.
- CPL_ERROR_INCOMPATIBLE_INPUT If the input and output images have incompatible sizes.
- CPL_ERROR_UNSUPPORTED_MODE If the output pixel buffer overlaps the input one (or the kernel), or the border/filter mode is unsupported.

References CPL_BORDER_CROP, CPL_BORDER_FILTER, cpl_ensure_code, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_DATA_NOT_FOUND, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, CPL_ERROR_UNSUPPORTED_MODE, CPL_FILTER_AVERAGE, CPL_FILTER_AVERAGE_FAST, CPL_FILTER_MEDIAN, CPL_FILTER_STDEV, CPL_FILTER_STDEV_FAST, cpl_image_copy(), cpl_image_get_data(), cpl_image_get_data_const(), cpl_image_get_size_x(), cpl_image_get_size_y(), cpl_image_get_type(), cpl_mask_get_data(), cpl_mask_get_data_const(), cpl_mask_get_size_x(), cpl_mask_get_size_y(), cpl_mask_is_empty(), cpl_mask_wrap(), CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT, cpl_type_get_sizeof(), and CPL_TYPE_INT.

Referenced by cpl_geom_img_offset_fine(), and cpl_vector_filter_median_create().

4.24.4.45 cpl_image_filter_median()

```
cpl_image* cpl_image_filter_median (
             const cpl_image * in,
             const cpl_matrix * ker )
```

Apply a spatial median filter to an image.

Parameters

in	Image to filter.
ker	the kernel

Returns

1 newly allocated image or NULL in error case

See also

cpl_image_filter_mask

Deprecated Replace this call with cpl_image_filter_mask() using CPL_FILTER_MEDIAN and CPL_BORDER_F↩ ILTER.

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

4.24.4.46 cpl_image_filter_morpho()

```
cpl_image* cpl_image_filter_morpho (
            const cpl_image * in,
            const cpl_matrix * ker )
```

Filter an image in spatial domain with a morpho kernel.

Parameters

in	Image to filter.
ker	Filter definition.

Returns

1 newly allocated image or NULL in error case.

See also

cpl_image_filter()

Deprecated Replace this call with cpl_image_filter() using CPL_FILTER_MORPHO and CPL_BORDER_FILTER.

References CPL_BORDER_FILTER, cpl_ensure, CPL_ERROR_NULL_INPUT, CPL_FILTER_MORPHO, cpl_↩ image_delete(), cpl_image_filter(), and cpl_image_new().

4.24.4.47 cpl_image_filter_stdev()

```
cpl_image* cpl_image_filter_stdev (
             const cpl_image * in,
             const cpl_matrix * ker )
```

Standard deviation filter.

Parameters

in	input image
ker	the kernel

Returns

a newly allocated filtered image or NULL on error

See also

cpl_image_filter_mask

Deprecated Replace this call with cpl_image_filter_mask() using CPL_FILTER_STDEV and CPL_BORDER_FI↩ LTER.

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

4.24.4.48 cpl_image_fit_gaussian()

```
cpl_error_code cpl_image_fit_gaussian (
             const cpl_image * im,
             cpl_size xpos,
             cpl_size ypos,
             cpl_size size,
             double * norm,
             double * xcen,
             double * ycen,
             double * sig_x,
             double * sig_y,
             double * fwhm_x,
             double * fwhm_y )
```

Apply a gaussian fit on an image sub window.

Parameters

im	the input image
xpos	the x position of the center (1 for the first pixel)
ypos	the y position of the center (1 for the first pixel)
size	the window size in pixels, at least 4
norm	the norm of the gaussian or NULL

Parameters

xcen	the x center of the gaussian or NULL
ycen	the y center of the gaussian or NULL
sig_x	the semi-major axis of the gaussian or NULL
sig_y	the semi-minor axis of the gaussian or NULL
fwhm_x	the FHHM in x or NULL
fwhm_y	the FHHM in y or NULL

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

See also

cpl_fit_image_gaussian()

cpl_image_iqe()

Deprecated If you need a 2D gaussian fit please use the function *cpl_fit_image_gaussian()*. Please note that on CPL versions earlier than 5.1.0 this function was wrongly documented: the parameters *sig_x* and *sig_y* were defined as "the sigma in x (or y) of the gaussian", while actually they returned the semi-major and semi-minor axes of the gaussian distribution at 1-sigma. ***PLEASE NOTE THAT IF YOU USED THIS FUNCTION FOR DETERMINING THE SPREAD OF A DISTRIBUTION ALONG THE X DIRECTION, THIS WAS VERY LIKELY OVERESTIMATED*** (because *sig_x* was always assigned the semi-major axis of the distribution ignoring the rotation), ***WHILE THE SPREAD ALONG THE Y DIRECTION WOULD BE UNDERESTIMATED***. In addition to that, even with circular distributions this function may lead to an underestimation of *sig_x* and *sig_y* (up to 25% underestimation in the case of noiseless data with a box 4 times the sigma, 1% underestimation in the case of noiseless data with a box 7 times the sigma). This latter problem is related to the function *cpl_image_iqe()*.

This function is only acceptable for determining the position of a peak.

References cpl_bivector_delete(), cpl_bivector_get_x_data(), cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_image_cast(), cpl_image_delete(), cpl_image_get_type(), cpl_image_iqe(), CPL_MATH_2PI, CPL_MATH_FWHM_SIG, and CPL_TYPE_FLOAT.

4.24.4.49 cpl_image_flip()

```
cpl_error_code cpl_image_flip (
            cpl_image * im,
            int angle )
```

Flip an image on a given mirror line.

Parameters

im	the image to flip.
angle	mirror line in polar coord. is theta = (PI/4) ∗ angle

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

This function operates locally on the pixel buffer.

angle can take one of the following values:

- 0 (theta=0) to flip the image around the horizontal
- 1 (theta=pi/4) to flip the image around y=x
- 2 (theta=pi/2) to flip the image around the vertical
- 3 (theta=3pi/4) to flip the image around y=-x

Images can be of type CPL_TYPE_INT, CPL_TYPE_FLOAT or CPL_TYPE_DOUBLE.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the angle is different from the allowed values
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References cpl_ensure_code, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INP↩UT, and cpl_mask_flip().

4.24.4.50 cpl_image_get()

```
double cpl_image_get (
            const cpl_image * image,
            cpl_size xpos,
            cpl_size ypos,
            int * pis_rejected )
```

Get the value of a pixel at a given position.

Parameters

image	Input image.
xpos	Pixel x position (FITS convention, 1 for leftmost)
ypos	Pixel y position (FITS convention, 1 for lowest)
pis_rejected	1 if the pixel is bad, 0 if good, negative on error

Returns

The pixel value (cast to a double) or undefined if ∗pis_rejected

See also

cpl_image_get_complex()

The return value is defined if the pixel is not flagged as rejected, i. e. when ∗pis_rejected == 0.

In case of an error, the _cpl_error_code_ code is set.

Images can be CPL_TYPE_FLOAT, CPL_TYPE_INT or CPL_TYPE_DOUBLE.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ACCESS_OUT_OF_RANGE if the passed position is not in the image
- CPL_ERROR_INVALID_TYPE if the image pixel type is not supported

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_INVALID_TYPE, CPL_ERR↩OR_NULL_INPUT, and CPL_TYPE_COMPLEX.

Referenced by cpl_fit_image_gaussian(), cpl_image_dump_window(), and cpl_image_get_fwhm().

4.24.4.51 cpl_image_get_absflux()

```
double cpl_image_get_absflux (
            const cpl_image * image )
```

Computes the sum of absolute values over an image.

Parameters

image	input image.

Returns

the absolute flux (sum of |pixels|) value

See also

cpl_image_get_min()

4.24.4.52 cpl_image_get_absflux_window()

```
double cpl_image_get_absflux_window (
             const cpl_image * image,
             cpl_size llx,
             cpl_size lly,
             cpl_size urx,
             cpl_size ury )
```

Computes the sum of absolute values over an image sub-window.

Parameters

image	input image.
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)

Returns

the absolute flux (sum of |pixels|) value

See also

cpl_image_get_min_window()

4.24.4.53 cpl_image_get_bpm()

```
cpl_mask* cpl_image_get_bpm (
             cpl_image * img )
```

Gets the bad pixels map.

Parameters

img	Image to query.

Returns

A pointer to the mask identifying the bad pixels or NULL.

The returned pointer refers to already allocated data. If the bad pixel map is NULL, an empty one is created.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, CPL_ERROR_NULL_INPUT, and cpl_mask_new().

Referenced by cpl_image_reject_from_mask().

4.24.4.54 cpl_image_get_bpm_const()

```
const cpl_mask* cpl_image_get_bpm_const (
            const cpl_image * img )
```

Gets the bad pixels map.

Parameters

img	Image to query.

Returns

A pointer to the mask identifying the bad pixels or NULL.

Note

NULL is returned if the image has no bad pixel map

See also

cpl_image_get_bpm

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_detector_interpolate_rejected(), cpl_fit_image_gaussian(), cpl_fit_imagelist_polynomial_↩window(), cpl_image_new_from_accepted(), cpl_imagelist_cast(), cpl_imagelist_collapse_create(), cpl_imagelist↩_empty(), cpl_imagelist_erase(), cpl_imagelist_set(), and cpl_mask_threshold_image().

4.24.4.55 cpl_image_get_centroid_x()

```
double cpl_image_get_centroid_x (
            const cpl_image * image )
```

Computes the x centroid value over the whole image.

Parameters

image	input image.

Returns

the x centroid value

See also

cpl_image_get_min_window()

4.24.4.56 cpl_image_get_centroid_x_window()

```
double cpl_image_get_centroid_x_window (
            const cpl_image * image,
            cpl_size llx,
            cpl_size lly,
            cpl_size urx,
            cpl_size ury )
```

Computes the x centroid value over an image sub-window.

Parameters

image	input image.
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)

Returns

the x centroid value

See also

cpl_image_get_min_window()

4.24.4.57 cpl_image_get_centroid_y()

```
double cpl_image_get_centroid_y (
            const cpl_image * image )
```

Computes the y centroid value over the whole image.

Parameters

image	input image.

Returns

the y centroid value

See also

cpl_image_get_min_window()

4.24.4.58 cpl_image_get_centroid_y_window()

```
double cpl_image_get_centroid_y_window (
            const cpl_image * image,
            cpl_size llx,
            cpl_size lly,
            cpl_size urx,
            cpl_size ury )
```

Computes the y centroid value over an image sub-window.

Parameters

image	input image.
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)

Returns

the y centroid value

See also

cpl_image_get_min_window()

4.24.4.59 cpl_image_get_complex()

```
double complex cpl_image_get_complex (
            const cpl_image * image,
            cpl_size xpos,
            cpl_size ypos,
            int * pis_rejected )
```

Get the value of a complex pixel at a given position.

Parameters

image	Input complex image.
xpos	Pixel x position (FITS convention, 1 for leftmost)
ypos	Pixel y position (FITS convention, 1 for lowest)
pis_rejected	1 if the pixel is bad, 0 if good, negative on error

Returns

The pixel value (cast to a double complex) or undefined if ∗pis_rejected

See also

cpl_image_get()

Note

This function is available iff the application includes complex.h

The return value is defined if the pixel is not flagged as rejected, i. e. when ∗pis_rejected == 0.

In case of an error, the _cpl_error_code_ code is set.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ACCESS_OUT_OF_RANGE if the passed position is not in the image

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_INVALID_TYPE, and CPL_↩ ERROR_NULL_INPUT.

Referenced by cpl_image_dump_window().

4.24.4.60 cpl_image_get_data()

```
void* cpl_image_get_data (
            cpl_image * img )
```

Gets the pixel data.

Parameters

img	Image to query.

Returns

A pointer to the image pixel data or NULL on error.

The returned pointer refers to already allocated data.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_detector_interpolate_rejected(), cpl_image_copy(), and cpl_image_filter_mask().

4.24.4.61 cpl_image_get_data_const()

```
const void* cpl_image_get_data_const (
            const cpl_image * img )
```

Gets the pixel data.

Parameters

img	Image to query.

Returns

A pointer to the image pixel data or NULL on error.

See also

cpl_image_get_data

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_image_filter(), cpl_image_filter_mask(), and cpl_mask_threshold_image().

4.24.4.62 cpl_image_get_data_double()

```
double* cpl_image_get_data_double (
            cpl_image * img )
```

Get the data as a double array.

Parameters

img	a cpl_image object

Returns

pointer to the double data array or NULL on error.

The returned pointer refers to already allocated data.

The pixels are stored in a one dimensional array. The pixel value PIXVAL at position (i, j) in the image - (0, 0) is the lower left pixel, i gives the column position from left to right, j gives the row position from bottom to top - is given by : PIXVAL = array[i + j∗nx]; where nx is the x size of the image and array is the data array returned by this function. array can be used to access or modify the pixel value in the image.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_TYPE_MISMATCH if the passed image type is not double

Referenced by cpl_image_fill_jacobian(), and cpl_image_fill_jacobian_polynomial().

4.24.4.63 cpl_image_get_data_double_complex()

```
double complex* cpl_image_get_data_double_complex (
            cpl_image * img )
```

Get the data as a double complex array.

Parameters

img	a cpl_image object

Returns

pointer to the double complex data array or NULL on error.

See also

cpl_image_get_data_double()

Note

This function is available iff the application includes complex.h

4.24.4.64 cpl_image_get_data_double_complex_const()

```
const double complex* cpl_image_get_data_double_complex_const (
            const cpl_image * img )
```

Get the data as a double complex array.

Parameters

img	a cpl_image object

Returns

pointer to the double complex data array or NULL on error.

See also

cpl_image_get_data_double_complex()

4.24.4.65 cpl_image_get_data_double_const()

```
const double* cpl_image_get_data_double_const (
            const cpl_image * img )
```

Get the data as a double array.

Parameters

img	a cpl_image object

Returns

pointer to the double data array or NULL on error.

See also

cpl_image_get_data_double

Referenced by cpl_fit_image_gaussian(), cpl_image_fill_jacobian(), and cpl_image_warp().

4.24.4.66 cpl_image_get_data_float()

```
float* cpl_image_get_data_float (
            cpl_image * img )
```

Get the data as a float array.

Parameters

img	a cpl_image object

Returns

pointer to the float data array or NULL on error.

See also

cpl_image_get_data_double()

Referenced by cpl_image_fill_jacobian(), cpl_image_fill_jacobian_polynomial(), and cpl_image_iqe().

4.24.4.67 cpl_image_get_data_float_complex()

```
float complex* cpl_image_get_data_float_complex (
             cpl_image * img )
```

Get the data as a float complex array.

Parameters

img	a cpl_image object

Returns

pointer to the float complex data array or NULL on error.

See also

cpl_image_get_data_double_complex()

4.24.4.68 cpl_image_get_data_float_complex_const()

```
const float complex* cpl_image_get_data_float_complex_const (
             const cpl_image * img )
```

Get the data as a float complex array.

Parameters

img	a cpl_image object

Returns

pointer to the float complex data array or NULL on error.

See also

cpl_image_get_data_double_complex()

4.24.4.69 cpl_image_get_data_float_const()

```
const float* cpl_image_get_data_float_const (
             const cpl_image * img )
```

Get the data as a float array.

Parameters

img	a cpl_image object

Returns

pointer to the float data array or NULL on error.

See also

cpl_image_get_data_float()

4.24.4.70 cpl_image_get_data_int()

```
int* cpl_image_get_data_int (
             cpl_image * img )
```

Get the data as a integer array.

Parameters

img	a cpl_image object

Returns

pointer to the integer data array or NULL on error.

See also

cpl_image_get_data_double()

Referenced by cpl_image_labelise_mask_create(), and cpl_imagelist_collapse_sigclip_create().

4.24.4.71 cpl_image_get_data_int_const()

```
const int* cpl_image_get_data_int_const (
            const cpl_image * img )
```

Get the data as a integer array.

Parameters

img	a cpl_image object

Returns

pointer to the integer data array or NULL on error.

See also

cpl_image_get_data_int()

4.24.4.72 cpl_image_get_flux()

```
double cpl_image_get_flux (
            const cpl_image * image )
```

Computes the sum of pixel values over an image.

Parameters

image	input image.

Returns

the flux value

See also

cpl_image_get_min()

References cpl_ensure, CPL_ERROR_NULL_INPUT, and cpl_image_get_flux_window().

4.24.4.73 cpl_image_get_flux_window()

```
double cpl_image_get_flux_window (
             const cpl_image * image,
             cpl_size llx,
             cpl_size lly,
             cpl_size urx,
             cpl_size ury )
```

Computes the sum of pixel values over an image sub-window.

Parameters

image	input image.
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)

Returns

the flux value

See also

cpl_image_get_min_window()

Referenced by cpl_image_get_flux().

4.24.4.74 cpl_image_get_fwhm()

```
cpl_error_code cpl_image_get_fwhm (
             const cpl_image * in,
             cpl_size xpos,
             cpl_size ypos,
             double * fwhm_x,
             double * fwhm_y )
```

Compute FWHM values in x and y for an object.

Parameters

in	the input image
xpos	the x position of the object (1 for the first pixel)
ypos	the y position of the object (1 for the first pixel)
fwhm_x	the computed FWHM in x or -1 on error
fwhm_y	the computed FWHM in y or -1 on error

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_

This function uses a basic method: start from the center of the object and go away until the half maximum value is reached in x and y.

For the FWHM in x (resp. y) to be computed, the image size in the x (resp. y) direction should be at least of 5 pixels.

If for any reason, one of the FHWMs cannot be computed, its returned value is -1.0, but an error is not necessarily raised. For example, if a 4 column image is passed, the fwhm_x would be -1.0, the fwhm_y would be correctly computed, and no error would be raised.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_DATA_NOT_FOUND if (xpos, ypos) specifies a rejected pixel or a pixel with a non-positive value
- CPL_ERROR_ACCESS_OUT_OF_RANGE if xpos or ypos is outside the image size range

References cpl_ensure_code, CPL_ERROR_DATA_NOT_FOUND, cpl_error_get_code(), CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), cpl_errorstate_is_equal(), cpl_image_get(), cpl_vector_delete(), cpl_vector_new_from_image_column(), and cpl_vector_new_from_image_row().

Referenced by cpl_apertures_get_fwhm().

4.24.4.75 cpl_image_get_interpolated()

```
double cpl_image_get_interpolated (
             const cpl_image * source,
             double xpos,
             double ypos,
             const cpl_vector * xprofile,
             double xradius,
             const cpl_vector * yprofile,
             double yradius,
             double * pconfid )
```

Interpolate a pixel.

Parameters

source	Interpolation source
xpos	Pixel x floating-point position (FITS convention)
ypos	Pixel y floating-point position (FITS convention)
xprofile	Interpolation weight as a function of the distance in X
xradius	Positive inclusion radius in the X-dimension
yprofile	Interpolation weight as a function of the distance in Y
yradius	Positive inclusion radius in the Y-dimension
pconfid	Confidence level of the interpolated value (range 0 to 1)

Returns

The interpolated pixel value, or undefined on error

See also

cpl_image_get()

If the X- and Y-radii are identical the area of inclusion is a circle, otherwise it is an ellipse, with the larger of the two radii as the semimajor axis and the other as the semiminor axis.

The radii are only required to be positive. However, for small radii, especially radii less than 1/sqrt(2), (xpos, ypos) may be located such that no source pixels are included in the interpolation, causing the interpolated pixel value to be undefined.

The X- and Y-profiles can be generated with cpl_vector_fill_kernel_profile(profile, radius). For profiles generated with cpl_vector_fill_kernel_profile() it is important to use the same radius both there and in cpl_image_get_interpolated().

A good profile length is CPL_KERNEL_DEF_SAMPLES, using radius CPL_KERNEL_DEF_WIDTH.

On error ∗pconfid is negative (unless pconfid is NULL). Otherwise, if ∗pconfid is zero, the interpolated pixel-value is undefined. Otherwise, if ∗pconfid is less than 1, the area of inclusion is close to the image border or contains rejected pixels.

The input image type can be CPL_TYPE_INT, CPL_TYPE_FLOAT and CPL_TYPE_DOUBLE.

Here is an example of a simple image unwarping (with error-checking omitted for brevity):

const double xyradius = CPL_KERNEL_DEF_WIDTH;

cpl_vector ∗ xyprofile = cpl_vector_new(CPL_KERNEL_DEF_SAMPLES); cpl_image ∗ unwarped = cpl_image_↩ new(nx, ny, CPL_TYPE_DOUBLE);

cpl_vector_fill_kernel_profile(xyprofile, CPL_KERNEL_DEFAULT, xyradius);

for (iv = 1; iv <= ny; iv++) { for (iu = 1; iu <= nx; iu++) { double confidence; const double x = my_unwarped_x(); const double y = my_unwarped_y();

const double value = cpl_image_get_interpolated(warped, x, y, xyprofile, xyradius, xyprofile, xyradius, &confidence);

if (confidence > 0) cpl_image_set(unwarped, iu, iv, value); else cpl_image_reject(unwarped, iu, iv); } }

cpl_vector_delete(xyprofile);

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if (one of) the input pointer(s) is NULL
- CPL_ERROR_ILLEGAL_INPUT if xradius, xprofile, yprofile and yradius are not as requested
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References cpl_ensure, cpl_error_get_code(), CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NULL_INPUT, cpl_image_get_size_x(), cpl_image_get_size_y(), cpl_mask_get_data_const(), CP↩ L_TYPE_DOUBLE, CPL_TYPE_FLOAT, CPL_TYPE_INT, cpl_vector_get_data_const(), and cpl_vector_get_size().

4.24.4.76 cpl_image_get_mad()

```
double cpl_image_get_mad (
            const cpl_image * image,
            double * sigma )
```

Computes median and median absolute deviation (MAD) on an image.

Parameters

image	Input image.
sigma	The median of the absolute median deviation of the good pixels

Returns

The median of the non-bad pixels

See also

cpl_image_get_mad_window()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported
- CPL_ERROR_DATA_NOT_FOUND if all the pixels are bad in the image

4.24.4.77 cpl_image_get_mad_window()

```
double cpl_image_get_mad_window (
             const cpl_image * image,
             cpl_size llx,
             cpl_size lly,
             cpl_size urx,
             cpl_size ury,
             double * sigma )
```

Computes median and median absolute deviation (MAD) on an image window.

Parameters

image	Input image.
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)
sigma	The median of the absolute median deviation of the good pixels

Returns

The median of the non-bad pixels

See also

cpl_image_get_median_window(), CPL_MATH_STD_MAD

For each non-bad pixel in the window the absolute deviation from the median is computed. The median of these absolute deviations in returned via sigma, while the median itself is returned by the function.

If the pixels are gaussian, the computed sigma is a robust and consistent estimate of the standard deviation in the sense that the standard deviation is approximately k $*$ MAD, where k is a constant equal to approximately 1.4826. CPL defines CPL_MATH_STD_MAD as this scaling constant.

Images can be CPL_TYPE_FLOAT, CPL_TYPE_INT or CPL_TYPE_DOUBLE. On error the _cpl_error_code_ code is set and the return value is undefined.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported
- CPL_ERROR_ILLEGAL_INPUT if the specified window is illegal
- CPL_ERROR_DATA_NOT_FOUND if all the pixels are bad in the image window

4.24.4.78 cpl_image_get_max()

```
double cpl_image_get_max (
            const cpl_image * image )
```

computes maximum pixel value over an image.

Parameters

image	input image.

Returns

the maximum value

See also

cpl_image_get_min()

4.24.4.79 cpl_image_get_max_window()

```
double cpl_image_get_max_window (
            const cpl_image * image,
```

```
            cpl_size llx,
            cpl_size lly,
            cpl_size urx,
            cpl_size ury )
```

computes maximum pixel value over an image sub-window.

Parameters

image	input image.
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)

Returns

the maximum value

See also

cpl_image_get_min_window()

4.24.4.80 cpl_image_get_maxpos()

```
cpl_error_code cpl_image_get_maxpos (
            const cpl_image * image,
            cpl_size * px,
            cpl_size * py )
```

Computes maximum pixel value and position over an image.

Parameters

image	Input image.
px	ptr to the x coordinate of the maximum position
py	ptr to the y coordinate of the maximum position

Returns

CPL_ERROR_NONE or the _cpl_error_code_ on error

See also

cpl_image_get_minpos()

References cpl_ensure_code, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_fit_image_gaussian().

4.24.4.81 cpl_image_get_maxpos_window()

```
cpl_error_code cpl_image_get_maxpos_window (
            const cpl_image * image,
            cpl_size llx,
            cpl_size lly,
            cpl_size urx,
            cpl_size ury,
            cpl_size * px,
            cpl_size * py )
```

Computes maximum pixel value and position over an image sub window.

Parameters

image	Input image.
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)
px	ptr to the x coordinate of the maximum position
py	ptr to the y coordinate of the maximum position

Returns

CPL_ERROR_NONE or the _cpl_error_code_ on error

See also

cpl_image_get_minpos_window()

References cpl_ensure_code, and CPL_ERROR_NULL_INPUT.

4.24.4.82 cpl_image_get_mean()

```
double cpl_image_get_mean (
            const cpl_image * image )
```

computes mean pixel value over an image.

Parameters

image	input image.

Returns

the mean value

See also

cpl_image_get_min()

Referenced by cpl_fit_image_gaussian().

4.24.4.83 cpl_image_get_mean_window()

```
double cpl_image_get_mean_window (
            const cpl_image * image,
            cpl_size llx,
            cpl_size lly,
            cpl_size urx,
            cpl_size ury )
```

computes mean pixel value over an image sub-window.

Parameters

image	input image.
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)

Returns

the mean value

See also

cpl_image_get_min_window()

4.24.4.84 cpl_image_get_median()

```
double cpl_image_get_median (
            const cpl_image * image )
```

computes median pixel value over an image.

Parameters

image	Input image.

Returns

the median value

See also

cpl_image_get_median_window()

In case of error, the _cpl_error_code_ code is set, and the returned double is undefined. Images can be CPL_TYPE_FLOAT, CPL_TYPE_INT or CPL_TYPE_DOUBLE.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

Referenced by cpl_fit_image_gaussian().

4.24.4.85 cpl_image_get_median_dev()

```
double cpl_image_get_median_dev (
            const cpl_image * image,
            double * sigma )
```

Computes median and mean absolute median deviation on an image window.

Parameters

image	Input image.
sigma	The mean of the absolute median deviation of the (good) pixels

Returns

The median of the non-bad pixels

See also

cpl_image_get_median_dev_window()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported
- CPL_ERROR_DATA_NOT_FOUND if all the pixels are bad in the image

4.24.4.86 cpl_image_get_median_dev_window()

```
double cpl_image_get_median_dev_window (
            const cpl_image * image,
            cpl_size llx,
            cpl_size lly,
            cpl_size urx,
            cpl_size ury,
            double * sigma )
```

Computes median and mean absolute median deviation on an image window.

Parameters

image	Input image.
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)
sigma	The mean of the absolute median deviation of the (good) pixels

Returns

The median of the non-bad pixels

See also

cpl_image_get_mad_window()

For each non-bad pixel in the window the absolute deviation from the median is computed. The mean of these absolute deviations in returned via sigma, while the median itself is returned by the function. The computed median and sigma may be a robust estimate of the mean and standard deviation of the pixels. The sigma is however still sensitive to outliers. See cpl_image_get_mad_window() for a more robust estimator.

Images can be CPL_TYPE_FLOAT, CPL_TYPE_INT or CPL_TYPE_DOUBLE. On error the _cpl_error_code_ code is set and the return value is undefined.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported
- CPL_ERROR_ILLEGAL_INPUT if the specified window is illegal
- CPL_ERROR_DATA_NOT_FOUND if all the pixels are bad in the image window

4.24.4.87 cpl_image_get_median_window()

```
double cpl_image_get_median_window (
            const cpl_image * image,
            cpl_size llx,
            cpl_size lly,
            cpl_size urx,
            cpl_size ury )
```

computes median pixel value over an image sub-window.

Parameters

image	Input image.
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)

Returns

The median value, or undefined on error

The specified bounds are included in the specified region.

In case of error, the _cpl_error_code_ code is set, and the returned value is undefined. Images can be CPL_TYPE_FLOAT, CPL_TYPE_INT or CPL_TYPE_DOUBLE.

For a finite population or sample, the median is the middle value of an odd number of values (arranged in ascending order) or any value between the two middle values of an even number of values. For an even number of elements in the array, the mean of the two central values is returned. Note that in this case, the median might not belong to the input array.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the window is outside the image or if there are only bad pixels in the window
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

4.24.4.88 cpl_image_get_min()

```
double cpl_image_get_min (
            const cpl_image * image )
```

computes minimum pixel value over an image.

Parameters

image	input image.

Returns

the minimum value

See also

cpl_image_get_min_window()

In case of error, the _cpl_error_code_ code is set, and the returned double is undefined. Images can be CPL_TYPE_FLOAT, CPL_TYPE_INT or CPL_TYPE_DOUBLE.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

4.24.4.89 cpl_image_get_min_window()

```
double cpl_image_get_min_window (
            const cpl_image * image,
            cpl_size llx,
            cpl_size lly,
            cpl_size urx,
            cpl_size ury )
```

computes minimum pixel value over an image sub-window.

Parameters

image	input image.
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)

Returns

the minimum value, or undefined on error

See also

cpl_stats_new_from_window()

Note

In case of error, the _cpl_error_code_ code is set.

The specified bounds are included in the specified region.

Images can be CPL_TYPE_FLOAT, CPL_TYPE_INT or CPL_TYPE_DOUBLE.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

4.24.4.90 cpl_image_get_minpos()

```
cpl_error_code cpl_image_get_minpos (
            const cpl_image * image,
            cpl_size * px,
            cpl_size * py )
```

Computes minimum pixel value and position over an image.

Parameters

image	Input image.
px	ptr to the x coordinate of the minimum position
py	ptr to the y coordinate of the minimum position

Returns

CPL_ERROR_NONE or the _cpl_error_code_ on error

See also

cpl_image_get_minpos_window()

Images can be CPL_TYPE_FLOAT, CPL_TYPE_INT or CPL_TYPE_DOUBLE.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure_code, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_fit_image_gaussian().

4.24.4.91 cpl_image_get_minpos_window()

```
cpl_error_code cpl_image_get_minpos_window (
            const cpl_image * image,
            cpl_size llx,
            cpl_size lly,
            cpl_size urx,
            cpl_size ury,
            cpl_size * px,
            cpl_size * py )
```

Computes minimum pixel value and position over an image sub window.

Parameters

image	Input image.
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)
px	ptr to the x coordinate of the minimum position

Returns

CPL_ERROR_NONE or the _cpl_error_code_ on error

See also

cpl_image_get_min_window()

The specified bounds are included in the specified region.

Images can be CPL_TYPE_FLOAT, CPL_TYPE_INT or CPL_TYPE_DOUBLE.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure_code, and CPL_ERROR_NULL_INPUT.

4.24.4.92 cpl_image_get_size_x()

```
cpl_size cpl_image_get_size_x (
            const cpl_image * img )
```

Get the image x size.

Parameters

img	a cpl_image object

Returns

The image x size, or -1 on NULL input

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_detector_interpolate_rejected(), cpl_fit_image_gaussian(), cpl_fit_imagelist_polynomial(), cpl_fit_imagelist_polynomial_window(), cpl_geom_img_offset_fine(), cpl_geom_img_offset_saa(), cpl_image_collapse_create(), cpl_image_copy(), cpl_image_fill_jacobian(), cpl_image_fill_jacobian_polynomial(), cpl_image_filter(), cpl_image_filter_mask(), cpl_image_get_interpolated(), cpl_image_iqe(), cpl_image_new_from_accepted(), cpl_image_rebin(), cpl_image_warp(), cpl_imagelist_collapse_minmax_create(), cpl_imagelist_collapse_sigclip_create(), cpl_imagelist_is_uniform(), cpl_imagelist_set(), cpl_imagelist_swap_axis_create(), cpl_mask_threshold_image(), cpl_mask_threshold_image_create(), cpl_plot_image(), cpl_plot_image_col(), cpl_plot_image_row(), and cpl_test_get_bytes_image().

4.24.4.93 cpl_image_get_size_y()

```
cpl_size cpl_image_get_size_y (
            const cpl_image * img )
```

Get the image y size.

Parameters

img	a cpl_image object

Returns

The image y size, or -1 on NULL input

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_detector_interpolate_rejected(), cpl_fit_image_gaussian(), cpl_fit_imagelist_polynomial(), cpl_fit_imagelist_polynomial_window(), cpl_geom_img_offset_fine(), cpl_geom_img_offset_saa(), cpl_image_collapse_create(), cpl_image_copy(), cpl_image_fill_jacobian(), cpl_image_fill_jacobian_polynomial(), cpl_image_filter(), cpl_image_filter_mask(), cpl_image_get_interpolated(), cpl_image_iqe(), cpl_image_new_from_accepted(), cpl_image_warp(), cpl_imagelist_collapse_minmax_create(), cpl_imagelist_collapse_sigclip_create(), cpl_imagelist_is_uniform(), cpl_imagelist_set(), cpl_imagelist_swap_axis_create(), cpl_mask_threshold_image(), cpl_mask_threshold_image_create(), cpl_plot_image(), cpl_plot_image_col(), cpl_plot_image_row(), and cpl_test_get_bytes_image().

4.24.4.94 cpl_image_get_sqflux()

```
double cpl_image_get_sqflux (
            const cpl_image * image )
```

Computes the sum of squared values over an image.

Parameters

image	input image.

Returns

the sqaure flux

See also

cpl_image_get_min()

4.24.4.95 cpl_image_get_sqflux_window()

```
double cpl_image_get_sqflux_window (
            const cpl_image * image,
            cpl_size llx,
            cpl_size lly,
            cpl_size urx,
            cpl_size ury )
```

Computes the sum of squared values over an image sub-window.

Parameters

image	input image.
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)

Returns

the square flux

See also

cpl_image_get_min_window()

4.24.4.96 cpl_image_get_stdev()

```
double cpl_image_get_stdev (
            const cpl_image * image )
```

computes pixel standard deviation over an image.

Parameters

image	input image.

Returns

the standard deviation value

See also

cpl_image_get_min()

4.24.4.97 cpl_image_get_stdev_window()

```
double cpl_image_get_stdev_window (
            const cpl_image * image,
            cpl_size llx,
            cpl_size lly,
            cpl_size urx,
            cpl_size ury )
```

computes pixel standard deviation over an image sub-window.

Parameters

image	input image.
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)

Returns

the standard deviation value

See also

cpl_image_get_min_window()

4.24.4.98 cpl_image_get_type()

```
cpl_type cpl_image_get_type (
            const cpl_image * img )
```

Get the image type.

Parameters

img	a cpl_image object

Returns

The image type or CPL_TYPE_INVALID on NULL input.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, CPL_ERROR_NULL_INPUT, and CPL_TYPE_INVALID.

Referenced by cpl_detector_interpolate_rejected(), cpl_fit_image_gaussian(), cpl_geom_img_offset_fine(), cpl_geom_img_offset_saa(), cpl_image_add_scalar(), cpl_image_and(), cpl_image_and_scalar(), cpl_image_collapse_window_create(), cpl_image_copy(), cpl_image_divide_scalar(), cpl_image_exponential(), cpl_image_fill_jacobian(), cpl_image_fill_jacobian_polynomial(), cpl_image_filter(), cpl_image_filter_mask(), cpl_image_fit_gaussian(), cpl_image_hypot(), cpl_image_logarithm(), cpl_image_multiply_scalar(), cpl_image_not(), cpl_image_or(), cpl_image_or_scalar(), cpl_image_power(), cpl_image_reject_value(), cpl_image_subtract_scalar(), cpl_image_warp(), cpl_image_xor(), cpl_image_xor_scalar(), cpl_imagelist_cast(), cpl_imagelist_collapse_minmax_create(), cpl_imagelist_collapse_sigclip_create(), cpl_imagelist_is_uniform(), cpl_imagelist_set(), cpl_imagelist_swap_axis_create(), cpl_mask_threshold_image(), and cpl_test_get_bytes_image().

4.24.4.99 cpl_image_hypot()

```
cpl_error_code cpl_image_hypot (
            cpl_image * self,
            const cpl_image * first,
            const cpl_image * second )
```

The pixel-wise Euclidean distance function of the images.

Parameters

self	Pre-allocated image to hold the result
first	First operand, or NULL for an in-place operation
second	Second operand

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

The Euclidean distance function is useful for gaussian error propagation on addition/subtraction operations.

For pixel values a and b the Euclidean distance c is defined as: $$c = sqrt{a^2 + b^2}$$

first may be NULL, in this case the distance is computed in-place on self using second as the other operand.

Images can be of type CPL_TYPE_FLOAT or CPL_TYPE_DOUBLE.

If both input operands are of type CPL_TYPE_FLOAT the distance is computed in single precision (using hypotf()), otherwise in double precision (using hypot()).

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if the images have different sizes
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_image_get_type(), CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT, and cpl_type_get_name().

4.24.4.100 cpl_image_iqe()

```
cpl_bivector* cpl_image_iqe (
             const cpl_image * in,
             cpl_size llx,
             cpl_size lly,
             cpl_size urx,
             cpl_size ury )
```

Compute an image quality estimation for an object.

Parameters

in	the input image
llx	
lly	the zone to analyse ((1, 1) for the first pixel)
urx	The zone must be at least 4 by 4 pixels
ury	

Returns

a newly allocated cpl_bivector containing the results or NULL in error case.

This function makes internal use of the iqe() MIDAS function (called here cpl_iqe()) written by P. Grosbol. Refer to the MIDAS documentation for more details. This function has proven to give good results over the years when called from RTD. The goal is to provide the exact same functionality in CPL as the one provided in RTD. The code is simply copied from the MIDAS package, it is not maintained by the CPL team.

The returned object must be deallocated with cpl_bivector_delete(). It contains in the first vector the computed values, and in the second one, the associated errors. The computed values are:

- x position of the object
- y position of the object
- FWHM along the major axis
- FWHM along the minor axis
- the angle of the major axis with the horizontal in degrees
- the peak value of the object
- the background computed

The bad pixels map of the image is not taken into account. The input image must be of type float.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if (one of) the input pointer(s) is NULL
- CPL_ERROR_ILLEGAL_INPUT if the specified zone is not valid or if the computation fails on this zone
- CPL_ERROR_INVALID_TYPE if the input image has the wrong type

Note

This function may lead to a strong underestimation of the sigmas and FWHMs (up to 25% underestimation in the case of noiseless data with a box 4 times the sigma, 1% underestimation in the case of noiseless data with a box 7 times the sigma).

References cpl_bivector_get_x_data(), cpl_bivector_get_y_data(), cpl_bivector_new(), cpl_ensure, CPL_ERR↩ OR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NULL_INPUT, cpl_image_delete(), cpl_↩ image_extract(), cpl_image_get_data_float(), cpl_image_get_size_x(), cpl_image_get_size_y(), and CPL_TYP↩ E_FLOAT.

Referenced by cpl_image_fit_gaussian().

4.24.4.101 cpl_image_is_rejected()

```
int cpl_image_is_rejected (
              const cpl_image * im,
              cpl_size x,
              cpl_size y )
```

Test if a pixel is good or bad.

Parameters

im	the input image
x	the x pixel position in the image (first pixel is 1)
y	the y pixel position in the image (first pixel is 1)

Returns

1 if the pixel is bad, 0 if the pixel is good, negative on error.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ACCESS_OUT_OF_RANGE if the specified position is out of the image

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_NULL_INPUT, and cpl_↩ mask_get().

4.24.4.102 cpl_image_labelise_mask_create()

```
cpl_image* cpl_image_labelise_mask_create (
              const cpl_mask * in,
              cpl_size * nbobjs )
```

Labelise a mask to differentiate different objects.

Parameters

in	mask to labelise
nbobjs	If non-NULL, number of objects found on success

Returns

A newly allocated label image or NULL on error

This function labelises all blobs in a mask. All 4-neighbour connected zones set to 1 in the input mask will end up in the returned integer image as zones where all pixels are set to the same (unique for this blob in this image) label. A non-recursive flood-fill is applied to label the zones. The flood-fill is dimensioned by the number of lines in the image, and the maximal number of lines possibly covered by a blob. The returned image must be deallocated with cpl_image_delete()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if the input mask is NULL

References cpl_ensure, CPL_ERROR_NULL_INPUT, cpl_free(), cpl_image_get_data_int(), cpl_image_new(), cpl_malloc(), cpl_mask_get_data_const(), cpl_mask_get_size_x(), cpl_mask_get_size_y(), and CPL_TYPE_INT.

Referenced by cpl_apertures_extract_mask().

4.24.4.103 cpl_image_load()

```
cpl_image* cpl_image_load (
             const char * filename,
             cpl_type im_type,
             cpl_size pnum,
             cpl_size xtnum )
```

Load an image from a FITS file.

Parameters

filename	Name of the file to load from.
im_type	Type of the created image
pnum	Plane number in the Data Unit (0 for first)
xtnum	Extension number in the file (0 for primary HDU)

Returns

1 newly allocated image or NULL on error

This function loads an image from a FITS file (NAXIS=2 or 3).

The returned image has to be deallocated with cpl_image_delete().

The passed type for the output image can be : CPL_TYPE_FLOAT, CPL_TYPE_DOUBLE or CPL_TYPE_INT.

This type is there to specify the type of the cpl_image that will be created by the function. It is totally independant from the way the data are stored in the FITS file. A FITS image containg float pixels can be loaded as a cpl_image of type double. In this case, the user would specify CPL_TYPE_DOUBLE as im_type.

Additionally, CPL_TYPE_UNSPECIFIED can be passed to inherit the FITS file type. The type of the image is defined based on the BITPIX information of the FITS file. After a successful call, the type of the created image can be accessed via cpl_image_get_type().

'xtnum' specifies from which extension the image should be loaded. This could be 0 for the main data section (files without extension), or any number between 1 and N, where N is the number of extensions present in the file.

The requested plane number runs from 0 to nplanes-1, where nplanes is the number of planes present in the requested data section.

The created image has an empty bad pixel map.

Examples:

```
// Load as a float image the only image in FITS file (a.fits) without ext.
// and NAXIS=2.
cpl_image * im = cpl_image_load("a.fits", CPL_TYPE_FLOAT, 0, 0);
// Load as a double image the first plane in a FITS cube (a.fits) without
// extension, NAXIS=3 and NAXIS3=128
cpl_image * im = cpl_image_load("a.fits", CPL_TYPE_DOUBLE, 0, 0);
// Load as an integer image the third plane in a FITS cube (a.fits) without
// extension, NAXIS=3 and NAXIS3=128
cpl_image * im = cpl_image_load("a.fits", CPL_TYPE_INT, 2, 0);
// Load as a double image the first plane from extension 5
cpl_image * im = cpl_image_load("a.fits", CPL_TYPE_DOUBLE, 0, 5);
// Load as a double image the third plane in extension 5
cpl_image * im = cpl_image_load("a.fits", CPL_TYPE_DOUBLE, 2, 5);
```

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_FILE_IO if the file cannot be opened or does not exist
- CPL_ERROR_BAD_FILE_FORMAT if the data cannot be loaded from the file
- CPL_ERROR_INVALID_TYPE if the passed pixel type is not supported
- CPL_ERROR_ILLEGAL_INPUT if the passed extension number is negative
- CPL_ERROR_DATA_NOT_FOUND if the specified extension has no image data

4.24.4.104 cpl_image_load_window()

```
cpl_image* cpl_image_load_window (
              const char * filename,
              cpl_type im_type,
              cpl_size pnum,
              cpl_size xtnum,
              cpl_size llx,
              cpl_size lly,
              cpl_size urx,
              cpl_size ury )
```

Load an image from a FITS file.

Parameters

filename	Name of the file to load from.
im_type	Type of the created image
pnum	Plane number in the Data Unit (0 for first)
xtnum	Extension number in the file.
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)

Returns

1 newly allocated image or NULL on error

See also

cpl_image_load()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_FILE_IO if the file does not exist
- CPL_ERROR_BAD_FILE_FORMAT if the data cannot be loaded from the file
- CPL_ERROR_INVALID_TYPE if the passed pixel type is not supported
- CPL_ERROR_ILLEGAL_INPUT if the passed position is invalid

4.24.4.105 cpl_image_logarithm()

```
cpl_error_code cpl_image_logarithm (
            cpl_image * self,
            double base )
```

Compute the elementwise logarithm of the image.

Parameters

self	Image to be modified in place.
base	Base of the logarithm.

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

Modifies the image by computing the base-scalar logarithm of each of its pixels.

Images can be of type CPL_TYPE_INT, CPL_TYPE_FLOAT or CPL_TYPE_DOUBLE.

Pixels for which the logarithm is not defined are rejected and set to zero.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported
- CPL_ERROR_ILLEGAL_INPUT if base is non-positive
- CPL_ERROR_DIVISION_BY_ZERO if the base equals 1

References cpl_ensure_code, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INP↩UT, cpl_image_get_type(), CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT, cpl_type_get_name(), and CPL_TYPE_INT.

Referenced by cpl_image_logarithm_create(), and cpl_imagelist_logarithm().

4.24.4.106 cpl_image_logarithm_create()

```
cpl_image* cpl_image_logarithm_create (
             const cpl_image * image,
             double base )
```

Create a new image by taking the elementwise logarithm of an image.

Parameters

image	Image to take logarithm of
base	Base of the logarithm.

Returns

1 newly allocated image or NULL in case of an error

See also

cpl_image_logarithm

cpl_image_add_scalar_create

References cpl_ensure, cpl_error_get_code(), cpl_image_delete(), cpl_image_duplicate(), and cpl_image_↩logarithm().

4.24.4.107 cpl_image_move()

```
cpl_error_code cpl_image_move (
             cpl_image * im,
             cpl_size nb_cut,
             const cpl_size * new_pos )
```

Permute tiles in an image.

Parameters

im	The image to modify
nb_cut	The number of cuts in x and y
new_pos	Array with the nb_cut^2 permuted positions

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

Note

The permutation array *must* contain nb_cut-squared elements

nb_cut^2 defines in how many tiles the images will be permuted. Each tile will then be moved to another place defined in new_pos. nb_cut equal 1 will leave the image unchanged, 2 is used to permute the four image quadrants, etc.. new_pos contains nb_cut^2 values between 1 and nb_cut^2, i.e. a permutation of the values from 1 to nb_cut^2. The zone positions are counted from the lower left part of the image. It is not allowed to move two tiles to the same position (the relation between th new tiles positions and the initial position is bijective !). The array with the permuted positions must contain nb_cut^2 values, the function is unable to verify this.

The image x and y sizes have to be multiples of nb_cut.

```
Example:

16   17   18            6    5    4
13   14   15            3    2    1

10   11   12   ----->  12   11   10
 7    8    9            9    8    7

 4    5    6           18   17   16
 1    2    3           15   14   13

 image 3x6             cpl_image_move(image, 3, new_pos);
                       with new_pos = {9,8,7,6,5,4,3,2,1};
```

The bad pixels are moved in the same way.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if nb_cut is not strictly positive or cannot divide one of the image sizes or if the new_pos array does not contain a permutation of the values from 1 through nb_cut squared.
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_free(), cpl_malloc(), and cpl_mask_move().

Referenced by cpl_image_fft().

4.24.4.108 cpl_image_multiply()

```
cpl_error_code cpl_image_multiply (
            cpl_image * im1,
            const cpl_image * im2 )
```

Multiply two images, store the result in the first image.

Parameters

im1	first operand.
im2	second operand.

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

See also

cpl_image_add()

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NONE, CPL_ERROR_N↩ ULL_INPUT, CPL_TYPE_DOUBLE_COMPLEX, and CPL_TYPE_FLOAT_COMPLEX.

4.24.4.109 cpl_image_multiply_create()

```
cpl_image* cpl_image_multiply_create (
             const cpl_image * image1,
             const cpl_image * image2 )
```

Multiply two images.

Parameters

image1	first operand
image2	second operand

Returns

1 newly allocated image or NULL on error

See also

cpl_image_add_create()

4.24.4.110 cpl_image_multiply_scalar()

```
cpl_error_code cpl_image_multiply_scalar (
             cpl_image * self,
             double scalar )
```

Elementwise multiplication of an image with a scalar.

Parameters

self	Image to be modified in place.
scalar	Number to multiply with

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

See also

cpl_image_add_scalar()

References cpl_ensure_code, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INP↩UT, and cpl_image_get_type().

Referenced by cpl_image_fill_test_create(), cpl_image_multiply_scalar_create(), and cpl_imagelist_multiply_↩scalar().

4.24.4.111 cpl_image_multiply_scalar_create()

```
cpl_image* cpl_image_multiply_scalar_create (
             const cpl_image * image,
             double factor )
```

Create a new image by multiplication of a scalar and an image.

Parameters

image	Image to be multiplied
factor	Number to multiply with

Returns

1 newly allocated image or NULL in case of an error

See also

cpl_image_multiply_scalar

cpl_image_add_scalar_create

References cpl_ensure, cpl_error_get_code(), cpl_image_delete(), cpl_image_duplicate(), and cpl_image_↩multiply_scalar().

4.24.4.112 cpl_image_new()

```
cpl_image* cpl_image_new (
              cpl_size nx,
              cpl_size ny,
              cpl_type type )
```

Allocate an image structure and pixel buffer for a image.

Parameters

nx	Size in x (the number of columns)
ny	Size in y (the number of rows)
type	The pixel type

Returns

1 newly allocated cpl_image or NULL in case of an error

Allocates space for the cpl_image structure and sets the dimensions and type of pixel data. The pixel buffer is allocated and initialised to zero. The pixel array will contain nx∗ny values being the image pixels from the lower left to the upper right line by line.

Supported pixel types are CPL_TYPE_INT, CPL_TYPE_FLOAT, CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT_C↩ OMPLEX and CPL_TYPE_DOUBLE_COMPLEX.

The returned cpl_image must be deallocated using cpl_image_delete().

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_ILLEGAL_INPUT if nx or ny is non-positive
- CPL_ERROR_INVALID_TYPE if the passed pixel type is not supported

Referenced by cpl_geom_img_offset_fine(), cpl_image_fill_jacobian(), cpl_image_fill_test_create(), cpl_image↩ _filter_linear(), cpl_image_filter_morpho(), cpl_image_labelise_mask_create(), and cpl_image_new_from_↩ accepted().

4.24.4.113 cpl_image_new_from_mask()

```
cpl_image* cpl_image_new_from_mask (
              const cpl_mask * mask )
```

Create an int image from a mask.

Parameters

mask	the original mask

Returns

1 newly allocated cpl_image or NULL on error

The created image is of type CPL_TYPE_INT.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, CPL_ERROR_NULL_INPUT, cpl_image_wrap_int(), cpl_malloc(), cpl_mask_get_data_↩const(), cpl_mask_get_size_x(), and cpl_mask_get_size_y().

4.24.4.114 cpl_image_normalise()

```
cpl_error_code cpl_image_normalise (
            cpl_image * image,
            cpl_norm mode )
```

Normalise pixels in an image.

Parameters

image	Image operand.
mode	Normalisation mode.

Returns

CPL_ERROR_NONE, or the relevant _cpl_error_code_ on error.

Normalises an image according to a given criterion.

Possible normalisations are:

- CPL_NORM_SCALE sets the pixel interval to [0,1].
- CPL_NORM_MEAN sets the mean value to 1.
- CPL_NORM_FLUX sets the flux to 1.
- CPL_NORM_ABSFLUX sets the absolute flux to 1.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure_code, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_image_normalise_create(), and cpl_imagelist_normalise().

4.24.4.115 cpl_image_normalise_create()

```
cpl_image* cpl_image_normalise_create (
            const cpl_image * image_in,
            cpl_norm mode )
```

Create a new normalised image from an existing image.

Parameters

image_in	Image operand.
mode	Normalisation mode.

Returns

1 newly allocated image or NULL on error

See also

cpl_image_normalise

Stores the result in a newly allocated image and returns it. The returned image must be deallocated using cpl_image_delete().

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, CPL_ERROR_NULL_INPUT, cpl_image_delete(), cpl_image_duplicate(), and cpl_image_normalise().

4.24.4.116 cpl_image_not()

```
cpl_error_code cpl_image_not (
            cpl_image * self,
            const cpl_image * first )
```

The bit-wise complement (not) of an image with integer pixels.

Parameters

self	Pre-allocated image to hold the result
first	First operand, or NULL for an in-place operation

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

Note

CPL_TYPE_INT is required

See also

cpl_mask_not() for the equivalent logical operation

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if the images have different sizes
- CPL_ERROR_INVALID_TYPE if the passed image type is as required

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_image_get_type(), cpl_type_get_name(), cpl_type_get_sizeof(), and CPL_TYPE_INT.

4.24.4.117 cpl_image_or()

```
cpl_error_code cpl_image_or (
            cpl_image * self,
            const cpl_image * first,
            const cpl_image * second )
```

The bit-wise or of two images with integer pixels.

Parameters

self	Pre-allocated image to hold the result
first	First operand, or NULL for an in-place operation
second	Second operand

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

Note

CPL_TYPE_INT is required

See also

cpl_mask_or() for the equivalent logical operation

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if the images have different sizes
- CPL_ERROR_INVALID_TYPE if the passed image type is as required

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_INVALID_TYPE, CPL_E↩RROR_NULL_INPUT, cpl_image_get_type(), cpl_type_get_name(), and CPL_TYPE_INT.

4.24.4.118 cpl_image_or_scalar()

```
cpl_error_code cpl_image_or_scalar (
            cpl_image * self,
            const cpl_image * first,
            cpl_bitmask second )
```

The bit-wise or of a scalar and an image with integer pixels.

Parameters

self	Pre-allocated image to hold the result
first	First operand, or NULL for an in-place operation
second	Second operand (scalar)

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

Note

CPL_TYPE_INT is required

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if the images have different sizes
- CPL_ERROR_INVALID_TYPE if the passed image type is as required

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_INVALID_TYPE, CPL_E↩RROR_NONE, CPL_ERROR_NULL_INPUT, cpl_image_get_type(), cpl_type_get_name(), cpl_type_get_sizeof(), and CPL_TYPE_INT.

4.24.4.119 cpl_image_power()

```
cpl_error_code cpl_image_power (
            cpl_image * self,
            double exponent )
```

Compute the elementwise power of the image.

Parameters

self	Image to be modified in place.
exponent	Scalar exponent.

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

Modifies the image by lifting each of its pixels to exponent.

Images can be of type CPL_TYPE_INT, CPL_TYPE_FLOAT or CPL_TYPE_DOUBLE.

Pixels for which the power to the given exponent is not defined are rejected and set to zero.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References cpl_ensure_code, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INP↩UT, cpl_image_get_type(), CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT, cpl_type_get_name(), and CPL_TYPE_INT.

Referenced by cpl_image_power_create(), and cpl_imagelist_power().

4.24.4.120 cpl_image_power_create()

```
cpl_image* cpl_image_power_create (
              const cpl_image * image,
              double exponent )
```

Create a new image by elementwise raising of an image to a power.

Parameters

image	Image to raise to a power
exponent	scalar exponent

Returns

1 newly allocated image or NULL in case of an error

See also

cpl_image_power

cpl_image_add_scalar_create

References cpl_ensure, cpl_error_get_code(), cpl_image_delete(), cpl_image_duplicate(), and cpl_image_power().

4.24.4.121 cpl_image_rebin()

```
cpl_image* cpl_image_rebin (
              const cpl_image * image,
              cpl_size xstart,
              cpl_size ystart,
              cpl_size xstep,
              cpl_size ystep )
```

Rebin an image.

Parameters

image	The image to rebin
xstart	start x position of binning (starting from 1...)
ystart	start y position of binning (starting from 1...)
xstep	Bin size in x.
ystep	Bin size in y.

Returns

The newly allocated rebinned image or NULL in case of error

If both bin sizes in x and y are = 2, an image with (about) a quarter of the pixels of the input image will be created. Each new pixel will be the sum of the values of all contributing input pixels. If a bin is incomplete (i.e., the input image size is not a multiple of the bin sizes), it is not computed.

xstep and ystep must not be greater than the sizes of the rebinned region.

The input image type can be CPL_TYPE_INT, CPL_TYPE_FLOAT and CPL_TYPE_DOUBLE. If the image has bad pixels, they will be propagated to the rebinned image "pessimistically", i.e., if at least one of the contributing input pixels is bad, then the corresponding output pixel will also be flagged "bad". If you need an image of "weights" for each rebinned pixel, just cast the input image bpm into a CPL_TYPE_INT image, and apply cpl_image_rebin() to it too.

The returned image must be deallocated using cpl_image_delete().

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if the input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if xstep, ystep, xstart, ystart are not positive
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NULL_↩INPUT, cpl_image_get_size_x(), cpl_mask_get_data(), cpl_mask_get_data_const(), and cpl_mask_new().

4.24.4.122 cpl_image_reject()

```
cpl_error_code cpl_image_reject (
              cpl_image * im,
              cpl_size x,
              cpl_size y )
```

Set a pixel as bad in an image.

Parameters

im	the input image
x	the x pixel position in the image (first pixel is 1)
y	the y pixel position in the image (first pixel is 1)

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ACCESS_OUT_OF_RANGE if the specified position is out of the image

References cpl_ensure_code, CPL_ERROR_ACCESS_OUT_OF_RANGE, and CPL_ERROR_NULL_INPUT.

4.24.4.123 cpl_image_reject_from_mask()

```
cpl_error_code cpl_image_reject_from_mask (
            cpl_image * im,
            const cpl_mask * map )
```

Set the bad pixels in an image as defined in a mask.

Parameters

im	the input image
map	the mask defining the bad pixels

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

If the input image has a bad pixel map prior to the call, it is overwritten.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if the input image or the input map is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if the image and the map have different sizes

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NONE, CPL_ERROR_N↩ULL_INPUT, cpl_image_get_bpm(), cpl_mask_get_data(), cpl_mask_get_data_const(), cpl_mask_get_size_x(), and cpl_mask_get_size_y().

Referenced by cpl_image_warp(), cpl_image_warp_polynomial(), and cpl_imagelist_swap_axis_create().

4.24.4.124 cpl_image_reject_value()

```
cpl_error_code cpl_image_reject_value (
            cpl_image * self,
            cpl_value mode )
```

Reject pixels with the specified special value(s)

Parameters

self	Input image to modify
mode	Bit field specifying which special value(s) to reject

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INVALID_TYPE if mode is 1, e.g. due to a logical or (||) of the allowed options or if the pixel type is complex
- CPL_ERROR_UNSUPPORTED_MODE if mode is otherwise different from the allowed options.

References CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_image_get↩ _type(), CPL_TYPE_DOUBLE, CPL_TYPE_DOUBLE_COMPLEX, CPL_TYPE_FLOAT, CPL_TYPE_FLOAT_C↩ OMPLEX, and CPL_TYPE_INT.

4.24.4.125 cpl_image_save()

```
cpl_error_code cpl_image_save (
            const cpl_image * self,
            const char * filename,
            cpl_type type,
            const cpl_propertylist * plist,
            unsigned mode )
```

Save an image to a FITS file.

Parameters

self	Image to write to disk or NULL
filename	Name of the file to write to
type	The type used to represent the data in the file
plist	Property list for the output header or NULL
mode	The desired output options

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_propertylist_save()

This function saves an image to a FITS file. If a property list is provided, it is written to the header where the image is written. The image may be NULL, in this case only the propertylist is saved.

Supported image types are CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT, CPL_TYPE_INT.

The type used in the file can be one of: CPL_TYPE_UCHAR (8 bit unsigned), CPL_TYPE_SHORT (16 bit signed), CPL_TYPE_USHORT (16 bit unsigned), CPL_TYPE_INT (32 bit signed), CPL_TYPE_FLOAT (32 bit floating point), or CPL_TYPE_DOUBLE (64 bit floating point). Additionally, the special value CPL_TYPE_UNSPECIFIED is allowed. This value means that the type used for saving is the pixel type of the input image. Using the image pixel type as saving type ensures that the saving incurs no loss of information.

Supported output modes are CPL_IO_CREATE (create a new file) and CPL_IO_EXTEND (append a new extension to an existing file).

Upon success the image will reside in a FITS data unit with NAXIS = 2. Is it possible to save a single image in a FITS data unit with NAXIS = 3, see cpl_imagelist_save().

When the data written to disk are of an integer type, the output mode CPL_IO_EXTEND can be combined (via bit-wise or) with an option for tile-compression. This compression of integer data is lossless. The options are: CPL_IO_COMPRESS_GZIP, CPL_IO_COMPRESS_RICE, CPL_IO_COMPRESS_HCOMPRESS, CPL_IO_CO↩MPRESS_PLIO. With compression the type must be CPL_TYPE_UNSPECIFIED or CPL_TYPE_INT.

Note that in append mode the file must be writable (and do not take for granted that a file is writable just because it was created by the same application, as this depends from the system *umask*).

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the type or the mode is not supported
- CPL_ERROR_INVALID_TYPE if the passed pixel type is not supported
- CPL_ERROR_FILE_NOT_CREATED if the output file cannot be created
- CPL_ERROR_FILE_IO if the data cannot be written to the file

References CPL_ERROR_NONE, and cpl_propertylist_save().

4.24.4.126 cpl_image_set()

```
cpl_error_code cpl_image_set (
            cpl_image * image,
            cpl_size xpos,
            cpl_size ypos,
            double value )
```

Set the pixel at the given position to the given value.

Parameters

image	input image.
xpos	Pixel x position (FITS convention, 1 for leftmost)
ypos	Pixel y position (FITS convention, 1 for lowest)
value	New pixel value

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

Images can be CPL_TYPE_FLOAT, CPL_TYPE_INT, CPL_TYPE_DOUBLE.

If the pixel is flagged as rejected, this flag is removed.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ACCESS_OUT_OF_RANGE if the passed position is not in the image
- CPL_ERROR_INVALID_TYPE if the image pixel type is not supported

References cpl_ensure_code, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_INVALID_TYPE, CP↩L_ERROR_NULL_INPUT, and CPL_TYPE_COMPLEX.

4.24.4.127 cpl_image_set_bpm()

```
cpl_mask* cpl_image_set_bpm (
              cpl_image * self,
              cpl_mask * bpm )
```

Replace the bad pixel map of the image.

Parameters

self	Image to modfify
bpm	Bad Pixel Map (BPM) to set, replacing the old one, or NULL

Returns

A pointer to the old mask of bad pixels, or NULL

Note

NULL is returned if the image had no bad pixel map, while a non-NULL returned mask must be deallocated by the caller using cpl_mask_delete().

See also

cpl_image_get_bpm_const()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if self is NULL

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_image_unset_bpm().

4.24.4.128 cpl_image_set_complex()

```
cpl_error_code cpl_image_set_complex (
            cpl_image * image,
            cpl_size xpos,
            cpl_size ypos,
            double complex value )
```

Set the pixel at the given position to the given complex value.

Parameters

image	input image.
xpos	Pixel x position (FITS convention, 1 for leftmost)
ypos	Pixel y position (FITS convention, 1 for lowest)
value	New pixel value

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_image_set()

Note

This function is available iff the application includes complex.h

Images can be CPL_TYPE_FLOAT_COMPLEX or CPL_TYPE_DOUBLE_COMPLEX.

If the pixel is flagged as rejected, this flag is removed.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ACCESS_OUT_OF_RANGE if the passed position is not in the image
- CPL_ERROR_INVALID_TYPE if the image pixel type is not supported

References cpl_ensure_code, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_NULL_INPUT, and C↩PL_TYPE_FLOAT_COMPLEX.

4.24.4.129 cpl_image_shift()

```
cpl_error_code cpl_image_shift (
            cpl_image * self,
            cpl_size dx,
            cpl_size dy )
```

Shift an image by integer offsets.

Parameters

self	The image to shift in place
dx	The shift in X
dy	The shift in Y

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

The new zones (in the result image) where no new value is computed are set to 0 and flagged as bad pixels. The shift values have to be valid: -nx $<$ dx $<$ nx and -ny $<$ dy $<$ ny

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the requested shift is bigger than the image size

References cpl_ensure_code, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_mask_new(), cpl_mask_↩ shift(), and cpl_type_get_sizeof().

4.24.4.130 cpl_image_subtract()

```
cpl_error_code cpl_image_subtract (
            cpl_image * im1,
            const cpl_image * im2 )
```

Subtract two images, store the result in the first image.

Parameters

im1	first operand.
im2	second operand.

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

See also

cpl_image_add()

4.24.4.131 cpl_image_subtract_create()

```
cpl_image* cpl_image_subtract_create (
             const cpl_image * image1,
             const cpl_image * image2 )
```

Subtract two images.

Parameters

image1	first operand
image2	second operand

Returns

1 newly allocated image or NULL on error

See also

cpl_image_add_create()

4.24.4.132 cpl_image_subtract_scalar()

```
cpl_error_code cpl_image_subtract_scalar (
             cpl_image * self,
             double scalar )
```

Elementwise subtraction of a scalar from an image.

Parameters

self	Image to be modified in place.
scalar	Number to subtract

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

See also

cpl_image_add_scalar()

References cpl_ensure_code, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INP↩UT, and cpl_image_get_type().

Referenced by cpl_image_subtract_scalar_create(), and cpl_imagelist_subtract_scalar().

4.24.4.133 cpl_image_subtract_scalar_create()

```
cpl_image* cpl_image_subtract_scalar_create (
             const cpl_image * image,
             double subtrahend )
```

Create an image by elementwise subtraction of a scalar from an image.

Parameters

image	Image to be subtracted from
subtrahend	Number to subtract

Returns

1 newly allocated image or NULL in case of an error

See also

cpl_image_subtract_scalar

cpl_image_add_scalar_create

References cpl_ensure, cpl_error_get_code(), cpl_image_delete(), cpl_image_duplicate(), and cpl_image_↩subtract_scalar().

4.24.4.134 cpl_image_threshold()

```
cpl_error_code cpl_image_threshold (
             cpl_image * image_in,
             double lo_cut,
             double hi_cut,
             double assign_lo_cut,
             double assign_hi_cut )
```

Threshold an image to a given interval.

Parameters

image_in	Image to threshold.
lo_cut	Lower bound.
hi_cut	Higher bound.
assign_lo_cut	Value to assign to pixels below low bound.
assign_hi_cut	Value to assign to pixels above high bound.

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

Pixels outside of the provided interval are assigned the given values.

Use FLT_MIN and FLT_MAX for floating point images and DBL_MIN and DBL_MAX for double images for the lo_cut and hi_cut to avoid any pixel replacement.

Images can be of type CPL_TYPE_INT, CPL_TYPE_FLOAT or CPL_TYPE_DOUBLE. lo_cut must be smaller than or equal to hi_cut.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported
- CPL_ERROR_ILLEGAL_INPUT if lo_cut is greater than hi_cut

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_↩ NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_imagelist_threshold().

4.24.4.135 cpl_image_turn()

```
cpl_error_code cpl_image_turn (
            cpl_image * self,
            int rot )
```

Rotate an image by a multiple of 90 degrees clockwise.

Parameters

self	The image to rotate in place.
rot	The multiple: -1 is a rotation of 90 deg counterclockwise.

Returns

CPL_ERROR_NONE on success, otherwise the relevant _cpl_error_code_

Note

The dimension of a rectangular image is changed.

Images can be of type CPL_TYPE_INT, CPL_TYPE_FLOAT or CPL_TYPE_DOUBLE.

The definition of the rotation relies on the FITS convention: The lower left corner of the image is at (1,1), x increasing from left to right, y increasing from bottom to top.

For rotations of +90 or -90 degrees on rectangular non-1D-images, the pixel buffer is temporarily duplicated.

rot may be any integer value, its modulo 4 determines the rotation:

- -3 to turn 270 degrees counterclockwise.
- -2 to turn 180 degrees counterclockwise.
- -1 to turn 90 degrees counterclockwise.
- 0 to not turn
- +1 to turn 90 degrees clockwise (same as -3)
- +2 to turn 180 degrees clockwise (same as -2).
- +3 to turn 270 degrees clockwise (same as -1).

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References cpl_ensure_code, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INP↩UT, and cpl_mask_turn().

4.24.4.136 cpl_image_unset_bpm()

```
cpl_mask* cpl_image_unset_bpm (
              cpl_image * self )
```

Remove the bad pixel map from the image.

Parameters

self	image to process

Returns

A pointer to the cpl_mask of bad pixels, or NULL

Note

NULL is returned if the image has no bad pixel map

The returned mask must be deallocated using cpl_mask_delete().

See also

cpl_image_set_bpm()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, CPL_ERROR_NULL_INPUT, and cpl_image_set_bpm().

Referenced by cpl_image_accept_all(), cpl_imagelist_cast(), cpl_imagelist_empty(), cpl_imagelist_erase(), and cpl_imagelist_set().

4.24.4.137 cpl_image_unwrap()

```
void* cpl_image_unwrap (
            cpl_image * d )
```

Free memory associated to an cpl_image object, but the pixel buffer.

Parameters

d	Image to destroy.

Returns

A pointer to the data array or NULL if the input is NULL.

Frees all memory associated to an cpl_image, except the pixel buffer. cpl_image_unwrap() is provided for images that are constructed by passing a pixel buffer to one of cpl_image_wrap_{double,float,int}().

Note

The pixel buffer must subsequently be deallocated. Failure to do so will result in a memory leak.

References cpl_free(), and cpl_mask_delete().

Referenced by cpl_vector_cycle(), and cpl_vector_filter_median_create().

4.24.4.138 cpl_image_warp()

```
cpl_error_code cpl_image_warp (
             cpl_image * out,
             const cpl_image * in,
             const cpl_image * deltax,
             const cpl_image * deltay,
             const cpl_vector * xprofile,
             double xradius,
             const cpl_vector * yprofile,
             double yradius )
```

Warp an image.

Parameters

out	Pre-allocated destination image to hold the result
in	Source image to warp
deltax	The x shift of each pixel, same image size as out
deltay	The y shift of each pixel, same image size as out
xprofile	Interpolation weight as a function of the distance in X
xradius	Positive inclusion radius in the X-dimension
yprofile	Interpolation weight as a function of the distance in Y
yradius	Positive inclusion radius in the Y-dimension

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_image_get_interpolated()

The pixel value at the (integer) position (u, v) in the destination image is interpolated from the (typically non-integer) pixel position (x, y) in the source image, where

```
x = u - deltax(u, v),
y = v - deltay(u, v).
```

The identity transform is thus given by deltax and deltay filled with zeros.

The first pixel is (1, 1), located in the lower left. 'out' and 'in' may have different sizes, while deltax and deltay must have the same size as 'out'. deltax and deltay must have pixel type CPL_TYPE_DOUBLE.

Beware that extreme transformations may lead to blank images.

'out' and 'in' may be of type CPL_TYPE_INT, CPL_TYPE_FLOAT or CPL_TYPE_DOUBLE.

Examples of profiles and radius are:

```
xprofile = cpl_vector_new(CPL_KERNEL_DEF_SAMPLES);
cpl_vector_fill_kernel_profile(profile, CPL_KERNEL_DEFAULT,
      CPL_KERNEL_DEF_WIDTH);
xradius = CPL_KERNEL_DEF_WIDTH;
```

In case a correction for flux conservation were required, please create a correction map using the function cpl_image_fill_jacobian().

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if (one of) the input pointer(s) is NULL
- CPL_ERROR_ILLEGAL_INPUT if the input images sizes are incompatible or if the delta images are not of type CPL_TYPE_DOUBLE
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_image_get_data_double_const(), cpl_image_get_size_x(), cpl_image_get_size_y(), cpl_image_get_type(), cpl_image_reject_from_mask(), cpl_mask_delete(), cpl_mask_get_data(), cpl_mask_get_data_const(), cpl_mask_new(), CPL_TYPE_DOUBLE, cpl_vector_get_data_const(), and cpl_vector_get_size().

4.24.4.139 cpl_image_warp_polynomial()

```
cpl_error_code cpl_image_warp_polynomial (
            cpl_image * out,
            const cpl_image * in,
            const cpl_polynomial * poly_x,
            const cpl_polynomial * poly_y,
            const cpl_vector * xprofile,
            double xradius,
            const cpl_vector * yprofile,
            double yradius )
```

Warp an image according to a 2D polynomial transformation.

Parameters

out	Pre-allocated image to hold the result
in	Image to warp.
poly_x	Defines source x-pos corresponding to destination (u,v).
poly_y	Defines source y-pos corresponding to destination (u,v).
xprofile	Interpolation weight as a function of the distance in X
xradius	Positive inclusion radius in the X-dimension
yprofile	Interpolation weight as a function of the distance in Y
yradius	Positive inclusion radius in the Y-dimension

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_image_get_interpolated()

'out' and 'in' may have different dimensions and types.

The pair of 2D polynomials are used internally like this

```
x = cpl_polynomial_eval(poly_x, (u, v));
y = cpl_polynomial_eval(poly_y, (u, v));
```

where (u,v) are (integer) pixel positions in the destination image and (x,y) are the corresponding pixel positions (typically non-integer) in the source image.

The identity transform (poly_x(u,v) = u, poly_y(u,v) = v) would thus overwrite the 'out' image with the 'in' image, starting from the lower left if the two images are of different sizes.

Beware that extreme transformations may lead to blank images.

The input image type may be CPL_TYPE_INT, CPL_TYPE_FLOAT or CPL_TYPE_DOUBLE.

In case a correction for flux conservation were required, please create a correction map using the function cpl_image_fill_jacobian_polynomial().

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if (one of) the input pointer(s) is NULL
- CPL_ERROR_ILLEGAL_INPUT if the polynomial dimensions are not 2
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References cpl_ensure, cpl_ensure_code, cpl_error_get_code(), CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_image_reject_from_mask(), cpl_mask_delete(), cpl_mask_get_data(), cpl_mask_get_data_const(), cpl_mask_new(), cpl_polynomial_get_dimension(), cpl_vector_delete(), cpl_vector_get_data(), cpl_vector_get_data_const(), cpl_vector_get_size(), and cpl_vector_new().

4.24.4.140 cpl_image_wrap_double()

```
cpl_image* cpl_image_wrap_double (
            cpl_size nx,
            cpl_size ny,
            double * pixels )
```

Create a double image using an existing pixel buffer.

Parameters

nx	Size in x (the number of columns)
ny	Size in y (the number of rows)
pixels	double * pixel data

Returns

1 newly allocated cpl_image or NULL in case of an error

See also

cpl_image_new

The pixel array is set to point to that of the argument. The pixel array must contain nx∗ny doubles.

The allocated image must be deallocated with cpl_image_unwrap().

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if nx or ny is non-positive or zero.

References cpl_ensure, CPL_ERROR_NULL_INPUT, and CPL_TYPE_DOUBLE.

Referenced by cpl_image_divide_create(), cpl_vector_cycle(), and cpl_vector_filter_median_create().

4.24.4.141 cpl_image_wrap_double_complex()

```
cpl_image* cpl_image_wrap_double_complex (
             cpl_size nx,
             cpl_size ny,
             double complex * pixels )
```

Create a double complex image using an existing pixel buffer.

Parameters

nx	Size in x (the number of columns)
ny	Size in y (the number of rows)
pixels	double complex ∗ pixel data.

Returns

1 newly allocated cpl_image or NULL on error

See also

cpl_image_wrap_double()

Note

This function is available iff the application includes complex.h

References cpl_ensure, CPL_ERROR_NULL_INPUT, and CPL_TYPE_DOUBLE_COMPLEX.

Referenced by cpl_image_divide_create().

4.24.4.142 cpl_image_wrap_float()

```
cpl_image* cpl_image_wrap_float (
              cpl_size nx,
              cpl_size ny,
              float * pixels )
```

Create a float image using an existing pixel buffer.

Parameters

nx	Size in x (the number of columns)
ny	Size in y (the number of rows)
pixels	float ∗ pixel data.

Returns

1 newly allocated cpl_image or NULL on error

See also

cpl_image_wrap_double()

References cpl_ensure, CPL_ERROR_NULL_INPUT, and CPL_TYPE_FLOAT.

Referenced by cpl_image_divide_create().

4.24.4.143 cpl_image_wrap_float_complex()

```
cpl_image* cpl_image_wrap_float_complex (
              cpl_size nx,
              cpl_size ny,
              float complex * pixels )
```

Create a float complex image using an existing pixel buffer.

Parameters

nx	Size in x (the number of columns)
ny	Size in y (the number of rows)
pixels	float complex ∗ pixel data.

Returns

1 newly allocated cpl_image or NULL on error

See also

cpl_image_wrap_double_complex()

References cpl_ensure, CPL_ERROR_NULL_INPUT, and CPL_TYPE_FLOAT_COMPLEX.

Referenced by cpl_image_divide_create().

4.24.4.144 cpl_image_wrap_int()

```
cpl_image* cpl_image_wrap_int (
             cpl_size nx,
             cpl_size ny,
             int * pixels )
```

Create an integer image using an existing pixel buffer.

Parameters

nx	Size in x (the number of columns)
ny	Size in y (the number of rows)
pixels	int * pixel data.

Returns

1 newly allocated cpl_image or NULL on error

See also

cpl_image_wrap_double()

References cpl_ensure, CPL_ERROR_NULL_INPUT, and CPL_TYPE_INT.

Referenced by cpl_image_divide_create(), and cpl_image_new_from_mask().

4.24.4.145 cpl_image_xor()

```
cpl_error_code cpl_image_xor (
             cpl_image * self,
             const cpl_image * first,
             const cpl_image * second )
```

The bit-wise xor of two images with integer pixels.

Parameters

self	Pre-allocated image to hold the result
first	First operand, or NULL for an in-place operation
second	Second operand

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

Note

CPL_TYPE_INT is required

See also

cpl_mask_xor() for the equivalent logical operation

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if the images have different sizes
- CPL_ERROR_INVALID_TYPE if the passed image type is as required

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_INVALID_TYPE, CPL_E↩RROR_NULL_INPUT, cpl_image_get_type(), cpl_type_get_name(), and CPL_TYPE_INT.

4.24.4.146 cpl_image_xor_scalar()

```
cpl_error_code cpl_image_xor_scalar (
            cpl_image * self,
            const cpl_image * first,
            cpl_bitmask second )
```

The bit-wise xor of a scalar and an image with integer pixels.

Parameters

self	Pre-allocated image to hold the result
first	First operand, or NULL for an in-place operation
second	Second operand (scalar)

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_ on error

Note

CPL_TYPE_INT is required

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if the images have different sizes
- CPL_ERROR_INVALID_TYPE if the passed image type is as required

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_INVALID_TYPE, CPL_E↩RROR_NONE, CPL_ERROR_NULL_INPUT, cpl_image_get_type(), cpl_type_get_name(), cpl_type_get_sizeof(), and CPL_TYPE_INT.

4.24.4.147 cpl_vector_new_from_image_column()

```
cpl_vector* cpl_vector_new_from_image_column (
            const cpl_image * image_in,
            cpl_size pos )
```

Extract a column from an image.

Parameters

image↩ _in	Input image
pos	Position of the column (1 for the left one)

Returns

1 newly allocated cpl_vector or NULL on error

Images can be of type CPL_TYPE_INT, CPL_TYPE_FLOAT or CPL_TYPE_DOUBLE. The returned vector must be deallocated using cpl_vector_delete().

The bad pixels map is not taken into account in this function.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if pos is not valid
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NULL_↩INPUT, cpl_vector_delete(), cpl_vector_get_data(), and cpl_vector_new().

Referenced by cpl_image_get_fwhm().

4.24.4.148 cpl_vector_new_from_image_row()

```
cpl_vector* cpl_vector_new_from_image_row (
            const cpl_image * image_in,
            cpl_size pos )
```

Extract a row from an image.

Parameters

image_in	Input image
pos	Position of the row (1 for the bottom one)

Returns

1 newly allocated cpl_vector or NULL on error

Images can be of type CPL_TYPE_INT, CPL_TYPE_FLOAT or CPL_TYPE_DOUBLE. The returned vector must be deallocated using cpl_vector_delete().

The bad pixels map is not taken into account in this function.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if pos is not valid
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NULL_INPUT, cpl_vector_delete(), cpl_vector_get_data(), and cpl_vector_new().

Referenced by cpl_image_get_fwhm().

4.25 Library Initialization

Functions

- void cpl_end (void)

  Stop the internal subsystems of CPL.
- const char * cpl_get_description (unsigned self)

  Create a string of version numbers of CPL and its libraries.
- void cpl_init (unsigned self)

  Initialise the CPL core library.

4.25.1 Detailed Description

The module provides the CPL library startup routine. The startup routine initialises CPL internal data structures. For this reason, any application using functions from the CPL libraries **must** call the startup routine prior to calling any other CPL function.

Synopsis:

```
#include <cpl_init.h>
```

4.25.2 Function Documentation

4.25.2.1 cpl_end()

```
void cpl_end (
              void  )
```

Stop the internal subsystems of CPL.

Returns

Nothing.

Note

Currently, the behaviour of any CPL function becomes undefined after this function is called.

This function must be called after any other CPL function is called.

References cpl_fits_set_mode(), CPL_FITS_STOP_CACHING, and cpl_msg_stop().

4.25.2.2 cpl_get_description()

```
const char* cpl_get_description (
              unsigned self )
```

Create a string of version numbers of CPL and its libraries.

Parameters

self	CPL_DESCRIPTION_DEFAULT

Returns

A pointer to a constant character array

References cpl_msg_warning().

4.25.2.3 cpl_init()

```
void cpl_init (
            unsigned self )
```

Initialise the CPL core library.

Parameters

self	*CPL_INIT_DEFAULT* is the only supported value

Returns

Nothing.

Note

The function must be called once before any other CPL function.

See also

cpl_fits_set_mode()

This function sets up the library internal subsystems, which other CPL functions expect to be in a defined state. In particular, the CPL memory management and the CPL messaging systems are initialised by this function call.

One of the internal subsystems of CPL handles memory allocation. The default CPL memory mode is defined during the build procedure, this default can be changed during the call to cpl_init() via the environment variable *CPL_MEMORY_MODE*. The valid values are 0: Use the default system functions for memory handling 1: Exit if a memory-allocation fails, provide checking for memory leaks, limited reporting of memory allocation and limited protection on deallocation of invalid pointers. 2: Exit if a memory-allocation fails, provide checking for memory leaks, extended reporting of memory allocation and protection on deallocation of invalid pointers. Any other value (including NULL) will leave the default memory mode unchanged.

This function also reads the environment variable *CPL_IO_MODE*. Iff set to 1, cpl_fits_set_mode() is called with CPL_FITS_START_CACHING.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_INCOMPATIBLE_INPUT if there is an inconsistency between the run- time and compile-time versions of a library that CPL depends on internally, e.g. CFITSIO. This error may occur with dynamic linking. If it does occur, the use of CPL may lead to unexpected behaviour.

4.26 Library Version Information

Functions

- unsigned int cpl_version_get_binary_age (void)
  Get the library's binary age.
- unsigned int cpl_version_get_binary_version (void)
  Get the library's binary version number.
- unsigned int cpl_version_get_interface_age (void)
  Get the library's interface age.
- unsigned int cpl_version_get_major (void)
  Get the library's major version number.
- unsigned int cpl_version_get_micro (void)
  Get the library's micro version number.
- unsigned int cpl_version_get_minor (void)
  Get the library's minor version number.
- const char * cpl_version_get_version (void)
  Get the library's version string.

4.26.1 Detailed Description

This module provides functions to access the library's version information.

The library version applies to all component libraries of the Common pipeline library, and changes if any of the component libraries changes.

The library version is a version code made up of three components, the major, minor and micro version number, and is usually represented by a string of the form "major.minor.micro".

Synopsis:

```
#include <cpl_version.h>
```

4.26.2 Function Documentation

4.26.2.1 cpl_version_get_binary_age()

```
unsigned int cpl_version_get_binary_age (
            void  )
```

Get the library's binary age.

Returns

The function returns the library's binary age.

The function returns the binary age of the library.

4.26.2.2 cpl_version_get_binary_version()

```
unsigned int cpl_version_get_binary_version (
            void  )
```

Get the library's binary version number.

Returns

The function returns the library's version number.

The function returns the version number of the library encoded as a single integer number.

4.26.2.3 cpl_version_get_interface_age()

```
unsigned int cpl_version_get_interface_age (
            void  )
```

Get the library's interface age.

Returns

The function returns the library's interface age.

The function returns the interface age of the library.

4.26.2.4 cpl_version_get_major()

```
unsigned int cpl_version_get_major (
            void  )
```

Get the library's major version number.

Returns

The function returns the library's major version number.

The function returns the major version number of the library.

4.26.2.5 cpl_version_get_micro()

```
unsigned int cpl_version_get_micro (
            void  )
```

Get the library's micro version number.

Returns

The function returns the library's micro version number.

The function returns the micro version number of the library.

4.26.2.6 cpl_version_get_minor()

```
unsigned int cpl_version_get_minor (
            void  )
```

Get the library's minor version number.

Returns

The function returns the library's minor version number.

The function returns the minor version number of the library.

4.26.2.7 cpl_version_get_version()

```
const char* cpl_version_get_version (
            void  )
```

Get the library's version string.

Returns

The function returns the library's version string.

The function returns the package version of the library as a string. The returned version string is composed of the major, minor and, possibly, the micro version of the library separated by dots. The micro version may not be there if it is `0`.

4.27 Masks of pixels

Functions

- cpl_error_code cpl_mask_and (cpl_mask *in1, const cpl_mask *in2)

  Performs a logical AND of one mask onto another.
- cpl_error_code cpl_mask_closing (cpl_mask *in, const cpl_matrix *ker)

  Compute a morphological closing.
- cpl_mask * cpl_mask_collapse_create (const cpl_mask *in, int dir)

  Collapse a mask.
- cpl_error_code cpl_mask_copy (cpl_mask *in1, const cpl_mask *in2, cpl_size x_pos, cpl_size y_pos)

  Insert a mask in an other one.
- cpl_size cpl_mask_count (const cpl_mask *in)

  Get the number of occurences of CPL_BINARY_1.
- cpl_size cpl_mask_count_window (const cpl_mask *self, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury)

  Get the number of occurences of CPL_BINARY_1 in a window.
- void cpl_mask_delete (cpl_mask *m)

  Delete a cpl_mask.
- cpl_error_code cpl_mask_dilation (cpl_mask *in, const cpl_matrix *ker)

  Compute a morphological dilation.
- cpl_error_code cpl_mask_dump_window (const cpl_mask *self, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury, FILE *stream)

  Dump a mask.
- cpl_mask * cpl_mask_duplicate (const cpl_mask *in)

  Duplicates a cpl_mask.
- cpl_error_code cpl_mask_erosion (cpl_mask *in, const cpl_matrix *ker)

  Compute a morphological erosion.
- cpl_mask * cpl_mask_extract (const cpl_mask *in, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury)

  Extract a mask from an other one.
- cpl_mask * cpl_mask_extract_subsample (const cpl_mask *in, cpl_size xstep, cpl_size ystep)

  Subsample a mask.
- cpl_error_code cpl_mask_filter (cpl_mask *self, const cpl_mask *other, const cpl_mask *kernel, cpl_filter_mode filter, cpl_border_mode border)

  Filter a mask using a binary kernel.
- cpl_error_code cpl_mask_flip (cpl_mask *in, int angle)

  Flip a mask on a given mirror line.
- cpl_binary cpl_mask_get (const cpl_mask *self, cpl_size xpos, cpl_size ypos)

  Get the value of a mask at a given position.
- cpl_binary * cpl_mask_get_data (cpl_mask *in)

  Get a pointer to the data part of the mask.
- const cpl_binary * cpl_mask_get_data_const (const cpl_mask *in)

  Get a pointer to the data part of the mask.
- cpl_size cpl_mask_get_size_x (const cpl_mask *in)

  Get the x size of the mask.
- cpl_size cpl_mask_get_size_y (const cpl_mask *in)

  Get the y size of the mask.
- cpl_boolean cpl_mask_is_empty (const cpl_mask *self)

  Return CPL_TRUE iff a mask has no elements set (to CPL_BINARY_1)
- cpl_boolean cpl_mask_is_empty_window (const cpl_mask *self, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury)

  Return CPL_TRUE iff a mask has no elements set in the window.

- cpl_mask * cpl_mask_load (const char *filename, cpl_size pnum, cpl_size xtnum)

  Load a mask from a FITS file.
- cpl_mask * cpl_mask_load_window (const char *filename, cpl_size pnum, cpl_size xtnum, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury)

  Load a mask from a FITS file.
- cpl_error_code cpl_mask_move (cpl_mask *in, cpl_size nb_cut, const cpl_size *new_pos)

  Permute tiles in a mask.
- cpl_mask * cpl_mask_new (cpl_size nx, cpl_size ny)

  Create a new cpl_mask.
- cpl_error_code cpl_mask_not (cpl_mask *in)

  Performs a logical NOT on a mask.
- cpl_error_code cpl_mask_opening (cpl_mask *in, const cpl_matrix *ker)

  Compute a morphological opening.
- cpl_error_code cpl_mask_or (cpl_mask *in1, const cpl_mask *in2)

  Performs a logical OR of one mask onto another.
- cpl_error_code cpl_mask_save (const cpl_mask *self, const char *filename, const cpl_propertylist *plist, unsigned mode)

  Save a mask to a FITS file.
- cpl_error_code cpl_mask_set (cpl_mask *self, cpl_size xpos, cpl_size ypos, cpl_binary value)

  Set a value in a mask at a given position.
- cpl_error_code cpl_mask_shift (cpl_mask *self, cpl_size dx, cpl_size dy)

  Shift a mask.
- cpl_error_code cpl_mask_threshold_image (cpl_mask *self, const cpl_image *image, double lo_cut, double hi_cut, cpl_binary inval)

  Select parts of an image with provided thresholds.
- cpl_mask * cpl_mask_threshold_image_create (const cpl_image *in, double lo_cut, double hi_cut)

  Select parts of an image with provided thresholds.
- cpl_error_code cpl_mask_turn (cpl_mask *self, int rot)

  Rotate a mask by a multiple of 90 degrees clockwise.
- void * cpl_mask_unwrap (cpl_mask *m)

  Delete a cpl_mask except the data array.
- cpl_mask * cpl_mask_wrap (cpl_size nx, cpl_size ny, cpl_binary *data)

  Create a cpl_mask from existing data.
- cpl_error_code cpl_mask_xor (cpl_mask *in1, const cpl_mask *in2)

  Performs a logical XOR of one mask onto another.

4.27.1 Detailed Description

This module provides functions to handle masks of pixels

These masks are useful for object detection routines or bad pixel map handling. Morphological routines (erosion, dilation, closing and opening) and logical operations are provided. A cpl_mask is a kind of binary array whose elements are of type cpl_binary and can take only two values: either CPL_BINARY_0 or CPL_BINARY_1.

The element indexing follows the FITS convention in the sense that the lower left element in a CPL mask has index (1, 1).

Synopsis:

```
#include "cpl_mask.h"
```

4.27.2 Function Documentation

4.27.2.1 cpl_mask_and()

```
cpl_error_code cpl_mask_and (
            cpl_mask * in1,
            const cpl_mask * in2 )
```

Performs a logical AND of one mask onto another.

Parameters

in1	first mask, to be modified
in2	second mask

Returns

CPL_ERROR_NONE on success, otherwise the relevant _cpl_error_code_

See also

cpl_image_and()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the two masks have different sizes

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, and CPL_ERROR_NULL_INPUT.

4.27.2.2 cpl_mask_closing()

```
cpl_error_code cpl_mask_closing (
            cpl_mask * in,
            const cpl_matrix * ker )
```

Compute a morphological closing.

Parameters

in	input mask to filter
ker	binary kernel (0 for 0, any other value is considered as 1)

Returns

CPL_ERROR_NONE on success or the _cpl_error_code_ on failure

See also

cpl_mask_filter()

Deprecated Replace this call with cpl_mask_filter() using CPL_FILTER_CLOSING and CPL_BORDER_ZERO.

4.27.2.3 cpl_mask_collapse_create()

```
cpl_mask* cpl_mask_collapse_create (
            const cpl_mask * in,
            int dir )
```

Collapse a mask.

Parameters

in	input mask to collapse
dir	collapsing direction

Returns

the newly allocated mask or NULL on error

Note

The returned mask must be deallocated using cpl_mask_delete().

direction 0 collapses along y, producing a nx by 1 mask direction 1 collapses along x, producing a 1 by ny mask

The resulting mask element is set to CPL_BINARY_1 iff all elements of the associated column (resp. row) in the input mask are set to CPL_BINARY_1.

```
Direction 0 collapse:

1 0 1    Input mask.
0 1 1
0 0 1
-----
0 0 1    Only the third element is set to CPL_BINARY_1 since all input
         elements of that column are set to CPL_BINARY_1.
```

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT
- CPL_ERROR_ILLEGAL_INPUT

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_malloc(), and cpl_↩ mask_wrap().

4.27.2.4 cpl_mask_copy()

```
cpl_error_code cpl_mask_copy (
            cpl_mask * in1,
            const cpl_mask * in2,
            cpl_size x_pos,
            cpl_size y_pos )
```

Insert a mask in an other one.

Parameters

in1	mask in which in2 is inserted
in2	mask to insert
x_pos	the x pixel position in in1 where the lower left pixel of in2 should go (from 1 to the x size of in1)
y_pos	the y pixel position in in1 where the lower left pixel of in2 should go (from 1 to the y size of in1)

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if in1 or in2 is NULL
- CPL_ERROR_ILLEGAL_INPUT if x_pos, y_pos is outside in1

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_IN↩PUT, cpl_mask_get_data(), and cpl_mask_get_data_const().

Referenced by cpl_image_copy().

4.27.2.5 cpl_mask_count()

```
cpl_size cpl_mask_count (
            const cpl_mask * in )
```

Get the number of occurences of CPL_BINARY_1.

Parameters

in	the input mask

Returns

the number of occurences of CPL_BINARY_1 or -1 on error

See also

cpl_mask_count_window()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, CPL_ERROR_NULL_INPUT, and cpl_mask_count_window().

Referenced by cpl_image_count_rejected(), and cpl_image_dump_structure().

4.27.2.6 cpl_mask_count_window()

```
cpl_size cpl_mask_count_window (
            const cpl_mask * self,
            cpl_size llx,
            cpl_size lly,
            cpl_size urx,
            cpl_size ury )
```

Get the number of occurences of CPL_BINARY_1 in a window.

Parameters

self	The mask to count
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)

Returns

the number of occurences of CPL_BINARY_1 or -1 on error

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the window coordinates are invalid

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, and CPL_MIN.

Referenced by cpl_mask_count().

4.27.2.7 cpl_mask_delete()

```
void cpl_mask_delete (
            cpl_mask * m )
```

Delete a cpl_mask.

Parameters

m	cpl_mask to delete

Returns

void

The function deallocates the memory used by the mask *m*. If *m* is `NULL`, nothing is done, and no error is set.

References cpl_free().

Referenced by cpl_fit_imagelist_polynomial_window(), cpl_image_accept_all(), cpl_image_copy(), cpl_image_delete(), cpl_image_divide(), cpl_image_divide_create(), cpl_image_unwrap(), cpl_image_warp(), cpl_image_warp_polynomial(), cpl_imagelist_swap_axis_create(), cpl_mask_flip(), cpl_mask_move(), cpl_mask_threshold_image_create(), cpl_mask_turn(), and cpl_vector_filter_median_create().

4.27.2.8 cpl_mask_dilation()

```
cpl_error_code cpl_mask_dilation (
            cpl_mask * in,
            const cpl_matrix * ker )
```

Compute a morphological dilation.

Parameters

in	input mask to filter
ker	binary kernel (0 for 0, any other value is considered as 1)

Returns

CPL_ERROR_NONE on success or the _cpl_error_code_ on failure

See also

cpl_mask_filter()

Deprecated Replace this call with cpl_mask_filter() using CPL_FILTER_DILATION and CPL_BORDER_ZERO.

4.27.2.9 cpl_mask_dump_window()

```
cpl_error_code cpl_mask_dump_window (
            const cpl_mask * self,
```

```
            cpl_size llx,
            cpl_size lly,
            cpl_size urx,
            cpl_size ury,
            FILE * stream )
```

Dump a mask.

Parameters

self	mask to dump
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)
stream	Output stream, accepts `stdout` or `stderr`

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_FILE_IO if a write operation fails
- CPL_ERROR_ACCESS_OUT_OF_RANGE if the defined window is not in the mask
- CPL_ERROR_ILLEGAL_INPUT if the window definition is wrong (e.g llx $>$ urx)

References cpl_ensure_code, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_FILE_IO, CPL_ERR↩OR_ILLEGAL_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and CPL_SIZE_FORMAT.

4.27.2.10 cpl_mask_duplicate()

```
cpl_mask* cpl_mask_duplicate (
            const cpl_mask * in )
```

Duplicates a cpl_mask.

Parameters

in	the mask to duplicate

Returns

1 newly allocated cpl_mask or NULL on error

The returned object must be deallocated using cpl_mask_delete().

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if in is NULL

References CPL_ERROR_NULL_INPUT, cpl_malloc(), and cpl_mask_wrap().

Referenced by cpl_image_average_create(), cpl_image_cast(), cpl_image_divide(), cpl_image_divide_create(), cpl_image_duplicate(), cpl_mask_flip(), cpl_mask_move(), and cpl_mask_turn().

4.27.2.11 cpl_mask_erosion()

```
cpl_error_code cpl_mask_erosion (
            cpl_mask * in,
            const cpl_matrix * ker )
```

Compute a morphological erosion.

Parameters

in	input mask to filter
ker	binary kernel (0 for 0, any other value is considered as 1)

Returns

CPL_ERROR_NONE on success or the _cpl_error_code_ on failure

See also

cpl_mask_filter()

Deprecated Replace this call with cpl_mask_filter() using CPL_FILTER_EROSION and CPL_BORDER_ZERO.

4.27.2.12 cpl_mask_extract()

```
cpl_mask* cpl_mask_extract (
            const cpl_mask * in,
            cpl_size llx,
            cpl_size lly,
            cpl_size urx,
            cpl_size ury )
```

Extract a mask from an other one.

Parameters

in	input mask
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)

Returns

1 newly allocated mask or NULL on error.

Note

The returned mask must be deallocated using cpl_mask_delete().

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_ILLEGAL_INPUT if the zone falls outside the mask
- CPL_ERROR_NULL_INPUT if the input mask is NULL

References cpl_errorstate_get(), cpl_errorstate_is_equal(), and cpl_mask_new().

4.27.2.13 cpl_mask_extract_subsample()

```
cpl_mask* cpl_mask_extract_subsample (
            const cpl_mask * in,
            cpl_size xstep,
            cpl_size ystep )
```

Subsample a mask.

Parameters

in	input mask
xstep	Take every xstep pixel in x
ystep	Take every ystep pixel in y

Returns

the newly allocated mask or NULL on error case

See also

cpl_image_extract_subsample()

The returned mask must be deallocated using cpl_mask_delete().

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if in is NULL
- CPL_ERROR_ILLEGAL_INPUT if xstep and ystep are not greater than zero

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_mask_get_data(), cpl_mask_get_data_const(), and cpl_mask_new().

Referenced by cpl_image_extract_subsample().

4.27.2.14 cpl_mask_filter()

```
cpl_error_code cpl_mask_filter (
            cpl_mask * self,
            const cpl_mask * other,
            const cpl_mask * kernel,
            cpl_filter_mode filter,
            cpl_border_mode border )
```

Filter a mask using a binary kernel.

Parameters

self	Pre-allocated mask to hold the filtered result
other	Mask to filter
kernel	Elements to use, if set to CPL_BINARY_1
filter	CPL_FILTER_EROSION, CPL_FILTER_DILATION, CPL_FILTER_OPENING, CPL_FILTER_CLOSING
border	CPL_BORDER_NOP, CPL_BORDER_ZERO or CPL_BORDER_COPY

Returns

CPL_ERROR_NONE or the relevant CPL error code

The two masks must have equal dimensions.

The kernel must have an odd number of rows and an odd number of columns.

At least one kernel element must be set to CPL_BINARY_1.

For erosion and dilation: In-place filtering is not supported, but the input buffer may overlap all but the 1+h first rows of the output buffer, where 1+2∗h is the number of rows in the kernel.

For opening and closing: Opening is implemented as an erosion followed by a dilation, and closing is implemented as a dilation followed by an erosion. As such a temporary, internal buffer the size of self is allocated and used. Consequently, in-place opening and closing is supported with no additional overhead, it is achieved by passing the same mask as both self and other.

Duality and idempotency: Erosion and Dilation have the duality relations: not(dil(A,B)) = er(not(A), B) and not(er(↩ A,B)) = dil(not(A), B).

Opening and closing have similar duality relations: not(open(A,B)) = close(not(A), B) and not(close(A,B)) = open(not(A), B).

Opening and closing are both idempotent, i.e. open(A,B) = open(open(A,B),B) and close(A,B) = close(close(A,B),B).

The above duality and idempotency relations do *not* hold on the mask border (with the currently supported border modes).

Unnecessary large kernels: Adding an empty border to a given kernel should not change the outcome of the filtering. However doing so widens the border of the mask to be filtered and therefore has an effect on the filtering of the mask border. Since an unnecessary large kernel is also more costly to apply, such kernels should be avoided.

A 1 x 3 erosion filtering example (error checking omitted for brevity)

```
cpl_mask * kernel = cpl_mask_new(1, 3);
cpl_mask_not(kernel);
cpl_mask_filter(filtered, raw, kernel, CPL_FILTER_EROSION,
                                       CPL_BORDER_NOP);
cpl_mask_delete(kernel);
```

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL.
- CPL_ERROR_ILLEGAL_INPUT if the kernel has a side of even length.
- CPL_ERROR_DATA_NOT_FOUND If the kernel is empty.
- CPL_ERROR_ACCESS_OUT_OF_RANGE If the kernel has a side longer than the input mask.
- CPL_ERROR_INCOMPATIBLE_INPUT If the input and output masks have incompatible sizes.
- CPL_ERROR_UNSUPPORTED_MODE If the output pixel buffer overlaps the input one (or the kernel), or the border/filter mode is unsupported.

References CPL_BORDER_COPY, CPL_BORDER_CROP, CPL_BORDER_NOP, CPL_BORDER_ZERO, cpl_ensure_code, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_DATA_NOT_FOUND, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NULL_INPUT, CPL_ERROR_UNSUPPORTED_MODE, CPL_FILTER_CLOSING, CPL_FILTER_EROSION, CPL_FILTER_OPENING, cpl_mask_get_data(), cpl_mask_get_data_const(), cpl_mask_get_size_x(), cpl_mask_get_size_y(), and cpl_mask_is_empty().

4.27.2.15 cpl_mask_flip()

```
cpl_error_code cpl_mask_flip (
            cpl_mask * in,
            int angle )
```

Flip a mask on a given mirror line.

Parameters

in	mask to flip
angle	mirror line in polar coord. is theta = (PI/4) * angle

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

angle can take one of the following values:

- 0 (theta=0) to flip the image around the horizontal
- 1 (theta=pi/4) to flip the image around y=x
- 2 (theta=pi/2) to flip the image around the vertical

- 3 (theta=3pi/4) to flip the image around y=-x

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if in is NULL
- CPL_ERROR_ILLEGAL_INPUT if angle is not as specified

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_IN↩PUT, cpl_mask_delete(), cpl_mask_duplicate(), cpl_mask_get_data(), and cpl_mask_get_data_const().

Referenced by cpl_image_flip().

4.27.2.16 cpl_mask_get()

```
cpl_binary cpl_mask_get (
             const cpl_mask * self,
             cpl_size xpos,
             cpl_size ypos )
```

Get the value of a mask at a given position.

Parameters

self	The input mask
xpos	Pixel x position (FITS convention, 1 for leftmost)
ypos	Pixel y position (FITS convention, 1 for lowest)

Returns

The mask value or undefined if an error code is set

The mask value can be either CPL_BINARY_0 or CPL_BINARY_1

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if xpos or ypos is out of bounds

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_image_dump_window(), and cpl_image_is_rejected().

4.27.2.17 cpl_mask_get_data()

```
cpl_binary* cpl_mask_get_data (
             cpl_mask * in )
```

Get a pointer to the data part of the mask.

Parameters

in	the input mask

Returns

Pointer to the data or NULL on error

The returned pointer refers to already allocated data.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_image_divide(), cpl_image_divide_create(), cpl_image_fill_rejected(), cpl_image_filter_mask(), cpl_image_rebin(), cpl_image_reject_from_mask(), cpl_image_warp(), cpl_image_warp_polynomial(), cpl_imagelist_swap_axis_create(), cpl_mask_copy(), cpl_mask_extract_subsample(), cpl_mask_filter(), cpl_mask_flip(), cpl_mask_move(), cpl_mask_threshold_image(), and cpl_mask_turn().

4.27.2.18 cpl_mask_get_data_const()

```
const cpl_binary* cpl_mask_get_data_const (
            const cpl_mask * in )
```

Get a pointer to the data part of the mask.

Parameters

in	the input mask

Returns

Pointer to the data or NULL on error

See also

cpl_mask_get_data

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_detector_interpolate_rejected(), cpl_image_filter_mask(), cpl_image_get_interpolated(), cpl_image_labelise_mask_create(), cpl_image_new_from_mask(), cpl_image_rebin(), cpl_image_reject_from_mask(), cpl_image_warp(), cpl_image_warp_polynomial(), cpl_mask_copy(), cpl_mask_extract_subsample(), cpl_mask_filter(), cpl_mask_flip(), cpl_mask_move(), cpl_mask_threshold_image(), cpl_mask_turn(), and cpl_plot_mask().

4.27.2.19 cpl_mask_get_size_x()

```
cpl_size cpl_mask_get_size_x (
            const cpl_mask * in )
```

Get the x size of the mask.

Parameters

in	the input mask

Returns

The mask x size, or -1 on NULL input

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_image_filter_mask(), cpl_image_labelise_mask_create(), cpl_image_new_from_mask(), cpl_↩ image_reject_from_mask(), cpl_mask_filter(), cpl_mask_threshold_image(), and cpl_plot_mask().

4.27.2.20 cpl_mask_get_size_y()

```
cpl_size cpl_mask_get_size_y (
            const cpl_mask * in )
```

Get the y size of the mask.

Parameters

in	the input mask

Returns

The mask y size, or -1 on NULL input

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_image_filter_mask(), cpl_image_labelise_mask_create(), cpl_image_new_from_mask(), cpl_↩ image_reject_from_mask(), cpl_mask_filter(), cpl_mask_threshold_image(), and cpl_plot_mask().

4.27.2.21 cpl_mask_is_empty()

```
cpl_boolean cpl_mask_is_empty (
            const cpl_mask * self )
```

Return CPL_TRUE iff a mask has no elements set (to CPL_BINARY_1)

Parameters

self	The mask to search

Returns

CPL_TRUE iff the mask has no elements set (to CPL_BINARY_1)

See also

cpl_mask_is_empty_window()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_fit_image_gaussian(), cpl_image_dump_window(), cpl_image_fill_rejected(), cpl_image_filter_mask(), and cpl_mask_filter().

4.27.2.22 cpl_mask_is_empty_window()

```
cpl_boolean cpl_mask_is_empty_window (
            const cpl_mask * self,
            cpl_size llx,
            cpl_size lly,
            cpl_size urx,
            cpl_size ury )
```

Return CPL_TRUE iff a mask has no elements set in the window.

Parameters

self	The mask to search
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)

Returns

CPL_TRUE iff the mask has no elements set (to CPL_BINARY_1)

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the window coordinates are not valid

4.27.2.23 cpl_mask_load()

```
cpl_mask* cpl_mask_load (
              const char * filename,
              cpl_size pnum,
              cpl_size xtnum )
```

Load a mask from a FITS file.

Parameters

filename	Name of the file to load from.
pnum	Plane number in the Data Unit (0 for first)
xtnum	Extension number in the file (0 for primary HDU)

Returns

1 newly allocated mask or NULL on error

See also

cpl_image_load()

This function loads a mask from a FITS file (NAXIS=2 or 3). Each non-zero pixel is set to CPL_BINARY_1.

The returned mask has to be deallocated with cpl_mask_delete().

'xtnum' specifies from which extension the mask should be loaded. This could be 0 for the main data section (files without extension), or any number between 1 and N, where N is the number of extensions present in the file.

The requested plane number runs from 0 to nplanes-1, where nplanes is the number of planes present in the requested data section.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_FILE_IO if the file cannot be opened or does not exist
- CPL_ERROR_BAD_FILE_FORMAT if the data cannot be loaded from the file
- CPL_ERROR_ILLEGAL_INPUT if the passed extension number is negative
- CPL_ERROR_DATA_NOT_FOUND if the specified extension has no mask data

4.27.2.24 cpl_mask_load_window()

```
cpl_mask* cpl_mask_load_window (
            const char * filename,
            cpl_size pnum,
            cpl_size xtnum,
            cpl_size llx,
            cpl_size lly,
            cpl_size urx,
            cpl_size ury )
```

Load a mask from a FITS file.

Parameters

filename	Name of the file to load from.
pnum	Plane number in the Data Unit (0 for first)
xtnum	Extension number in the file.
llx	Lower left x position (FITS convention, 1 for leftmost)
lly	Lower left y position (FITS convention, 1 for lowest)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)

Returns

1 newly allocated mask or NULL on error

See also

cpl_mask_load()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_FILE_IO if the file does not exist
- CPL_ERROR_BAD_FILE_FORMAT if the data cannot be loaded from the file
- CPL_ERROR_ILLEGAL_INPUT if the passed position is invalid

4.27.2.25 cpl_mask_move()

```
cpl_error_code cpl_mask_move (
            cpl_mask * in,
            cpl_size nb_cut,
            const cpl_size * new_pos )
```

Permute tiles in a mask.

Parameters

in	Mask to modify
nb_cut	The number of cuts in x and y
new_pos	Array with the nb_cut^2 permuted positions

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

See also

cpl_image_move()

Note

The permutation array *must* contain nb_cut-squared elements

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if nb_cut is not strictly positive or cannot divide one of the image sizes or if the new_pos array does not contain a permutation of the values from 1 through nb_cut squared.

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_INP↩UT, cpl_free(), cpl_malloc(), cpl_mask_delete(), cpl_mask_duplicate(), cpl_mask_get_data(), and cpl_mask_get↩_data_const().

Referenced by cpl_image_move().

4.27.2.26 cpl_mask_new()

```
cpl_mask* cpl_mask_new (
              cpl_size nx,
              cpl_size ny )
```

Create a new cpl_mask.

Parameters

nx	The number of elements in the X-direction
ny	The number of elements in the Y-direction

Returns

1 newly allocated cpl_mask or NULL on error

Note

The returned object must be deallocated using cpl_mask_delete().

The created cpl_mask elements are all set to CPL_BINARY_0.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_ILLEGAL_INPUT if nx or ny is negative

References cpl_calloc(), cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, and cpl_mask_wrap().

Referenced by cpl_geom_img_offset_fine(), cpl_image_copy(), cpl_image_divide(), cpl_image_divide_create(), cpl_image_get_bpm(), cpl_image_rebin(), cpl_image_shift(), cpl_image_warp(), cpl_image_warp_polynomial(), cpl_imagelist_swap_axis_create(), cpl_mask_extract(), cpl_mask_extract_subsample(), cpl_mask_threshold_↩image_create(), and cpl_vector_filter_median_create().

4.27.2.27 cpl_mask_not()

```
cpl_error_code cpl_mask_not (
            cpl_mask * in )
```

Performs a logical NOT on a mask.

Parameters

in	mask to be modified

Returns

CPL_ERROR_NONE on success, otherwise the relevant _cpl_error_code_

See also

cpl_image_not()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure_code, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_geom_img_offset_fine(), and cpl_vector_filter_median_create().

4.27.2.28 cpl_mask_opening()

```
cpl_error_code cpl_mask_opening (
            cpl_mask * in,
            const cpl_matrix * ker )
```

Compute a morphological opening.

Parameters

in	input mask to filter
ker	binary kernel (0 for 0, any other value is considered as 1)

Returns

CPL_ERROR_NONE on success or the _cpl_error_code_ on failure

See also

cpl_mask_filter()

Deprecated Replace this call with cpl_mask_filter() using CPL_FILTER_OPENING and CPL_BORDER_ZERO.

4.27.2.29 cpl_mask_or()

```
cpl_error_code cpl_mask_or (
            cpl_mask * in1,
            const cpl_mask * in2 )
```

Performs a logical OR of one mask onto another.

Parameters

in1	first mask, to be modified
in2	second mask

Returns

CPL_ERROR_NONE on success, otherwise the relevant _cpl_error_code_

See also

cpl_image_or()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the two masks have different sizes

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_fit_imagelist_polynomial_window(), cpl_image_average_create(), cpl_image_divide(), and cpl↩_image_divide_create().

4.27.2.30 cpl_mask_save()

```
cpl_error_code cpl_mask_save (
            const cpl_mask * self,
            const char * filename,
            const cpl_propertylist * plist,
            unsigned mode )
```

Save a mask to a FITS file.

Parameters

self	mask to write to disk
filename	Name of the file to write
plist	Property list for the output header or NULL
mode	The desired output options

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_propertylist_save()

This function saves a mask to a FITS file. If a property list is provided, it is written to the header where the mask is written.

The type used in the file is CPL_TYPE_UCHAR (8 bit unsigned).

Supported output modes are CPL_IO_CREATE (create a new file) and CPL_IO_EXTEND (append a new extension to an existing file)

The output mode CPL_IO_EXTEND can be combined (via bit-wise or) with an option for tile-compression. This compression is lossless. The options are: CPL_IO_COMPRESS_GZIP, CPL_IO_COMPRESS_RICE, CPL_IO_↩ COMPRESS_HCOMPRESS, CPL_IO_COMPRESS_PLIO.

Note that in append mode the file must be writable (and do not take for granted that a file is writable just because it was created by the same application, as this depends from the system *umask*).

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the mode is not supported
- CPL_ERROR_FILE_NOT_CREATED if the output file cannot be created
- CPL_ERROR_FILE_IO if the data cannot be written to the file

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_IN↩ PUT, CPL_IO_COMPRESS_GZIP, CPL_IO_COMPRESS_HCOMPRESS, CPL_IO_COMPRESS_PLIO, CPL_IO↩ _COMPRESS_RICE, CPL_IO_CREATE, CPL_IO_EXTEND, and CPL_IO_MAX.

4.27.2.31 cpl_mask_set()

```
cpl_error_code cpl_mask_set (
            cpl_mask * self,
            cpl_size xpos,
            cpl_size ypos,
            cpl_binary value )
```

Set a value in a mask at a given position.

Parameters

self	the input mask
xpos	Pixel x position (FITS convention, 1 for leftmost)
ypos	Pixel y position (FITS convention, 1 for lowest)
value	the value to set in the mask

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

The value can be either CPL_BINARY_0 or CPL_BINARY_1

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if xpos or ypos is out of bounds or if value is different from CPL_BINARY_0 and CPL_BINARY_1

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, and CPL_ERROR_NULL↩ _INPUT.

4.27.2.32 cpl_mask_shift()

```
cpl_error_code cpl_mask_shift (
            cpl_mask * self,
            cpl_size dx,
            cpl_size dy )
```

Shift a mask.

Parameters

self	Mask to shift in place
dx	Shift in X
dy	Shift in Y

Returns

the _cpl_error_code_ or CPL_ERROR_NONE

See also

cpl_image_turn()

The 'empty zone' in the shifted mask is set to CPL_BINARY_1. The shift values have to be valid: -nx $<$ dx $<$ nx and -ny $<$ dy $<$ ny

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if in is NULL
- CPL_ERROR_ILLEGAL_INPUT if the offsets are too big

References cpl_ensure_code, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_image_shift().

4.27.2.33 cpl_mask_threshold_image()

```
cpl_error_code cpl_mask_threshold_image (
            cpl_mask * self,
            const cpl_image * image,
            double lo_cut,
            double hi_cut,
            cpl_binary inval )
```

Select parts of an image with provided thresholds.

Parameters

self	Mask to flag according to threshold
image	Image to threshold.
lo_cut	Lower bound for threshold.
hi_cut	Higher bound for threshold.
inval	This value (CPL_BINARY_1 or CPL_BINARY_0) is assigned where the pixel value is not marked as rejected and is strictly inside the provided interval. The other positions are assigned the other value.

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

The input image type can be CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT or CPL_TYPE_INT. If lo_cut is greater than or equal to hi_cut, then the mask is filled with outval.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_UNSUPPORTED_MODE if the pixel type is unsupported
- CPL_ERROR_INCOMPATIBLE_INPUT if the mask and image have different sizes
- CPL_ERROR_ILLEGAL_INPUT if inval is not one of CPL_BINARY_1 or CPL_BINARY_0

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, CPL_ERROR_UNSUPPORTED_MODE, cpl_image_get_bpm_const(), cpl_image_get_data_const(), cpl_image_get_size_x(), cpl_image_get_size_y(), cpl_image_get_type(), cpl_mask_get_data(), cpl_mask_get_data_const(), cpl_mask_get_size_x(), cpl_mask_get_size_y(), CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT, and CPL_TYPE_INT.

Referenced by cpl_mask_threshold_image_create().

4.27.2.34 cpl_mask_threshold_image_create()

```
cpl_mask* cpl_mask_threshold_image_create (
            const cpl_image * in,
            double lo_cut,
            double hi_cut )
```

Select parts of an image with provided thresholds.

Parameters

in	Image to threshold.
lo_cut	Lower bound for threshold.
hi_cut	Higher bound for threshold.

Returns

1 newly allocated mask or NULL on error

Note

The returned mask must be deallocated with cpl_mask_delete()

See also

cpl_mask_threshold_image()

References cpl_image_get_size_x(), cpl_image_get_size_y(), cpl_mask_delete(), cpl_mask_new(), and cpl_mask_threshold_image().

4.27.2.35 cpl_mask_turn()

```
cpl_error_code cpl_mask_turn (
            cpl_mask * self,
            int rot )
```

Rotate a mask by a multiple of 90 degrees clockwise.

Parameters

self	Mask to rotate in place
rot	The multiple: -1 is a rotation of 90 deg counterclockwise.

Returns

CPL_ERROR_NONE on success, otherwise the relevant _cpl_error_code_

See also

cpl_image_turn()

rot may be any integer value, its modulo 4 determines the rotation:

- -3 to turn 270 degrees counterclockwise.
- -2 to turn 180 degrees counterclockwise.
- -1 to turn 90 degrees counterclockwise.
- 0 to not turn
- +1 to turn 90 degrees clockwise (same as -3)
- +2 to turn 180 degrees clockwise (same as -2).
- +3 to turn 270 degrees clockwise (same as -1).

The definition of the rotation relies on the FITS convention: The lower left corner of the image is at (1,1), x increasing from left to right, y increasing from bottom to top.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if self is NULL

References cpl_ensure_code, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_mask_delete(), cpl_mask↩_duplicate(), cpl_mask_get_data(), and cpl_mask_get_data_const().

Referenced by cpl_image_turn().

4.27.2.36 cpl_mask_unwrap()

```
void* cpl_mask_unwrap (
            cpl_mask * m )
```

Delete a cpl_mask except the data array.

Parameters

m	cpl_mask to delete

Returns

A pointer to the data array or NULL if the input is NULL.

Note

The data array must be deallocated, otherwise a memory leak occurs.

References cpl_free().

4.27.2.37 cpl_mask_wrap()

```
cpl_mask* cpl_mask_wrap (
              cpl_size nx,
              cpl_size ny,
              cpl_binary * data )
```

Create a cpl_mask from existing data.

Parameters

nx	number of element in x direction
ny	number of element in y direction
data	Pointer to array of nx∗ny cpl_binary

Returns

1 newly allocated cpl_mask or NULL in case of an error

Note

The returned object must be deallocated using cpl_mask_unwrap().

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if nx or ny is negative or zero

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, and cpl_malloc().

Referenced by cpl_image_filter_mask(), cpl_mask_collapse_create(), cpl_mask_duplicate(), and cpl_mask_new().

4.27.2.38 cpl_mask_xor()

```
cpl_error_code cpl_mask_xor (
            cpl_mask * in1,
            const cpl_mask * in2 )
```

Performs a logical XOR of one mask onto another.

Parameters

in1	first mask, to be modified
in2	second mask

Returns

CPL_ERROR_NONE on success, otherwise the relevant _cpl_error_code_

See also

cpl_image_xor()

Note

Passing the same mask twice will clear it

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the two masks have different sizes

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, and CPL_ERROR_NULL_INPUT.

4.28 Matrices

Functions

- cpl_error_code cpl_matrix_add (cpl_matrix *matrix1, const cpl_matrix *matrix2)

  Add two matrices.
- cpl_error_code cpl_matrix_add_scalar (cpl_matrix *matrix, double value)

  Add a scalar to a matrix.
- cpl_error_code cpl_matrix_append (cpl_matrix *matrix1, const cpl_matrix *matrix2, int mode)

  Append a matrix to another.
- cpl_error_code cpl_matrix_copy (cpl_matrix *matrix, const cpl_matrix *submatrix, cpl_size row, cpl_size col)

  Write the values of a matrix into another matrix.
- cpl_error_code cpl_matrix_decomp_chol (cpl_matrix *self)

  *Replace a matrix by its Cholesky-decomposition, L * transpose(L) = A.*
- void cpl_matrix_delete (cpl_matrix *matrix)

  Delete a matrix.
- cpl_error_code cpl_matrix_divide (cpl_matrix *matrix1, const cpl_matrix *matrix2)

  Divide a matrix by another element by element.
- cpl_error_code cpl_matrix_divide_scalar (cpl_matrix *matrix, double value)

  Divide a matrix by a scalar.
- void cpl_matrix_dump (const cpl_matrix *matrix, FILE *stream)

  Print a matrix.
- cpl_matrix * cpl_matrix_duplicate (const cpl_matrix *matrix)

  Make a copy of a matrix.
- cpl_error_code cpl_matrix_erase_columns (cpl_matrix *matrix, cpl_size start, cpl_size count)

  Delete columns from a matrix.
- cpl_error_code cpl_matrix_erase_rows (cpl_matrix *matrix, cpl_size start, cpl_size count)

  Delete rows from a matrix.
- cpl_error_code cpl_matrix_exponential (cpl_matrix *matrix, double base)

  Compute the exponential of matrix elements.
- cpl_matrix * cpl_matrix_extract (const cpl_matrix *matrix, cpl_size start_row, cpl_size start_column, cpl_size step_row, cpl_size step_column, cpl_size nrows, cpl_size ncolumns)

  Extract a submatrix from a matrix.
- cpl_matrix * cpl_matrix_extract_column (const cpl_matrix *matrix, cpl_size column)

  Copy a matrix column.
- cpl_matrix * cpl_matrix_extract_diagonal (const cpl_matrix *matrix, cpl_size diagonal)

  Extract a matrix diagonal.
- cpl_matrix * cpl_matrix_extract_row (const cpl_matrix *matrix, cpl_size row)

  Extract a matrix row.
- cpl_error_code cpl_matrix_fill (cpl_matrix *matrix, double value)

  Write the same value to all matrix elements.
- cpl_error_code cpl_matrix_fill_column (cpl_matrix *matrix, double value, cpl_size column)

  Write the same value to a matrix column.
- cpl_error_code cpl_matrix_fill_diagonal (cpl_matrix *matrix, double value, cpl_size diagonal)

  Write a given value to all elements of a given matrix diagonal.
- cpl_error_code cpl_matrix_fill_row (cpl_matrix *matrix, double value, cpl_size row)

  Write the same value to a matrix row.
- cpl_error_code cpl_matrix_fill_window (cpl_matrix *matrix, double value, cpl_size row, cpl_size col, cpl_size nrow, cpl_size ncol)

  Write the same value into a submatrix of a matrix.
- cpl_error_code cpl_matrix_flip_columns (cpl_matrix *matrix)

Reverse order of columns in matrix.

- cpl_error_code cpl_matrix_flip_rows (cpl_matrix *matrix)

  Reverse order of rows in matrix.
- double cpl_matrix_get (const cpl_matrix *matrix, cpl_size row, cpl_size column)

  Get the value of a matrix element.
- double * cpl_matrix_get_data (cpl_matrix *matrix)

  Get the pointer to a matrix data buffer, or `NULL` in case of error.
- const double * cpl_matrix_get_data_const (const cpl_matrix *matrix)

  Get the pointer to a matrix data buffer, or `NULL` in case of error.
- double cpl_matrix_get_determinant (const cpl_matrix *matrix)

  Compute the determinant of a matrix.
- double cpl_matrix_get_max (const cpl_matrix *matrix)

  Find the maximum value of matrix elements.
- cpl_error_code cpl_matrix_get_maxpos (const cpl_matrix *matrix, cpl_size *row, cpl_size *column)

  Find position of the maximum value of matrix elements.
- double cpl_matrix_get_mean (const cpl_matrix *matrix)

  Find the mean of all matrix elements.
- double cpl_matrix_get_median (const cpl_matrix *matrix)

  Find the median of matrix elements.
- double cpl_matrix_get_min (const cpl_matrix *matrix)

  Find the minimum value of matrix elements.
- cpl_error_code cpl_matrix_get_minpos (const cpl_matrix *matrix, cpl_size *row, cpl_size *column)

  Find position of minimum value of matrix elements.
- cpl_size cpl_matrix_get_ncol (const cpl_matrix *matrix)

  Get the number of columns of a matrix.
- cpl_size cpl_matrix_get_ncol_ (const cpl_matrix *matrix)

  Get the number of columns of a matrix.
- cpl_size cpl_matrix_get_nrow (const cpl_matrix *matrix)

  Get the number of rows of a matrix.
- double cpl_matrix_get_stdev (const cpl_matrix *matrix)

  Find the standard deviation of matrix elements.
- cpl_matrix * cpl_matrix_invert_create (const cpl_matrix *matrix)

  Find a matrix inverse.
- int cpl_matrix_is_diagonal (const cpl_matrix *matrix, double tolerance)

  Check if a matrix is diagonal.
- int cpl_matrix_is_identity (const cpl_matrix *matrix, double tolerance)

  Check for identity matrix.
- int cpl_matrix_is_zero (const cpl_matrix *matrix, double tolerance)

  Check for zero matrix.
- cpl_error_code cpl_matrix_logarithm (cpl_matrix *matrix, double base)

  Compute the logarithm of matrix elements.
- cpl_error_code cpl_matrix_multiply (cpl_matrix *matrix1, const cpl_matrix *matrix2)

  Multiply two matrices element by element.
- cpl_error_code cpl_matrix_multiply_scalar (cpl_matrix *matrix, double value)

  Multiply a matrix by a scalar.
- cpl_matrix * cpl_matrix_new (cpl_size rows, cpl_size columns)

  Create a zero matrix of given size.
- cpl_error_code cpl_matrix_power (cpl_matrix *matrix, double exponent)

  Compute a power of matrix elements.
- cpl_matrix * cpl_matrix_product_create (const cpl_matrix *matrix1, const cpl_matrix *matrix2)

  Rows-by-columns product of two matrices.

- cpl_error_code cpl_matrix_resize (cpl_matrix *matrix, cpl_size top, cpl_size bottom, cpl_size left, cpl_size right)

  Reframe a matrix.
- void cpl_matrix_set_ (cpl_matrix *matrix, cpl_size row, cpl_size column, double value)

  Write a value to a matrix element.
- cpl_error_code cpl_matrix_set_size (cpl_matrix *matrix, cpl_size rows, cpl_size columns)

  Resize a matrix.
- cpl_error_code cpl_matrix_shift (cpl_matrix *matrix, cpl_size rshift, cpl_size cshift)

  Shift matrix elements.
- cpl_matrix * cpl_matrix_solve (const cpl_matrix *coeff, const cpl_matrix *rhs)

  Solution of a linear system.
- cpl_error_code cpl_matrix_solve_chol (const cpl_matrix *self, cpl_matrix *rhs)

  *Solve a L*transpose(L)-system.*
- cpl_matrix * cpl_matrix_solve_normal (const cpl_matrix *coeff, const cpl_matrix *rhs)

  Solution of overdetermined linear equations in a least squares sense.
- cpl_matrix * cpl_matrix_solve_svd (const cpl_matrix *coeff, const cpl_matrix *rhs)

  Solve a linear system in a least square sense using an SVD factorization.
- cpl_matrix * cpl_matrix_solve_svd_threshold (const cpl_matrix *coeff, const cpl_matrix *rhs, int mode, double tolerance)

  Solve a linear system in a least square sense using an SVD factorization discarding singular values below a given threshold.
- cpl_error_code cpl_matrix_sort_columns (cpl_matrix *matrix, int mode)

  Sort matrix by columns.
- cpl_error_code cpl_matrix_sort_rows (cpl_matrix *matrix, int mode)

  Sort matrix by rows.
- cpl_error_code cpl_matrix_subtract (cpl_matrix *matrix1, const cpl_matrix *matrix2)

  Subtract a matrix from another.
- cpl_error_code cpl_matrix_subtract_scalar (cpl_matrix *matrix, double value)

  Subtract a scalar to a matrix.
- cpl_error_code cpl_matrix_swap_columns (cpl_matrix *matrix, cpl_size column1, cpl_size column2)

  Swap two matrix columns.
- cpl_error_code cpl_matrix_swap_rowcolumn (cpl_matrix *matrix, cpl_size row)

  Swap a matrix column with a matrix row.
- cpl_error_code cpl_matrix_swap_rows (cpl_matrix *matrix, cpl_size row1, cpl_size row2)

  Swap two matrix rows.
- cpl_error_code cpl_matrix_threshold_small (cpl_matrix *matrix, double tolerance)

  Rounding to zero very small numbers in matrix.
- cpl_matrix * cpl_matrix_transpose_create (const cpl_matrix *matrix)

  Create transposed matrix.
- void * cpl_matrix_unwrap (cpl_matrix *matrix)

  Delete a matrix, but not its data buffer.
- cpl_matrix * cpl_matrix_wrap (cpl_size rows, cpl_size columns, double *data)

  Create a new matrix from existing data.

4.28.1 Detailed Description

This module provides functions to create, destroy and use a *cpl_matrix*. The elements of a *cpl_matrix* with M rows and N columns are counted from 0,0 to M-1,N-1. The matrix element 0,0 is the one at the upper left corner of a matrix. The CPL matrix functions work properly only in the case the matrices elements do not contain garbage (such as `NaN` or infinity).

Synopsis:

```
#include <cpl_matrix.h>
```

4.28.2 Function Documentation

4.28.2.1 cpl_matrix_add()

```
cpl_error_code cpl_matrix_add (
            cpl_matrix * matrix1,
            const cpl_matrix * matrix2 )
```

Add two matrices.

Parameters

matrix1	Pointer to first matrix.
matrix2	Pointer to second matrix.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	An input matrix is a `NULL` pointer.
CPL_ERROR_INCOMPATIBLE_INPUT	The specified matrices do not have the same size.

Add two matrices element by element. The two matrices must have identical sizes. The result is written to the first matrix.

References CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.28.2.2 cpl_matrix_add_scalar()

```
cpl_error_code cpl_matrix_add_scalar (
            cpl_matrix * matrix,
            double value )
```

Add a scalar to a matrix.

Parameters

matrix	Pointer to matrix.
value	Value to add.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a NULL pointer.

Add the same value to each matrix element.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.28.2.3 cpl_matrix_append()

```
cpl_error_code cpl_matrix_append (
            cpl_matrix * matrix1,
            const cpl_matrix * matrix2,
            int mode )
```

Append a matrix to another.

Parameters

matrix1	Pointer to first matrix.
matrix2	Pointer to second matrix.
mode	Matrices connected horizontally (0) or vertically (1).

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	An input matrix is a NULL pointer.
CPL_ERROR_ILLEGAL_INPUT	*mode* is neither 0 nor 1.
CPL_ERROR_INCOMPATIBLE_INPUT	Matrices cannot be joined as indicated by *mode*.

If *mode* is set to 0, the matrices must have the same number of rows, and are connected horizontally with the first matrix on the left. If *mode* is set to 1, the matrices must have the same number of columns, and are connected vertically with the first matrix on top. The first matrix is expanded to include the values from the second matrix, while the second matrix is left untouched.

Note

The pointer to the first matrix data buffer may change, therefore pointers previously retrieved by calling `cpl_matrix_get_data()` should be discarded.

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_matrix_copy(), and cpl_matrix_resize().

4.28.2.4 cpl_matrix_copy()

```
cpl_error_code cpl_matrix_copy (
            cpl_matrix * matrix,
            const cpl_matrix * submatrix,
            cpl_size row,
            cpl_size col )
```

Write the values of a matrix into another matrix.

Parameters

matrix	Pointer to matrix to be modified.
submatrix	Pointer to matrix to get the values from.
row	Position of row 0 of *submatrix* in *matrix*.
col	Position of column 0 of *submatrix* in *matrix*.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* or *submatrix* are `NULL` pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	No overlap exists between the two matrices.

The values of *submatrix* are written to *matrix* starting at the indicated row and column. There are no restrictions on the sizes of *submatrix:* just the parts of *submatrix* overlapping *matrix* are copied. There are no restrictions on *row* and *col* either, that can also be negative. If the two matrices do not overlap, nothing is done, but an error condition is set.

References CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_matrix_append(), cpl_matrix_duplicate(), and cpl_matrix_resize().

4.28.2.5 cpl_matrix_decomp_chol()

```
cpl_error_code cpl_matrix_decomp_chol (
            cpl_matrix * self )
```

Replace a matrix by its Cholesky-decomposition, L ∗ transpose(L) = A.

Parameters

self	N by N symmetric positive-definite matrix to decompose

Returns

CPL_ERROR_NONE on success, or the relevant CPL error code

Note

Only the upper triangle of self is read, L is written in the lower triangle

If the matrix is singular the elements of self become undefined

See also

Golub & Van Loan, Matrix Computations, Algorithm 4.2.1 (Cholesky: Gaxpy Version).

Errors

CPL_ERROR_NULL_INPUT	An input pointer is `NULL`.
CPL_ERROR_ILLEGAL_INPUT	*self* is not an n by n matrix.
CPL_ERROR_SINGULAR_MAT↩ RIX	*self* is not symmetric, positive definite.

References cpl_ensure_code, CPL_ERROR_NULL_INPUT, and cpl_matrix_get_ncol_().

4.28.2.6 cpl_matrix_delete()

```
void cpl_matrix_delete (
             cpl_matrix * matrix )
```

Delete a matrix.

Parameters

matrix	Pointer to a matrix to be deleted.

Returns

Nothing.

This function frees all the memory associated to a matrix. If *matrix* is `NULL`, nothing is done.

References cpl_free().

Referenced by cpl_wcs_delete().

4.28.2.7 cpl_matrix_divide()

```
cpl_error_code cpl_matrix_divide (
            cpl_matrix * matrix1,
            const cpl_matrix * matrix2 )
```

Divide a matrix by another element by element.

Parameters

matrix1	Pointer to first matrix.
matrix2	Pointer to second matrix.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	An input matrix is a NULL pointer.
CPL_ERROR_INCOMPATIBLE_INPUT	The specified matrices do not have the same size.

Divide each element of the first matrix by the corresponding element of the second one. The two matrices must have the same number of rows and columns. The result is written to the first matrix. No check is made against a division by zero.

References CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.28.2.8 cpl_matrix_divide_scalar()

```
cpl_error_code cpl_matrix_divide_scalar (
            cpl_matrix * matrix,
            double value )
```

Divide a matrix by a scalar.

Parameters

matrix	Pointer to matrix.
value	Divisor.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.
CPL_ERROR_DIVISION_BY_ZERO	The input *value* is 0.0.

Divide each matrix element by the same value.

References CPL_ERROR_DIVISION_BY_ZERO, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.28.2.9 cpl_matrix_dump()

```
void cpl_matrix_dump (
            const cpl_matrix * matrix,
            FILE * stream )
```

Print a matrix.

Parameters

matrix	The matrix to print
stream	The output stream

Returns

Nothing.

This function is intended just for debugging. It just prints the elements of a matrix, ordered in rows and columns, to the specified stream or FILE pointer. If the specified stream is NULL, it is set to *stdout*. The function used for printing is the standard C `fprintf()`.

References CPL_SIZE_FORMAT.

4.28.2.10 cpl_matrix_duplicate()

```
cpl_matrix* cpl_matrix_duplicate (
            const cpl_matrix * matrix )
```

Make a copy of a matrix.

Parameters

matrix	Matrix to be duplicated.

Returns

Pointer to the new matrix, or NULL in case of error.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a NULL pointer.

A copy of the input matrix is created. To destroy the duplicated matrix the function cpl_matrix_delete() should be used.

References CPL_ERROR_NULL_INPUT, cpl_malloc(), and cpl_matrix_copy().

Referenced by cpl_matrix_get_determinant(), cpl_matrix_invert_create(), cpl_matrix_solve(), cpl_matrix_solve_↩ svd(), and cpl_matrix_solve_svd_threshold().

4.28.2.11 cpl_matrix_erase_columns()

```
cpl_error_code cpl_matrix_erase_columns (
            cpl_matrix * matrix,
            cpl_size start,
            cpl_size count )
```

Delete columns from a matrix.

Parameters

matrix	Pointer to matrix to be modified.
start	First column to delete.
count	Number of columns to delete.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a NULL pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The specified *start* is outside the *matrix* boundaries.
CPL_ERROR_ILLEGAL_INPUT	*count* is not positive.
CPL_ERROR_ILLEGAL_OUTPUT	Attempt to delete all the columns of *matrix*.

A portion of the matrix data is physically removed. The pointer to matrix data may change, therefore pointers previously retrieved by calling cpl_matrix_get_data() should be discarded. The specified segment can

extend beyond the end of the matrix, but the attempt to remove all matrix columns is flagged as an error because zero length matrices are illegal. Columns are counted starting from 0.

References CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_ILLEG↩AL_OUTPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl_realloc().

4.28.2.12 cpl_matrix_erase_rows()

```
cpl_error_code cpl_matrix_erase_rows (
            cpl_matrix * matrix,
            cpl_size start,
            cpl_size count )
```

Delete rows from a matrix.

Parameters

matrix	Pointer to matrix to be modified.
start	First row to delete.
count	Number of rows to delete.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The specified *start* is outside the *matrix* boundaries.
CPL_ERROR_ILLEGAL_INPUT	*count* is not positive.
CPL_ERROR_ILLEGAL_OUTPUT	Attempt to delete all the rows of *matrix*.

A portion of the matrix data is physically removed. The pointer to matrix data may change, therefore pointers previously retrieved by calling `cpl_matrix_get_data()` should be discarded. The specified segment can extend beyond the end of the matrix, but the attempt to remove all matrix rows is flagged as an error because zero length matrices are illegal. Rows are counted starting from 0.

References CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_ILLEG↩AL_OUTPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl_realloc().

4.28.2.13 cpl_matrix_exponential()

```
cpl_error_code cpl_matrix_exponential (
            cpl_matrix * matrix,
            double base )
```

Compute the exponential of matrix elements.

Parameters

matrix	Target matrix.
base	Exponential base.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	The input *base* is not positive.

Each matrix element is replaced by its exponential in the specified base. The base must be positive.

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.28.2.14 cpl_matrix_extract()

```
cpl_matrix* cpl_matrix_extract (
             const cpl_matrix * matrix,
             cpl_size start_row,
             cpl_size start_column,
             cpl_size step_row,
             cpl_size step_column,
             cpl_size nrows,
             cpl_size ncolumns )
```

Extract a submatrix from a matrix.

Parameters

matrix	Pointer to the input matrix.
start_row	Matrix row where to begin extraction.
start_column	Matrix column where to begin extraction.
step_row	Step between extracted rows.
step_column	Step between extracted columns.
nrows	Number of rows to extract.
ncolumns	Number of columns to extract.

Returns

Pointer to the new matrix, or `NULL` in case of error.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a NULL pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The start position is outside the *matrix* boundaries.
CPL_ERROR_ILLEGAL_INPUT	While *nrows* and *ncolumns* are greater than 1, the specified steps are not positive.

The new matrix will include the *nrows* x *ncolumns* values read from the input matrix elements starting from position (*start_row*, *start_column*), in a grid of steps *step_row* and *step_column*. If the extraction parameters exceed the input matrix boundaries, just the overlap is returned, and this matrix would have sizes smaller than *nrows* x *ncolumns*. To destroy the new matrix the function cpl_matrix_delete() should be used.

References CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL↩ _INPUT, and cpl_malloc().

Referenced by cpl_matrix_extract_column(), and cpl_matrix_extract_row().

4.28.2.15 cpl_matrix_extract_column()

```
cpl_matrix* cpl_matrix_extract_column (
              const cpl_matrix * matrix,
              cpl_size column )
```

Copy a matrix column.

Parameters

matrix	Pointer to matrix containing the column.
column	Sequence number of column to copy.

Returns

Pointer to new matrix, or NULL in case of error.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a NULL pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The *column* is outside the *matrix* boundaries.

If a MxN matrix is given in input, the extracted row is a new Mx1 matrix. The column number is counted from 0. To destroy the new matrix the function cpl_matrix_delete() should be used.

References CPL_ERROR_NULL_INPUT, and cpl_matrix_extract().

4.28.2.16 cpl_matrix_extract_diagonal()

```
cpl_matrix* cpl_matrix_extract_diagonal (
            const cpl_matrix * matrix,
            cpl_size diagonal )
```

Extract a matrix diagonal.

Parameters

matrix	Pointer to the matrix containing the diagonal.
diagonal	Sequence number of the diagonal to copy.

Returns

Pointer to the new matrix, or NULL in case of error.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a NULL pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The *diagonal* is outside the *matrix* boundaries (see description below).

If a MxN matrix is given in input, the extracted diagonal is a Mx1 matrix if N >= M, or a 1xN matrix if N < M. The diagonal number is counted from 0, corresponding to the matrix diagonal starting at element (0,0). A square matrix has just one diagonal; if M != N, the number of diagonals in the matrix is |M - N| + 1. To specify a *diagonal* sequence number outside this range would set an error condition, and a NULL pointer would be returned. To destroy the new matrix the function cpl_matrix_delete() should be used.

References CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_NULL_INPUT, and cpl_malloc().

4.28.2.17 cpl_matrix_extract_row()

```
cpl_matrix* cpl_matrix_extract_row (
            const cpl_matrix * matrix,
            cpl_size row )
```

Extract a matrix row.

Parameters

matrix	Pointer to the matrix containing the row.
row	Sequence number of row to copy.

Returns

Pointer to new matrix, or NULL in case of error.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a NULL pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The *row* is outside the *matrix* boundaries.

If a MxN matrix is given in input, the extracted row is a new 1xN matrix. The row number is counted from 0. To destroy the new matrix the function cpl_matrix_delete() should be used.

References CPL_ERROR_NULL_INPUT, and cpl_matrix_extract().

4.28.2.18 cpl_matrix_fill()

```
cpl_error_code cpl_matrix_fill (
            cpl_matrix * matrix,
            double value )
```

Write the same value to all matrix elements.

Parameters

matrix	Pointer to the matrix to access
value	Value to write

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a NULL pointer.

Write the same value to all matrix elements.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.28.2.19 cpl_matrix_fill_column()

```
cpl_error_code cpl_matrix_fill_column (
            cpl_matrix * matrix,
```

```
            double value,
            cpl_size column )
```

Write the same value to a matrix column.

Parameters

matrix	Pointer to the matrix to access
value	Value to write
column	Sequence number of column to overwrite

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The specified *column* is outside the *matrix* boundaries.

Write the same value to a matrix column. Columns are counted starting from 0.

References CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.28.2.20 cpl_matrix_fill_diagonal()

```
cpl_error_code cpl_matrix_fill_diagonal (
            cpl_matrix * matrix,
            double value,
            cpl_size diagonal )
```

Write a given value to all elements of a given matrix diagonal.

Parameters

matrix	Matrix to modify
value	Value to write to diagonal
diagonal	Number of diagonal to overwrite, 0 for main, positive for above main, negative for below main

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The specified *diagonal* is outside the *matrix* boundaries (see description below).

References CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and CPL_SIZE_FORMAT.

4.28.2.21 cpl_matrix_fill_row()

```
cpl_error_code cpl_matrix_fill_row (
            cpl_matrix * matrix,
            double value,
            cpl_size row )
```

Write the same value to a matrix row.

Parameters

matrix	Matrix to access
value	Value to write
row	Sequence number of row to overwrite.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The specified *row* is outside the *matrix* boundaries.

Write the same value to a matrix row. Rows are counted starting from 0.

References CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.28.2.22 cpl_matrix_fill_window()

```
cpl_error_code cpl_matrix_fill_window (
            cpl_matrix * matrix,
            double value,
```

```
            cpl_size row,
            cpl_size col,
            cpl_size nrow,
            cpl_size ncol )
```

Write the same value into a submatrix of a matrix.

Parameters

matrix	Pointer to matrix to be modified.
value	Value to write.
row	Start row of matrix submatrix.
col	Start column of matrix submatrix.
nrow	Number of rows of matrix submatrix.
ncol	Number of columns of matrix submatrix.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a NULL pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The specified start position is outside the *matrix* boundaries.
CPL_ERROR_ILLEGAL_INPUT	*nrow* or *ncol* are not positive.

The specified value is written to *matrix* starting at the indicated row and column; *nrow* and *ncol* can exceed the input matrix boundaries, just the range overlapping the *matrix* is used in that case.

References CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.28.2.23 cpl_matrix_flip_columns()

```
cpl_error_code cpl_matrix_flip_columns (
            cpl_matrix * matrix )
```

Reverse order of columns in matrix.

Parameters

matrix	Pointer to matrix to be reversed.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.

The order of the columns in the matrix is reversed in place.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl_matrix_swap_columns().

4.28.2.24 cpl_matrix_flip_rows()

```
cpl_error_code cpl_matrix_flip_rows (
            cpl_matrix * matrix )
```

Reverse order of rows in matrix.

Parameters

matrix	Pointer to matrix to be reversed.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.

The order of the rows in the matrix is reversed in place.

References CPL_ERROR_NULL_INPUT.

4.28.2.25 cpl_matrix_get()

```
double cpl_matrix_get (
            const cpl_matrix * matrix,
            cpl_size row,
            cpl_size column )
```

Get the value of a matrix element.

Parameters

matrix	Pointer to a matrix.
row	Matrix element row.
column	Matrix element column.

Returns

Value of the specified matrix element, or 0.0 in case of error.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The accessed element is beyond the *matrix* boundaries.

Get the value of a matrix element. The matrix rows and columns are counted from 0,0.

References CPL_ERROR_ACCESS_OUT_OF_RANGE, and CPL_ERROR_NULL_INPUT.

4.28.2.26 cpl_matrix_get_data()

```
double* cpl_matrix_get_data (
            cpl_matrix * matrix )
```

Get the pointer to a matrix data buffer, or `NULL` in case of error.

Parameters

matrix	Input matrix.

Returns

Pointer to the matrix data buffer.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.

A *cpl_matrix* object includes an array of values of type *double*. This function returns a pointer to this internal array, whose first element corresponds to the *cpl_matrix* element 0,0. The internal array contains in sequence all the *cpl_matrix* rows. For instance, in the case of a 3x4 matrix, the array elements
`0 1 2 3 4 5 6 7 8 9 10 11`

would correspond to the following matrix elements:

```
0  1  2  3
4  5  6  7
8  9 10 11
```

Note

Use at your own risk: direct manipulation of matrix data rules out any check performed by the matrix object interface, and may introduce
inconsistencies between the information maintained internally, and the actual matrix data and structure.

References CPL_ERROR_NULL_INPUT.

4.28.2.27 cpl_matrix_get_data_const()

```
const double* cpl_matrix_get_data_const (
            const cpl_matrix * matrix )
```

Get the pointer to a matrix data buffer, or `NULL` in case of error.

Parameters

matrix	Input matrix.

Returns

Pointer to the matrix data buffer.

See also

cpl_matrix_get_data

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_image_filter(), and cpl_wcs_platesol().

4.28.2.28 cpl_matrix_get_determinant()

```
double cpl_matrix_get_determinant (
            const cpl_matrix * matrix )
```

Compute the determinant of a matrix.

Parameters

matrix	Pointer to n by n matrix.

Returns

Matrix determinant. In case of error, 0.0 is returned.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	The input *matrix* is not square.
CPL_ERROR_UNSPECIFIED	The input *matrix* is near-singular with a determinant so close to zero that it cannot be represented by a double.

The input matrix must be a square matrix. In case of a 1x1 matrix, the matrix single element value is returned.

References cpl_array_wrap_int(), cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl↩_errorstate_get(), cpl_malloc(), and cpl_matrix_duplicate().

4.28.2.29 cpl_matrix_get_max()

```
double cpl_matrix_get_max (
             const cpl_matrix * matrix )
```

Find the maximum value of matrix elements.

Parameters

matrix	Pointer to matrix.

Returns

Maximum value. In case of error, 0.0 is returned.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.

The maximum value of all matrix elements is found.

References cpl_matrix_get_maxpos().

4.28.2.30 cpl_matrix_get_maxpos()

```
cpl_error_code cpl_matrix_get_maxpos (
            const cpl_matrix * matrix,
            cpl_size * row,
            cpl_size * column )
```

Find position of the maximum value of matrix elements.

Parameters

matrix	Input matrix.
row	Returned row position of maximum.
column	Returned column position of maximum.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a NULL pointer.

The position of the maximum value of all matrix elements is found. If more than one matrix element have a value corresponding to the maximum, the lowest element row number is returned in *row*. If more than one maximum matrix elements have the same row number, the lowest element column number is returned in *column*.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_matrix_get_max().

4.28.2.31 cpl_matrix_get_mean()

```
double cpl_matrix_get_mean (
            const cpl_matrix * matrix )
```

Find the mean of all matrix elements.

Parameters

matrix	Pointer to matrix.

Returns

Mean. In case of error 0.0 is returned.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.

The mean value of all matrix elements is calculated.

Note

This function works properly only in the case all the elements of the input matrix do not contain garbage (such as `NaN` or infinity).

References CPL_ERROR_NULL_INPUT.

4.28.2.32 cpl_matrix_get_median()

```
double cpl_matrix_get_median (
            const cpl_matrix * matrix )
```

Find the median of matrix elements.

Parameters

matrix	Pointer to matrix.

Returns

Median. In case of error 0.0 is returned.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.

The median value of all matrix elements is calculated.

4.28.2.33 cpl_matrix_get_min()

```
double cpl_matrix_get_min (
            const cpl_matrix * matrix )
```

Find the minimum value of matrix elements.

Parameters

matrix	Pointer to matrix.

Returns

Minimum value. In case of error, 0.0 is returned.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.

The minimum value of all matrix elements is found.

References cpl_matrix_get_minpos().

4.28.2.34 cpl_matrix_get_minpos()

```
cpl_error_code cpl_matrix_get_minpos (
            const cpl_matrix * matrix,
            cpl_size * row,
            cpl_size * column )
```

Find position of minimum value of matrix elements.

Parameters

matrix	Input matrix.
row	Returned row position of minimum.
column	Returned column position of minimum.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.

The position of the minimum value of all matrix elements is found. If more than one matrix element have a value corresponding to the minimum, the lowest element row number is returned in *row*. If more than one minimum matrix elements have the same row number, the lowest element column number is returned in *column*.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_matrix_get_min().

4.28.2.35 cpl_matrix_get_ncol()

```
cpl_size cpl_matrix_get_ncol (
            const cpl_matrix * matrix )
```

Get the number of columns of a matrix.

Parameters

matrix	Pointer to the matrix to examine.

Returns

Number of matrix columns, or zero in case of failure.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.

Determine the number of columns in a matrix.

References CPL_ERROR_NULL_INPUT, and cpl_matrix_get_ncol_().

Referenced by cpl_image_filter(), cpl_test_get_bytes_matrix(), and cpl_wcs_convert().

4.28.2.36 cpl_matrix_get_ncol_()

```
cpl_size cpl_matrix_get_ncol_ (
            const cpl_matrix * matrix )
```

Get the number of columns of a matrix.

Parameters

matrix	Pointer to the matrix to examine.

Returns

Number of matrix columns, or zero in case of failure.

Note

No error checking in this internal function!

See also

cpl_matrix_get_ncol

Referenced by cpl_matrix_decomp_chol(), cpl_matrix_get_ncol(), cpl_matrix_solve_chol(), cpl_polynomial_fit(), cpl_ppm_match_points(), and cpl_vector_fill_polynomial_fit_residual().

4.28.2.37 cpl_matrix_get_nrow()

```
cpl_size cpl_matrix_get_nrow (
            const cpl_matrix * matrix )
```

Get the number of rows of a matrix.

Parameters

matrix	Pointer to the matrix to examine.

Returns

Number of matrix rows, or zero in case of failure.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.

Determine the number of rows in a matrix.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_image_filter(), cpl_test_get_bytes_matrix(), cpl_wcs_convert(), and cpl_wcs_platesol().

4.28.2.38 cpl_matrix_get_stdev()

```
double cpl_matrix_get_stdev (
            const cpl_matrix * matrix )
```

Find the standard deviation of matrix elements.

Parameters

matrix	Pointer to matrix.

Returns

Standard deviation. In case of error, or if a matrix is 1x1, 0.0 is returned.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.

The standard deviation of all matrix elements is calculated.

Note

This function works properly only in the case all the elements of the input matrix do not contain garbage (such as `NaN` or infinity).

References CPL_ERROR_NULL_INPUT.

4.28.2.39 cpl_matrix_invert_create()

```
cpl_matrix* cpl_matrix_invert_create (
             const cpl_matrix * matrix )
```

Find a matrix inverse.

Parameters

matrix	Pointer to matrix to invert.

Returns

Inverse matrix. In case of error a `NULL` is returned.

Errors

CPL_ERROR_NULL_INPUT	Any input is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	*self* is not an n by n matrix.
CPL_ERROR_SINGULAR_MAT↩ RIX	*matrix* cannot be inverted.

The input must be a square matrix. To destroy the new matrix the function `cpl_matrix_delete()` should be used.

Note

When calling `cpl_matrix_invert_create()` with a nearly singular matrix, it is possible to get a result containin NaN values without any error code being set.

References cpl_array_wrap_int(), cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl↩_malloc(), and cpl_matrix_duplicate().

4.28.2.40 cpl_matrix_is_diagonal()

```
int cpl_matrix_is_diagonal (
            const cpl_matrix * matrix,
            double tolerance )
```

Check if a matrix is diagonal.

Parameters

matrix	Pointer to matrix to be checked.
tolerance	Max tolerated rounding to zero.

Returns

1 in case of diagonal matrix, 0 otherwise. If a `NULL` pointer is passed, or the input matrix is not square, -1 is returned.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.

A threshold may be specified to consider zero any number that is close enough to zero. If the specified *tolerance* is negative, a default of `DBL_EPSILON` is used. A zero tolerance may also be specified. No error is set if the input matrix is not square.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_matrix_is_identity().

4.28.2.41 cpl_matrix_is_identity()

```
int cpl_matrix_is_identity (
            const cpl_matrix * matrix,
            double tolerance )
```

Check for identity matrix.

Parameters

matrix	Pointer to matrix to be checked.
tolerance	Max tolerated rounding to zero, or to one.

Returns

1 in case of identity matrix, 0 otherwise. If a NULL pointer is passed, or the input matrix is not square, -1 is returned.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a NULL pointer.

A threshold may be specified to consider zero any number that is close enough to zero, and 1 any number that is close enough to 1. If the specified *tolerance* is negative, a default of DBL_EPSILON is used. A zero tolerance may also be specified. No error is set if the input matrix is not square.

References CPL_ERROR_NULL_INPUT, and cpl_matrix_is_diagonal().

4.28.2.42 cpl_matrix_is_zero()

```
int cpl_matrix_is_zero (
            const cpl_matrix * matrix,
            double tolerance )
```

Check for zero matrix.

Parameters

matrix	Pointer to matrix to be checked.
tolerance	Max tolerated rounding to zero.

Returns

1 in case of zero matrix, 0 otherwise. If a NULL pointer is passed, -1 is returned.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a NULL pointer.

After specific manipulations of a matrix some of its elements may theoretically be expected to be zero. However, because of numerical noise, such elements may turn out not to be exactly zero. In this specific case, if any of

the matrix element is not exactly zero, the matrix would not be classified as a null matrix. A threshold may be specified to consider zero any number that is close enough to zero. If the specified *tolerance* is negative, a default of DBL_EPSILON is used. A zero tolerance may also be specified.

References CPL_ERROR_NULL_INPUT.

4.28.2.43 cpl_matrix_logarithm()

```
cpl_error_code cpl_matrix_logarithm (
            cpl_matrix * matrix,
            double base )
```

Compute the logarithm of matrix elements.

Parameters

matrix	Pointer to matrix.
base	Logarithm base.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a NULL pointer.
CPL_ERROR_ILLEGAL_INPUT	The input *base*, or any *matrix* element, is not positive.

Each matrix element is replaced by its logarithm in the specified base. The base and all matrix elements must be positive. If this is not the case, the matrix would not be modified.

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.28.2.44 cpl_matrix_multiply()

```
cpl_error_code cpl_matrix_multiply (
            cpl_matrix * matrix1,
            const cpl_matrix * matrix2 )
```

Multiply two matrices element by element.

Parameters

matrix1	Pointer to first matrix.
matrix2	Pointer to second matrix.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	An input matrix is a NULL pointer.
CPL_ERROR_INCOMPATIBLE_INPUT	The specified matrices do not have the same size.

Multiply the two matrices element by element. The two matrices must have identical sizes. The result is written to the first matrix.

Note

To obtain the rows-by-columns product between two matrices, cpl_matrix_product_create() should be used instead.

References CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.28.2.45 cpl_matrix_multiply_scalar()

```
cpl_error_code cpl_matrix_multiply_scalar (
            cpl_matrix * matrix,
            double value )
```

Multiply a matrix by a scalar.

Parameters

matrix	Pointer to matrix.
value	Multiplication factor.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a NULL pointer.

Multiply each matrix element by the same factor.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.28.2.46 cpl_matrix_new()

```
cpl_matrix* cpl_matrix_new (
             cpl_size rows,
             cpl_size columns )
```

Create a zero matrix of given size.

Parameters

rows	Number of matrix rows.
columns	Number of matrix columns.

Returns

Pointer to new matrix, or NULL in case of error.

Errors

CPL_ERROR_ILLEGAL_INPUT	*rows* or *columns* are not positive numbers.

This function allocates and initialises to zero a matrix of given size. To destroy this matrix the function cpl_matrix_delete() should be used.

References cpl_calloc(), CPL_ERROR_ILLEGAL_INPUT, and cpl_malloc().

Referenced by cpl_matrix_resize(), cpl_matrix_solve_svd(), cpl_matrix_solve_svd_threshold(), and cpl_wcs_convert().

4.28.2.47 cpl_matrix_power()

```
cpl_error_code cpl_matrix_power (
             cpl_matrix * matrix,
             double exponent )
```

Compute a power of matrix elements.

Parameters

matrix	Pointer to matrix.
exponent	Constant exponent.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	Any *matrix* element is not compatible with the specified *exponent* (see description below).

Each matrix element is replaced by its power to the specified exponent. If the specified exponent is not negative, all matrix elements must be not negative; if the specified exponent is negative, all matrix elements must be positive; otherwise, an error condition is set and the matrix will be left unchanged. If the exponent is exactly 0.5 the (faster) `sqrt()` will be applied instead of `pow()`. If the exponent is zero, then any (non negative) matrix element would be assigned the value 1.0.

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.28.2.48 cpl_matrix_product_create()

```
cpl_matrix* cpl_matrix_product_create (
             const cpl_matrix * matrix1,
             const cpl_matrix * matrix2 )
```

Rows-by-columns product of two matrices.

Parameters

matrix1	Pointer to left side matrix.
matrix2	Pointer to right side matrix.

Returns

Pointer to product matrix, or `NULL` in case of error.

Errors

CPL_ERROR_NULL_INPUT	An input matrix is a `NULL` pointer.
CPL_ERROR_INCOMPATIBLE_INPUT	The number of columns of the first matrix is not equal to the number of rows of the second matrix.

Rows-by-columns product of two matrices. The number of columns of the first matrix must be equal to the number of rows of the second matrix. To destroy the new matrix the function `cpl_matrix_delete()` should be used.

References cpl_ensure, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NULL_INPUT, cpl_malloc(), and cpl_matrix_wrap().

Referenced by cpl_matrix_solve_normal().

4.28.2.49 cpl_matrix_resize()

```
cpl_error_code cpl_matrix_resize (
             cpl_matrix * matrix,
             cpl_size top,
             cpl_size bottom,
             cpl_size left,
             cpl_size right )
```

Reframe a matrix.

Parameters

matrix	Pointer to matrix to be modified.
top	Extra rows on top.
bottom	Extra rows on bottom.
left	Extra columns on left.
right	Extra columns on right.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_OUTPUT	Attempt to shrink *matrix* to zero size (or less).

The input matrix is reframed according to specifications. Extra rows and column on the sides might also be negative, as long as they are compatible with the matrix sizes: the input matrix would be reduced in size accordingly, but an attempt to remove all matrix columns and/or rows is flagged as an error because zero length matrices are illegal. The old matrix elements contained in the new shape are left unchanged, and new matrix elements added by the reshaping are initialised to zero. No reshaping (i.e., all the extra rows set to zero) would not be flagged as an error.

Note

The pointer to the matrix data buffer may change, therefore pointers previously retrieved by calling `cpl_matrix_get_data()` should be discarded.

References CPL_ERROR_ILLEGAL_OUTPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_free(), cpl_matrix_copy(), cpl_matrix_new(), and cpl_matrix_unwrap().

Referenced by cpl_matrix_append(), and cpl_matrix_set_size().

4.28.2.50 cpl_matrix_set_()

```
void cpl_matrix_set_ (
             cpl_matrix * matrix,
             cpl_size row,
             cpl_size column,
             double value )
```

Write a value to a matrix element.

Parameters

matrix	Input matrix.
row	Matrix element row.
column	Matrix element column.
value	Value to write.

Returns

void

Note

No error checking in this internal function!

See also

cpl_matrix_set

4.28.2.51 cpl_matrix_set_size()

```
cpl_error_code cpl_matrix_set_size (
            cpl_matrix * matrix,
            cpl_size rows,
            cpl_size columns )
```

Resize a matrix.

Parameters

matrix	Pointer to matrix to be resized.
rows	New number of rows.
columns	New number of columns.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_OUTPUT	Attempt to shrink *matrix* to zero size (or less).

The input matrix is resized according to specifications. The old matrix elements contained in the resized matrix are

left unchanged, and new matrix elements added by an increase of the matrix number of rows and/or columns are initialised to zero.

Note

The pointer to the matrix data buffer may change, therefore pointers previously retrieved by calling `cpl_matrix_get_data()` should be discarded.

References CPL_ERROR_NONE, and cpl_matrix_resize().

4.28.2.52 cpl_matrix_shift()

```
cpl_error_code cpl_matrix_shift (
            cpl_matrix * matrix,
            cpl_size rshift,
            cpl_size cshift )
```

Shift matrix elements.

Parameters

matrix	Pointer to matrix to be modified.
rshift	Shift in the vertical direction.
cshift	Shift in the horizontal direction.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.

The performed shift operation is cyclical (toroidal), i.e., matrix elements shifted out of one side of the matrix get shifted in from its opposite side. There are no restrictions on the values of the shift. Positive shifts are always in the direction of increasing row/column indexes.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_free(), and cpl_malloc().

4.28.2.53 cpl_matrix_solve()

```
cpl_matrix* cpl_matrix_solve (
            const cpl_matrix * coeff,
            const cpl_matrix * rhs )
```

Solution of a linear system.

Parameters

coeff	A non-singular N by N matrix.
rhs	A matrix containing one or more right-hand-sides.

Note

rhs must have N rows and may contain more than one column, which each represent an independent right-hand-side.

Returns

A newly allocated solution matrix with the size as rhs, or `NULL` on error.

Errors

CPL_ERROR_NULL_INPUT	Any input is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	*coeff* is not a square matrix.
CPL_ERROR_INCOMPATIBLE_INPUT	*coeff* and *rhs* do not have the same number of rows.
CPL_ERROR_SINGULAR_MATRIX	*coeff* is singular (to working precision).

Compute the solution of a system of N equations with N unknowns:

coeff $*$ X = rhs

coeff must be an NxN matrix, and *rhs* a NxM matrix. M greater than 1 means that multiple independent right-hand-sides are solved for. To destroy the solution matrix the function `cpl_matrix_delete()` should be used.

References cpl_array_wrap_int(), cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INCOMPATIBLE↩_INPUT, CPL_ERROR_NULL_INPUT, cpl_malloc(), and cpl_matrix_duplicate().

4.28.2.54 cpl_matrix_solve_chol()

```
cpl_error_code cpl_matrix_solve_chol (
            const cpl_matrix * self,
            cpl_matrix * rhs )
```

Solve a L$*$transpose(L)-system.

Parameters

self	N by N L$*$transpose(L)-matrix from cpl_matrix_decomp_chol()
rhs	M right-hand-sides to be replaced by their solution

Returns

CPL_ERROR_NONE on success, or the relevant CPL error code

See also

cpl_matrix_decomp_chol()

Note

Only the lower triangle of self is accessed

Errors

CPL_ERROR_NULL_INPUT	An input pointer is NULL.
CPL_ERROR_ILLEGAL_INPUT	*self* is not an n by n matrix.
CPL_ERROR_INCOMPATIBLE_INPUT	The specified matrices do not have the same number of rows.
CPL_ERROR_DIVISION_BY_ZERO	The main diagonal of L contains a zero. This error can only occur if the L∗transpose(L)-matrix does not come from a successful call to cpl_matrix_decomp_chol().

References cpl_ensure_code, CPL_ERROR_NULL_INPUT, and cpl_matrix_get_ncol_().

4.28.2.55 cpl_matrix_solve_normal()

```
cpl_matrix* cpl_matrix_solve_normal (
             const cpl_matrix * coeff,
             const cpl_matrix * rhs )
```

Solution of overdetermined linear equations in a least squares sense.

Parameters

coeff	The N by M matrix of coefficients, where N >= M.
rhs	An N by K matrix containing K right-hand-sides.

Note

rhs may contain more than one column, which each represent an independent right-hand-side.

Returns

A newly allocated M by K solution matrix, or NULL on error.

Errors

CPL_ERROR_NULL_INPUT	Any input is a NULL pointer.
CPL_ERROR_INCOMPATIBLE_INPUT	*coeff* and *rhs* do not have the same number of rows.
CPL_ERROR_SINGULAR_MATRIX	The matrix is (near) singular and a solution cannot be computed.

The following linear system of N equations and M unknowns is given:
`coeff * X = rhs`

where *coeff* is the NxM matrix of the coefficients, *X* is the MxK matrix of the unknowns, and *rhs* the NxK matrix containing the K right hand side(s).

The solution to the normal equations is known to be a least-squares solution, i.e. the 2-norm of coeff ∗ X - rhs is minimized by the solution to transpose(coeff) ∗ coeff ∗ X = transpose(coeff) ∗ rhs.

In the case that coeff is square (N is equal to M) it gives a faster and more accurate result to use cpl_matrix_solve().

The solution matrix should be deallocated with the function `cpl_matrix_delete()`.

References cpl_ensure, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NULL_INPUT, cpl_matrix_↩ product_create(), and cpl_matrix_transpose_create().

4.28.2.56 cpl_matrix_solve_svd()

```
cpl_matrix* cpl_matrix_solve_svd (
             const cpl_matrix * coeff,
             const cpl_matrix * rhs )
```

Solve a linear system in a least square sense using an SVD factorization.

Parameters

coeff	An N by M matrix of linear system coefficients, where N >= M
rhs	An N by 1 matrix with the right hand side of the system

Returns

A newly allocated M by 1 matrix containing the solution vector or NULL on error.

Errors

CPL_ERROR_NULL_INPUT	Any input is a NULL pointer.
CPL_ERROR_INCOMPATIBLE_INPUT	*coeff* and *rhs* do not have the same number of rows.
CPL_ILLEGAL_INPUT	The matrix *rhs* has more than one column.

The function solves a linear system of the form Ax = b for the solution vector x, where A is represented by the

argument *coeff* and b by the argument *rhs*. The linear system is solved using the singular value decomposition (SVD) of the coefficient matrix, based on a one-sided Jacobi orthogonalization. When solving the linear system all singular values are taken into account, regardless of their magnitude. This is equivalent to calling cpl_matrix_solve_svd_threshold() with mode set to 2 and a threshold value of 0.

The returned solution matrix should be deallocated using **cpl_matrix_delete()**.

See also

cpl_matrix_solve_svd_threshold()

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NULL_INPUT, cpl_matrix_duplicate(), cpl_matrix_new(), and cpl_vector_new().

4.28.2.57 cpl_matrix_solve_svd_threshold()

```
cpl_matrix* cpl_matrix_solve_svd_threshold (
             const cpl_matrix * coeff,
             const cpl_matrix * rhs,
             int mode,
             double tolerance )
```

Solve a linear system in a least square sense using an SVD factorization discarding singular values below a given threshold.

Parameters

coeff	An N by M matrix of linear system coefficients, where N >= M
rhs	An N by 1 matrix with the right hand side of the system
mode	Cutoff mode selector for small singular values.
tolerance	Factor used to compute the cutoff value if mode is set to 2.

Returns

A newly allocated M by 1 matrix containing the solution vector or NULL on error.

Errors

CPL_ERROR_NULL_INPUT	*coeff* or *rhs* is a NULL pointer.
CPL_ERROR_INCOMPATIBLE_INPUT	*coeff* and *rhs* do not have the same number of rows.
CPL_ILLEGAL_INPUT	The matrix *rhs* has more than one column, or an illegal *mode* or *tolerance* was given.

The function solves a linear system of the form Ax = b for the solution vector x, where A is represented by the argument *coeff* and b by the argument *rhs*. The linear system is solved using the singular value decomposition (SVD) of the coefficient matrix, based on a one-sided Jacobi orthogonalization. When solving the linear system singular values which are less or equal than a given cutoff value are treated as zero.

The argument *mode* is used to select the computation of the cutoff value for small singular values. If *mode* is set to 0 the machine precision (DBL_EPSILON) is used as the cutoff factor. If *mode* is 1, the cutoff factor is computed as $10 \times$ DBL_EPSILON $\times \max(N, M)$, and if *mode* is 2 the argument *tolerance*, a value in the range $[0, 1]$. is used as the cutoff factor. The actual cutoff value, is then given by the cutoff factor times the biggest singular value obtained from the SVD of the input matrix *coeff*.

The returned solution matrix should be deallocated using **cpl_matrix_delete()**.

See also

cpl_matrix_solve_svd()

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR↩
_NULL_INPUT, cpl_matrix_duplicate(), cpl_matrix_new(), and cpl_vector_new().

4.28.2.58 cpl_matrix_sort_columns()

```
cpl_error_code cpl_matrix_sort_columns (
            cpl_matrix * matrix,
            int mode )
```

Sort matrix by columns.

Parameters

matrix	Pointer to matrix to be sorted.
mode	Sorting mode: 0, by absolute value, otherwise by value.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a NULL pointer.

The matrix elements of the top row are used as reference for the column sorting, if there are identical the values of the second row are considered, etc. Columns with the largest values go on the right. If *mode* is equal to zero, the columns are sorted according to their absolute values (zeroes at left).

References cpl_calloc(), CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_free(), and cpl_malloc().

4.28.2.59 cpl_matrix_sort_rows()

```
cpl_error_code cpl_matrix_sort_rows (
            cpl_matrix * matrix,
            int mode )
```

Sort matrix by rows.

Parameters

matrix	Pointer to matrix to be sorted.
mode	Sorting mode: 0, by absolute value, otherwise by value.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a NULL pointer.

The matrix elements of the leftmost column are used as reference for the row sorting, if there are identical the values of the second column are considered, etc. Rows with the greater values go on top. If *mode* is equal to zero, the rows are sorted according to their absolute values (zeroes at bottom).

References cpl_calloc(), CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_free(), and cpl_malloc().

4.28.2.60 cpl_matrix_subtract()

```
cpl_error_code cpl_matrix_subtract (
            cpl_matrix * matrix1,
            const cpl_matrix * matrix2 )
```

Subtract a matrix from another.

Parameters

matrix1	Pointer to first matrix.
matrix2	Pointer to second matrix.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	An input matrix is a NULL pointer.
CPL_ERROR_INCOMPATIBLE_INPUT	The specified matrices do not have the same size.

Subtract the second matrix from the first one element by element. The two matrices must have identical sizes. The result is written to the first matrix.

References CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.28.2.61 cpl_matrix_subtract_scalar()

```
cpl_error_code cpl_matrix_subtract_scalar (
            cpl_matrix * matrix,
            double value )
```

Subtract a scalar to a matrix.

Parameters

matrix	Pointer to matrix.
value	Value to subtract.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a NULL pointer.

Subtract the same value to each matrix element.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.28.2.62 cpl_matrix_swap_columns()

```
cpl_error_code cpl_matrix_swap_columns (
            cpl_matrix * matrix,
            cpl_size column1,
            cpl_size column2 )
```

Swap two matrix columns.

Parameters

matrix	Pointer to matrix to be modified.
column1	One matrix column.
column2	Another matrix column.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	Any of the specified columns is outside the *matrix* boundaries.

The values of two given matrix columns are exchanged. Columns are counted starting from 0. If the same column number is given twice, nothing is done and no error is set.

References CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_matrix_flip_columns().

4.28.2.63 cpl_matrix_swap_rowcolumn()

```
cpl_error_code cpl_matrix_swap_rowcolumn (
            cpl_matrix * matrix,
            cpl_size row )
```

Swap a matrix column with a matrix row.

Parameters

matrix	Pointer to matrix to be modified.
row	Matrix row.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a `NULL` pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The specified *row* is outside the *matrix* boundaries.
CPL_ERROR_ILLEGAL_INPUT	The input *matrix* is not square.

The values of the indicated row are exchanged with the column having the same sequence number. Rows and columns are counted starting from 0.

References CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.28.2.64 cpl_matrix_swap_rows()

```
cpl_error_code cpl_matrix_swap_rows (
            cpl_matrix * matrix,
            cpl_size row1,
            cpl_size row2 )
```

Swap two matrix rows.

Parameters

matrix	Pointer to matrix to be modified.
row1	One matrix row.
row2	Another matrix row.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a NULL pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	Any of the specified rows is outside the *matrix* boundaries.

The values of two given matrix rows are exchanged. Rows are counted starting from 0. If the same row number is given twice, nothing is done and no error is set.

References cpl_ensure_code, CPL_ERROR_ACCESS_OUT_OF_RANGE, and CPL_ERROR_NULL_INPUT.

4.28.2.65 cpl_matrix_threshold_small()

```
cpl_error_code cpl_matrix_threshold_small (
            cpl_matrix * matrix,
            double tolerance )
```

Rounding to zero very small numbers in matrix.

Parameters

matrix	Pointer to matrix to be chopped.
tolerance	Max tolerated rounding to zero.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a NULL pointer.

After specific manipulations of a matrix some of its elements may theoretically be expected to be zero (for instance, as a result of multiplying a matrix by its inverse). However, because of numerical noise, such elements may turn out not to be exactly zero. With this function any very small number in the matrix is turned to exactly zero. If the *tolerance* is zero or negative, a default threshold of DBL_EPSILON is used.

References CPL_ERROR_NULL_INPUT.

4.28.2.66 cpl_matrix_transpose_create()

```
cpl_matrix* cpl_matrix_transpose_create (
            const cpl_matrix * matrix )
```

Create transposed matrix.

Parameters

matrix	Pointer to matrix to be transposed.

Returns

Pointer to transposed matrix. If a NULL pointer is passed, a NULL pointer is returned.

Errors

CPL_ERROR_NULL_INPUT	The input *matrix* is a NULL pointer.

The transposed of the input matrix is created. To destroy the new matrix the function cpl_matrix_delete() should be used.

References CPL_ERROR_NULL_INPUT, cpl_malloc(), and cpl_matrix_wrap().

Referenced by cpl_matrix_solve_normal().

4.28.2.67 cpl_matrix_unwrap()

```
void* cpl_matrix_unwrap (
            cpl_matrix * matrix )
```

Delete a matrix, but not its data buffer.

Parameters

matrix	Pointer to a matrix to be deleted.

Returns

Pointer to the internal data buffer.

This function deallocates all the memory associated to a matrix, with the exception of its data buffer. This type of destructor should be used on matrices created with the `cpl_matrix_wrap()` constructor, if the data buffer specified then was not allocated using the functions of the `cpl_memory` module. In such a case, the data buffer should be deallocated separately. See the documentation of the function `cpl_matrix_wrap()`. If *matrix* is `NULL`, nothing is done, and a `NULL` pointer is returned.

References cpl_free().

Referenced by cpl_matrix_resize(), and cpl_vector_fit_gaussian().

4.28.2.68 cpl_matrix_wrap()

```
cpl_matrix* cpl_matrix_wrap (
            cpl_size rows,
            cpl_size columns,
            double * data )
```

Create a new matrix from existing data.

Parameters

data	Existing data buffer.
rows	Number of matrix rows.
columns	Number of matrix columns.

Returns

Pointer to new matrix, or `NULL` in case of error.

Errors

CPL_ERROR_NULL_INPUT	The input *data* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	*rows* or *columns* are not positive numbers.

This function creates a new matrix that will encapsulate the given data. At any error condition, a `NULL` pointer would be returned. Note that the size of the input data array is not checked in any way, and it is expected to match the specified matrix sizes. The input array is supposed to contain in sequence all the new *cpl_matrix* rows. For instance, in the case of a 3x4 matrix, the input array should contain 12 elements

```
0 1 2 3 4 5 6 7 8 9 10 11
```

that would correspond to the matrix elements

```
0  1  2  3
4  5  6  7
8  9 10 11
```

The data buffer is not copied, so it should not be deallocated while the matrix is still in use: the function `cpl_matrix_delete()` would take care of deallocating it. To avoid problems with the memory managment, the specified data buffer should be allocated using the functions of the `cpl_memory` module, and also statically allocated data should be strictly avoided. If this were not the case, this matrix should not be destroyed using `cpl_matrix_delete()`, but `cpl_matrix_unwrap()` should be used instead; moreover, functions implying memory handling (as `cpl_matrix_set_size()`, or `cpl_matrix_delete_row()`) should not be used.

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, and cpl_malloc().

Referenced by cpl_fit_image_gaussian(), cpl_matrix_product_create(), cpl_matrix_transpose_create(), and cpl_↩ vector_fit_gaussian().

4.29 Memory Management Utilities

Functions

- void * cpl_calloc (size_t natoms, size_t nbytes)
  Allocate memory for natoms elements of size size.
- void cpl_free (void *memblk)
  Memory block deallocation.
- void * cpl_malloc (size_t nbytes)
  Allocate nbytes bytes.
- void cpl_memory_dump (void)
  Display the memory status.
- int cpl_memory_is_empty (void)
  Tell if there is some memory allocated.
- void * cpl_realloc (void *memblk, size_t nbytes)
  Change the size of a memory block.
- char * cpl_sprintf (const char *format,...)
  Create a string and fill it in an sprintf()-like manner.
- char * cpl_strdup (const char *string)
  Duplicate a string.
- char * cpl_vsprintf (const char *format, va_list arglist)
  Create a string and fill it in an vsprintf()-like manner.

4.29.1 Detailed Description

This module provides the CPL memory management utilities.

4.29.2 Function Documentation

4.29.2.1 cpl_calloc()

```
void* cpl_calloc (
            size_t natoms,
            size_t nbytes )
```

Allocate memory for *natoms* elements of size *size*.

Parameters

natoms	Number of atomic elements.
nbytes	Element size in bytes.

Returns

Pointer to the allocated memory block.

The function allocates memory suitable for storage of *natoms* elements of size *nbytes* bytes. The allocated memory is cleared, i.e. the value 0 is written to each single byte.

Note

If the memory subsytem has not been initialised before calling this function, the program execution is stopped printing a message to the error channel showing the current code position.

Referenced by cpl_apertures_sort_by_flux(), cpl_apertures_sort_by_max(), cpl_apertures_sort_by_npix(), cpl_array_cast(), cpl_array_dump(), cpl_array_new(), cpl_array_wrap_cplsize(), cpl_array_wrap_double(), cpl_array_wrap_double_complex(), cpl_array_wrap_float(), cpl_array_wrap_float_complex(), cpl_array_wrap_int(), cpl_array_wrap_long(), cpl_array_wrap_long_long(), cpl_array_wrap_string(), cpl_image_fft(), cpl_imagelist_new(), cpl_mask_new(), cpl_matrix_new(), cpl_matrix_sort_columns(), cpl_matrix_sort_rows(), cpl_polynomial_extract(), cpl_ppm_match_positions(), cpl_table_and_selected_window(), cpl_table_dump(), cpl_table_new(), cpl_table_or_selected_window(), cpl_table_select_row(), and cpl_wlcalib_slitmodel_new().

4.29.2.2 cpl_free()

```
void cpl_free (
            void * memblk )
```

Memory block deallocation.

Returns

Nothing.

Deallocates a memory block previously allocated by any of the CPL allocator functions.

See also

cpl_malloc(), cpl_calloc()

Referenced by cpl_apertures_delete(), cpl_apertures_sort_by_flux(), cpl_apertures_sort_by_max(), cpl_apertures_sort_by_npix(), cpl_array_cast(), cpl_array_delete(), cpl_array_dump(), cpl_array_unwrap(), cpl_bivector_delete(), cpl_bivector_unwrap_vectors(), cpl_fit_image_gaussian(), cpl_image_cast(), cpl_image_delete(), cpl_image_divide_create(), cpl_image_fft(), cpl_image_labelise_mask_create(), cpl_image_move(), cpl_image_unwrap(), cpl_imagelist_cast(), cpl_imagelist_unwrap(), cpl_mask_delete(), cpl_mask_move(), cpl_mask_unwrap(), cpl_matrix_delete(), cpl_matrix_resize(), cpl_matrix_shift(), cpl_matrix_sort_columns(), cpl_matrix_sort_rows(), cpl_matrix_unwrap(), cpl_polynomial_extract(), cpl_ppm_match_positions(), cpl_stats_delete(), cpl_table_and_selected_window(), cpl_table_delete(), cpl_table_or_selected_window(), cpl_table_select_all(), cpl_table_select_row(), cpl_table_unselect_all(), cpl_table_unselect_row(), cpl_vector_cycle(), cpl_vector_delete(), cpl_vector_get_median_const(), cpl_vector_unwrap(), cpl_wcs_delete(), and cpl_wlcalib_slitmodel_delete().

4.29.2.3 cpl_malloc()

```
void* cpl_malloc (
            size_t nbytes )
```

Allocate *nbytes* bytes.

Parameters

nbytes	Number of bytes.

Returns

Pointer to the allocated memory block.

The function allocates *nbytes* bytes of memory. The allocated memory is not cleared.

Note

If the memory subsytem has not been initialised before calling this function, the program execution is stopped printing a message to the error channel showing the current code position.

Referenced by cpl_apertures_sort_by_flux(), cpl_apertures_sort_by_max(), cpl_apertures_sort_by_npix(), cpl_array_dump(), cpl_bivector_duplicate(), cpl_bivector_new(), cpl_bivector_wrap_vectors(), cpl_fit_image_gaussian(), cpl_fit_imagelist_polynomial_window(), cpl_image_cast(), cpl_image_divide_create(), cpl_image_duplicate(), cpl_image_labelise_mask_create(), cpl_image_move(), cpl_image_new_from_mask(), cpl_imagelist_cast(), cpl_mask_collapse_create(), cpl_mask_duplicate(), cpl_mask_move(), cpl_mask_wrap(), cpl_matrix_duplicate(), cpl_matrix_extract(), cpl_matrix_extract_diagonal(), cpl_matrix_get_determinant(), cpl_matrix_invert_create(), cpl_matrix_new(), cpl_matrix_product_create(), cpl_matrix_shift(), cpl_matrix_solve(), cpl_matrix_sort_columns(), cpl_matrix_sort_rows(), cpl_matrix_transpose_create(), cpl_matrix_wrap(), cpl_plot_bivectors(), cpl_plot_columns(), cpl_plot_vectors(), cpl_ppm_match_positions(), cpl_stats_new_from_image_window(), cpl_table_dump(), cpl_table_duplicate(), cpl_table_sort(), cpl_table_unselect_row(), cpl_vector_cycle(), cpl_vector_get_median_const(), cpl_vector_new(), cpl_vector_wrap(), and cpl_wcs_new_from_propertylist().

4.29.2.4 cpl_memory_dump()

```
void cpl_memory_dump (
             void  )
```

Display the memory status.

4.29.2.5 cpl_memory_is_empty()

```
int cpl_memory_is_empty (
             void  )
```

Tell if there is some memory allocated.

Returns

-1 if the model is off, 1 if it is empty, 0 otherwise

4.29.2.6 cpl_realloc()

```
void* cpl_realloc (
             void * memblk,
             size_t nbytes )
```

Change the size of a memory block.

Parameters

memblk	Pointer to the memory to re-allocate.
nbytes	New memory block size in bytes.

Returns

Pointer to the allocated memory block.

The function changes the size of an already allocated memory block *memblk* to the new size *nbytes* bytes. The contents is unchanged to the minimum of old and new size; newly allocated memory is not initialized. If *memblk* is `NULL` the call to **cpl_realloc()** is equivalent to **cpl_malloc()**, and if *nbytes* is 0 the call is equivalent to **cpl_free()**. Unless *memblk* is `NULL`, it must have been returned by a previous call to **cpl_malloc()**, **cpl_calloc()**, or **cpl_realloc()**.

Note

- The returned memory block returned on successfull allocation may not be the same as the one pointed to by *memblk*. Existing references pointing to locations within the original memory block might be invalidated!
- If the memory subsytem has not been initialised before calling this function, the program execution is stopped printing a message to the error channel showing the current code position.

See also

cpl_malloc(), cpl_calloc()

Referenced by cpl_imagelist_set(), cpl_matrix_erase_columns(), cpl_matrix_erase_rows(), and cpl_vector_set_↩size().

4.29.2.7 cpl_sprintf()

```
char* cpl_sprintf (
            const char * format,
             ...  )
```

Create a string and fill it in an sprintf()-like manner.

Parameters

format	The format string
...	Variable argument list for format

Returns

The created string or NULL on error

Note

The created string must be deallocated with cpl_free()

See also

cpl_vsprintf()

The allocated memory is exactly what is needed to hold the string.

Errors

CPL_ERROR_NULL_INPUT	The format string is `NULL`.
CPL_ERROR_ILLEGAL_INPUT	The format string has an invalid format.

Example of usage (note how strlen() is not needed to get the string length):

```
int error;
int length;
char * cp_cmd = cpl_sprintf("cp %s %s/%s%n", long_file, new_dir,
                                              new_file, &length);
assert( cp_cmd != NULL);
assert( length == (int)strlen(cp_cmd));
error = system(cp_cmd);
assert(!error);
cpl_free(cp_cmd);
```

References cpl_ensure, cpl_error_get_code(), CPL_ERROR_NULL_INPUT, and cpl_vsprintf().

Referenced by cpl_array_dump(), cpl_msg_progress(), cpl_plot_bivectors(), cpl_plot_columns(), cpl_plot_vectors(), and cpl_plugin_get_version_string().

4.29.2.8 cpl_strdup()

```
char* cpl_strdup (
            const char * string )
```

Duplicate a string.

Parameters

string	String to be duplicated.

Returns

Newly allocated copy of the original string.

Duplicates the input string *string*. The newly allocated copy returned to the caller can be deallocated using **cpl_free()**.

See also

cpl_free()

4.29.2.9 cpl_vsprintf()

```
char* cpl_vsprintf (
            const char * format,
            va_list arglist )
```

Create a string and fill it in an vsprintf()-like manner.

Parameters

format	The format string
arglist	The argument list for the format

Returns

The created string or NULL on error

Note

The created string must be deallocated with cpl_free()

See also

vsprintf()

The allocated memory is exactly what is needed to hold the string.

Errors

CPL_ERROR_NULL_INPUT	The format string is `NULL`.
CPL_ERROR_ILLEGAL_INPUT	The format string has an invalid format.

Referenced by cpl_sprintf().

4.30 Messages

Functions

- void cpl_msg_debug (const char ∗component, const char ∗format,...)

  Display a debug message.
- void cpl_msg_error (const char ∗component, const char ∗format,...)

  Display an error message.
- const char ∗ cpl_msg_get_domain (void)

  Get the domain name.
- cpl_msg_severity cpl_msg_get_level (void)

  Get current terminal verbosity level.
- cpl_msg_severity cpl_msg_get_log_level (void)

  Get current log verbosity level.
- const char ∗ cpl_msg_get_log_name (void)

  Get the log file name.
- void cpl_msg_indent (int level)

  Set the indentation level.
- void cpl_msg_indent_less (void)

  Decrease the message indentation by one indentation step.
- void cpl_msg_indent_more (void)

  Increase the message indentation by one indentation step.
- void cpl_msg_info (const char ∗component, const char ∗format,...)

  Display an information message.
- void cpl_msg_info_overwritable (const char ∗component, const char ∗format,...)

  Display an overwritable information message.
- cpl_error_code cpl_msg_init (void)

  Initialise the messaging system.
- void cpl_msg_progress (const char ∗component, int i, int iter, const char ∗format,...)

  Display a progress message predicting the time required in a loop.
- void cpl_msg_set_component_off (void)

  Disable the component tag in output messages.
- void cpl_msg_set_component_on (void)

  Attach the component tag to output messages.
- void cpl_msg_set_domain (const char ∗name)

  Set the domain name.
- void cpl_msg_set_domain_off (void)

  Disable the domain tag in output messages.
- void cpl_msg_set_domain_on (void)

  Attach the domain tag to output messages.
- void cpl_msg_set_indentation (int step)

  Set the indentation step.
- void cpl_msg_set_level (cpl_msg_severity verbosity)

  Set verbosity level of output to terminal.
- void cpl_msg_set_level_from_env (void)

  Set verbosity level of terminal output using an environment variable.
- cpl_error_code cpl_msg_set_log_level (cpl_msg_severity verbosity)

  Open and initialise a log file.
- cpl_error_code cpl_msg_set_log_name (const char ∗name)

  Set the log file name.

- void cpl_msg_set_threadid_off (void)

  Disable the thread id tag to output messages.
- void cpl_msg_set_threadid_on (void)

  Attach a thread id tag to output messages.
- void cpl_msg_set_time_off (void)

  Disable the time tag in output messages.
- void cpl_msg_set_time_on (void)

  Attach a time tag to output messages.
- void cpl_msg_set_width (int width)

  Set the maximum width of the displayed text.
- void cpl_msg_stop (void)

  Turn the messaging system off.
- cpl_error_code cpl_msg_stop_log (void)

  Close the current log file.
- void cpl_msg_warning (const char *component, const char *format,...)

  Display a warning message.

4.30.1 Detailed Description

This module provides functions to display and log messages. The following operations are supported:

```
- Enable messages output to terminal or to log file.
- Optionally adding informative tags to messages.
- Setting width for message line wrapping.
- Control the message indentation level.
- Filtering messages according to their severity level.
```

To activate and deactivate the messaging system, the functions `cpl_msg_init()` and `cpl_msg_stop()` need to be used. However, since they are called anyway by the functions `cpl_init()` and `cpl_end()`, there is generally no need to call them explicitly, and starting from CPL 2.1 they are deprecated. These functions would typically be called at the beginning and at the end of a program. An attempt to use an uninitialised messaging system would generate a warning message. More functions may also be used to configure the messaging system, and here is an example of a possible initialisation:

```
...
cpl_msg_set_time_on();
cpl_msg_set_component_on();
cpl_msg_set_domain("Source detection");
cpl_msg_set_domain_on();
cpl_msg_set_level(CPL_MSG_ERROR);
cpl_msg_set_log_level(CPL_MSG_DEBUG);
...
```

The functions of these kind, are meant to configure the messaging system, defining its "style", once and for all. For this reason such functions are not supposed to be called from threads. Three different tags may be attached to any message: *time*, *domain*, and *component*. The *time* tag is the time of printing of the message, and can optionally be turned on or off with the functions `cpl_msg_set_time_on()` and `cpl_msg_set_time_off()`. The *domain* tag is an identifier of the main program running (typically, a pipeline recipe), and can be optionally turned on or off with the functions `cpl_msg_set_domain_on()` and `cpl_msg_set_domain_off()`. Finally, the *component* tag is used to identify a component of the program running (typically, a function), and can be optionally turned on or off with the functions `cpl_msg_set_component_on()` and `cpl_msg_set_component_off()`. As a default, none of the above tags are attached to messages sent to terminal. However, all tags are always used in messages sent to the log file. A further tag, the *severity* tag, can never be turned off. This tag depends on the function used to print a message, that can be either `cpl_msg_debug()`, `cpl_msg_info()`, `cpl_msg_warning()`, or `cpl_msg_error()`. The *time* and *severity* tags are all prepended to any message, and are not affected by the message indentation controlled by the functions `cpl_msg_indent()`, `cpl_msg_indent_more()`, `cpl_msg_indent_less()`, and `cpl_msg_set_indentation()`.

Synopsis:

```
#include <cpl_msg.h>
```

4.30.2 Function Documentation

4.30.2.1 cpl_msg_debug()

```
void cpl_msg_debug (
             const char * component,
             const char * format,
              ...  )
```

Display a debug message.

Returns

Nothing.

Parameters

component	Name of the function generating the message.
format	Format string.
...	Variable argument list associated to the format string.

See the description of the function `cpl_msg_error()`.

Referenced by cpl_frameset_labelise(), cpl_msg_progress(), and cpl_ppm_match_points().

4.30.2.2 cpl_msg_error()

```
void cpl_msg_error (
             const char * component,
             const char * format,
              ...  )
```

Display an error message.

Returns

Nothing.

Parameters

component	Name of the component generating the message.
format	Format string.
...	Variable argument list associated to the format string.

The *format* string should follow the usual `printf()` convention. Newline characters shouldn't generally be used, as the message would be split automatically according to the width specified with **cpl_msg_set_width()**. Inserting a newline character would enforce breaking a line of text even before the current row is filled. Newline characters at the end of the *format* string are not required. If *component* is a `NULL` pointer, it would be set to the string "<empty field>". If *format* is a `NULL` pointer, the message "<empty message>" would be printed.

4.30.2.3 cpl_msg_get_domain()

```
const char* cpl_msg_get_domain (
            void  )
```

Get the *domain* name.

Returns

Pointer to "domain" string.

This routine returns always the same pointer to the statically allocated buffer containing the "domain" string.

4.30.2.4 cpl_msg_get_level()

```
cpl_msg_severity cpl_msg_get_level (
            void  )
```

Get current terminal verbosity level.

Returns

Current verbosity level.

Get current verbosity level set for the output to terminal.

4.30.2.5 cpl_msg_get_log_level()

```
cpl_msg_severity cpl_msg_get_log_level (
            void  )
```

Get current log verbosity level.

Returns

Current verbosity level.

Get current verbosity level set for the output to the log file.

4.30.2.6 cpl_msg_get_log_name()

```
const char* cpl_msg_get_log_name (
            void  )
```

Get the log file name.

Returns

Logfile name

The name of the log file is returned.

4.30.2.7 cpl_msg_indent()

```
void cpl_msg_indent (
            int level )
```

Set the indentation level.

Returns

Nothing.

Parameters

level	Indentation level.

Any message printed after a call to this function would be indented by a number of characters equal to the *level* multiplied by the indentation step specified with the function cpl_msg_set_indentation(). Specifying a negative indentation level would set the indentation level to zero.

4.30.2.8 cpl_msg_indent_less()

```
void cpl_msg_indent_less (
            void  )
```

Decrease the message indentation by one indentation step.

Returns

Nothing.

Calling this function is equivalent to decrease the indentation level by 1. If the indentation level is already 0, it is not decreased.

4.30.2.9 cpl_msg_indent_more()

```
void cpl_msg_indent_more (
            void  )
```

Increase the message indentation by one indentation step.

Returns

Nothing.

Calling this function is equivalent to increase the indentation level by 1. See function cpl_msg_indent().

Referenced by cpl_ppm_match_points().

4.30.2.10 cpl_msg_info()

```
void cpl_msg_info (
            const char * component,
            const char * format,
             ...  )
```

Display an information message.

Returns

Nothing.

Parameters

component	Name of the function generating the message.
format	Format string.
...	Variable argument list associated to the format string.

See the description of the function cpl_msg_error().

Referenced by cpl_msg_info_overwritable().

4.30.2.11 cpl_msg_info_overwritable()

```
void cpl_msg_info_overwritable (
            const char * component,
            const char * format,
             ...  )
```

Display an overwritable information message.

Returns

Nothing.

Parameters

component	Name of the function generating the message.
format	Format string.
...	Variable argument list associated to the format string.

See the description of the function `cpl_msg_error()`. The severity of the message issued by `cpl_msg_info_overwritable()` is the same as the severity of a message issued using `cpl_msg_info()`. The only difference with the `cpl_msg_info()` function is that the printed message would be overwritten by a new message issued using again cpl_msg_info_overwritable(), if no other meassages were issued with other messaging functions in between. This function would be used typically in loops, as in the following example:

```
iter = 1000;
for (i = 0; i < iter; i++) {
    cpl_msg_info_overwritable(cpl_func,
                  "Median computation... %d out of %d", i, iter);
    <median computation would take place here>
}
```

Note

In the current implementation, an overwritable message is obtained by not adding the new-line character ('\n') at the end of the message (contrary to what `cpl_msg_info()` does).

References cpl_msg_info().

4.30.2.12 cpl_msg_init()

```
cpl_error_code cpl_msg_init (
             void  )
```

Initialise the messaging system.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_FILE_ALREADY_OPEN	The messaging system was already initialised.
CPL_ERROR_DUPLICATING_STREAM	`stdout` and `stderr` streams cannot be duplicated.
CPL_ERROR_ASSIGNING_STREAM	A stream cannot be associated with a file descriptor.

This function needs to be called to activate the messaging system, typically at the beginning of a program. An attempt to use any messaging function before turning the system on would generate a warning message. The

messaging system needs to be deactivated by calling the function cpl_msg_stop(). However, since these functions are called anyway by the functions cpl_init() and cpl_end(), there is generally no need to call them explicitly, and starting from CPL 2.1 they are deprecated.

When cpl_msg_init() is called, the *stdout* and *stderr* streams are duplicated for greater flexibility of the system. The terminal width is determined (if possible), and the resized window signal handler is deployed to monitor possible changes of the terminal window width. If the width of the output device cannot be determined, lines of text are not splitted when written to output. If line splitting is not wanted, the function cpl_msg_set_width() should be called specifying a non positive width.

References CPL_ERROR_ASSIGNING_STREAM, CPL_ERROR_DUPLICATING_STREAM, and CPL_ERROR↩
_FILE_ALREADY_OPEN.

4.30.2.13 cpl_msg_progress()

```
void cpl_msg_progress (
             const char * component,
             int i,
             int iter,
             const char * format,
              ...  )
```

Display a progress message predicting the time required in a loop.

Returns

Nothing.

Parameters

component	Name of the function generating the message.
i	Iteration, starting with 0 and less than iter.
iter	Total number of iterations.
format	Format string.
...	Variable argument list associated to the format string.

See also

cpl_msg_info()

Deprecated Use standard calls such as cpl_msg_info() instead.

References cpl_msg_debug(), and cpl_sprintf().

4.30.2.14 cpl_msg_set_component_off()

```
void cpl_msg_set_component_off (
            void  )
```

Disable the *component* tag in output messages.

Returns

Nothing.

The *component* tag is turned off in messages written to terminal, unless the verbosity level is set to `CPL_MSG_DEBUG`. The *component* tag cannot be turned off in messages written to the log file.

Note

This function is meant to configure once and for all the behaviour and the "style" of the messaging system, and it is not supposed to be called in threads.

4.30.2.15 cpl_msg_set_component_on()

```
void cpl_msg_set_component_on (
            void  )
```

Attach the *component* tag to output messages.

Returns

Nothing.

As a default, the *component* tag is attached just to messages written to the log file. This function must be called to display the *component* tag also in messages written to terminal. To turn the *component* tag off the function `cpl_msg_set_component_off()` should be called. However, the *component* tag is always shown when the verbosity level is set to `CPL_MSG_DEBUG`.

Note

This function is meant to configure once and for all the behaviour and the "style" of the messaging system, and it is not supposed to be called in threads.

4.30.2.16 cpl_msg_set_domain()

```
void cpl_msg_set_domain (
            const char * name )
```

Set the *domain* name.

Parameters

name	Any task identifier, typically a pipeline recipe name.

Returns

Nothing.

This routine should be called at a pipeline recipe start, and before a possible call to the function cpl_msg_set_log_level() or the proper task identifier would not appear in the log file header. The *domain* tag is attached to messages sent to terminal only if the function `cpl_msg_set_domain_on()` is called. If the *domain* tag is on and no domain tag was specified, the string "Undefined domain" (or something analogous) would be attached to all messages. To turn the *domain* tag off the function `cpl_msg_set_domain_off()` should be called. If *name* is a `NULL` pointer, nothing is done, and no error is set.

Note

This function is meant to configure once and for all the behaviour and the "style" of the messaging system, and it is not supposed to be called in threads.

4.30.2.17 cpl_msg_set_domain_off()

```
void cpl_msg_set_domain_off (
             void  )
```

Disable the *domain* tag in output messages.

Returns

Nothing.

The *domain* tag is turned off in messages written to terminal.

Note

This function is meant to configure once and for all the behaviour and the "style" of the messaging system, and it is not supposed to be called in threads.

4.30.2.18 cpl_msg_set_domain_on()

```
void cpl_msg_set_domain_on (
             void  )
```

Attach the *domain* tag to output messages.

Returns

Nothing.

As a default, the *domain* tag is just written to the header of the log file. This function must be called to attach the *domain* tag to all messages written to terminal. If the *domain* tag is on and no domain tag was specified, the string "Undefined domain" (or something analogous) would be attached to all messages. To turn the *domain* tag off the function `cpl_msg_set_domain_off()` must be called.

Note

This function is meant to configure once and for all the behaviour and the "style" of the messaging system, and it is not supposed to be called in threads.

4.30.2.19 cpl_msg_set_indentation()

```
void cpl_msg_set_indentation (
             int step )
```

Set the indentation step.

Parameters

step	Indentation step.

Returns

Nothing.

To maintain a consistent message display style, this routine should be called at most once, and just at program start. A message line might be moved leftward or rightward by a number of characters that is a multiple of the specified indentation step. Setting the indentation step to zero or to a negative number would eliminate messages indentation. If this function is not called, the indentation step defaults to 2.

Note

This function is meant to configure once and for all the behaviour and the "style" of the messaging system, and it is not supposed to be called in threads.

4.30.2.20 cpl_msg_set_level()

```
void cpl_msg_set_level (
            cpl_msg_severity verbosity )
```

Set verbosity level of output to terminal.

Parameters

verbosity	Verbosity level.

Returns

Nothing.

The *verbosity* specifies the lowest severity level that a message should have for being displayed to terminal. If this function is not called, the verbosity level defaults to `CPL_MSG_INFO`.

Note

This function is not supposed to be called in threads.

Referenced by cpl_msg_set_level_from_env().

4.30.2.21 cpl_msg_set_level_from_env()

```
void cpl_msg_set_level_from_env (
            void  )
```

Set verbosity level of terminal output using an environment variable.

Returns

void

See also

cpl_msg_set_level

Note

This function can be used for run-time control of the verbosity level of unit test modules.

The CPL verbosity level of output to terminal is set according to the environment variable CPL_MSG_LEVEL↩ : debug: CPL_MSG_DEBUG info: CPL_MSG_INFO warning: CPL_MSG_WARNING error: CPL_MSG_ERROR off: CPL_MSG_OFF

Any other value (including NULL) will cause the function to do nothing.

References cpl_msg_set_level().

4.30.2.22 cpl_msg_set_log_level()

```
cpl_error_code cpl_msg_set_log_level (
            cpl_msg_severity verbosity )
```

Open and initialise a log file.

Parameters

verbosity	Verbosity level.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_FILE_ALREADY_OPEN	A log file was already started.
CPL_ERROR_FILE_NOT_CREATED	Log file cannot be created.

If the specified *verbosity* level is different from CPL_MSG_OFF, a log file is created and initialised with a header containing the start logging time, the *domain* identifier set by the function cpl_msg_set_domain(), and the chosen *verbosity* level. The *verbosity* specifies the lowest severity level that a message should have to be written to the log file. The name of the created log file may be previously set with the function cpl_msg_set_log_name(), otherwise it is left to a default ".logfile". The log file name can be obtained by calling the function cpl_msg_get_log_name(). Typically this function is called at the beginning of a program. Calling it while a log file is already open has no effect, but it will return an error code.

Note

This function is meant to configure once and for all the behaviour and the "style" of the messaging system, and it is not supposed to be called in threads.

References CPL_ERROR_FILE_ALREADY_OPEN.

4.30.2.23 cpl_msg_set_log_name()

```
cpl_error_code cpl_msg_set_log_name (
            const char * name )
```

Set the log file name.

Parameters

name	Name of log file.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	The specified *name* is a `NULL` pointer.
CPL_ERROR_FILE_ALREADY_OPEN	A log file was already started.
CPL_ERROR_ILLEGAL_INPUT	The specified *name* is longer than `CPL_MAX_LOGFILE_NAME` characters (including the terminating '\0').

This function is used to set the log file name, and can only be called before the log is opened by `cpl_msg_set_log_level()`. If this function is not called, or the specified *name* is longer than `CPL_MAX_LOGFILE_NAME` characters, the log file name is left to its default, ".logfile".

Note

This function is meant to configure once and for all the behaviour and the "style" of the messaging system, and it is not supposed to be called in threads.

References CPL_ERROR_FILE_ALREADY_OPEN, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.30.2.24 cpl_msg_set_threadid_off()

```
void cpl_msg_set_threadid_off (
             void  )
```

Disable the *thread* id tag to output messages.

Returns

Nothing.

The *thread* id tag is turned off in messages written to terminal. The *thread* id tag cannot be turned off in messages written to the log file.

Note

This function is meant to configure once and for all the behaviour and the "style" of the messaging system, and it is not supposed to be called in threads.

4.30.2.25 cpl_msg_set_threadid_on()

```
void cpl_msg_set_threadid_on (
             void  )
```

Attach a *thread* id tag to output messages.

Returns

Nothing.

As a default, *thread* ids tags are attached just to messages written to the log file. This function must be called to display the *thread* id tag also in messages written to terminal. To turn the *thread* id tag off the function `cpl_msg_set_threadid_off()` should be called.

Note

This function is meant to configure once and for all the behaviour and the "style" of the messaging system, and it is not supposed to be called in threads.

4.30.2.26 cpl_msg_set_time_off()

```
void cpl_msg_set_time_off (
             void  )
```

Disable the *time* tag in output messages.

Returns

Nothing.

The *time* tag is turned off in messages written to terminal. The *time* tag cannot be turned off in messages written to the log file.

Note

This function is meant to configure once and for all the behaviour and the "style" of the messaging system, and it is not supposed to be called in threads.

4.30.2.27 cpl_msg_set_time_on()

```
void cpl_msg_set_time_on (
             void  )
```

Attach a *time* tag to output messages.

Returns

Nothing.

As a default, *time* tags are attached just to messages written to the log file. This function must be called to display the *time* tag also in messages written to terminal. To turn the *time* tag off the function `cpl_msg_set_time_off()` should be called.

Note

This function is meant to configure once and for all the behaviour and the "style" of the messaging system, and it is not supposed to be called in threads.

4.30.2.28 cpl_msg_set_width()

```
void cpl_msg_set_width (
             int width )
```

Set the maximum width of the displayed text.

Parameters

width	Max width of the displayed text.

Returns

Nothing.

If a message is longer than *width* characters, it would be broken into shorter lines before being displayed to terminal. However, words longer than *width* would not be broken, and in this case longer lines would be printed. This function is automatically called by the messaging system every time the terminal window is resized, and the width is set equal to the new width of the terminal window. If *width* is zero or negative, long message lines would not be broken. Lines are never broken in log files.

4.30.2.29 cpl_msg_stop()

```
void cpl_msg_stop (
             void  )
```

Turn the messaging system off.

Returns

Nothing

This function needs to be called to turn the messaging system off, typically at the end of a program. To turn the messaging system on the function `cpl_msg_init()` needs to be called. However, since these functions are called anyway by the functions `cpl_init()` and `cpl_end()`, there is generally no need to call them explicitly, and starting from CPL 2.1 they are deprecated.

When `cpl_msg_stop()` is called, the default resized window signal handler is restored, and the duplicated output streams are closed. If a log file is still open, it is closed, and the log verbosity level is set to CPL_MSG_OFF. If the messaging system is not on, nothing is done, and no error is set.

References cpl_msg_stop_log().

Referenced by cpl_end().

4.30.2.30 cpl_msg_stop_log()

```
cpl_error_code cpl_msg_stop_log (
            void  )
```

Close the current log file.

Returns

`CPL_ERROR_NONE` on success.

The log file is closed. The name of the created log file is always the same, and can be obtained by calling the function `cpl_msg_get_log_name()`. An attempt to close a non existing log file would not generate an error condition. This routine may be called in case the logging should be terminated before the end of a program. Otherwise, the function `cpl_msg_stop()` would automatically close the log file when called at the end of the program.

References CPL_ERROR_NONE.

Referenced by cpl_msg_stop().

4.30.2.31 cpl_msg_warning()

```
void cpl_msg_warning (
            const char * component,
            const char * format,
             ...  )
```

Display a warning message.

Returns

Nothing.

Parameters

component	Name of the function generating the message.
format	Format string.
...	Variable argument list associated to the format string.

See the description of the function `cpl_msg_error()`.

Referenced by cpl_get_description(), and cpl_property_delete().

4.31 Multi Frames

Modules

- Regular Expression Filter

Typedefs

- typedef struct _cpl_multiframe_ cpl_multiframe

  The opaque multi-frame data type.
- typedef enum _cpl_multiframe_id_mode_ cpl_multiframe_id_mode

  The flags indicating the naming scheme to use for multi-frame datasets.

Enumerations

- enum _cpl_multiframe_id_mode_ {
  CPL_MULTIFRAME_ID_SET,
  CPL_MULTIFRAME_ID_PREFIX,
  CPL_MULTIFRAME_ID_JOIN }

  The flags indicating the naming scheme to use for multi-frame datasets.

Functions

- cpl_error_code cpl_multiframe_add_empty (cpl_multiframe *self, const char *id)

  Add a placeholder to a multi-frame container.
- cpl_error_code cpl_multiframe_append_datagroup (cpl_multiframe *self, const char *id, const cpl_frame *frame, cpl_size nsets, const char **names, const cpl_regex **filter1, const cpl_regex **filter2, const char **properties, unsigned int flags)

  Adds a group of dataset references given by name to a multi-frame container object.
- cpl_error_code cpl_multiframe_append_datagroup_from_position (cpl_multiframe *self, const char *id, const cpl_frame *frame, cpl_size nsets, cpl_size *positions, const cpl_regex **filter1, const cpl_regex **filter2, const char **properties, unsigned int flags)

  Adds a group of dataset references given by position to a multi-frame container object.
- cpl_error_code cpl_multiframe_append_dataset (cpl_multiframe *self, const char *id, const cpl_frame *frame, const char *name, const cpl_regex *filter1, const cpl_regex *filter2, unsigned int flags)

  Adds a dataset reference given by name to a multi-frame container object.
- cpl_error_code cpl_multiframe_append_dataset_from_position (cpl_multiframe *self, const char *id, const cpl_frame *frame, cpl_size position, const cpl_regex *filter1, const cpl_regex *filter2, unsigned int flags)

  Adds a dataset reference given by position to a multi-frame container object.
- void cpl_multiframe_delete (cpl_multiframe *self)

  Destroys a multi-frame container object.
- cpl_size cpl_multiframe_get_size (const cpl_multiframe *self)

  Get the size of a multi-frame container object.
- cpl_multiframe * cpl_multiframe_new (const cpl_frame *head, const char *id, cpl_regex *filter)

  Create a new multi-frame container object.
- cpl_error_code cpl_multiframe_write (cpl_multiframe *self, const char *filename)

  Write a multi-frame container to a file.

4.31.1 Detailed Description

This module implements the `cpl_multiframe` container type. A multi frame contains references to datasets (FITS extensions) which may be distributed across several physical files. These references can then be merged into a new product file.

Synopsis:

```
#include <cpl_multiframe.h>
```

Note

This feature should be considered as experimental!

4.31.2 Typedef Documentation

4.31.2.1 cpl_multiframe

```
typedef struct _cpl_multiframe_ cpl_multiframe
```

The opaque multi-frame data type.

4.31.2.2 cpl_multiframe_id_mode

```
typedef enum _cpl_multiframe_id_mode_ cpl_multiframe_id_mode
```

The flags indicating the naming scheme to use for multi-frame datasets.

4.31.3 Enumeration Type Documentation

4.31.3.1 _cpl_multiframe_id_mode_

```
enum _cpl_multiframe_id_mode_
```

The flags indicating the naming scheme to use for multi-frame datasets.

Enumerator

CPL_MULTIFRAME_ID_SET	Use the given identifier as dataset name
CPL_MULTIFRAME_ID_PREFIX	Create the dataset name from the given identifier by appending the name of the source dataset.
CPL_MULTIFRAME_ID_JOIN	Create the dataset name by concatenating the names of the involved source datasets. The given identifier is used as separator.

4.31.4 Function Documentation

4.31.4.1 cpl_multiframe_add_empty()

```
cpl_error_code cpl_multiframe_add_empty (
            cpl_multiframe * self,
            const char * id )
```

Add a placeholder to a multi-frame container.

Parameters

self	The multi-frame object.
id	Unique dataset identifier.

Returns

The function returns CPL_ERROR_NONE on success, and an appropriate error code otherwise.

The function adds an empty dataset, a placeholder to a multi-frame container. An empty dataset is special since it is not attached to an underlying data file. When the multi-frame object is written to a file, an empty dataset appears as a named, but otherwise empty unit.

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_ILLEGAL_OUTPUT, and CPL_ERROR_NONE.

4.31.4.2 cpl_multiframe_append_datagroup()

```
cpl_error_code cpl_multiframe_append_datagroup (
            cpl_multiframe * self,
            const char * id,
            const cpl_frame * frame,
            cpl_size nsets,
            const char ** names,
            const cpl_regex ** filter1,
            const cpl_regex ** filter2,
            const char ** properties,
            unsigned int flags )
```

Adds a group of dataset references given by name to a multi-frame container object.

Parameters

self	The multi-frame object.
id	Unique dataset identifier.
frame	The source data frame from which the datasets are taken.
nsets	The number of datasets to be merged.
names	The names of the source datasets in the source data frame.
filter1	Property filters to apply to the primary header of each source dataset.
filter2	Property filters to apply to the extension header of each source dataset.
properties	Property names to be updated.
flags	Flag controlling the creation of the dataset's target id.

Returns

The function returns CPL_ERROR_NONE on success, and an appropriate error code otherwise.

The function is equivalent to cpl_multiframe_append_datagroup_from_position(), but the source datasets to be added are looked up in the source data frame *frame* using their names given in the array *names*.

References CPL_ERROR_DATA_NOT_FOUND, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_fits_find_extension(), cpl_frame_get_filename(), CPL_MULTIFRAME_ID_JOIN, and CPL_MULTIFRAME_ID_PREFIX.

4.31.4.3 cpl_multiframe_append_datagroup_from_position()

```
cpl_error_code cpl_multiframe_append_datagroup_from_position (
            cpl_multiframe * self,
            const char * id,
            const cpl_frame * frame,
            cpl_size nsets,
            cpl_size * positions,
            const cpl_regex ** filter1,
            const cpl_regex ** filter2,
            const char ** properties,
            unsigned int flags )
```

Adds a group of dataset references given by position to a multi-frame container object.

Parameters

self	The multi-frame object.
id	Unique dataset identifier.
frame	The source data frame from which the datasets are taken.
nsets	The number of datasets to be merged.
positions	Positions of the source datasets in the source data frame.
filter1	Property filters to apply to the primary header of each source dataset.
filter2	Property filters to apply to the extension header of each source dataset.
properties	Property names to be updated.
flags	Flag controlling the creation of the dataset's target id.

Returns

The function returns CPL_ERROR_NONE on success, and an appropriate error code otherwise.

The function adds *nsets* new dataset entry to the multi-frame *self*. The datasets to add are taken from the source data frame *frame* and are specified by the first *nsets* positions passed through the array *positions*. Before each selected dataset is added to the multi-frame *self*, the dataset's primary and supplementary properties are merged. If the filter arguments are given, i.e. the respective entries in *filter1* and/or *filter2* are non NULL, they are applied to the primary and the supplementary properties before both are merged. The arrays *filter1* and *filter2* must be given, and they must have *nsets* elements. The array elements, i.e. an individual filter may be set to NULL if no filter should be applied.

The creation of the dataset's target id is controlled by the argument *flags*. It can be only set to CPL_MULTIFRAME_ID_PREFIX or CPL_MULTIFRAME_ID_JOIN. If *flags* is set to CPL_MULTIFRAME_ID_PREFIX then

id is used as prefix for the current dataset's original name (extension name). If the dataset to be appended does not have a name, *id* is used as the dataset identifier. If *flags* is set to CPL_MULTIFRAME_ID_JOIN, the dataset's identifier is created by concatenating the dataset name found in the primary properties, and the dataset name taken from the supplementary properties, using *id* as separator string. If no dataset name is found in the supplementary properties, only the dataset name found in the primary properties is used as identifier and the given separator is not appended. Note that for this last method it is an error if there is no dataset name present in the primary properties of the source dataset.

The argument *id* may be the empty string for the method CPL_MULTIFRAME_ID_JOIN. For the method CPL↩ _MULTIFRAME_ID_PREFIX this is an error.

If *properties* is given it has to be a NULL terminated array of property names. For each specified property name their value is changed according to the naming scheme selected by *flags*, i.e. the value is either prefixed by *id*, or it is set to the concatenation of the source dataset name found in its primary properties, *id*, and its original value. This can be used to correctly change properties used to reference one of the other datasets in the given group through their value. If a given property is not found, it is ignored.

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_frame_get_filename(), CPL_MU↩ LTIFRAME_ID_JOIN, and CPL_MULTIFRAME_ID_PREFIX.

4.31.4.4 cpl_multiframe_append_dataset()

```
cpl_error_code cpl_multiframe_append_dataset (
             cpl_multiframe * self,
             const char * id,
             const cpl_frame * frame,
             const char * name,
             const cpl_regex * filter1,
             const cpl_regex * filter2,
             unsigned int flags )
```

Adds a dataset reference given by name to a multi-frame container object.

Parameters

self	The multi-frame object.
id	Unique dataset identifier.
frame	The source data frame from which the dataset is taken.
name	Name of the source dataset in the source data frame.
filter1	Property filter to apply to the primary header of the source dataset.
filter2	Property filter to apply to the extension header of the source dataset.
flags	Flag controlling the creation of the dataset's target id.

Returns

The function returns CPL_ERROR_NONE on success, and an appropriate error code otherwise.

The function adds a new dataset entry to the multi-frame *self*. The dataset to add is looked up in the source data frame *frame* using its name *name*. It is an error if no dataset with the given name *name* is found. Before the selected dataset is added to the multi-frame *self*, the dataset's primary and supplementary properties are merged. If the filter

arguments are given, i.e. *filter1* and/or *filter2* are non NULL, they are applied to the primary and the supplementary properties before both are merged.

The creation of the dataset's target id is controlled by the argument *flags*. It can be set to one of the values defined by the enumeration cpl_multiframe_id_mode. If *flags* is set to CPL_MULTIFRAME_ID_SET, the argument *id* is used as dataset identifier. If *flags* is set to CPL_MULTIFRAME_ID_PREFIX then *id* is used as prefix for the dataset's original name (extension name). If the dataset to be appended does not have a name, *id* is used as the full dataset identifier. If *flags* is set to CPL_MULTIFRAME_ID_JOIN, the dataset's identifier is created by concatenating the dataset name found in the primary properties, and the dataset name taken from the supplementary properties, using *id* as separator string. If no dataset name is found in the supplementary properties, only the dataset name found in the primary properties is used as identifier and the given separator is not appended. Note that for this last method it is an error if there is no dataset name present in the primary properties of the source dataset.

The argument *id* may be the empty string for the methods CPL_MULTIFRAME_ID_SET and CPL_MULTIFR↩AME_ID_JOIN. For the method CPL_MULTIFRAME_ID_PREFIX this is an error.

References CPL_ERROR_DATA_NOT_FOUND, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_frame_get_filename(), CPL_MULTIFRAME_ID_JOIN, CPL_MULTIFRAME_ID_PREFIX, and CPL_MULTIF↩RAME_ID_SET.

4.31.4.5 cpl_multiframe_append_dataset_from_position()

```
cpl_error_code cpl_multiframe_append_dataset_from_position (
            cpl_multiframe * self,
            const char * id,
            const cpl_frame * frame,
            cpl_size position,
            const cpl_regex * filter1,
            const cpl_regex * filter2,
            unsigned int flags )
```

Adds a dataset reference given by position to a multi-frame container object.

Parameters

self	The multi-frame object.
id	Unique dataset identifier.
frame	The source data frame from which the dataset is taken.
position	Position of the source dataset in the source data frame.
filter1	Property filter to apply to the primary header of the source dataset.
filter2	Property filter to apply to the extension header of the source dataset.
flags	Flag controlling the creation of the dataset's target id.

Returns

The function returns CPL_ERROR_NONE on success, and an appropriate error code otherwise.

The function adds a new dataset entry to the multi-frame *self*. The dataset to add is the taken from position *position* of the source data frame *frame*. Before the selected dataset is added to the multi-frame *self*, the dataset's primary and supplementary properties are merged. If the filter arguments are given, i.e. *filter1* and/or *filter2* are non NULL, they are applied to the primary and the supplementary properties before both are merged.

The creation of the dataset's target id is controlled by the argument *flags*. It can be set to one of the values defined by the enumeration cpl_multiframe_id_mode. If *flags* is set to CPL_MULTIFRAME_ID_SET, the argument *id* is used as dataset identifier. If *flags* is set to CPL_MULTIFRAME_ID_PREFIX then *id* is used as prefix for the dataset's original name (extension name). If the dataset to be appended does not have a name, *id* is used as the full dataset identifier. If *flags* is set to CPL_MULTIFRAME_ID_JOIN, the dataset's identifier is created by concatenating the dataset name found in the primary properties, and the dataset name taken from the supplementary properties, using *id* as separator string. If no dataset name is found in the supplementary properties, only the dataset name found in the primary properties is used as identifier and the given separator is not appended. Note that for this last method it is an error if there is no dataset name present in the primary properties of the source dataset.

The argument *id* may be the empty string for the methods CPL_MULTIFRAME_ID_SET and CPL_MULTIFR↩AME_ID_JOIN. For the method CPL_MULTIFRAME_ID_PREFIX this is an error.

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_frame_get_filename(), CPL_MU↩LTIFRAME_ID_JOIN, CPL_MULTIFRAME_ID_PREFIX, and CPL_MULTIFRAME_ID_SET.

4.31.4.6 cpl_multiframe_delete()

```
void cpl_multiframe_delete (
            cpl_multiframe * self )
```

Destroys a multi-frame container object.

Parameters

self	The multi-frame container.

Returns

Nothing.

The function deallocates the multi-frame container *self*.

4.31.4.7 cpl_multiframe_get_size()

```
cpl_size cpl_multiframe_get_size (
            const cpl_multiframe * self )
```

Get the size of a multi-frame container object.

Parameters

self	The multi-frame object.

Returns

The function returns the current number of datasets referenced by the multi-frame container.

The function returns the number of dataset entries stored in the multi-frame *self*.

References CPL_ERROR_NULL_INPUT.

4.31.4.8 cpl_multiframe_new()

```
cpl_multiframe* cpl_multiframe_new (
            const cpl_frame * head,
            const char * id,
            cpl_regex * filter )
```

Create a new multi-frame container object.

Parameters

head	The first dataset of the multi-frame object
id	Unique dataset identifier.
filter	Filter to be applied to the dataset properties on merge.

Returns

The function returns a newly allocated multi-frame object on success, or `NULL` otherwise.

The function allocates the memory for a multi-frame container and adds the frame *master* as the head, i.e. the first and emtpy dataset of the multi-frame object. A unique dataset identifier *id* may be given. The identifier *id* may be the empty string, in which case it is ignored when writing the multi-frame object to a file. Furthermore a regular expression filter object may be given which will be applied to each of the properties of the dataset *head*. Only those properties of em head, which pass the filter *filter* will be propagated to the created multi-frame container.

References CPL_ERROR_NULL_INPUT.

4.31.4.9 cpl_multiframe_write()

```
cpl_error_code cpl_multiframe_write (
            cpl_multiframe * self,
            const char * filename )
```

Write a multi-frame container to a file.

Parameters

self	The multi-frame container object.
filename	Name of the file to which the multi-frame object is written.

Returns

The function returns `CPL_ERROR_NONE` on success, and an appropriate error code otherwise.

The function writes the multi-frame object *self* to the file *filename*. A multi-frame object contains only the properties and the references to the data units, which may be located in different files. Only when this function is called the all referenced datasets are copied and written to the output file.

References CPL_ERROR_DATA_NOT_FOUND, CPL_ERROR_FILE_NOT_CREATED, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.32 Parameter Lists

Typedefs

- typedef struct _cpl_parameterlist_ cpl_parameterlist

  The opaque parameter list data type.

Functions

- cpl_error_code cpl_parameterlist_append (cpl_parameterlist *self, cpl_parameter *parameter)

  Append a parameter to a parameter list.
- void cpl_parameterlist_delete (cpl_parameterlist *self)

  Destroy a parameter list.
- void cpl_parameterlist_dump (const cpl_parameterlist *self, FILE *stream)

  Dump the contents of a parameter list to the given stream.
- cpl_parameter * cpl_parameterlist_find (cpl_parameterlist *self, const char *name)

  Find a parameter with the given name in a parameter list.
- const cpl_parameter * cpl_parameterlist_find_const (const cpl_parameterlist *self, const char *name)

  Find a parameter with the given name in a parameter list.
- cpl_parameter * cpl_parameterlist_find_context (cpl_parameterlist *self, const char *context)

  Find a parameter which belongs to the given context in a parameter list.
- const cpl_parameter * cpl_parameterlist_find_context_const (const cpl_parameterlist *self, const char *context)

  Find a parameter which belongs to the given context in a parameter list.
- cpl_parameter * cpl_parameterlist_find_tag (cpl_parameterlist *self, const char *tag)

  Find a parameter with the given tag in a parameter list.
- const cpl_parameter * cpl_parameterlist_find_tag_const (const cpl_parameterlist *self, const char *tag)

  Find a parameter with the given tag in a parameter list.
- cpl_parameter * cpl_parameterlist_find_type (cpl_parameterlist *self, cpl_type type)

  Find a parameter of the given type in a parameter list.
- const cpl_parameter * cpl_parameterlist_find_type_const (const cpl_parameterlist *self, cpl_type type)

  Find a parameter of the given type in a parameter list.
- cpl_parameter * cpl_parameterlist_get_first (cpl_parameterlist *self)

  Get the first parameter in the given parameter list.
- const cpl_parameter * cpl_parameterlist_get_first_const (const cpl_parameterlist *self)

  Get the first parameter in the given parameter list.
- cpl_parameter * cpl_parameterlist_get_last (cpl_parameterlist *self)

  Get the last parameter in the given list.
- const cpl_parameter * cpl_parameterlist_get_last_const (const cpl_parameterlist *self)

  Get the last parameter in the given list.
- cpl_parameter * cpl_parameterlist_get_next (cpl_parameterlist *self)

  Get the next parameter in the given list.
- const cpl_parameter * cpl_parameterlist_get_next_const (const cpl_parameterlist *self)

  Get the next parameter in the given list.
- cpl_size cpl_parameterlist_get_size (const cpl_parameterlist *self)

  Get the current size of a parameter list.
- cpl_parameterlist * cpl_parameterlist_new (void)

  Create a new parameter list.

4.32.1 Detailed Description

The module implements a parameter list data type, a container for the cpl_parameter type. It provides a convenient way to pass a set of parameters, as a whole, to a function.

It is used in the plugin interface (cf. Plugin Interface), for instance, to pass the parameters a recipe accepts from the plugin to the calling application and vice versa.

All functions expect a valid pointer to a parameter list as input, unless otherwise specified.

Synopsis:

```
#include <cpl_parameterlist.h>
```

4.32.2 Typedef Documentation

4.32.2.1 cpl_parameterlist

```
typedef struct _cpl_parameterlist_ cpl_parameterlist
```

The opaque parameter list data type.

4.32.3 Function Documentation

4.32.3.1 cpl_parameterlist_append()

```
cpl_error_code cpl_parameterlist_append (
            cpl_parameterlist * self,
            cpl_parameter * parameter )
```

Append a parameter to a parameter list.

Parameters

self	A parameter list.
parameter	The parameter to append.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The parameter *parameter* is appended to the parameter list *self*.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.32.3.2 cpl_parameterlist_delete()

```
void cpl_parameterlist_delete (
            cpl_parameterlist * self )
```

Destroy a parameter list.

Parameters

self	The parameter list to destroy.

Returns

Nothing.

The function destroys the parameter list object *self* and all parameters it possibly contains. The individual parameters are destroyed using the parameter destructor (cf. Parameters). If *self* is `NULL`, nothing is done and no error is set.

References cpl_parameter_delete().

4.32.3.3 cpl_parameterlist_dump()

```
void cpl_parameterlist_dump (
            const cpl_parameterlist * self,
            FILE * stream )
```

Dump the contents of a parameter list to the given stream.

Parameters

self	The parameter list.
stream	The output stream to use.

Returns

Nothing.

The function dumps the debugging information for each parameter found in the parameter list *self* to the output stream *stream*. The debugging information for each individual parameter is dumped using **cpl_parameter_dump()**. If *self* is `NULL` the function does nothing.

See also

cpl_parameter_dump()

References cpl_parameter_dump().

4.32.3.4 cpl_parameterlist_find()

```
cpl_parameter* cpl_parameterlist_find (
            cpl_parameterlist * self,
            const char * name )
```

Find a parameter with the given name in a parameter list.

Parameters

self	A parameter list.
name	The parameter name to search for.

Returns

The function returns a handle for the first parameter with the name *name*, or `NULL` if no such parameter was found. If an error occurs the function returns `NULL` and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.

The function searches the parameter list *self* for the first occurrence of a parameter with the fully qualified name *name*. If no parameter with this name exists, the function returns `NULL`.

References cpl_error_set_where, cpl_errorstate_get(), cpl_errorstate_is_equal(), and cpl_parameterlist_find_↩ const().

4.32.3.5 cpl_parameterlist_find_const()

```
const cpl_parameter* cpl_parameterlist_find_const (
            const cpl_parameterlist * self,
            const char * name )
```

Find a parameter with the given name in a parameter list.

Parameters

self	A parameter list.
name	The parameter name to search for.

Returns

The function returns a handle for the first parameter with the name *name*, or NULL if no such parameter was found. If an error occurs the function returns NULL and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a NULL pointer.

The function searches the parameter list *self* for the first occurrence of a parameter with the fully qualified name *name*. If no parameter with this name exists, the function returns NULL.

References CPL_ERROR_NULL_INPUT, and cpl_parameter_get_name().

Referenced by cpl_parameterlist_find().

4.32.3.6 cpl_parameterlist_find_context()

```
cpl_parameter* cpl_parameterlist_find_context (
            cpl_parameterlist * self,
            const char * context )
```

Find a parameter which belongs to the given context in a parameter list.

Parameters

self	A parameter list.
context	The parameter context to search for.

Returns

The function returns a handle for the first parameter with the context *context*, or NULL if no such parameter was found. If an error occurs the function returns NULL and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *context* is a NULL pointer.

The function searches the parameter list *self* for the first occurrence of a parameter which belongs to the context *context*. If no parameter with this type exists, the function returns NULL.

References cpl_error_set_where, cpl_errorstate_get(), cpl_errorstate_is_equal(), and cpl_parameterlist_find_↩ context_const().

4.32.3.7 cpl_parameterlist_find_context_const()

```
const cpl_parameter* cpl_parameterlist_find_context_const (
            const cpl_parameterlist * self,
            const char * context )
```

Find a parameter which belongs to the given context in a parameter list.

Parameters

self	A parameter list.
context	The parameter context to search for.

Returns

The function returns a handle for the first parameter with the context *context*, or NULL if no such parameter was found. If an error occurs the function returns NULL and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *context* is a NULL pointer.

The function searches the parameter list *self* for the first occurrence of a parameter which belongs to the context *context*. If no parameter with this type exists, the function returns NULL.

References CPL_ERROR_NULL_INPUT, and cpl_parameter_get_context().

Referenced by cpl_parameterlist_find_context().

4.32.3.8 cpl_parameterlist_find_tag()

```
cpl_parameter* cpl_parameterlist_find_tag (
            cpl_parameterlist * self,
            const char * tag )
```

Find a parameter with the given tag in a parameter list.

Parameters

self	A parameter list.
tag	The parameter tag to search for.

Returns

The function returns a handle for the first parameter with the tag *tag*, or NULL if no such parameter was found. If an error occurs the function returns NULL and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *tag* is a NULL pointer.

The function searches the parameter list *self* for the first occurrence of a parameter with the user tag *tag*. If no parameter with this tag exists, the function returns NULL.

References cpl_error_set_where, cpl_errorstate_get(), cpl_errorstate_is_equal(), and cpl_parameterlist_find_tag↩_const().

4.32.3.9 cpl_parameterlist_find_tag_const()

```
const cpl_parameter* cpl_parameterlist_find_tag_const (
            const cpl_parameterlist * self,
            const char * tag )
```

Find a parameter with the given tag in a parameter list.

Parameters

self	A parameter list.
tag	The parameter tag to search for.

Returns

The function returns a handle for the first parameter with the tag *tag*, or NULL if no such parameter was found. If an error occurs the function returns NULL and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *tag* is a NULL pointer.

The function searches the parameter list *self* for the first occurrence of a parameter with the user tag *tag*. If no parameter with this tag exists, the function returns NULL.

References CPL_ERROR_NULL_INPUT, and cpl_parameter_get_tag().

Referenced by cpl_parameterlist_find_tag().

4.32.3.10 cpl_parameterlist_find_type()

```
cpl_parameter* cpl_parameterlist_find_type (
            cpl_parameterlist * self,
            cpl_type type )
```

Find a parameter of the given type in a parameter list.

Parameters

self	A parameter list.
type	The parameter type to search for.

Returns

The function returns a handle for the first parameter with the type *type*, or NULL if no such parameter was found. If an error occurs the function returns NULL and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function searches the parameter list *self* for the first occurrence of a parameter whose value is of the type *type*. If no parameter with this type exists, the function returns NULL.

References cpl_error_set_where, cpl_errorstate_get(), cpl_errorstate_is_equal(), and cpl_parameterlist_find_↩ type_const().

4.32.3.11 cpl_parameterlist_find_type_const()

```
const cpl_parameter* cpl_parameterlist_find_type_const (
            const cpl_parameterlist * self,
            cpl_type type )
```

Find a parameter of the given type in a parameter list.

Parameters

self	A parameter list.
type	The parameter type to search for.

Returns

The function returns a handle for the first parameter with the type *type*, or NULL if no such parameter was found. If an error occurs the function returns NULL and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function searches the parameter list *self* for the first occurrence of a parameter whose value is of the type *type*. If no parameter with this type exists, the function returns NULL.

References CPL_ERROR_NULL_INPUT, and cpl_parameter_get_type().

Referenced by cpl_parameterlist_find_type().

4.32.3.12 cpl_parameterlist_get_first()

```
cpl_parameter* cpl_parameterlist_get_first (
            cpl_parameterlist * self )
```

Get the first parameter in the given parameter list.

Parameters

self	A parameter list.

Returns

The function returns a handle for the first parameter in the list, or NULL if the list is empty. If an error occurs the function returns NULL and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function returns the first parameter in the parameter list *self*, if it exists. If there is no first parameter, i.e. if the list is empty, NULL is returned. The function updates the internal search position cache.

References cpl_error_set_where, cpl_errorstate_get(), cpl_errorstate_is_equal(), and cpl_parameterlist_get_first_const().

4.32.3.13 cpl_parameterlist_get_first_const()

```
const cpl_parameter* cpl_parameterlist_get_first_const (
            const cpl_parameterlist * self )
```

Get the first parameter in the given parameter list.

Parameters

self	A parameter list.

Returns

The function returns a handle for the first parameter in the list, or NULL if the list is empty. If an error occurs the function returns NULL and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function returns the first parameter in the parameter list *self*, if it exists. If there is no first parameter, i.e. if the list is empty, NULL is returned. The function updates the internal search position cache.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_parameterlist_get_first().

4.32.3.14 cpl_parameterlist_get_last()

```
cpl_parameter* cpl_parameterlist_get_last (
            cpl_parameterlist * self )
```

Get the last parameter in the given list.

Parameters

self	A parameter list.

Returns

The function returns a handle for the last parameter in the list. If an error occurs the function returns NULL and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_ILLEGAL_INPUT	The parameter *self* is an empty list.

The function returns the last parameter stored in the parameter list *self*. The list *self* **must not** be empty.

References cpl_error_set_where, cpl_errorstate_get(), cpl_errorstate_is_equal(), and cpl_parameterlist_get_last↩_const().

4.32.3.15 cpl_parameterlist_get_last_const()

```
const cpl_parameter* cpl_parameterlist_get_last_const (
            const cpl_parameterlist * self )
```

Get the last parameter in the given list.

Parameters

self	A parameter list.

Returns

The function returns a handle for the last parameter in the list. If an error occurs the function returns NULL and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_ILLEGAL_INPUT	The parameter *self* is an empty list.

The function returns the last parameter stored in the parameter list *self*. The list *self* **must not** be empty.

References CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_parameterlist_get_last().

4.32.3.16 cpl_parameterlist_get_next()

```
cpl_parameter* cpl_parameterlist_get_next (
            cpl_parameterlist * self )
```

Get the next parameter in the given list.

Parameters

self	A parameter list.

Returns

The function returns a handle for the next parameter in the list, or NULL if there are no more parameters in the list. If an error occurs the function returns NULL and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function returns the next parameter in the parameter list *self* if it exists and NULL otherwise. The function uses the last cached search position to determine the most recently accessed parameter. This means that the function only works as expected if the *self* has been initialised by a call to **cpl_parameterlist_get_first()**, and if no function updating the internal cache was called between two subsequent calls to this function.

References cpl_error_set_where, cpl_errorstate_get(), cpl_errorstate_is_equal(), and cpl_parameterlist_get_next↩_const().

4.32.3.17 cpl_parameterlist_get_next_const()

```
const cpl_parameter* cpl_parameterlist_get_next_const (
            const cpl_parameterlist * self )
```

Get the next parameter in the given list.

Parameters

self	A parameter list.

Returns

The function returns a handle for the next parameter in the list, or NULL if there are no more parameters in the list. If an error occurs the function returns NULL and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function returns the next parameter in the parameter list *self* if it exists and NULL otherwise. The function uses the last cached search position to determine the most recently accessed parameter. This means that the function only works as expected if the *self* has been initialised by a call to **cpl_parameterlist_get_first_const()**, and if no function updating the internal cache was called between two subsequent calls to this function.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_parameterlist_get_next().

4.32.3.18 cpl_parameterlist_get_size()

```
cpl_size cpl_parameterlist_get_size (
            const cpl_parameterlist * self )
```

Get the current size of a parameter list.

Parameters

self	A parameter list

Returns

The parameter list's current size, or 0 if the list is empty. If an error occurs the function returns 0 and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function reports the current number of elements stored in the parameter list *self*.

References CPL_ERROR_NULL_INPUT.

4.32.3.19 cpl_parameterlist_new()

```
cpl_parameterlist* cpl_parameterlist_new (
            void  )
```

Create a new parameter list.

Returns

A pointer to the newly created parameter list.

The function creates a new parameter list object. The created object must be destroyed using the parameter list destructor **cpl_parameterlist_delete()**.

4.33 Parameters

Typedefs

- typedef struct _cpl_parameter_ cpl_parameter

  The opaque parameter data type.
- typedef enum _cpl_parameter_class_ cpl_parameter_class

  The parameter class data type.
- typedef enum _cpl_parameter_mode_ cpl_parameter_mode

  The parameter mode data type.

Enumerations

- enum _cpl_parameter_class_ {
  CPL_PARAMETER_CLASS_INVALID,
  CPL_PARAMETER_CLASS_VALUE,
  CPL_PARAMETER_CLASS_RANGE,
  CPL_PARAMETER_CLASS_ENUM }

  Supported parameter classes.
- enum _cpl_parameter_mode_ {
  CPL_PARAMETER_MODE_CLI,
  CPL_PARAMETER_MODE_ENV,
  CPL_PARAMETER_MODE_CFG }

  Supported parameter modes.

Functions

- void cpl_parameter_delete (cpl_parameter *self)

  Delete a parameter.
- cpl_error_code cpl_parameter_disable (cpl_parameter *self, cpl_parameter_mode mode)

  Deactivate a parameter for the given mode.
- void cpl_parameter_dump (const cpl_parameter *self, FILE *stream)

  Dump the parameter debugging information to the given stream.
- cpl_parameter * cpl_parameter_duplicate (const cpl_parameter *self)

  Create a copy of a parameter.
- cpl_error_code cpl_parameter_enable (cpl_parameter *self, cpl_parameter_mode mode)

  Activates a parameter for the given mode.
- const char * cpl_parameter_get_alias (const cpl_parameter *self, cpl_parameter_mode mode)

  Get the parameter's alias name for the given mode.
- int cpl_parameter_get_bool (const cpl_parameter *self)

  Get the value of the given boolean parameter.
- cpl_parameter_class cpl_parameter_get_class (const cpl_parameter *self)

  Get the parameter's class.
- const char * cpl_parameter_get_context (const cpl_parameter *self)

  Get the context of a parameter.
- int cpl_parameter_get_default_bool (const cpl_parameter *self)

  Get the default value of the given boolean parameter.
- double cpl_parameter_get_default_double (const cpl_parameter *self)

  Get the default value of the given double parameter.
- int cpl_parameter_get_default_flag (const cpl_parameter *self)

Get the presence status flag of the given parameter.

- int cpl_parameter_get_default_int (const cpl_parameter *self)

  Get the default value of the given integer parameter.
- const char * cpl_parameter_get_default_string (const cpl_parameter *self)

  Get the default value of the given string parameter.
- double cpl_parameter_get_double (const cpl_parameter *self)

  Get the value of the given double parameter.
- double cpl_parameter_get_enum_double (const cpl_parameter *self, int position)

  Get the possible values for a double enumeration.
- int cpl_parameter_get_enum_int (const cpl_parameter *self, int position)

  Get the possible values for an integer enumeration.
- int cpl_parameter_get_enum_size (const cpl_parameter *self)

  Get the number of alternatives of an enumeration parameter.
- const char * cpl_parameter_get_enum_string (const cpl_parameter *self, int position)

  Get the possible values for a string enumeration.
- const char * cpl_parameter_get_help (const cpl_parameter *self)

  Get the description of a parameter.
- int cpl_parameter_get_id (const cpl_parameter *self)

  Get the numerical identifier of the given parameter.
- int cpl_parameter_get_int (const cpl_parameter *self)

  Get the value of the given integer parameter.
- const char * cpl_parameter_get_name (const cpl_parameter *self)

  Get the name of a parameter.
- double cpl_parameter_get_range_max_double (const cpl_parameter *self)

  Get the maximum value of a double range parameter.
- int cpl_parameter_get_range_max_int (const cpl_parameter *self)

  Get the maximum value of an integer range parameter.
- double cpl_parameter_get_range_min_double (const cpl_parameter *self)

  Get the minimum value of a double range parameter.
- int cpl_parameter_get_range_min_int (const cpl_parameter *self)

  Get the minimum value of an integer range parameter.
- const char * cpl_parameter_get_string (const cpl_parameter *self)

  Get the value of the given string parameter.
- const char * cpl_parameter_get_tag (const cpl_parameter *self)

  Get the parameter's user tag.
- cpl_type cpl_parameter_get_type (const cpl_parameter *self)

  Get the parameter's value type.
- int cpl_parameter_is_enabled (const cpl_parameter *self, cpl_parameter_mode mode)

  Get the parameter's activity status for the environment context.
- cpl_parameter * cpl_parameter_new_enum (const char *name, cpl_type type, const char *description, const char *context,...)

  Create a new enumeration parameter.
- cpl_parameter * cpl_parameter_new_enum_from_array (const char *name, cpl_type type, const char *description, const char *context, cpl_size position, cpl_size size, const void *alternatives)

  Create a new enumeration parameter from an array of alternative values.
- cpl_parameter * cpl_parameter_new_range (const char *name, cpl_type type, const char *description, const char *context,...)

  Create a new range parameter.
- cpl_parameter * cpl_parameter_new_value (const char *name, cpl_type type, const char *description, const char *context,...)

  Create a new value parameter.

- cpl_error_code cpl_parameter_set_alias (cpl_parameter *self, cpl_parameter_mode mode, const char *alias)

  Set alias names for the given parameter.
- cpl_error_code cpl_parameter_set_bool (cpl_parameter *self, int value)

  Assign a boolean value to a parameter.
- cpl_error_code cpl_parameter_set_default_bool (cpl_parameter *self, int value)

  Modify the default value of a boolean parameter.
- cpl_error_code cpl_parameter_set_default_double (cpl_parameter *self, double value)

  Modify the default value of a double parameter.
- cpl_error_code cpl_parameter_set_default_flag (cpl_parameter *self, int status)

  Change the presence status flag of the given parameter.
- cpl_error_code cpl_parameter_set_default_int (cpl_parameter *self, int value)

  Modify the default value of an integer parameter.
- cpl_error_code cpl_parameter_set_default_string (cpl_parameter *self, const char *value)

  Modify the default value of a string parameter.
- cpl_error_code cpl_parameter_set_double (cpl_parameter *self, double value)

  Assign a double value to a parameter.
- cpl_error_code cpl_parameter_set_id (cpl_parameter *self, int id)

  Set the numerical identifier of the given parameter.
- cpl_error_code cpl_parameter_set_int (cpl_parameter *self, int value)

  Assign an integer value to a parameter.
- cpl_error_code cpl_parameter_set_string (cpl_parameter *self, const char *value)

  Assign a string value to a parameter.
- cpl_error_code cpl_parameter_set_tag (cpl_parameter *self, const char *tag)

  Set the tag of the given parameter.

4.33.1 Detailed Description

This module implements a class of data types which can be used to manage context specific, named parameters. They provide a standard way to pass, for instance, command line information to different components of an application.

The fundamental parts of a parameter are its name, a context to which it belongs (a specific component of an application for instance), its current value and a default value.

The implementation supports three classes of parameters:

- A plain value
- A range of values
- An enumeration value

When a parameter is created it is created for a particular value type. The type of a parameter's current and default value may be (cf. Type codes):

- `CPL_TYPE_BOOL`
- `CPL_TYPE_INT`
- `CPL_TYPE_DOUBLE`
- `CPL_TYPE_STRING`

When a value is assigned to a parameter, the different parameter classes behave differently. For a plain value, no checks are applied to the data value to be assigned. For a range or an enumeration, on the other hand, the value to be stored is validated with respect to the minimum and maximum value of the range and the list of enumeration alternatives respectively. If the value is invalid, an error is reported.

Note

The validation of parameter values on assignment is not yet implemented.

The functions to create a new parameter of a particular class are polymorphic and accept any value type mentioned above to initialise the parameter to create. This is accomplished by using a variable argument list. The number and type of the parameters expected by these functions is determined from the parameter class and the indicated value type.

The constructor of a plain value expects a single value, the parameter's default value, which *must* be of the appropriate type, as argument while the constructor of a range parameter expects the default value, followed by the minimum value and the maximum value. The constructor of an enumeration parameter expects as arguments the default value, the number of possible enumeration values followed by the list of the possible enumeration values. Note that the default value must be a member of the list of possible enumeration values.

An example of how parameters of the different classes are created and destroyed is shown below:

```
cpl_parameter *value;
cpl_parameter *range;
cpl_parameter *enumeration;
...
value = cpl_parameter_new_value("Value", CPL_TYPE_INT,
                                "This is a plain value of type integer",
                                "Example",
                                -1);
range = cpl_parameter_new_range("Range", CPL_TYPE_DOUBLE,
                                "This is a value range of type double",
                                "Example",
                                0.5, 0.0, 1.0);
enumeration = cpl_parameter_new_enum("Enum", CPL_TYPE_STRING,
                                     "This is an enumeration of type "
                                     "string",
                                     "Example",
                                     "first", 3, "first", "second",
                                     "third");
...
cpl_parameter_delete(value);
cpl_parameter_delete(range);
cpl_parameter_delete(enumeration);
```

After the parameter has been created and initialised the parameters current value equals its default value. The initialisation values are copied into the parameter, i.e. if an initialiser value resides in dynamically allocated memory, it can be deallocated after the parameter has been initialised without affecting the created parameter.

Note

A variable argument list allows only limited type checking. Therefore, code explicitly what you mean and double check the constructor calls that the number of arguments and the types of the elements of the variable argument list are what the constructor expects. For instance, if a constructor takes a floating-point argument do not forget to put the decimal point at the end!

Synopsis:

```
#include <cpl_parameter.h>
```

4.33.2 Typedef Documentation

4.33.2.1 cpl_parameter

```
typedef struct _cpl_parameter_ cpl_parameter
```

The opaque parameter data type.

4.33.2.2 cpl_parameter_class

```
typedef enum _cpl_parameter_class_ cpl_parameter_class
```

The parameter class data type.

4.33.2.3 cpl_parameter_mode

```
typedef enum _cpl_parameter_mode_ cpl_parameter_mode
```

The parameter mode data type.

4.33.3 Enumeration Type Documentation

4.33.3.1 _cpl_parameter_class_

```
enum _cpl_parameter_class_
```

Supported parameter classes.

Enumerator

CPL_PARAMETER_CLASS_INVALID	Parameter of undefined or invalid class.
CPL_PARAMETER_CLASS_VALUE	Parameter representing a plain value.
CPL_PARAMETER_CLASS_RANGE	Parameter representing a range of values.
CPL_PARAMETER_CLASS_ENUM	Parameter representing an enumeration value.

4.33.3.2 _cpl_parameter_mode_

```
enum _cpl_parameter_mode_
```

Supported parameter modes.

The parameter mode is used to set or get context specific parameter attributes.

Enumerator

CPL_PARAMETER_MODE_CLI	Command line mode of the parameter.
CPL_PARAMETER_MODE_ENV	Environment variable mode of the parameter.
CPL_PARAMETER_MODE_CFG	Configuration file mode of the parameter.

4.33.4 Function Documentation

4.33.4.1 cpl_parameter_delete()

```
void cpl_parameter_delete (
            cpl_parameter * self )
```

Delete a parameter.

Parameters

self	The parameter to delete.

Returns

Nothing.

The function deletes a parameter of any class and type. All memory resources acquired by the parameter during its usage are released. If the parameter *self* is `NULL`, nothing is done and no error is set.

References CPL_PARAMETER_CLASS_VALUE.

Referenced by cpl_parameterlist_delete().

4.33.4.2 cpl_parameter_disable()

```
cpl_error_code cpl_parameter_disable (
            cpl_parameter * self,
            cpl_parameter_mode mode )
```

Deactivate a parameter for the given mode.

Parameters

self	A parameter.
mode	The parameter mode.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	The parameter mode *mode* is not supported.

The function disables the parameter *self* for the given mode *mode*.

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, cpl_parameter_get_name(), CPL_PARAMETER_MODE_CFG, CPL_PARAMETER_M↩ ODE_CLI, and CPL_PARAMETER_MODE_ENV.

4.33.4.3 cpl_parameter_dump()

```
void cpl_parameter_dump (
            const cpl_parameter * self,
            FILE * stream )
```

Dump the parameter debugging information to the given stream.

Parameters

self	The parameter.
stream	The output stream to use.

Returns

Nothing.

The function dumps the contents of of the parameter *self* to the output stream *stream*. If *stream* is `NULL` the function writes to the standard output. If *self* is `NULL` the function does nothing.

References CPL_PARAMETER_CLASS_ENUM, CPL_PARAMETER_CLASS_RANGE, CPL_PARAMETER_C↩ LASS_VALUE, CPL_TYPE_BOOL, CPL_TYPE_DOUBLE, CPL_TYPE_INT, and CPL_TYPE_STRING.

Referenced by cpl_parameterlist_dump().

4.33.4.4 cpl_parameter_duplicate()

```
cpl_parameter* cpl_parameter_duplicate (
            const cpl_parameter * self )
```

Create a copy of a parameter.

Parameters

self	The parameter to copy.

Returns

The copy of the given parameter, or NULL in case of an error. In the latter case an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function returns a copy of the parameter *self*. The copy is a deep copy, i.e. all parameter properties are copied.

References CPL_ERROR_NULL_INPUT, cpl_error_set, and CPL_PARAMETER_CLASS_VALUE.

4.33.4.5 cpl_parameter_enable()

```
cpl_error_code cpl_parameter_enable (
            cpl_parameter * self,
            cpl_parameter_mode mode )
```

Activates a parameter for the given mode.

Parameters

self	A parameter.
mode	The parameter mode.

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_ILLEGAL_INPUT	The parameter mode *mode* is not supported.

The function enables the parameter *self* for the given mode *mode*.

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, cpl_parameter_get_name(), CPL_PARAMETER_MODE_CFG, CPL_PARAMETER_M↩ ODE_CLI, and CPL_PARAMETER_MODE_ENV.

4.33.4.6 cpl_parameter_get_alias()

```
const char* cpl_parameter_get_alias (
            const cpl_parameter * self,
            cpl_parameter_mode mode )
```

Get the parameter's alias name for the given mode.

Parameters

self	A parameter.
mode	The parameter mode.

Returns

The function returns the parameter's context specific alias name. If no alias name was previously assigned the function returns NULL. If an error occurs the function returns NULL and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_ILLEGAL_INPUT	The parameter mode *mode* is not supported.

The function retrieves the read-only alias name of the parameter *self*. The context for which an alias is retrieved is selected by the *mode*. The functions accepts the parameter modes CPL_PARAMETER_MODE_CFG, CPL_PA↩RAMETER_MODE_CLI and CPL_PARAMETER_MODE_ENV.

See also

cpl_parameter_mode

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, cpl_parameter_get_name(), CPL_PARAMETER_MODE_CFG, CPL_PARAMETER_MODE_CLI, and CPL_PAR↩AMETER_MODE_ENV.

4.33.4.7 cpl_parameter_get_bool()

```
int cpl_parameter_get_bool (
            const cpl_parameter * self )
```

Get the value of the given boolean parameter.

Parameters

self	A boolean parameter.

Returns

The function returns the integer representation of the parameter's boolean value. TRUE is represented as non-zero value; whereas 0 indicates FALSE. If an error occurs the function returns 0 and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_TYPE_MISMATCH	The type of the parameter *self* is not of type CPL_TYPE_BOOL.

The current boolean value of the parameter *self* is retrieved.

References CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMAT↩CH, cpl_parameter_get_name(), cpl_parameter_get_type(), CPL_TYPE_BOOL, and cpl_type_get_name().

4.33.4.8 cpl_parameter_get_class()

```
cpl_parameter_class cpl_parameter_get_class (
            const cpl_parameter * self )
```

Get the parameter's class.

Parameters

self	A parameter.

Returns

The function returns the parameter's class identifier. The function returns CPL_PARAMETER_CLASS_IN↩VALID if *self* is not a valid parameter, but NULL. In this case an appropriate error code is also set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The class identifier of the parameter *self* is retrieved.

References CPL_ERROR_NULL_INPUT, cpl_error_set, and CPL_PARAMETER_CLASS_INVALID.

4.33.4.9 cpl_parameter_get_context()

```
const char* cpl_parameter_get_context (
            const cpl_parameter * self )
```

Get the context of a parameter.

Parameters

self	A parameter.

Returns

The function returns the context of the parameter, or NULL if *self* is not a valid parameter but NULL. In the latter case an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function returns the read-only context to which the parameter *self* belongs. The parameter's context **must** not be modified using the returned pointer.

References CPL_ERROR_NULL_INPUT, and cpl_error_set.

Referenced by cpl_parameterlist_find_context_const().

4.33.4.10 cpl_parameter_get_default_bool()

```
int cpl_parameter_get_default_bool (
            const cpl_parameter * self )
```

Get the default value of the given boolean parameter.

Parameters

self	A boolean parameter.

Returns

The function returns the integer representation of the parameter's default boolean value. TRUE is represented as non-zero value; whereas 0 indicates FALSE. If an error occurs the function returns 0 and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The type of the parameter *self* is not of type `CPL_TYPE_BOOL`.

The current boolean default value of the parameter *self* is retrieved.

References CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMAT↩CH, cpl_parameter_get_name(), cpl_parameter_get_type(), CPL_TYPE_BOOL, and cpl_type_get_name().

4.33.4.11 cpl_parameter_get_default_double()

```
double cpl_parameter_get_default_double (
            const cpl_parameter * self )
```

Get the default value of the given double parameter.

Parameters

self	An double parameter.

Returns

The function returns the parameter's default double value. If an error occurs the function returns 0 and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The type of the parameter *self* is not of type `CPL_TYPE_DOUBLE`.

The current double default value of the parameter *self* is retrieved.

References CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMAT↩CH, cpl_parameter_get_name(), cpl_parameter_get_type(), CPL_TYPE_DOUBLE, and cpl_type_get_name().

4.33.4.12 cpl_parameter_get_default_flag()

```
int cpl_parameter_get_default_flag (
            const cpl_parameter * self )
```

Get the presence status flag of the given parameter.

Parameters

self	A parameter.

Returns

The function returns the current setting of the parameters presence flag. If the parameter is present the function returns 1 and 0 otherwise. If an error occurs the function returns 0 and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function indicates whether the given parameter *self* was seen by an application while processing the input from the command line, the environment and/or a configuration file. If the parameter was seen the application may set this status flag and query it later using this function.

See also

cpl_parameter_set_default_flag()

References CPL_ERROR_NULL_INPUT, and cpl_error_set.

4.33.4.13 cpl_parameter_get_default_int()

```
int cpl_parameter_get_default_int (
            const cpl_parameter * self )
```

Get the default value of the given integer parameter.

Parameters

self	An integer parameter.

Returns

The function returns the parameter's default integer value. If an error occurs the function returns 0 and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The type of the parameter *self* is not of type `CPL_TYPE_INT`.

The current integer default value of the parameter *self* is retrieved.

References CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMAT↩
CH, cpl_parameter_get_name(), cpl_parameter_get_type(), cpl_type_get_name(), and CPL_TYPE_INT.

4.33.4.14 cpl_parameter_get_default_string()

```
const char* cpl_parameter_get_default_string (
            const cpl_parameter * self )
```

Get the default value of the given string parameter.

Parameters

self	An string parameter.

Returns

The function returns the parameter's default string value. If an error occurs the function returns 0 and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The type of the parameter *self* is not of type `CPL_TYPE_STRING`.

The current read-only string default value of the parameter *self* is retrieved. If *self* is not a valid pointer the error code `CPL_ERROR_NULL_INPUT` is set on return. Also, if the parameter *self* is not an string parameter, i.e. the parameter's type is different from `CPL_TYPE_INT`, the error code `CPL_ERROR_TYPE_MISMATCH` is set.

References CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMAT↩
CH, cpl_parameter_get_name(), cpl_parameter_get_type(), cpl_type_get_name(), and CPL_TYPE_STRING.

4.33.4.15 cpl_parameter_get_double()

```
double cpl_parameter_get_double (
            const cpl_parameter * self )
```

Get the value of the given double parameter.

Parameters

self	An double parameter.

Returns

The function returns the parameter's double value. If an error occurs the function returns 0 and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The type of the parameter *self* is not of type `CPL_TYPE_INT`.

The current double value of the parameter *self* is retrieved.

References CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMAT↩CH, cpl_parameter_get_name(), cpl_parameter_get_type(), CPL_TYPE_DOUBLE, and cpl_type_get_name().

4.33.4.16 cpl_parameter_get_enum_double()

```
double cpl_parameter_get_enum_double (
            const cpl_parameter * self,
            int position )
```

Get the possible values for a double enumeration.

Parameters

self	An enumeration parameter
position	Position to look up in the list of possible values.

Returns

The double value at position *position* in the list of possible values of the enumeration *self*. In case of an error the function returns 0. and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The class of the parameter *self* is not of the kind `CPL_↩PARAMETER_CLASS_ENUM`, or *self* is not of type `CP↩L_TYPE_DOUBLE`.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The requested index position *position* is out of range.

The function retrieves the double value at position *position* from the list of enumeration values the parameter *self* can possibly take. For any valid enumeration parameter this list contains at leas one value. The possible values are counted starting from 0.

References CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMATCH, CPL_PARAMETER_CLASS_ENUM, cpl_parameter_get_enum_size(), cpl_parameter_get_name(), cpl_parameter_get_type(), CPL_TYPE_DOUBLE, and cpl_type_get_name().

4.33.4.17 cpl_parameter_get_enum_int()

```
int cpl_parameter_get_enum_int (
            const cpl_parameter * self,
            int position )
```

Get the possible values for an integer enumeration.

Parameters

self	An enumeration parameter
position	Position to look up in the list of possible values.

Returns

The integer value at position *position* in the list of possible values of the enumeration *self*. In case of an error the function returns 0 and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The class of the parameter *self* is not of the kind `CPL_PARAMETER_CLASS_ENUM`, or *self* is not of type `CPL_TYPE_INT`.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The requested index position *position* is out of range.

The function retrieves the integer value at position *position* from the list of enumeration values the parameter *self* can possibly take. For any valid enumeration parameter this list contains at least one value. The possible values are counted starting from 0.

References CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMATCH, CPL_PARAMETER_CLASS_ENUM, cpl_parameter_get_enum_size(), cpl_parameter_get_name(), cpl_parameter_get_type(), cpl_type_get_name(), and CPL_TYPE_INT.

4.33.4.18 cpl_parameter_get_enum_size()

```
int cpl_parameter_get_enum_size (
            const cpl_parameter * self )
```

Get the number of alternatives of an enumeration parameter.

Parameters

self	An enumeration parameter.

Returns

The number of possible values the parameter *self* can take, or 0 if *self* does not point to a valid parameter, or the parameter is not an enumeration parameter. In case of an error the function also sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The class of the parameter *self* is not of the kind `CPL_PARAMET↩ ER_CLASS_ENUM`.

The function retrieves the number of possible alternatives for an enumeration parameter, which is always greater or equal than 1.

References CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMAT↩ CH, CPL_PARAMETER_CLASS_ENUM, cpl_parameter_get_name(), cpl_parameter_get_type(), and cpl_type_↩ get_name().

Referenced by cpl_parameter_get_enum_double(), cpl_parameter_get_enum_int(), and cpl_parameter_get_↩ enum_string().

4.33.4.19 cpl_parameter_get_enum_string()

```
const char* cpl_parameter_get_enum_string (
            const cpl_parameter * self,
            int position )
```

Get the possible values for a string enumeration.

Parameters

self	An enumeration parameter
position	Position to look up in the list of possible values.

Returns

The string value at position *position* in the list of possible values of the enumeration *self*. In case of an error the function returns NULL and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_TYPE_MISMATCH	The class of the parameter *self* is not of the kind CPL_↩ PARAMETER_CLASS_ENUM, or *self* is not of type CP↩ L_TYPE_STRING.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The requested index position *position* is out of range.

The function retrieves the read-only string value at position *position* from the list of enumeration values the parameter *self* can possibly take. For any valid enumeration parameter this list contains at least one value. The possible values are counted starting from 0.

References CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error↩ _set_message, CPL_ERROR_TYPE_MISMATCH, CPL_PARAMETER_CLASS_ENUM, cpl_parameter_get_↩ enum_size(), cpl_parameter_get_name(), cpl_parameter_get_type(), cpl_type_get_name(), and CPL_TYPE_ST↩ RING.

4.33.4.20 cpl_parameter_get_help()

```
const char* cpl_parameter_get_help (
            const cpl_parameter * self )
```

Get the description of a parameter.

Parameters

self	A parameter.

Returns

The function returns the parameter description, or NULL if no description has been set. The function returns NULL if an error occurs and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function returns the read-only short help of the parameter *self*. The parameter description **must** not be modified using the returned pointer.

References CPL_ERROR_NULL_INPUT, and cpl_error_set.

4.33.4.21 cpl_parameter_get_id()

```
int cpl_parameter_get_id (
            const cpl_parameter * self )
```

Get the numerical identifier of the given parameter.

Parameters

self	A parameter.

Returns

The function returns the parameter's numerical identifier. If an error occurs the function returns 0 and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function looks up the numerical identifier of the given parameter *self*. A numerical identifier may be assigned to a parameter using **cpl_parameter_set_id()**.

See also

cpl_parameter_set_id()

References CPL_ERROR_NULL_INPUT, and cpl_error_set.

4.33.4.22 cpl_parameter_get_int()

```
int cpl_parameter_get_int (
            const cpl_parameter * self )
```

Get the value of the given integer parameter.

Parameters

self	An integer parameter.

Returns

The function returns the parameter's integer value. If an error occurs the function returns 0 and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The type of the parameter *self* is not of type `CPL_TYPE_INT`.

The current integer value of the parameter *self* is retrieved.

References CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMAT↩CH, cpl_parameter_get_name(), cpl_parameter_get_type(), cpl_type_get_name(), and CPL_TYPE_INT.

4.33.4.23 cpl_parameter_get_name()

```
const char* cpl_parameter_get_name (
            const cpl_parameter * self )
```

Get the name of a parameter.

Parameters

self	A parameter.

Returns

The function returns the parameter's unique name, or `NULL` if *self* is not a valid parameter but `NULL`. In the latter case an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function returns the read-only unique name of the parameter *self*. The parameter name **must** not be modified using the returned pointer.

References CPL_ERROR_NULL_INPUT, and cpl_error_set.

Referenced by cpl_parameter_disable(), cpl_parameter_enable(), cpl_parameter_get_alias(), cpl_parameter↩_get_bool(), cpl_parameter_get_default_bool(), cpl_parameter_get_default_double(), cpl_parameter_get_↩default_int(), cpl_parameter_get_default_string(), cpl_parameter_get_double(), cpl_parameter_get_enum_↩double(), cpl_parameter_get_enum_int(), cpl_parameter_get_enum_size(), cpl_parameter_get_enum_string(),

cpl_parameter_get_int(), cpl_parameter_get_range_max_double(), cpl_parameter_get_range_max_int(), cpl_parameter_get_range_min_double(), cpl_parameter_get_range_min_int(), cpl_parameter_get_string(), cpl_parameter_is_enabled(), cpl_parameter_set_alias(), cpl_parameter_set_bool(), cpl_parameter_set_default_bool(), cpl_parameter_set_default_double(), cpl_parameter_set_default_int(), cpl_parameter_set_default_string(), cpl_parameter_set_double(), cpl_parameter_set_int(), cpl_parameter_set_string(), and cpl_parameterlist_find_const().

4.33.4.24 cpl_parameter_get_range_max_double()

```
double cpl_parameter_get_range_max_double (
            const cpl_parameter * self )
```

Get the maximum value of a double range parameter.

Parameters

self	A double range parameter.

Returns

The function returns the maximum double value the range parameter *self* can take. In case of an error, the function returns 0. and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The class of the parameter *self* is not of the kind `CPL_PARAMETER_CLASS_RANGE`, or *self* is not of type `CPL_TYPE_DOUBLE`.

The function retrieves the double value defined to be the upper limit of the double range parameter *self*.

References CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMATCH, CPL_PARAMETER_CLASS_RANGE, cpl_parameter_get_name(), cpl_parameter_get_type(), CPL_TYPE_DOUBLE, and cpl_type_get_name().

4.33.4.25 cpl_parameter_get_range_max_int()

```
int cpl_parameter_get_range_max_int (
            const cpl_parameter * self )
```

Get the maximum value of an integer range parameter.

Parameters

self	An integer range parameter.

Returns

The function returns the maximum integer value the range parameter *self* can take. In case of an error, the function returns 0 and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The class of the parameter *self* is not of the kind `CPL_PARAMETER_CLASS_RANGE`, or *self* is not of type `CPL_TYPE_INT`.

The function retrieves the integer value defined to be the upper limit of the integer range parameter *self*.

References CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMATCH, CPL_PARAMETER_CLASS_RANGE, cpl_parameter_get_name(), cpl_parameter_get_type(), cpl_type_get_name(), and CPL_TYPE_INT.

4.33.4.26 cpl_parameter_get_range_min_double()

```
double cpl_parameter_get_range_min_double (
            const cpl_parameter * self )
```

Get the minimum value of a double range parameter.

Parameters

self	A double range parameter.

Returns

The function returns the minimum double value the range parameter *self* can take. In case of an error, the function returns 0. and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The class of the parameter *self* is not of the kind `CPL_PARAMETER_CLASS_RANGE`, or *self* is not of type `CPL_TYPE_DOUBLE`.

The function retrieves the double value defined to be the lower limit of the double range parameter *self*.

References CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMATCH, CPL_PARAMETER_CLASS_RANGE, cpl_parameter_get_name(), cpl_parameter_get_type(), CPL_TYPE_DOUBLE, and cpl_type_get_name().

4.33.4.27 cpl_parameter_get_range_min_int()

```
int cpl_parameter_get_range_min_int (
            const cpl_parameter * self )
```

Get the minimum value of an integer range parameter.

Parameters

self	An integer range parameter.

Returns

The function returns the minimum integer value the range parameter *self* can take. In case of an error, the function returns 0 and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The class of the parameter *self* is not of the kind `CPL_PARAMETER_CLASS_RANGE`, or *self* is not of type `CPL_TYPE_INT`.

The function retrieves the integer value defined to be the lower limit of the integer range parameter *self*.

References CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMATCH, CPL_PARAMETER_CLASS_RANGE, cpl_parameter_get_name(), cpl_parameter_get_type(), cpl_type_get_name(), and CPL_TYPE_INT.

4.33.4.28 cpl_parameter_get_string()

```
const char* cpl_parameter_get_string (
            const cpl_parameter * self )
```

Get the value of the given string parameter.

Parameters

self	A string parameter.

Returns

The function returns the parameter's string value. If an error occurs the function returns `NULL` and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The type of the parameter *self* is not of type `CPL_TYPE_STRING`.

The current string value of the parameter *self* is retrieved.

References CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMAT↩CH, cpl_parameter_get_name(), cpl_parameter_get_type(), cpl_type_get_name(), and CPL_TYPE_STRING.

4.33.4.29 cpl_parameter_get_tag()

```
const char* cpl_parameter_get_tag (
            const cpl_parameter * self )
```

Get the parameter's user tag.

Parameters

self	A parameter.

Returns

The function returns the parameter's user tag, or `NULL` if no user tag was previously set. The function returns `NULL` if an error occurs and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The current read-only setting of the user definable tag of the given parameter *self* is retrieved.

References CPL_ERROR_NULL_INPUT, and cpl_error_set.

Referenced by cpl_parameterlist_find_tag_const().

4.33.4.30 cpl_parameter_get_type()

```
cpl_type cpl_parameter_get_type (
            const cpl_parameter * self )
```

Get the parameter's value type.

Parameters

self	A parameter.

Returns

The function returns the parameter's value type. The function returns `CPL_TYPE_INVALID` if *self* is not a valid parameter, but `NULL`. In this case an appropriate error is also set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The type identifier of the parameter's value is retrieved from the given parameter *self*.

References CPL_ERROR_NULL_INPUT, cpl_error_set, and CPL_TYPE_INVALID.

Referenced by cpl_parameter_get_bool(), cpl_parameter_get_default_bool(), cpl_parameter_get_default_↩ double(), cpl_parameter_get_default_int(), cpl_parameter_get_default_string(), cpl_parameter_get_double(), cpl_parameter_get_enum_double(), cpl_parameter_get_enum_int(), cpl_parameter_get_enum_size(), cpl↩ _parameter_get_enum_string(), cpl_parameter_get_int(), cpl_parameter_get_range_max_double(), cpl_↩ parameter_get_range_max_int(), cpl_parameter_get_range_min_double(), cpl_parameter_get_range_min_int(), cpl_parameter_get_string(), cpl_parameter_set_bool(), cpl_parameter_set_default_bool(), cpl_parameter_set↩ _default_double(), cpl_parameter_set_default_int(), cpl_parameter_set_default_string(), cpl_parameter_set_↩ double(), cpl_parameter_set_int(), cpl_parameter_set_string(), and cpl_parameterlist_find_type_const().

4.33.4.31 cpl_parameter_is_enabled()

```
int cpl_parameter_is_enabled (
            const cpl_parameter * self,
            cpl_parameter_mode mode )
```

Get the parameter's activity status for the environment context.

Parameters

self	A parameter.
mode	The parameter mode.

Returns

The function returns a non-zero value if the parameter is enabled for the environment context, or 0 if the parameter is not active. If an error occurs the function returns 0 and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	The parameter mode *mode* is not supported.

The function returns whether the parameter is enabled for the environment context or not. If the parameter is enabled for the context the application may modify the parameter's value if the parameter is referenced in the context specific input of the application. If the parameter is disabled, the application must not modify the parameter's value.

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, cpl_parameter_get_name(), CPL_PARAMETER_MODE_CFG, CPL_PARAMETER_MODE_CLI, and CPL_PAR↩AMETER_MODE_ENV.

4.33.4.32 cpl_parameter_new_enum()

```
cpl_parameter* cpl_parameter_new_enum (
            const char * name,
            cpl_type type,
            const char * description,
            const char * context,
             ...  )
```

Create a new enumeration parameter.

Parameters

name	The parameter's unique name
type	The type of the parameter's current and default value.
description	An optional comment or description of the parameter.
context	The context to which the parameter belongs.
...	Arguments used for parameter initialisation.

Returns

The function returns the newly created parameter, or `NULL` if the parameter could not be created. In the latter case an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *name* is a `NULL` pointer.
CPL_ERROR_INVALID_TYPE	An unsupported type was passed as *type*.
CPL_ERROR_ILLEGAL_INPUT	The provided parameter initialisation data is invalid.

The function allocates memory for an enumeration parameter with the name *name* and a value type *type*. Optionally a comment describing the parameter may be passed by *description* and the context to which the parameter belongs may be given by *context*.

To properly initialise the newly created enumeration, the parameter's default value (together with the number of alternatives following) and the list of enumeration alternatives must be passed as the variable argument list arguments. The list of enumeration alternatives must contain the default value! Apart from the number of possible alternatives, which must be an argument of type `int`, the type of all initialisation arguments must match the parameter's value type as it is indicated by *type*. Enumeration parameters can only be created for the value types integer, double or string.

The following example creates a string enumeration parameter with the unique name `application.setup.enum` which belongs to the context `application.setup` with the default `first` and 3 enumeration alternatives.

Example:

```
const char *first = "first";
const char *second = "second";
const char *third = "third";
cpl_parameter *p = cpl_parameter_new_enum("application.setup.enum",
                                          CPL_TYPE_STRING,
                                          "An enum of type string",
                                          "application.setup",
                                          first, 3, first, second,
                                          third);
```

See also

cpl_parameter_new_value(), cpl_parameter_new_range(), cpl_parameter_new_enum_from_array()

References CPL_ERROR_NULL_INPUT, and cpl_error_set.

4.33.4.33 cpl_parameter_new_enum_from_array()

```
cpl_parameter* cpl_parameter_new_enum_from_array (
             const char * name,
             cpl_type type,
             const char * description,
             const char * context,
             cpl_size position,
             cpl_size size,
             const void * alternatives )
```

Create a new enumeration parameter from an array of alternative values.

Parameters

name	The parameter's unique name
type	The type of the parameter's current and default value.
description	An optional comment or description of the parameter.
context	The context to which the parameter belongs.
position	The index of the alternative to use as default value.
size	The size of the enumeration alternatives list.
alternatives	The array of enumeration alternatives.

Returns

The function returns the newly created parameter, or NULL if the parameter could not be created. In the latter case an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *name* is a NULL pointer.
CPL_ERROR_INVALID_TYPE	An unsupported type was passed as *type*.
CPL_ERROR_ILLEGAL_INPUT	The provided parameter initialisation data is invalid.

The function allocates memory for an enumeration parameter with the name *name* and a value type *type*. Optionally a comment describing the parameter may be passed by *description* and the context to which the parameter belongs may be given by *context*.

To properly initialise the newly created enumeration the index *position* of the parameter's default value in the array of alternatives *alternatives* must be given. The size of the array of alternatives is given by *size*. The type of all alternatives given must match the parameter's value type as it is indicated by *type*. Enumeration parameters can only be created for the value types integer, double or string.

The following example creates a string enumeration parameter with the unique name application.setup.enum which belongs to the context application.setup with the default first and 3 enumeration alternatives.

Example:

```
const char *alternatives = {"first", "second", "third"};
cpl_parameter *p =
    cpl_parameter_new_enum_array_from_array("application.setup.enum",
                                            CPL_TYPE_STRING,
                                            "An enum of type string",
                                            "application.setup",
                                            0, 3, alternatives);
```

See also

cpl_parameter_new_value(), cpl_parameter_new_range(), cpl_parameter_new_enum()

References CPL_ERROR_NULL_INPUT, and cpl_error_set.

4.33.4.34 cpl_parameter_new_range()

```
cpl_parameter* cpl_parameter_new_range (
            const char * name,
            cpl_type type,
            const char * description,
            const char * context,
             ...  )
```

Create a new range parameter.

Parameters

name	The parameter's unique name
type	The type of the parameter's current and default value.
description	An optional comment or description of the parameter.
context	The context to which the parameter belongs.
...	Arguments used for parameter initialisation.

Returns

The function returns the newly created parameter, or `NULL` if the parameter could not be created. In the latter case an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *name* is a `NULL` pointer.
CPL_ERROR_INVALID_TYPE	An unsupported type was passed as *type*.
CPL_ERROR_ILLEGAL_INPUT	The provided parameter initialisation data is invalid.

The function allocates memory for a range parameter with the name *name* and a value type *type*. Optionally a comment describing the parameter may be passed by *description* and the context to which the parameter belongs may be given by *context*.

To properly initialise the newly created range the parameters default value together with the minimum and maximum value of the range has to be passed as the variable argument list arguments. The type of all initialisation arguments must match the parameter's value type as it is indicated by *type*. Range parameters can only be created for the value types integer or double.

The following example creates a double range parameter with the unique name `application.setup.range` which belongs to the context `application.setup` with the default `def` and the range boundary values *minimum* and *maximum*.

Example:

```
double def = 0.5;
double min = 0.0;
double max = 1.0;
cpl_parameter *p = cpl_parameter_new_range("application.setup.range",
                                           CPL_TYPE_DOUBLE,
                                           "A range of type double",
                                           "application.setup",
                                           def, min, max);
```

See also

cpl_parameter_new_value(), cpl_parameter_new_enum()

References CPL_ERROR_NULL_INPUT, and cpl_error_set.

4.33.4.35 cpl_parameter_new_value()

```
cpl_parameter* cpl_parameter_new_value (
             const char * name,
             cpl_type type,
             const char * description,
             const char * context,
              ...  )
```

Create a new value parameter.

Parameters

name	The parameter's unique name
type	The type of the parameter's current and default value.
description	An optional comment or description of the parameter.
context	The context to which the parameter belongs.
...	Arguments used for parameter initialisation.

Returns

The function returns the newly created parameter, or `NULL` if the parameter could not be created. In the latter case an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *name* is a `NULL` pointer.
CPL_ERROR_INVALID_TYPE	An unsupported type was passed as *type*.
CPL_ERROR_ILLEGAL_INPUT	The provided parameter initialisation data is invalid.

The function allocates memory for a value parameter with the name *name* and a value type *type*. Optionally a comment describing the parameter may be passed by *description* and the context to which the parameter belongs may be given by *context*.

The newly created parameter is initialised with the default value passed to the function as the only variable argument list argument. The type of the value must match the parameter's value type as it is indicated by *type*.

The following example creates an integer value parameter with the unique name `application.setup.value` which belongs to the context `application.setup` with the default value `def`

Example:

```
int def = 1;
cpl_parameter *p = cpl_parameter_new_value("application.setup.value",
                                           CPL_TYPE_INT,
                                           "An integer value",
                                           "application.setup",
                                           def);
```

See also

cpl_parameter_new_range(), cpl_parameter_new_enum()

References CPL_ERROR_NULL_INPUT, and cpl_error_set.

4.33.4.36 cpl_parameter_set_alias()

```
cpl_error_code cpl_parameter_set_alias (
            cpl_parameter * self,
            cpl_parameter_mode mode,
            const char * alias )
```

Set alias names for the given parameter.

Parameters

self	A parameter
mode	The parameter mode for which the alias should be set.
alias	The parameter's alias.

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_ILLEGAL_INPUT	The parameter mode *mode* is not supported.

The function assigns an alternative name to the parameter *self* for the given mode *mode*. This alias name may be used instead of the fully qualified parameter name in the context for which they have been set. If the alias name is NULL a previously set alias is deleted.

The context for which an alias is set is selected by the *mode*. The functions accepts the parameter modes CPL_↩PARAMETER_MODE_CFG, CPL_PARAMETER_MODE_CLI and CPL_PARAMETER_MODE_ENV.

See also

cpl_parameter_mode

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, cpl_parameter_get_name(), CPL_PARAMETER_MODE_CFG, CPL_PARAMETER_M↩ODE_CLI, and CPL_PARAMETER_MODE_ENV.

4.33.4.37 cpl_parameter_set_bool()

```
cpl_error_code cpl_parameter_set_bool (
            cpl_parameter * self,
            int value )
```

Assign a boolean value to a parameter.

Parameters

self	The parameter the value is assigned to.
value	The boolean value to be assigned.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The type of the parameter *self* is not of type `CPL_TYPE_BOOL`.

The function assigns a boolean value *value* to a parameter of type `CPL_TYPE_BOOL`. Any non-zero value passed as *value* is interpreted as *true*.

References CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMAT↩CH, cpl_parameter_get_name(), cpl_parameter_get_type(), CPL_TYPE_BOOL, and cpl_type_get_name().

4.33.4.38 cpl_parameter_set_default_bool()

```
cpl_error_code cpl_parameter_set_default_bool (
            cpl_parameter * self,
            int value )
```

Modify the default value of a boolean parameter.

Parameters

self	The parameter whose default value is modified.
value	The boolean value to be assigned.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The type of the parameter *self* is not of type `CPL_TYPE_BOOL`.

The function changes the default value of the parameter *self* to the boolean value *value*. The parameter *self* must

be of type CPL_TYPE_BOOL. Any non-zero value passed as *value* is interpreted as *true*.

References CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMAT↩CH, cpl_parameter_get_name(), cpl_parameter_get_type(), CPL_TYPE_BOOL, and cpl_type_get_name().

4.33.4.39 cpl_parameter_set_default_double()

```
cpl_error_code cpl_parameter_set_default_double (
            cpl_parameter * self,
            double value )
```

Modify the default value of a double parameter.

Parameters

self	The parameter whose default value is modified.
value	The double value to be assigned.

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_TYPE_MISMATCH	The type of the parameter *self* is not of type CPL_TYPE_DOUBLE.

The function changes the default value of the parameter *self* to the double value *value*. The parameter *self* must be of type CPL_TYPE_DOUBLE.

References CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMAT↩CH, cpl_parameter_get_name(), cpl_parameter_get_type(), CPL_TYPE_DOUBLE, and cpl_type_get_name().

4.33.4.40 cpl_parameter_set_default_flag()

```
cpl_error_code cpl_parameter_set_default_flag (
            cpl_parameter * self,
            int status )
```

Change the presence status flag of the given parameter.

Parameters

self	A parameter.
status	The presence status value to assign.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function sets the presence status flag of the given parameter *self* to the value *status*. Any non-zero value means that the parameter is present. If the presence status should be changed to 'not present' the argument *status* must be 0.

See also

cpl_parameter_get_default_flag()

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl_error_set.

4.33.4.41 cpl_parameter_set_default_int()

```
cpl_error_code cpl_parameter_set_default_int (
            cpl_parameter * self,
            int value )
```

Modify the default value of an integer parameter.

Parameters

self	The parameter whose default value is modified.
value	The integer value to be assigned.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The type of the parameter *self* is not of type `CPL_TYPE_INT`.

The function changes the default value of the parameter *self* to the integer value *value*. The parameter *self* must be of type `CPL_TYPE_INT`.

References CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMAT↩
CH, cpl_parameter_get_name(), cpl_parameter_get_type(), cpl_type_get_name(), and CPL_TYPE_INT.

4.33.4.42 cpl_parameter_set_default_string()

```
cpl_error_code cpl_parameter_set_default_string (
            cpl_parameter * self,
            const char * value )
```

Modify the default value of a string parameter.

Parameters

self	The parameter whose default value is modified.
value	The string value to be assigned.

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_TYPE_MISMATCH	The type of the parameter *self* is not of type CPL_TYPE_STRING.

The function changes the default value of the parameter *self* to the string value *value*. The parameter *self* must be of type CPL_TYPE_STRING.

References CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMAT↩
CH, cpl_parameter_get_name(), cpl_parameter_get_type(), cpl_type_get_name(), and CPL_TYPE_STRING.

4.33.4.43 cpl_parameter_set_double()

```
cpl_error_code cpl_parameter_set_double (
            cpl_parameter * self,
            double value )
```

Assign a double value to a parameter.

Parameters

self	The parameter the value is assigned to.
value	The double value to be assigned.

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_TYPE_MISMATCH	The type of the parameter *self* is not of type CPL_TYPE_DOUBLE.

The function assigns a double value *value* to a parameter of type CPL_TYPE_DOUBLE.

References CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMAT↩CH, cpl_parameter_get_name(), cpl_parameter_get_type(), CPL_TYPE_DOUBLE, and cpl_type_get_name().

4.33.4.44 cpl_parameter_set_id()

```
cpl_error_code cpl_parameter_set_id (
            cpl_parameter * self,
            int id )
```

Set the numerical identifier of the given parameter.

Parameters

self	A parameter.
id	Numerical identifier to assign.

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function assigns a numerical identifier to the parameter *self*. The numerical value to be assigned to the parameter's numerical identifier member is passed as the argument *id*. The function does not do any checks on the numerical value of *id*. The numerical identifier may be used by an application to, for instance, assign a sequence number to the parameter.

See also

cpl_parameter_get_id()

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl_error_set.

4.33.4.45 cpl_parameter_set_int()

```
cpl_error_code cpl_parameter_set_int (
            cpl_parameter * self,
            int value )
```

Assign an integer value to a parameter.

Parameters

self	The parameter the value is assigned to.
value	The integer value to be assigned.

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_TYPE_MISMATCH	The type of the parameter *self* is not of type CPL_TYPE_INT.

The function assigns an integer value *value* to a parameter of type CPL_TYPE_INT.

References CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMAT↩CH, cpl_parameter_get_name(), cpl_parameter_get_type(), cpl_type_get_name(), and CPL_TYPE_INT.

4.33.4.46 cpl_parameter_set_string()

```
cpl_error_code cpl_parameter_set_string (
            cpl_parameter * self,
            const char * value )
```

Assign a string value to a parameter.

Parameters

self	The parameter the value is assigned to.
value	The string value to be assigned.

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The type of the parameter *self* is not of type `CPL_TYPE_STRING`.

The function assigns a string value *value* to a parameter of type `CPL_TYPE_STRING`.

References CPL_ERROR_NULL_INPUT, cpl_error_set, cpl_error_set_message, CPL_ERROR_TYPE_MISMAT↩CH, cpl_parameter_get_name(), cpl_parameter_get_type(), cpl_type_get_name(), and CPL_TYPE_STRING.

4.33.4.47 cpl_parameter_set_tag()

```
cpl_error_code cpl_parameter_set_tag (
            cpl_parameter * self,
            const char * tag )
```

Set the tag of the given parameter.

Parameters

self	A parameter.
tag	Tag string to assign.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function assigns a user definable tag *tag* to the parameter *self*. The function does not check the passed string *tag* in any way. The tag may be used by an application but it cannot rely on the contents of the parameter's tag.

See also

cpl_parameter_get_tag()

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl_error_set.

4.34 Plotting of CPL objects

Functions

- cpl_error_code cpl_plot_bivector (const char *pre, const char *options, const char *post, const cpl_bivector *bivector)

  Plot a bivector.
- cpl_error_code cpl_plot_bivectors (const char *pre, const char **options, const char *post, const cpl_bivector **bivectors, cpl_size nbvec)

  Plot an array of bivectors.
- cpl_error_code cpl_plot_column (const char *pre, const char *options, const char *post, const cpl_table *tab, const char *xlab, const char *ylab)

  Plot a column of a table.
- cpl_error_code cpl_plot_columns (const char *pre, const char *options, const char *post, const cpl_table *tab, const char **labels, cpl_size nlabels)

  Plot severals column of a table.
- cpl_error_code cpl_plot_image (const char *pre, const char *options, const char *post, const cpl_image *image)

  Plot an image.
- cpl_error_code cpl_plot_image_col (const char *pre, const char *options, const char *post, const cpl_image *image, cpl_size firstcol, cpl_size lastcol, cpl_size colstep)

  Plot a range of image columns.
- cpl_error_code cpl_plot_image_row (const char *pre, const char *options, const char *post, const cpl_image *image, cpl_size firstrow, cpl_size lastrow, cpl_size rowstep)

  Plot a range of image rows.
- cpl_error_code cpl_plot_mask (const char *pre, const char *options, const char *post, const cpl_mask *mask)

  Plot a mask.
- cpl_error_code cpl_plot_vector (const char *pre, const char *options, const char *post, const cpl_vector *vector)

  Plot a vector.
- cpl_error_code cpl_plot_vectors (const char *pre, const char *options, const char *post, const cpl_vector **vectors, cpl_size nvec)

  Plot an array of vectors.

4.34.1 Detailed Description

This module provides functions to plot basic CPL objects

This module is offered to help during the development process. The functions offered should NOT be used in any operational environment. For that reason, the support of those remains limited, and no functionality extension can be expected from the CPL team.

The created plot windows can be closed by pressing the 'q' key like you would do with a normal gnuplot window.

The default behaviour of the plotting is to use gnuplot (with option -persist). The user can control the actual plotting-command used to create the plot by setting the environment variable CPL_PLOTTER. Currently, if CPL_PLOTTER is set it must contain the string 'gnuplot'. Setting it to 'cat > my_gnuplot_$$.txt' causes a number of ASCII-files to be created, which each produce a plot when given as standard input to gnuplot.

A finer control of the plotting options can be obtained by writing an executable script, e.g. my_gnuplot, that executes gnuplot after setting the desired gnuplot options (e.g. set terminal pslatex color) and then setting CPL_PLOTTER to my_gnuplot.

Images can be plotted not only with gnuplot, but also using the pnm format. This is controlled with the environment variable CPL_IMAGER. If CPL_IMAGER is set to a string that does not contain the word gnuplot, the recipe will generate the plot in pnm format. E.g. setting CPL_IMAGER to 'display - &' will produce a gray-scale image using the image viewer display.

```
#include "cpl_plot.h"
```

4.34.2 Function Documentation

4.34.2.1 cpl_plot_bivector()

```
cpl_error_code cpl_plot_bivector (
            const char * pre,
            const char * options,
            const char * post,
            const cpl_bivector * bivector )
```

Plot a bivector.

Parameters

pre	An optional string with pre-plot commands
options	An optional string with plotting options
post	An optional string with post-plot commands
bivector	The bivector to plot

Returns

CPL_ERROR_NONE or the relevant CPL_ERROR_# on error

The bivector must have a positive number of elements.

See also

also `cpl_mplot_open()`.

Possible *cpl_error_code* set in this function:

- CPL_ERROR_FILE_IO
- CPL_ERROR_NULL_INPUT
- CPL_ERROR_ILLEGAL_INPUT
- CPL_ERROR_UNSUPPORTED_MODE if plotting is unsupported on the specific run-time system.

References cpl_bivector_get_size(), cpl_bivector_get_x_data_const(), cpl_bivector_get_y_data_const(), and CP↩L_ERROR_NONE.

Referenced by cpl_plot_column().

4.34.2.2 cpl_plot_bivectors()

```
cpl_error_code cpl_plot_bivectors (
            const char * pre,
            const char ** options,
            const char * post,
            const cpl_bivector ** bivectors,
            cpl_size nbvec )
```

Plot an array of bivectors.

Parameters

pre	An optional string with pre-plot commands
options	Array of strings with plotting options
post	An optional string with post-plot commands
bivectors	The bivectors array to plot
nbvec	The number of bivectors, at least one is required

Returns

CPL_ERROR_NONE or the relevant CPL_ERROR_# on error

Each bivector in the array defines a sequence of points to be plotted. The bivectors can have different size.

The *options* array must be of same size as the *bivectors* array. The i'th string in the array specifies the plotting options for the i'th bivector.

See also

also `cpl_mplot_open()`.

Possible *cpl_error_code* set in this function:

- CPL_ERROR_FILE_IO
- CPL_ERROR_NULL_INPUT
- CPL_ERROR_DATA_NOT_FOUND
- CPL_ERROR_UNSUPPORTED_MODE if plotting is unsupported on the specific run-time system.

References cpl_ensure_code, CPL_ERROR_DATA_NOT_FOUND, CPL_ERROR_NULL_INPUT, cpl_errorstate↩_get(), cpl_malloc(), CPL_SIZE_FORMAT, and cpl_sprintf().

4.34.2.3 cpl_plot_column()

```
cpl_error_code cpl_plot_column (
            const char * pre,
            const char * options,
            const char * post,
            const cpl_table * tab,
            const char * xlab,
            const char * ylab )
```

Plot a column of a table.

Parameters

pre	An optional string with pre-plot commands
options	An optional string with plotting options
post	An optional string with post-plot commands
tab	The table to plot
xlab	The label of the column used in x
ylab	The label of the column used in y

Returns

CPL_ERROR_NONE or the relevant CPL_ERROR_# on error

See also

also `cpl_mplot_open()`.

If xlab is NULL, the sequence number is used for X.

Possible *cpl_error_code* set in this function:

- CPL_ERROR_FILE_IO
- CPL_ERROR_NULL_INPUT
- CPL_ERROR_ILLEGAL_INPUT
- CPL_ERROR_DATA_NOT_FOUND
- CPL_ERROR_INVALID_TYPE
- CPL_ERROR_UNSUPPORTED_MODE if plotting is unsupported on the specific run-time system.

References cpl_bivector_unwrap_vectors(), cpl_bivector_wrap_vectors(), cpl_ensure_code, CPL_ERROR_DATA_NOT_FOUND, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), cpl_errorstate_is_equal(), cpl_plot_bivector(), cpl_table_cast_column(), cpl_table_count_invalid(), cpl_table_delete(), cpl_table_duplicate_column(), cpl_table_erase_invalid(), cpl_table_get_column_type(), cpl_table_get_data_double(), cpl_table_get_data_double_const(), cpl_table_get_nrow(), cpl_table_has_column(), cpl_table_new(), CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT, CPL_TYPE_INT, cpl_vector_delete(), cpl_vector_duplicate(), cpl_vector_get_size(), cpl_vector_set(), cpl_vector_unwrap(), and cpl_vector_wrap().

4.34.2.4 cpl_plot_columns()

```
cpl_error_code cpl_plot_columns (
             const char * pre,
             const char * options,
             const char * post,
             const cpl_table * tab,
             const char ** labels,
             cpl_size nlabels )
```

Plot severals column of a table.

Parameters

pre	An optional string with pre-plot commands
options	An optional string with plotting options
post	An optional string with post-plot commands
tab	The table to plot
labels	The labels of the columns
nlabels	The number of labels

Returns

CPL_ERROR_NONE or the relevant CPL_ERROR_# on error

See also

also `cpl_mplot_open()`.

If xlab is NULL, the sequence number is used for X.

Possible *cpl_error_code* set in this function:

- CPL_ERROR_FILE_IO
- CPL_ERROR_NULL_INPUT
- CPL_ERROR_ILLEGAL_INPUT
- CPL_ERROR_DATA_NOT_FOUND
- CPL_ERROR_INVALID_TYPE
- CPL_ERROR_UNSUPPORTED_MODE if plotting is unsupported on the specific run-time system.

References cpl_ensure_code, CPL_ERROR_DATA_NOT_FOUND, CPL_ERROR_INVALID_TYPE, CPL_ERR↩OR_NULL_INPUT, cpl_errorstate_get(), cpl_malloc(), CPL_SIZE_FORMAT, cpl_sprintf(), cpl_table_cast_column(), cpl_table_duplicate_column(), cpl_table_get_column_type(), cpl_table_get_data_double_const(), cpl_table_get_↩nrow(), cpl_table_has_column(), cpl_table_new(), CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT, and CPL_TYPE_I↩NT.

4.34.2.5 cpl_plot_image()

```
cpl_error_code cpl_plot_image (
             const char * pre,
             const char * options,
             const char * post,
             const cpl_image * image )
```

Plot an image.

Parameters

pre	An optional string with pre-plot commands
options	An optional string with plotting options
post	An optional string with post-plot commands
image	The image to plot

Returns

CPL_ERROR_NONE or the relevant CPL_ERROR_# on error

The image must have a positive number of pixels.

See also

also `cpl_image_open()`.

If the specified plotting command does not contain the string 'gnuplot', the plotting command is assumed to be able to parse a pgm (P5) image from stdin. Valid examples of such a command may include 'cat > myplot$$.pgm' and 'display - &'.

The 'pre' and 'post' commands are ignored in PGM-plots, while the 'options' string is written as a comment in the header of the image.

See also cpl_plot_vector().

Possible *cpl_error_code* set in this function:

- CPL_ERROR_FILE_IO
- CPL_ERROR_NULL_INPUT
- CPL_ERROR_ILLEGAL_INPUT
- CPL_ERROR_UNSUPPORTED_MODE if plotting is unsupported on the specific run-time system.

References CPL_ERROR_NONE, cpl_image_get_size_x(), and cpl_image_get_size_y().

4.34.2.6 cpl_plot_image_col()

```
cpl_error_code cpl_plot_image_col (
            const char * pre,
            const char * options,
            const char * post,
            const cpl_image * image,
            cpl_size firstcol,
            cpl_size lastcol,
            cpl_size colstep )
```

Plot a range of image columns.

Parameters

pre	An optional string with pre-plot commands
options	An optional string with plotting options
post	An optional string with post-plot commands
image	The image to plot
firstcol	The first column to plot (1 for first)
lastcol	The last column to plot
colstep	The positive column stride

Returns

CPL_ERROR_NONE or the relevant CPL_ERROR_# on error

The image must have a positive number of pixels.

lastcol shall be greater than or equal to firstcol.

See also

also `cpl_mplot_open()`.

Possible *cpl_error_code* set in this function:

- CPL_ERROR_FILE_IO
- CPL_ERROR_NULL_INPUT
- CPL_ERROR_ILLEGAL_INPUT
- CPL_ERROR_ACCESS_OUT_OF_RANGE (firstcol or lastcol are out of range)
- CPL_ERROR_UNSUPPORTED_MODE if plotting is unsupported on the specific run-time system.

References cpl_ensure_code, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), cpl_image_get_size_x(), and cpl_image_get_size_y().

4.34.2.7 cpl_plot_image_row()

```
cpl_error_code cpl_plot_image_row (
             const char * pre,
             const char * options,
             const char * post,
             const cpl_image * image,
             cpl_size firstrow,
             cpl_size lastrow,
             cpl_size rowstep )
```

Plot a range of image rows.

Parameters

pre	An optional string with pre-plot commands
options	An optional string with plotting options
post	An optional string with post-plot commands
image	The image to plot
firstrow	The first row to plot (1 for first)
lastrow	The last row to plot
rowstep	The positive row stride

Returns

CPL_ERROR_NONE or the relevant CPL_ERROR_# on error

The image must have a positive number of pixels.

lastrow shall be greater than or equal to firstrow.

See also

also `cpl_mplot_open()`.

Possible *cpl_error_code* set in this function:

- CPL_ERROR_FILE_IO
- CPL_ERROR_NULL_INPUT
- CPL_ERROR_ILLEGAL_INPUT
- CPL_ERROR_ACCESS_OUT_OF_RANGE (firstrow or lastrow are out of range)
- CPL_ERROR_UNSUPPORTED_MODE if plotting is unsupported on the specific run-time system.

References cpl_ensure_code, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, CP↩L_ERROR_NULL_INPUT, cpl_errorstate_get(), cpl_image_get_size_x(), and cpl_image_get_size_y().

4.34.2.8 cpl_plot_mask()

```
cpl_error_code cpl_plot_mask (
            const char * pre,
            const char * options,
            const char * post,
            const cpl_mask * mask )
```

Plot a mask.

Parameters

pre	An optional string with pre-plot commands
options	An optional string with plotting options
post	An optional string with post-plot commands
mask	The mask to plot

Returns

CPL_ERROR_NONE or the relevant CPL_ERROR_# on error

If the specified plotting command does not contain the string 'gnuplot', the plotting command is assumed to be able to parse a pgm (P5) mask from stdin. Valid examples of such a command may include 'cat > myplot$$.pgm' and 'display - &'.

The 'pre' and 'post' commands are ignored in PGM-plots, while the 'options' string is written as a comment in the header of the mask.

See also cpl_plot_vector().

Possible *cpl_error_code* set in this function:

- CPL_ERROR_FILE_IO
- CPL_ERROR_NULL_INPUT
- CPL_ERROR_ILLEGAL_INPUT
- CPL_ERROR_UNSUPPORTED_MODE if plotting is unsupported on the specific run-time system.

References CPL_ERROR_NONE, cpl_mask_get_data_const(), cpl_mask_get_size_x(), and cpl_mask_get_size↩_y().

4.34.2.9 cpl_plot_vector()

```
cpl_error_code cpl_plot_vector (
            const char * pre,
            const char * options,
            const char * post,
            const cpl_vector * vector )
```

Plot a vector.

Parameters

pre	An optional string with pre-plot commands
options	An optional string with plotting options
post	An optional string with post-plot commands
vector	The vector to plot

Returns

CPL_ERROR_NONE or the relevant CPL_ERROR_# on error

The vector must have a positive number of elements.

Possible *cpl_error_code* set in this function:

- CPL_ERROR_FILE_IO
- CPL_ERROR_NULL_INPUT
- CPL_ERROR_ILLEGAL_INPUT
- CPL_ERROR_UNSUPPORTED_MODE if plotting is unsupported on the specific run-time system.

References CPL_ERROR_NONE, cpl_vector_get_data_const(), and cpl_vector_get_size().

4.34.2.10 cpl_plot_vectors()

```
cpl_error_code cpl_plot_vectors (
            const char * pre,
            const char * options,
            const char * post,
            const cpl_vector ** vectors,
            cpl_size nvec )
```

Plot an array of vectors.

Parameters

pre	An optional string with pre-plot commands
options	An optional string with plotting options
post	An optional string with post-plot commands
vectors	The vectors array to plot
nvec	The number of vectors

Returns

CPL_ERROR_NONE or the relevant CPL_ERROR_# on error

The array should contain at least 3 vectors, the first one can be NULL.

The non-NULL vectors must have the same number of elements. The first vector gives the x-axis. If NULL, the index is used.

See also

also `cpl_mplot_open()`.

Possible *cpl_error_code* set in this function:

- CPL_ERROR_FILE_IO
- CPL_ERROR_NULL_INPUT
- CPL_ERROR_ILLEGAL_INPUT
- CPL_ERROR_UNSUPPORTED_MODE if plotting is unsupported on the specific run-time system.

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), cpl_malloc(), CPL_SIZE_FORMAT, cpl_sprintf(), cpl_vector_get_data_const(), and cpl_vector_get_size().

4.35 Plugin Interface

The basic plugin interface definition.

Classes

- struct _cpl_plugin_

  The type representation of the generic plugin interface.

Macros

- #define CPL_PLUGIN_API

  Plugin API version.

Typedefs

- typedef struct _cpl_plugin_ cpl_plugin

  The plugin data type.
- typedef enum _cpl_plugin_type_ cpl_plugin_type

  Data type used to store the plugin type code.

Enumerations

- enum _cpl_plugin_type_ {
  CPL_PLUGIN_TYPE_NONE = 0,
  CPL_PLUGIN_TYPE_RECIPE = 1 << 0,
  CPL_PLUGIN_TYPE_RECIPE_V2 = (1 << 1) | CPL_PLUGIN_TYPE_RECIPE }

  Definition of plugin types.

Functions

- cpl_error_code cpl_plugin_copy (cpl_plugin *self, const cpl_plugin *other)

  Copy a plugin.
- void cpl_plugin_delete (cpl_plugin *self)

  Destroy a plugin.
- void cpl_plugin_dump (const cpl_plugin *self, FILE *stream)

  Dump the plugin debugging information to the given stream.
- unsigned int cpl_plugin_get_api (const cpl_plugin *self)

  Get the version number of the plugin interface implementation.
- const char * cpl_plugin_get_author (const cpl_plugin *self)

  Get the name of the plugin author.
- const char * cpl_plugin_get_copyright (const cpl_plugin *self)

  Get the license and copyright information of a plugin.
- cpl_plugin_func cpl_plugin_get_deinit (const cpl_plugin *self)

  Get the cleanup handler of a plugin.
- const char * cpl_plugin_get_description (const cpl_plugin *self)

  Get the detailed description of a plugin.

- const char * cpl_plugin_get_email (const cpl_plugin *self)

  Get the contact information of a plugin.
- cpl_plugin_func cpl_plugin_get_exec (const cpl_plugin *self)

  Get the execution handler of a plugin.
- int cpl_plugin_get_info (cpl_pluginlist *cpl_plugin_list)

  Append the plugin information to the given list.
- cpl_plugin_func cpl_plugin_get_init (const cpl_plugin *self)

  Get the initialisation handler of a plugin.
- const char * cpl_plugin_get_name (const cpl_plugin *self)

  Get the name of a plugin.
- const char * cpl_plugin_get_synopsis (const cpl_plugin *self)

  Get the short description of a plugin.
- unsigned long cpl_plugin_get_type (const cpl_plugin *self)

  Get the type of a plugin.
- char * cpl_plugin_get_type_string (const cpl_plugin *self)

  Get the type of a plugin as string.
- unsigned long cpl_plugin_get_version (const cpl_plugin *self)

  Get the version number of a plugin.
- char * cpl_plugin_get_version_string (const cpl_plugin *self)

  Get the version number of a plugin as a string.
- cpl_error_code cpl_plugin_init (cpl_plugin *self, unsigned int api, unsigned long version, unsigned long type, const char *name, const char *synopsis, const char *description, const char *author, const char *email, const char *copyright, cpl_plugin_func create, cpl_plugin_func execute, cpl_plugin_func destroy)

  Initialise a plugin.
- cpl_plugin * cpl_plugin_new (void)

  Create a new, empty plugin interface.
- cpl_error_code cpl_plugin_set_api (cpl_plugin *self, unsigned int api)

  Set the plugin interface version number.
- cpl_error_code cpl_plugin_set_author (cpl_plugin *self, const char *author)

  Set the name of the plugin author.
- cpl_error_code cpl_plugin_set_copyright (cpl_plugin *self, const char *copyright)

  Set the license and copyright information of a plugin.
- cpl_error_code cpl_plugin_set_deinit (cpl_plugin *self, cpl_plugin_func func)

  Set the cleanup handler of a plugin.
- cpl_error_code cpl_plugin_set_description (cpl_plugin *self, const char *description)

  Set the detailed description of a plugin.
- cpl_error_code cpl_plugin_set_email (cpl_plugin *self, const char *email)

  Set the contact information of a plugin.
- cpl_error_code cpl_plugin_set_exec (cpl_plugin *self, cpl_plugin_func func)

  Set the execution handler of a plugin.
- cpl_error_code cpl_plugin_set_init (cpl_plugin *self, cpl_plugin_func func)

  Set the initialisation handler of a plugin.
- cpl_error_code cpl_plugin_set_name (cpl_plugin *self, const char *name)

  Set the name of a plugin.
- cpl_error_code cpl_plugin_set_synopsis (cpl_plugin *self, const char *synopsis)

  Set the short description of a plugin.
- cpl_error_code cpl_plugin_set_type (cpl_plugin *self, unsigned long type)

  Set the type of a plugin.
- int cpl_plugin_set_version (cpl_plugin *self, unsigned long version)

  Set the version number of a plugin.

4.35.1 Detailed Description

The basic plugin interface definition.

This module defines the basic plugin interface. The plugin interface provides the possibility to dynamically load and execute software modules. The basic plugin interface defined here serves as 'superclass' for context specific plugins. All context specific plugins inherit this basic plugin interface. A plugin context is represented by a type code, i.e. the different plugin contexts are represented as different plugin types.

Most of the time an application using the plugin interface is dealing only with this basic, plugin type independent part of the interface. It provides the application with the necessary information about a particular plugin implementation and the services to initialise, execute and cleanup a dynamically loaded module.

In this way plugin type specific details may remain hidden from the application until the plugin type and its implementation details are known through querying the basic plugin interface part.

To obtain a filled plugin interface structure the application will call the function `cpl_plugin_get_info()`, which has the following prototype:

```
#include <cpl_pluginlist.h>
cpl_plugin_get_info(cpl_pluginlist *list);
```

For each plugin library (a shared object library containing one or more plugins) this function must be implemented by the plugin developer. Its purpose is to fill a plugin interface structure for each plugin the plugin library contains and add it to a list provided by the application. This list of plugin interfaces provides the application with all the details on how to communicate with a particular plugin.

As an example on how to create a context specific plugin, i.e. how to create a new plugin type, you may have a look at Recipes.

Synopsis:

```
#include <cpl_plugin.h>
```

4.35.2 Macro Definition Documentation

4.35.2.1 CPL_PLUGIN_API

```
#define CPL_PLUGIN_API
```

Plugin API version.

4.35.3 Typedef Documentation

4.35.3.1 cpl_plugin

```
typedef struct _cpl_plugin_ cpl_plugin
```

The plugin data type.

This defines the (public) plugin data type.

4.35.3.2 cpl_plugin_type

```
typedef enum _cpl_plugin_type_ cpl_plugin_type
```

Data type used to store the plugin type code.

4.35.4 Enumeration Type Documentation

4.35.4.1 _cpl_plugin_type_

```
enum _cpl_plugin_type_
```

Definition of plugin types.

Predefined plugin types supported by the Common Pipeline Library itself.

Enumerator

CPL_PLUGIN_TYPE_NONE	Plugin is of unknown or undefined type
CPL_PLUGIN_TYPE_RECIPE	Plugin is a complete data reduction task, i.e. a sequence of individual data reduction steps, turning a raw frame into a 'final' product.
CPL_PLUGIN_TYPE_RECIPE_V2	Plugin is a recipe, i.e. a complete data reduction task. In addition, this recipe version provides extra data about the required input data. This plugin is a subclass of `CPL_PLUGIN_TYPE_RECIPE`.

4.35.5 Function Documentation

4.35.5.1 cpl_plugin_copy()

```
cpl_error_code cpl_plugin_copy (
            cpl_plugin * self,
            const cpl_plugin * other )
```

Copy a plugin.

Parameters

self	A plugin.
other	The plugin structure to copy.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *other* is a `NULL` pointer.

The function copies all members of the plugin *other* to the plugin *self*. The plugin *other* and its copy *self* do not share any resources after the copy operation. If either *self*, or *other* are invalid pointers, the function returns immediately.

Note that the plugins *self* and *other* do not need to be of the same kind (see below).

Attention

If a **derived** plugin (a `cpl_recipe` for instance) is passed to this function, after casting it to its base, a `cpl_plugin` this function most likely will not work as expected, since **data** slicing will happen. The function is only aware of the `cpl_plugin` part of the derived plugin and only these data members are copied. Any additional data members defined by the derived plugin are therefore lost, unless they are copied explicitly.

The should be used function as follows. If necessary, create the target plugin using the appropriate constructor, or allocate a memory block of the appropriate size to hold the **derived** plugin type. Use this function to copy the `cpl_plugin` part of your derived plugin. Copy additional data members of the target plugin explicitly.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.35.5.2 cpl_plugin_delete()

```
void cpl_plugin_delete (
             cpl_plugin * self )
```

Destroy a plugin.

Parameters

self	The plugin to destroy.

Returns

Nothing.

The function destroys a plugin. First, the memory used by the members of the `cpl_plugin` type is released and then the plugin itself is destroyed.

Attention

The function may also be used to destroy plugins which have been derived from the `cpl_plugin` type. But if the derived plugin type defines members which are dynamically allocated, they have to be destroyed explicitly before (in the plugin's cleanup handler for instance) the derived plugin is passed to this function, or the references to these memory blocks have to be kept and cleaned up later. Otherwise memory leaks may result. If the plugin *self* is `NULL`, nothing is done and no error is set.

Referenced by cpl_pluginlist_delete().

4.35.5.3 cpl_plugin_dump()

```
void cpl_plugin_dump (
            const cpl_plugin * self,
            FILE * stream )
```

Dump the plugin debugging information to the given stream.

Parameters

self	The plugin.
stream	The output stream to use.

Returns

Nothing.

The function dumps the contents of of the plugin *self* to the output stream *stream*. If *stream* is `NULL` the function writes to the standard output. If *self* is `NULL` the function does nothing.

References cpl_plugin_get_api(), cpl_plugin_get_author(), cpl_plugin_get_copyright(), cpl_plugin_get_deinit(), cpl_plugin_get_description(), cpl_plugin_get_email(), cpl_plugin_get_exec(), cpl_plugin_get_init(), cpl_plugin_get_name(), cpl_plugin_get_synopsis(), cpl_plugin_get_type(), cpl_plugin_get_type_string(), cpl_plugin_get_version(), and cpl_plugin_get_version_string().

Referenced by cpl_pluginlist_dump().

4.35.5.4 cpl_plugin_get_api()

```
unsigned int cpl_plugin_get_api (
            const cpl_plugin * self )
```

Get the version number of the plugin interface implementation.

Parameters

self	A plugin.

Returns

The version number of the plugin interface implementation the plugin *self* complies with. If an error occurs the function returns 0 and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

This function returns the plugin interface version number, i.e. the version number describing the layout of the plugin data type itself. Valid version numbers are always greater or equal to 1.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_dump().

4.35.5.5 cpl_plugin_get_author()

```
const char* cpl_plugin_get_author (
             const cpl_plugin * self )
```

Get the name of the plugin author.

Parameters

self	A plugin.

Returns

The function returns a pointer to the plugin author's name string. If an error occurs the function returns `NULL` and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

This function returns a reference to the name of the author of the plugin *self*. The plugin author's name must not be modified using the returned pointer. The appropriate method should be used instead!

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_dump().

4.35.5.6 cpl_plugin_get_copyright()

```
const char* cpl_plugin_get_copyright (
            const cpl_plugin * self )
```

Get the license and copyright information of a plugin.

Parameters

self	A plugin.

Returns

The function returns a pointer to the plugin's copyright information string. If an error occurs the function returns NULL and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

This function returns a reference to the license and copyright information of the plugin *self*. The copyright information must not be modified using the returned pointer. The appropriate method should be used instead!

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_dump().

4.35.5.7 cpl_plugin_get_deinit()

```
cpl_plugin_func cpl_plugin_get_deinit (
            const cpl_plugin * self )
```

Get the cleanup handler of a plugin.

Parameters

self	A plugin.

Returns

The plugin's cleanup handler. If an error occurs the function returns NULL and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function returns the cleanup handler of the plugin *self*.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_dump().

4.35.5.8 cpl_plugin_get_description()

```
const char* cpl_plugin_get_description (
            const cpl_plugin * self )
```

Get the detailed description of a plugin.

Parameters

self	A plugin.

Returns

The function returns a pointer to the plugin's detailed description. If an error occurs the function returns `NULL` and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

This function returns a reference to the detailed description (a description of the algorithm for instance) of the plugin *self*. The plugin's description must not be modified using this pointer. Use the appropriate method instead!

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_dump().

4.35.5.9 cpl_plugin_get_email()

```
const char* cpl_plugin_get_email (
            const cpl_plugin * self )
```

Get the contact information of a plugin.

Parameters

self	A plugin.

Returns

The function returns a pointer to the plugin author's contact information string. If an error occurs the function returns `NULL` and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

This function returns a reference to the e-mail address of the author of the plugin *self*. The plugin author's e-mail address must not be modified using the returned pointer. The appropriate method should be used instead!

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_dump().

4.35.5.10 cpl_plugin_get_exec()

```
cpl_plugin_func cpl_plugin_get_exec (
            const cpl_plugin * self )
```

Get the execution handler of a plugin.

Parameters

self	A plugin.

Returns

The plugin's execution function. If an error occurs the function returns `NULL` and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function returns the execution function of the plugin *self*.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_dump().

4.35.5.11 cpl_plugin_get_info()

```
int cpl_plugin_get_info (
            cpl_pluginlist * cpl_plugin_list )
```

Append the plugin information to the given list.

Parameters

cpl_plugin_list	A plugin list.

Returns

The function must return 0 on success and 1 in case of an error.

This function must be implemented by plugin developers. There must be one such implementation per plugin library, regardless of how many plugins the library actually offers, provided that there is at least one plugin implemented in this library.

This prototype is only provided in order to allow the compiler to do some basic checks when compiling a plugin library. To have the prototype available when you are compiling your plugin library, you must add the line
`#include <cpl_plugininfo.h>`

to your plugin source file.

The purpose of this function is to create a plugin object for each plugin implementation provided by the plugin library, fill the basic plugin interface (the `cpl_plugin` part of the created plugin object) and append the created object to the plugin list *list*.

The list will be provided by the application which is going to use the plugin and it may be expected that *list* points to a valid plugin list when this function is called.

4.35.5.12 cpl_plugin_get_init()

```
cpl_plugin_func cpl_plugin_get_init (
            const cpl_plugin * self )
```

Get the initialisation handler of a plugin.

Parameters

self	A plugin.

Returns

The plugin's initalization function. If an error occurs the function returns `NULL` and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function returns the initalisation function of the plugin *self*.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_dump().

4.35.5.13 cpl_plugin_get_name()

```
const char* cpl_plugin_get_name (
            const cpl_plugin * self )
```

Get the name of a plugin.

Parameters

self	A plugin.

Returns

The function returns a pointer to the plugin's unique name string. If an error occurs the function returns `NULL` and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

This function returns a reference to the unique name of the plugin *self*. The plugin's name must not be modified using the returned pointer. The appropriate method should be used instead.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_dump(), and cpl_pluginlist_find().

4.35.5.14 cpl_plugin_get_synopsis()

```
const char* cpl_plugin_get_synopsis (
            const cpl_plugin * self )
```

Get the short description of a plugin.

Parameters

self	A plugin.

Returns

The function returns a pointer to the plugin's short description. If an error occurs the function returns `NULL` and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

This function returns a reference to the short description (its purpose for instance) of the plugin *self*. The plugin's short description must not be modified using this pointer. Use the appropriate method instead!

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_dump().

4.35.5.15 cpl_plugin_get_type()

```
unsigned long cpl_plugin_get_type (
            const cpl_plugin * self )
```

Get the type of a plugin.

Parameters

self	A plugin.

Returns

The function returns the plugin type code. If an error occurs the function returns CPL_PLUGIN_TYPE_NONE and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

This function returns the type (cf. cpl_plugin_type) of the plugin *self*.

References CPL_ERROR_NULL_INPUT, and CPL_PLUGIN_TYPE_NONE.

Referenced by cpl_plugin_dump().

4.35.5.16 cpl_plugin_get_type_string()

```
char* cpl_plugin_get_type_string (
            const cpl_plugin * self )
```

Get the type of a plugin as string.

Parameters

self	A plugin.

Returns

The function returns the string representation of the plugin type. If an error occurs the function returns NULL and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

This function returns the plugin type of *self* as a string. The type string is placed into a newly allocated buffer. This buffer must be deallocated using **cpl_free()** by the caller if it is no longer needed.

References CPL_ERROR_NULL_INPUT, CPL_PLUGIN_TYPE_NONE, and CPL_PLUGIN_TYPE_RECIPE.

Referenced by cpl_plugin_dump().

4.35.5.17 cpl_plugin_get_version()

```
unsigned long cpl_plugin_get_version (
            const cpl_plugin * self )
```

Get the version number of a plugin.

Parameters

self	A plugin.

Returns

The plugin's version number. If an error occurs the function returns 0 and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

This function returns the version number of the plugin *self*.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_dump().

4.35.5.18 cpl_plugin_get_version_string()

```
char* cpl_plugin_get_version_string (
            const cpl_plugin * self )
```

Get the version number of a plugin as a string.

Parameters

self	A plugin.

Returns

The string representation of the plugin's version number. If an error occurs the function returns `NULL` and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

This function returns the version number of the plugin *self* as a string. The function assumes that the integer representation of the plugin version can be decoded into a version string of the usual form: "major.minor.micro"

The resulting string is placed in a newly allocated buffer. This buffer must be deallocated using **cpl_free()** by the caller if it is no longer needed.

References CPL_ERROR_NULL_INPUT, and cpl_sprintf().

Referenced by cpl_plugin_dump().

4.35.5.19 cpl_plugin_init()

```
cpl_error_code cpl_plugin_init (
            cpl_plugin * self,
            unsigned int api,
            unsigned long version,
            unsigned long type,
            const char * name,
            const char * synopsis,
            const char * description,
            const char * author,
            const char * email,
            const char * copyright,
            cpl_plugin_func create,
            cpl_plugin_func execute,
            cpl_plugin_func destroy )
```

Initialise a plugin.

Parameters

self	The plugin to initialise.
api	The plugin interface version number.
version	The plugin's version number.
type	The plugin's type.
name	The plugin's unique name.
synopsis	The plugin's short description (purpose, synopsis ...).
description	The plugin's detailed description.
author	The plugin's author name.
email	The plugin author's e-mail address.
copyright	The plugin's copyright and licensing information.
create	The function used to create the plugin.
execute	The function used to execute the plugin.
destroy	The function used to destroy the plugin.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function fills the `cpl_plugin` part of a plugin structure. For information on which information is required and which is optional please refer to the documentation of the plugin structure *cpl_plugin*.

If *self* is not a valid pointer, the function returns immediately.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_plugin_set_api(), cpl_plugin_set_author(), cpl_plugin_set_copyright(), cpl_plugin_set_deinit(), cpl_plugin_set_description(), cpl_plugin_set_email(), cpl_↩ plugin_set_exec(), cpl_plugin_set_init(), cpl_plugin_set_name(), cpl_plugin_set_synopsis(), cpl_plugin_set_type(), and cpl_plugin_set_version().

4.35.5.20 cpl_plugin_new()

```
cpl_plugin* cpl_plugin_new (
            void  )
```

Create a new, empty plugin interface.

Returns

The pointer to a newly allocated plugin or NULL if it could not be created.

The function allocates memory for a cpl_plugin and initialises it. The function returns a handle for the newly created plugin interface object. The created plugin interface must be destroyed using the plugin interface destructor **cpl_plugin_delete()**.

References _cpl_plugin_::api.

4.35.5.21 cpl_plugin_set_api()

```
cpl_error_code cpl_plugin_set_api (
            cpl_plugin * self,
            unsigned int api )
```

Set the plugin interface version number.

Parameters

self	A plugin
api	The plugin interface version to set.

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

This function sets the version number of the plugin interface of the plugin *self* to the version *api*.

Attention

The plugin interface version describes the internal layout of the plugin interface. It should be used by an application to decide whether a particular plugin can be used or not, i.e. if the plugin interface supported by the application is compatible with the interface presented by the plugin itself.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_init().

4.35.5.22 cpl_plugin_set_author()

```
cpl_error_code cpl_plugin_set_author (
            cpl_plugin * self,
            const char * author )
```

Set the name of the plugin author.

Parameters

self	A plugin
author	The name of the plugin author.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *author* is a `NULL` pointer.

This function copies the plugin author's name from the string *author* to the plugin *self*.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_init().

4.35.5.23 cpl_plugin_set_copyright()

```
cpl_error_code cpl_plugin_set_copyright (
            cpl_plugin * self,
            const char * copyright )
```

Set the license and copyright information of a plugin.

Parameters

self	A plugin.
copyright	The plugin's license information.

Note

The license information must be compatible with that of CPL.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *copyright* is a `NULL` pointer.

This function copies the plugin's license information from the string *copyright* to the plugin *self*.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_init().

4.35.5.24 cpl_plugin_set_deinit()

```
cpl_error_code cpl_plugin_set_deinit (
            cpl_plugin * self,
            cpl_plugin_func func )
```

Set the cleanup handler of a plugin.

Parameters

self	A plugin
func	The plugin's cleanup handler.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

This function installs the function *func* as the cleanup handler of the plugin *self*. The registered function is called after the plugin has been executed to release any acquired resources.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_init().

4.35.5.25 cpl_plugin_set_description()

```
cpl_error_code cpl_plugin_set_description (
            cpl_plugin * self,
            const char * description )
```

Set the detailed description of a plugin.

Parameters

self	A plugin.
description	The plugin's detailed description, or `NULL`.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

This function copies the detailed description text from the string *description* to the plugin *self*. The C formatting characters `'\n'` and `'\t'` may be embedded in the string *description*.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_init().

4.35.5.26 cpl_plugin_set_email()

```
cpl_error_code cpl_plugin_set_email (
            cpl_plugin * self,
            const char * email )
```

Set the contact information of a plugin.

Parameters

self	A plugin.
email	The plugin author's contact information.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

Returns

The function returns 0 on success, or a non-zero value otherwise. If *self* is not a valid pointer the function returns 1.

This function copies the plugin author contact information from the string *email* to the plugin *self*.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_init().

4.35.5.27 cpl_plugin_set_exec()

```
cpl_error_code cpl_plugin_set_exec (
            cpl_plugin * self,
            cpl_plugin_func func )
```

Set the execution handler of a plugin.

Parameters

self	A plugin.
func	The plugin's execution function.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

This function installs the function *func* as the execution function of the plugin *self*. The registered function must be called by an application after the plugin's initialisation function was called. Calling the registered function executes the plugin.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_init().

4.35.5.28 cpl_plugin_set_init()

```
cpl_error_code cpl_plugin_set_init (
            cpl_plugin * self,
            cpl_plugin_func func )
```

Set the initialisation handler of a plugin.

Parameters

self	A plugin
func	The plugin's initialisation function

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

This function installs the function *func* as the initialisation function of the plugin *self*. The registered function must be called by an application before the plugin is executed.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_init().

4.35.5.29 cpl_plugin_set_name()

```
cpl_error_code cpl_plugin_set_name (
            cpl_plugin * self,
            const char * name )
```

Set the name of a plugin.

Parameters

self	A plugin.
name	The plugin's unique name.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.

This function assigns the name *name* to the plugin *self*.

Attention

Since plugins are selected through their name this name should be choosen carefully in order to avoid name clashes with other plugins.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_init().

4.35.5.30 cpl_plugin_set_synopsis()

```
cpl_error_code cpl_plugin_set_synopsis (
            cpl_plugin * self,
            const char * synopsis )
```

Set the short description of a plugin.

Parameters

self	A plugin.
synopsis	The plugin's synopsis, or NULL.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

This function copies the short description text from the string *synopsis* to the plugin *self*.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_init().

4.35.5.31 cpl_plugin_set_type()

```
cpl_error_code cpl_plugin_set_type (
            cpl_plugin * self,
            unsigned long type )
```

Set the type of a plugin.

Parameters

self	A plugin.
type	The plugin type to set.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

This function sets the type (cf. cpl_plugin_type) of the plugin *self* to *type*.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_init().

4.35.5.32 cpl_plugin_set_version()

```
int cpl_plugin_set_version (
            cpl_plugin * self,
            unsigned long version )
```

Set the version number of a plugin.

Parameters

self	A plugin
version	The plugin's version number to set.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

This function sets the version number of the plugin interface of the plugin *self* to the version *api*.

This function sets the version number of the plugin *self* to *version*.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_plugin_init().

4.36 Plugin List

Typedefs

- typedef struct _cpl_pluginlist_ cpl_pluginlist

  The opaque plugin list data type.

Functions

- cpl_error_code cpl_pluginlist_append (cpl_pluginlist *self, const cpl_plugin *plugin)

  Append a plugin to a plugin list.
- void cpl_pluginlist_delete (cpl_pluginlist *self)

  Delete a plugin list.
- void cpl_pluginlist_dump (const cpl_pluginlist *self, FILE *stream)

  Dump the contents of a plugin list to the given stream.
- cpl_plugin * cpl_pluginlist_find (cpl_pluginlist *self, const char *name)

  Find a plugin with a given name in a plugin list.
- cpl_plugin * cpl_pluginlist_get_first (cpl_pluginlist *self)

  Get the first plugin of a plugin list.
- cpl_plugin * cpl_pluginlist_get_last (cpl_pluginlist *self)

  Get the last plugin of a plugin list.
- cpl_plugin * cpl_pluginlist_get_next (cpl_pluginlist *self)

  Get the next plugin from a plugin list.
- int cpl_pluginlist_get_size (cpl_pluginlist *self)

  Get the current size of a plugin list.
- cpl_pluginlist * cpl_pluginlist_new (void)

  Creates an empty plugin list.
- cpl_error_code cpl_pluginlist_prepend (cpl_pluginlist *self, const cpl_plugin *plugin)

  Prepend a plugin to a plugin list.

4.36.1 Detailed Description

This module implements a list container for plugin objects and provides the facilities to query, to traverse and to update the container. The purpose of this container is to be able to store references to available plugins and to handle sets of plugins as a whole.

Since the plugin list just stores pointers to `cpl_plugin`, a plugin list may contain plugins of different kind at the same time, because all context specific plugins inherit the `cpl_plugin` type (see Plugin Interface).

Synopsis:

```
#include <cpl_pluginlist.h>
```

4.36.2 Typedef Documentation

4.36.2.1 cpl_pluginlist

```
typedef struct _cpl_pluginlist_ cpl_pluginlist
```

The opaque plugin list data type.

4.36.3 Function Documentation

4.36.3.1 cpl_pluginlist_append()

```
cpl_error_code cpl_pluginlist_append (
            cpl_pluginlist * self,
            const cpl_plugin * plugin )
```

Append a plugin to a plugin list.

Parameters

self	A plugin list.
plugin	The plugin to append.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *plugin* is a `NULL` pointer.

The plugin *plugin* is inserted into the plugin list *self* after the last element.

If *self* does not point to a valid plugin list, or if *plugin* is not a valid pointer the function returns immediately.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.36.3.2 cpl_pluginlist_delete()

```
void cpl_pluginlist_delete (
            cpl_pluginlist * self )
```

Delete a plugin list.

Parameters

self	The plugin list to delete.

Returns

Nothing.

The function deletes the plugin list *self* and destroys all plugins the list potentially contains. If *self* is `NULL`, nothing is done and no error is set.

References cpl_plugin_delete().

4.36.3.3 cpl_pluginlist_dump()

```
void cpl_pluginlist_dump (
            const cpl_pluginlist * self,
            FILE * stream )
```

Dump the contents of a plugin list to the given stream.

Parameters

self	The plugin list.
stream	The output stream to use.

Returns

Nothing.

The function dumps the debugging information for each plugin found in the plugin list *self* to the output stream *stream*. The debugging information for each individual plugin is dumped using **cpl_plugin_dump()**. If *self* is `NULL` the function does nothing.

See also

cpl_plugin_dump()

References cpl_plugin_dump().

4.36.3.4 cpl_pluginlist_find()

```
cpl_plugin* cpl_pluginlist_find (
            cpl_pluginlist * self,
            const char * name )
```

Find a plugin with a given name in a plugin list.

Parameters

self	The plugin list to query.
name	The plugin's unique name to look for.

Returns

The first plugin with the given name *name*, or `NULL` if it no plugin with this name could be found. The function returns `NULL` if an error occurs and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.

This function searches the plugin list *self* for a plugin with the unique name *name* and returns a pointer to the first one found. If no plugin with the given name is found the function returns `NULL`.

References CPL_ERROR_NULL_INPUT, and cpl_plugin_get_name().

4.36.3.5 cpl_pluginlist_get_first()

```
cpl_plugin* cpl_pluginlist_get_first (
            cpl_pluginlist * self )
```

Get the first plugin of a plugin list.

Parameters

self	A plugin list.

Returns

The first plugin stored in the plugin list *self*, or `NULL` if the list is empty. The function returns `NULL` if an error occurs and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function returns a handle to the first plugin stored in the *self*.

References CPL_ERROR_NULL_INPUT.

4.36.3.6 cpl_pluginlist_get_last()

```
cpl_plugin* cpl_pluginlist_get_last (
            cpl_pluginlist * self )
```

Get the last plugin of a plugin list.

Parameters

self	A plugin list.

Returns

The last plugin stored in the plugin list *self*, or NULL if the list is empty. The function returns NULL if an error occurs and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function returns a pointer to the last plugin stored in the plugin list *self*.

References CPL_ERROR_NULL_INPUT.

4.36.3.7 cpl_pluginlist_get_next()

```
cpl_plugin* cpl_pluginlist_get_next (
            cpl_pluginlist * self )
```

Get the next plugin from a plugin list.

Parameters

self	A plugin list.

Returns

The function returns the next plugin in the list, or NULL if the end of the list has been reached. The function returns NULL if an error occurs and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_ILLEGAL_INPUT	The function was called without initialising the plugin list *self* by calling **cpl_pluginlist_first()**.

The function returns the next plugin in *self*. To find the next plugin, the plugin list caches the position of the most recently obtained plugin. This requires a call to **cpl_pluginlist_get_first()** prior to calling this function in order to properly initialise the internal cache.

If the end of *self* has been reached the internal cache is reset to the first plugin in the list.

References CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_INPUT.

4.36.3.8 cpl_pluginlist_get_size()

```
int cpl_pluginlist_get_size (
            cpl_pluginlist * self )
```

Get the current size of a plugin list.

Parameters

self	A plugin list.

Returns

The plugin list's current size, or 0 if the list is empty. The function returns 0 if an error occurs and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function reports the current number of plugins stored in the plugin list *self*. If *self* does not point to a valid plugin list the function returns 0.

References CPL_ERROR_NULL_INPUT.

4.36.3.9 cpl_pluginlist_new()

```
cpl_pluginlist* cpl_pluginlist_new (
            void  )
```

Creates an empty plugin list.

Returns

The newly created plugin list, or `NULL` if it could not be created.

The function allocates memory for a plugin list object and initialises it to be empty.

4.36.3.10 cpl_pluginlist_prepend()

```
cpl_error_code cpl_pluginlist_prepend (
            cpl_pluginlist * self,
            const cpl_plugin * plugin )
```

Prepend a plugin to a plugin list.

Parameters

self	A plugin list.
plugin	The plugin to prepend.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *plugin* is a `NULL` pointer.

The plugin *plugin* is inserted into the plugin list *self* before the first element.

If *self* does not point to a valid plugin list, or if *plugin* is not a valid pointer the function returns immediately.

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.37 Point pattern matching module

Functions

- cpl_array ∗ cpl_ppm_match_points (const cpl_matrix ∗data, cpl_size use_data, double err_data, const cpl↩ _matrix ∗pattern, cpl_size use_pattern, double err_pattern, double tolerance, double radius, cpl_matrix ∗∗mdata, cpl_matrix ∗∗mpattern, double ∗lin_scale, double ∗lin_angle)

  Match 2-D distributions of points.
- cpl_bivector ∗ cpl_ppm_match_positions (const cpl_vector ∗peaks, const cpl_vector ∗lines, double min_disp, double max_disp, double tolerance, cpl_array ∗∗seq_peaks, cpl_array ∗∗seq_lines)

  Match 1-D patterns.

4.37.1 Detailed Description

Synopsis:

```
#include "cpl_ppm.h"
```

4.37.2 Function Documentation

4.37.2.1 cpl_ppm_match_points()

```
cpl_array* cpl_ppm_match_points (
             const cpl_matrix * data,
             cpl_size use_data,
             double err_data,
             const cpl_matrix * pattern,
             cpl_size use_pattern,
             double err_pattern,
             double tolerance,
             double radius,
             cpl_matrix ** mdata,
             cpl_matrix ** mpattern,
             double * lin_scale,
             double * lin_angle )
```

Match 2-D distributions of points.

Parameters

data	List of data points (e.g., detected stars positions).
use_data	Number of *data* points used for preliminary match.
err_data	Error on *data* points positions.
pattern	List of pattern points (e.g., expected stars positions).
use_pattern	Number of *pattern* points used for preliminary match.
err_pattern	Error on *pattern* points positions.
tolerance	Max relative difference of angles and scales from their median value for match acceptance.
radius	Search radius applied in final matching (*data* units).
mdata	List of identified *data* points.
mpattern	List of matching *pattern* points.
lin_scale	Linear transformation scale factor.
lin_angle	Linear transformation rotation angle.

Returns

Indexes of identified data points (pattern-to-data).

A point is described here by its coordinates on a cartesian plane. The input matrices *data* and *pattern* must have 2 rows, as their column vectors are the points coordinates.

This function attemps to associate points in *data* to points in *pattern*, under the assumption that a transformation limited to scaling, rotation, and translation, would convert positions in *pattern* into positions in *data*. Association between points is also indicated in the following as "match", or "identification".

Point identification is performed in two steps. In the first step only a subset of the points is identified (preliminary match). In the second step the identified points are used to define a first-guess transformation from *pattern* points to *data* points, that is applied to identify all the remaining points as well. The second step would be avoided if a *use_pattern* equal to the number of points in *pattern* is given, and exactly *use_pattern* points would be identified already in the first step.

First step:

All possible triangles (sub-patterns) are built using the first *use_data* points from *data* and the first *use_pattern* points from *pattern*. The values of *use_data* and *use_pattern* must always be at least 3 (however, see the note at the end), and should not be greater than the length of the corresponding lists of points. The point-matching algorithm goes as follow:

```
For every triplet of points:
   Select one point as the reference. The triangle coordinates
   are defined by
               ((Rmin/Rmax)^2, theta_min - theta_max)
   where Rmin (Rmax) is the shortest (longest) distance from the
   reference point to one of the two other points, and theta_min
   (theta_max) is the view angle in [0; 2pi[ to the nearest
   (farthest) point.
   Triangles are computed by using each point in the triplet
   as reference, thereby computing 3 times as many triangles
   as needed.
   The accuracy of triangle patterns is robust against distortions
   (i.e., systematic inaccuracies in the points positions) of the
   second order. This is because, if the points positions had
   constant statistical uncertainty, the relative uncertainty in
   the triangle coordinates would be inversely proportional to
   the triangle size, while if second order distortions are
   present the systematic error on points position would be
   directly proportional to the triangle size.
For every triangle derived from the @em pattern points:
   Match with nearest triangle derived from @em data points
   if their distance in the parameter space is less than their
   uncertainties (propagated from the points positions uncertainties
   @em err_data and @em err_pattern). For every matched pair of
   triangles, record their scale ratio, and their orientation
   difference. Note that if both @em err_data and @em err_pattern
   are zero, the tolerance in triangle comparison will also be
   zero, and therefore no match will be found.
Get median scale ratio and median angle of rotation, and reject
matches with a relative variation greater than @em tolerance from
the median of either quantities. The estimator of all the rotation
angles a_i is computed as
            atan( med sin(a_i) / med cos(a_i) )
```

Second step:

From the safely matched triangles, a list of identified points is derived, and the best transformation from *pattern* points to *data* points (in terms of best rotation angle, best scaling factor, and best shift) is applied to attempt the identification of all the points that are still without match. This matching is made by selecting for each *pattern* point the *data* point which is closest to its transformed position, and at a distance less than *radius*.

The returned array of integers is as long as the number of points in *pattern*, and each element reports the position of the matching point in *data* (counted starting from zero), or is invalid if no match was found for the *pattern* point. For instance, if element N of the array has value M, it means that the Nth point in *pattern* matches the Mth point in *data*. A NULL pointer is returned in case no point was identified.

If *mdata* and *mpattern* are both valid pointers, two more matrices will be returned with the coordinates of the identified points. These two matrices will both have the same size: 2 rows, and as many columns as successfully identified points. Matching points will be in the same column of both matrices. Those matrix should in the end be destroyed using cpl_matrix_delete().

If *lin_scale* is a valid pointer, it is returned with a good estimate of the scale (distance_in_data = lin_scale $*$ distance_in_pattern). This makes sense only in case the transformation between *pattern* and *data* is an affine transformation. In case of failure, *lin_scale* is set to zero.

If *lin_angle* is a valid pointer, it is returned with a good estimate of the rotation angle between *pattern* and *data* in degrees (counted counterclockwise, from -180 to +180, and with data_orientation = pattern_orientation + lin_angle). This makes sense only in case the transformation between *pattern* and *data* is an affine transformation. In case of failure, *lin_angle* is set to zero.

The returned values for *lin_scale* and *lin_angle* have the only purpose of providing a hint on the relation between *pattern* points and *data* points. This function doesn't attempt in any way to determine or even suggest a possible transformation between *pattern* points and *data* points: this function just matches points, and it is entriely a responsibility of the caller to fit the appropriate transformation between one coordinate system and the other. A polynomial transformation of degree 2 from *pattern* to *data* may be fit in the following way (assuming that *mpattern* and *mdata* are available):

```
int             degree  = 2;
int             npoints = cpl_matrix_get_ncol(mdata);
double         *dpoints = cpl_matrix_get_data(mdata);
cpl_vector     *data_x  = cpl_vector_wrap(npoints, dpoints);
cpl_vector     *data_y  = cpl_vector_wrap(npoints, dpoints + npoints);
cpl_polynomial *x_trans = cpl_polynomial_new(degree);
cpl_polynomial *y_trans = cpl_polynomial_new(degree);
cpl_polynomial_fit(x_trans, mpattern, NULL, data_x, NULL, CPL_FALSE,
                   NULL, degree);
cpl_polynomial_fit(y_trans, mpattern, NULL, data_y, NULL, CPL_FALSE,
                   NULL, degree);
```

Note

The basic requirement for using this function is that the searched point pattern (or at least most of it) is contained in the data. As an indirect consequence of this, it would generally be appropriate to have more points in *data* than in *pattern* (and analogously, to have *use_data* greater than *use_pattern*), even if this is not strictly necessary.

Also, *pattern* and *data* should not contain too few points (say, less than 5 or 4) or the identification may risk to be incorrect: more points enable the construction of many more triangles, reducing the risk of ambiguity (multiple valid solutions). Special situations, involving regularities in patterns (as, for instance, input *data* containing just three equidistant points, or the case of a regular grid of points) would certainly provide an answer, and this answer would very likely be wrong (the human brain would fail as well, and for exactly the same reasons).

The reason why a two steps approach is encouraged here is mainly to enable an efficient use of this function: in principle, constructing all possible triangles using *all* of the available points is never wrong, but it could become very slow: a list of N points implies the evaluation of N$*$(N-1)$*$(N-2)/2 triangles, and an even greater number of comparisons between triangles. The possibility of evaluating first a rough transformation based on a limited number of identified points, and then using this transformation for recovering all the remaining points, may significantly speed up the whole identification process. However it should again be ensured that the main requirement (i.e., that the searched point pattern must be contained in the data) would still be valid for the selected subsets of points: a random choice would likely lead to a matching failure (due to too few, or no, common points).

A secondary reason for the two steps approach is to limit the effect of another class of ambiguities, happening when either or both of the input matrices contains a very large number of uniformely distributed points. The likelihood to find several triangles that are similar by chance, and at all scales and orientations, may increase to unacceptable levels.

A real example may clarify a possible way of using this function: let *data* contain the positions (in pixel) of detected stars on a CCD. Typically hundreds of star positions would be available, but only the brightest ones may be used for preliminary identification. The input *data* positions will therefore be opportunely ordered from the brightest to the

dimmest star positions. In order to identify stars, a star catalogue is needed. From a rough knowledge of the pointing position of the telescope and of the size of the field of view, a subset of stars can be selected from the catalogue: they will be stored in the *pattern* list, ordered as well by their brightness, and with their RA and Dec coordinates converted into standard coordinates (a gnomonic coordinate system centered on the telescope pointing, i.e., a cartesian coordinate system), no matter in what units of arc, and no matter what orientation of the field. For the first matching step, the 10 brightest catalogue stars may be selected (selecting less stars would perhaps be unsafe, selecting more would likely make the program slower without producing any better result). Therefore *use_pattern* would be set to 10. From the data side, it would generally be appropriate to select twice as many stars positions, just to ensure that the searched pattern is present. Therefore *use_data* would be set to 20. A reasonable value for *tolerance* and for *radius* would be respectively 0.1 (a 10% variation of scales and angles) and 20 (pixels).

References CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL↩ _INPUT, cpl_error_set, cpl_matrix_get_ncol_(), cpl_msg_debug(), and cpl_msg_indent_more().

4.37.2.2 cpl_ppm_match_positions()

```
cpl_bivector* cpl_ppm_match_positions (
             const cpl_vector * peaks,
             const cpl_vector * lines,
             double min_disp,
             double max_disp,
             double tolerance,
             cpl_array ** seq_peaks,
             cpl_array ** seq_lines )
```

Match 1-D patterns.

Parameters

peaks	List of observed positions (e.g., of emission peaks)
lines	List of positions in searched pattern (e.g., wavelengths)
min_disp	Min expected scale (e.g., spectral dispersion in A/pixel)
max_disp	Max expected scale (e.g., spectral dispersion in A/pixel)
tolerance	Tolerance for interval ratio comparison
seq_peaks	Returned: index of identified peaks in input *peaks*
seq_lines	Returned: index of identified lines in input *lines*

Returns

List of all matching points positions

This function attempts to find the reference pattern *lines* in a list of observed positions *peaks*. In the following documentation a terminology drawn from the context of arc lamp spectra calibration is used for simplicity: the reference pattern is then a list of wavelengths corresponding to a set of reference arc lamp emission lines - the so-called line catalog; while the observed positions are the positions (in pixel) on the CCD, measured along the dispersion direction, of any significant peak of the signal. To identify the observed peaks means to associate them with the right reference wavelength. This is attempted here with a point-pattern matching technique, where the pattern is contained in the vector *lines*, and is searched in the vector *peak*.

In order to work, this method just requires a rough expectation value of the spectral dispersion (in Angstrom/pixel), and a line catalog. The line catalog *lines* should just include lines that are expected somewhere in the CCD exposure of the calibration lamp (note, however, that a catalog including extra lines at its blue and/or red ends is still allowed).

Typically, the arc lamp lines candidates *peak* will include light contaminations, hot pixels, and other unwanted signal, but only in extreme cases does this prevent the pattern-recognition algorithm from identifying all the spectral lines. The pattern is detected even in the case *peak* contained more arc lamp lines than actually listed in the input line catalog.

This method is based on the assumption that the relation between wavelengths and CCD positions is with good approximation *locally* linear (this is always true, for any modern spectrograph).

The ratio between consecutive intervals pairs in wavelength and in pixel is invariant to linear transformations, and therefore this quantity can be used in the recognition of local portions of the searched pattern. All the examined sub-patterns will overlap, leading to the final identification of the whole pattern, notwithstanding the overall non-linearity of the relation between pixels and wavelengths.

Ambiguous cases, caused by exceptional regularities in the pattern, or by a number of undetected (but expected) peaks that disrupt the pattern on the data, are recovered by linear interpolation and extrapolation of the safely identified peaks.

More details about the applied algorithm can be found in the comments to the function code.

The *seq_peaks* and *seq_lines* are array reporting the positions of matching peaks and wavelengths in the input *peaks* and *lines* vectors. This functionality is not yet supported: this arguments should always be set to NULL or a CPL_ERROR_UNSUPPORTED_MODE would be set.

References cpl_bivector_wrap_vectors(), cpl_calloc(), cpl_error_set, CPL_ERROR_UNSUPPORTED_MODE, cpl_free(), cpl_malloc(), cpl_vector_get_data_const(), cpl_vector_get_size(), and cpl_vector_wrap().

4.38 Polynomials

Functions

- cpl_error_code cpl_polynomial_add (cpl_polynomial *self, const cpl_polynomial *first, const cpl_polynomial *second)

  Add two polynomials of the same dimension.
- int cpl_polynomial_compare (const cpl_polynomial *self, const cpl_polynomial *other, double tol)

  Compare the coefficients of two polynomials.
- cpl_error_code cpl_polynomial_copy (cpl_polynomial *self, const cpl_polynomial *other)

  Copy the contents of one polynomial into another one.
- void cpl_polynomial_delete (cpl_polynomial *self)

  Delete a cpl_polynomial.
- cpl_error_code cpl_polynomial_derivative (cpl_polynomial *self, cpl_size dim)

  Compute a first order partial derivative.
- cpl_error_code cpl_polynomial_dump (const cpl_polynomial *self, FILE *stream)

  Dump a polynomial as ASCII to a stream.
- cpl_polynomial * cpl_polynomial_duplicate (const cpl_polynomial *self)

  Duplicate a polynomial.
- double cpl_polynomial_eval (const cpl_polynomial *self, const cpl_vector *x)

  Evaluate the polynomial at the given point using Horner's rule.
- double cpl_polynomial_eval_1d (const cpl_polynomial *self, double x, double *pd)

  Evaluate a univariate (1D) polynomial using Horner's rule.
- double cpl_polynomial_eval_1d_diff (const cpl_polynomial *self, double a, double b, double *ppa)

  Evaluate p(a) - p(b) using Horner's rule.
- double cpl_polynomial_eval_2d (const cpl_polynomial *self, double x, double y, double *gradient)

  Evaluate a bivariate (2D) polynomial using Horner's rule and compute the derivatives if needed.
- double cpl_polynomial_eval_3d (const cpl_polynomial *self, double x, double y, double z, double *gradient)

  Evaluate a 3D polynomial using Horner's rule and compute the the derivatives if needed.
- cpl_polynomial * cpl_polynomial_extract (const cpl_polynomial *self, cpl_size dim, const cpl_polynomial *other)

  Collapse one dimension of a multi-variate polynomial by composition.
- cpl_error_code cpl_polynomial_fit (cpl_polynomial *self, const cpl_matrix *samppos, const cpl_boolean *sampsym, const cpl_vector *fitvals, const cpl_vector *fitsigm, cpl_boolean dimdeg, const cpl_size *mindeg, const cpl_size *maxdeg)

  Fit a polynomial to a set of samples in a least squares sense.
- cpl_polynomial * cpl_polynomial_fit_1d_create (const cpl_vector *x_pos, const cpl_vector *values, cpl_size degree, double *pmse)

  Fit a 1D-polynomial to a 1D-signal in a least squares sense.
- cpl_polynomial * cpl_polynomial_fit_2d_create (const cpl_bivector *xy_pos, const cpl_vector *values, cpl_size degree, double *pmse)

  Fit a 2D-polynomial to a 2D-surface in a least squares sense.
- double cpl_polynomial_get_coeff (const cpl_polynomial *self, const cpl_size *pows)

  Get a coefficient of the polynomial.
- cpl_size cpl_polynomial_get_degree (const cpl_polynomial *self)

  The degree of the polynomial.
- cpl_size cpl_polynomial_get_dimension (const cpl_polynomial *self)

  The dimension of the polynomial.
- cpl_error_code cpl_polynomial_multiply (cpl_polynomial *self, const cpl_polynomial *first, const cpl_↩ polynomial *second)

  Multiply two polynomials of the same dimension.

- cpl_error_code cpl_polynomial_multiply_scalar (cpl_polynomial *self, const cpl_polynomial *other, double factor)

  Multiply a polynomial with a scalar.
- cpl_error_code cpl_polynomial_set_coeff (cpl_polynomial *self, const cpl_size *pows, double value)

  Set a coefficient of the polynomial.
- cpl_error_code cpl_polynomial_shift_1d (cpl_polynomial *p, cpl_size i, double u)

  Modify p, p(x0, x1, ..., xi, ...) := (x0, x1, ..., xi+u, ...)
- cpl_error_code cpl_polynomial_solve_1d (const cpl_polynomial *p, double x0, double *px, cpl_size mul)

  A real solution to p(x) = 0 using Newton-Raphsons method.
- cpl_error_code cpl_polynomial_subtract (cpl_polynomial *self, const cpl_polynomial *first, const cpl_polynomial *second)

  Subtract two polynomials of the same dimension.
- cpl_error_code cpl_vector_fill_polynomial (cpl_vector *v, const cpl_polynomial *p, double x0, double d)

  Evaluate a 1D-polynomial on equidistant points using Horner's rule.
- cpl_error_code cpl_vector_fill_polynomial_fit_residual (cpl_vector *self, const cpl_vector *fitvals, const cpl_vector *fitsigm, const cpl_polynomial *fit, const cpl_matrix *samppos, double *rechisq)

  Compute the residual of a polynomial fit.

4.38.1 Detailed Description

This module provides functions to handle uni- and multivariate polynomials.

Comparing the two polynomials

```
* P1(x) = p0 + p1.x + p4.x^2
*
```

and

```
* P2(x,y) = p0 + p1.x + p2.y + p3.x.y + p4.x^2 + p5.y^2
*
```

P1(x) may evaluate to more accurate results than P2(x,0), especially around the roots.

Note that a polynomial like P3(z) = p0 + p1.z + p2.z^2 + p3.z^3, z=x^4 is preferable to p4(x) = p0 + p1.x^4 + p2.x^8 + p3.x^12.

Polynomials are evaluated using Horner's method. For multivariate polynomials the evaluation is performed one dimension at a time, starting with the lowest dimension and proceeding upwards through the higher dimensions.

Access to a coefficient of an N-dimensional polynomial has complexity O(N).

4.38.2 Function Documentation

4.38.2.1 cpl_polynomial_add()

```
cpl_error_code cpl_polynomial_add (
            cpl_polynomial * self,
            const cpl_polynomial * first,
            const cpl_polynomial * second )
```

Add two polynomials of the same dimension.

Parameters

self	The polynomial to hold the result
first	The 1st polynomial to add
second	The 2nd polynomial to add

Returns

CPL_ERROR_NONE or the relevant CPL error code

Note

self may be passed also as first and/or second

Possible CPL error code set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if the polynomials do not have identical dimensions

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NULL_INPUT, and cpl_↩ polynomial_get_dimension().

4.38.2.2 cpl_polynomial_compare()

```
int cpl_polynomial_compare (
             const cpl_polynomial * self,
             const cpl_polynomial * other,
             double tol )
```

Compare the coefficients of two polynomials.

Parameters

self	The 1st polynomial
other	The 2nd polynomial
tol	The absolute (non-negative) tolerance

Returns

0 when equal, positive when different, negative on error.

The two polynomials are considered equal iff they have identical dimensions and the absolute difference between their coefficients does not exceed the given tolerance.

This means that the following two polynomials per definition are considered different: P1(x1) = 3∗x1 different from P2(x1,x2) = 3∗x1.

If all input parameters are valid and self and other point to the same polynomial the function returns 0.

If two polynomials have different dimensions, the return value is this (positive) difference.

If two 1D-polynomials differ, the return value is 1 plus the degree of the lowest order differing coefficient.

If for a higher dimension they differ, it is 1 plus the degree of a differing coefficient.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if tol is negative

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_INPUT.

4.38.2.3 cpl_polynomial_copy()

```
cpl_error_code cpl_polynomial_copy (
            cpl_polynomial * self,
            const cpl_polynomial * other )
```

Copy the contents of one polynomial into another one.

Parameters

self	Pre-allocated output polynomial
other	Input polynomial

Returns

CPL_ERROR_NONE on success or else the relevant _cpl_error_code_

self and other must point to different polynomials.

If self already contains coefficients, then they are overwritten.

This is the only function that can modify the dimension of a polynomial.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if self and out point to the same polynomial

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, and CPL_ERROR_NULL_INPUT.

4.38.2.4 cpl_polynomial_delete()

```
void cpl_polynomial_delete (
            cpl_polynomial * self )
```

Delete a cpl_polynomial.

Parameters

self	Polynomial to deallocate

Returns

void

See also

cpl_polynomial_new()

Note

If *self* is `NULL`, nothing is done and no error is set.

The function deallocates the memory used by the polynomial *self*.

Referenced by cpl_image_fill_jacobian_polynomial().

4.38.2.5 cpl_polynomial_derivative()

```
cpl_error_code cpl_polynomial_derivative (
            cpl_polynomial * self,
            cpl_size dim )
```

Compute a first order partial derivative.

Parameters

self	The polynomial to be modified in place
dim	The dimension to differentiate (zero for first dimension)

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_

The dimension of the polynomial is preserved, even if the operation may cause the polynomial to become independent of the dimension dim variable.

The call requires n FLOPs, where n is the number of (non-zero) polynomial coefficients whose power in dimension dim is at least 1.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if dim is negative.

- CPL_ERROR_ACCESS_OUT_OF_RANGE if dim exceeds the dimension of self.

References cpl_ensure_code, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_image_fill_jacobian_polynomial().

4.38.2.6 cpl_polynomial_dump()

```
cpl_error_code cpl_polynomial_dump (
            const cpl_polynomial * self,
            FILE * stream )
```

Dump a polynomial as ASCII to a stream.

Parameters

self	The polynomial to process
stream	Output stream eq. `stdout` or `stderr`

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_

Each coefficient is preceded by its integer power(s) and written on a single line. If the polynomial has non-zero coefficients, only those are printed, otherwise the (zero-valued) constant term is printed.

For an N-dimensional polynomial each line thus consists of N power(s) and their coefficient.

Comment lines start with the hash character.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_FILE_IO if the write operation fails

References cpl_ensure_code, CPL_ERROR_FILE_IO, CPL_ERROR_NULL_INPUT, and CPL_SIZE_FORMAT.

4.38.2.7 cpl_polynomial_duplicate()

```
cpl_polynomial* cpl_polynomial_duplicate (
            const cpl_polynomial * self )
```

Duplicate a polynomial.

Parameters

self	The non-NULL polynomial to process

Returns

A newly allocated cpl_polynomial or NULL on error

See also

cpl_polynomial_new()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_image_fill_jacobian_polynomial().

4.38.2.8 cpl_polynomial_eval()

```
double cpl_polynomial_eval (
            const cpl_polynomial * self,
            const cpl_vector * x )
```

Evaluate the polynomial at the given point using Horner's rule.

Parameters

self	The polynomial to access
x	Point of evaluation

Note

The length of x must match the polynomial dimension.

Returns

The computed value or undefined on error.

A polynomial with no non-zero coefficients evaluates as 0.

With n coefficients the complexity is about 2n FLOPs.

For multiple evaluations the input vector can be updated like this:

```
double xy[2];
cpl_vector * xyvec = cpl_vector_wrap(2, xy);
for (cpl_size i = 0; i < nmany; i++) {
    double value;
    xy[0] = my_x(i);
    xy[1] = my_y(i);
    value = cpl_polynomial_eval(poly2d, xyvec);
    my_func(value);
}
(void)cpl_vector_unwrap(xyvec);
```

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if the length of x differs from the dimension of the polynomial

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_image_fill_jacobian_polynomial().

4.38.2.9 cpl_polynomial_eval_1d()

```
double cpl_polynomial_eval_1d (
            const cpl_polynomial * self,
            double x,
            double * pd )
```

Evaluate a univariate (1D) polynomial using Horner's rule.

Parameters

self	The 1D-polynomial
x	The point of evaluation
pd	Iff pd is non-NULL, the derivative evaluated at x

Returns

The result or undefined on error.

See also

cpl_polynomial_eval()

A polynomial with no non-zero coefficents evaluates to 0 with a derivative that does likewise.

The result is computed as p_0 + x ∗ (p_1 + x ∗ (p_2 + ... x ∗ p_n)) and requires 2n FLOPs where n+1 is the number of coefficients.

If the derivative is requested it is computed using a nested Horner rule. This requires about 4n FLOPs.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INVALID_TYPE if the polynomial is not (1D) univariate

References cpl_ensure, CPL_ERROR_INVALID_TYPE, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_wlcalib_find_best_1d().

4.38.2.10 cpl_polynomial_eval_1d_diff()

```
double cpl_polynomial_eval_1d_diff (
            const cpl_polynomial * self,
            double a,
            double b,
            double * ppa )
```

Evaluate p(a) - p(b) using Horner's rule.

Parameters

self	The 1D-polynomial
a	The evaluation point of the minuend
b	The evaluation point of the subtrahend
ppa	Iff ppa is not NULL, p(a)

Returns

The difference or undefined on error

See also

cpl_polynomial_eval_1d()

The call requires about 4n FLOPs where n is the number of coefficients in self, which is the same as that required for two separate polynomial evaluations. cpl_polynomial_eval_1d_diff() is however more accurate.

ppa may be NULL. If it is not, ∗ppa is set to self(a), which is calculated at no extra cost.

The underlying algorithm is the same as that used in cpl_polynomial_eval_1d() when the derivative is also requested.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INVALID_TYPE if the polynomial has the wrong dimension

References cpl_ensure, CPL_ERROR_INVALID_TYPE, and CPL_ERROR_NULL_INPUT.

4.38.2.11 cpl_polynomial_eval_2d()

```
double cpl_polynomial_eval_2d (
            const cpl_polynomial * self,
            double x,
            double y,
            double * gradient )
```

Evaluate a bivariate (2D) polynomial using Horner's rule and compute the derivatives if needed.

Parameters

self	The 2D-polynomial
x	The x component of the evaluation point
y	The y component of the evaluation point
gradient	Iff gradient is non-NULL, the 2D gradient array containing the X and Y components of the gradient vector at the evaluated point

Returns

The result or undefined on error.

See also

cpl_polynomial_eval()

The result is computed as p_0 + x ∗ (p_1 + x ∗ (p_2 + ... x ∗ p_n))

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INVALID_TYPE if the polynomial is not 2D

References cpl_ensure, CPL_ERROR_INVALID_TYPE, and CPL_ERROR_NULL_INPUT.

4.38.2.12 cpl_polynomial_eval_3d()

```
double cpl_polynomial_eval_3d (
            const cpl_polynomial * self,
            double x,
            double y,
            double z,
            double * gradient )
```

Evaluate a 3D polynomial using Horner's rule and compute the the derivatives if needed.

Parameters

self	The 3D-polynomial
x	The x component of the evaluation point
y	The y component of the evaluation point
z	The z component of the evaluation point
gradient	Iff gradient is non-NULL, the 3D gradient array containing the X, Y and Z components of the gradient vector at the evaluated point

Returns

The result or undefined on error.

See also

cpl_polynomial_eval()

The result is computed as p_0 + x ∗ (p_1 + x ∗ (p_2 + ... x ∗ p_n))

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INVALID_TYPE if the polynomial is not 3D

References cpl_ensure, CPL_ERROR_INVALID_TYPE, and CPL_ERROR_NULL_INPUT.

4.38.2.13 cpl_polynomial_extract()

```
cpl_polynomial* cpl_polynomial_extract (
            const cpl_polynomial * self,
            cpl_size dim,
            const cpl_polynomial * other )
```

Collapse one dimension of a multi-variate polynomial by composition.

Parameters

self	The multi-variate polynomial
dim	The dimension to collapse (zero for first dimension)
other	The polynomial to replace dimension dim of self

Returns

The collapsed polynomial or NULL on error

See also

cpl_polynomial_eval()

The dimension of the polynomial self must be one greater than that of the other polynomial. Given these two polynomials the dimension dim of self is collapsed by creating a new polynomial from self(x0, x1, ..., x{dim-1}, other(x0, x1, ..., x{dim-1}, x{dim+1}, x{dim+2}, ..., x{n-1}), x{dim+1}, x{dim+2}, ..., x{n-1}).

The created polynomial thus has a dimension which is one less than the polynomial self and which is equal to that of the other polynomial. Collapsing one dimension of a 1D-polynomial is equivalent to evaluating it, which can be done with cpl_polynomial_eval_1d().

FIXME: The other polynomial must currently have a degree of zero, i.e. it must be a constant.

The collapse uses Horner's rule and requires for n coefficients requires about 2n FLOPs.

The returned object is a newly allocated cpl_polynomial that must be deallocated by the caller using cpl_polynomial_delete().

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INVALID_TYPE if the polynomial is uni-variate.
- CPL_ERROR_ILLEGAL_INPUT if dim is negative.
- CPL_ERROR_ACCESS_OUT_OF_RANGE if dim exceeds the dimension of self.
- CPL_ERROR_INCOMPATIBLE_INPUT if other has the wrong dimension.
- CPL_ERROR_UNSUPPORTED_MODE if other is not of degree 0.

References cpl_calloc(), cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INP↩UT, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NULL_INPUT, CPL↩_ERROR_UNSUPPORTED_MODE, cpl_free(), cpl_polynomial_get_coeff(), and cpl_polynomial_get_degree().

4.38.2.14 cpl_polynomial_fit()

```
cpl_error_code cpl_polynomial_fit (
             cpl_polynomial * self,
             const cpl_matrix * samppos,
             const cpl_boolean * sampsym,
             const cpl_vector * fitvals,
             const cpl_vector * fitsigm,
             cpl_boolean dimdeg,
             const cpl_size * mindeg,
             const cpl_size * maxdeg )
```

Fit a polynomial to a set of samples in a least squares sense.

Parameters

self	Polynomial of dimension d to hold the coefficients
samppos	Matrix of p sample positions, with d rows and p columns
sampsym	NULL, or d booleans, true iff the sampling is symmetric
fitvals	Vector of the p values to fit
fitsigm	Uncertainties of the sampled values, or NULL for all ones
dimdeg	True iff there is a fitting degree per dimension

Returns

CPL_ERROR_NONE on success, else the relevant _cpl_error_code_

Note

Currently only 1D (uni-variate), 2D (bi-variate) and 3D (tri-variate) polynomials are supported. For all but uni-variate polynomials mindeg must be zero.

See also

cpl_vector_fill_polynomial_fit_residual()

Any pre-set non-zero coefficients in self are overwritten or reset by the fit.

For 1D-polynomials N = 1 + maxdeg - mindeg coefficients are fitted. A non-zero mindeg ensures that the fitted polynomial has a fix-point at zero.

For multi-variate polynomials the fit depends on dimdeg:

If dimdeg is false, an n-degree coefficient is fitted iff mindeg <= n <= maxdeg. For a 2D-polynomial this means that N $*$ (N + 1) / 2 coefficients are fitted. For a 3D-polynomial, it is N $*$ (N + 1) $*$ (N + 2) / 6.

If dimdeg is true, nci = 1 + maxdeg[i] - mindeg[i] coefficients are fitted for dimension i, i.e. for a 2D-polynomial N = nc1 $*$ nc2 coefficients are fitted.

The number of distinct samples should exceed the number of coefficients to fit. The number of distinct samples may also equal the number of coefficients to fit, but in this case the fit has another meaning (any non-zero residual is due to rounding errors, not a fitting error). It is an error to try to fit more coefficients than there are distinct samples.

If the relative uncertainties of the sampled values are known, they may be passed via fitsigm. NULL means that all uncertainties equals one. This uncertainties can be applied to the least squares method by converting them to weights (w = 1.0 / sigma^2).

sampsym is ignored if mindeg is nonzero, otherwise the caller may use sampsym to indicate an a priori knowledge that the sampling positions are symmetric. NULL indicates no knowledge of such symmetry. sampsym[i] may be set to true iff the sampling is symmetric around u_i, where u_i is the mean of the sampling positions in dimension i.

In 1D this implies that the sampling points as pairs average u_0 (with an odd number of samples one sample must equal u_0). E.g. both x = (1, 2, 4, 6, 7) and x = (1, 6, 4, 2, 7) have sampling symmetry, while x = (1, 2, 4, 6) does not.

In 2D this implies that the sampling points are symmetric in the 2D-plane. For the first dimension sampling symmetry means that the sampling is line- symmetric around y = u_1, while for the second dimension, sampling symmetry implies line-symmetry around x = u_2. Point symmetry around (x,y) = (u_1, u_2) means that both sampsym[0] and sampsym[1] may be set to true.

Knowledge of symmetric sampling allows the fit to be both faster and eliminates certain round-off errors.

For higher order fitting the fitting problem known as "Runge's phenomenon" is minimized using the socalled "↩ Chebyshev nodes" as sampling points. For Chebyshev nodes sampsym can be set to CPL_TRUE.

Warning: An increase in the polynomial degree will normally reduce the fitting error. However, due to rounding errors and the limited accuracy of the solver of the normal equations, an increase in the polynomial degree may at some point cause the fitting error to *increase*. In some cases this happens with an increase of the polynomial degree from 8 to 9.

The fit is done in the following steps: 1) If fitsigm is non-NULL. The factors are applied to the values. 2) If mindeg is zero, the sampling positions are first transformed into Xhat_i = X_i - mean(X_i), i=1, .., dimension. 3) The Vandermonde matrix is formed from Xhat. If fitsigm is non-NULL, the weights are also taken into account. 4) The normal equations of the Vandermonde matrix is solved. 5) If mindeg is zero, the resulting polynomial in Xhat is transformed back to X.

For a univariate (1D) fit the call requires 6MN + N^3/3 + 7/2N^2 + O(M) FLOPs where M is the number of data points and where N is the number of polynomial coefficients to fit, N = 1 + maxdeg - mindeg.

For a bivariate fit the call requires MN^2 + N^3/3 + O(MN) FLOPs where M is the number of data points and where N is the number of polynomial coefficients to fit.

Examples of usage:

```
cpl_polynomial  * fit1d     = cpl_polynomial_new(1);
cpl_matrix      * samppos1d = my_sampling_points_1d(); // 1-row matrix
cpl_vector      * fitvals   = my_sampling_values();
const cpl_boolean sampsym   = CPL_TRUE;
const cpl_size    maxdeg1d  = 4; // Fit 5 coefficients
cpl_error_code    error1d
    = cpl_polynomial_fit(fit1d, samppos1d, &sampsym, fitvals, NULL,
                         CPL_FALSE, NULL, &maxdeg1d);
cpl_polynomial  * fit2d      = cpl_polynomial_new(2);
cpl_matrix      * samppos2d  = my_sampling_points_2d(); // 2-row matrix
cpl_vector      * fitvals    = my_sampling_values();
const cpl_size    maxdeg2d[] = {2, 1}; // Fit 6 coefficients
cpl_error_code    error2d
    = cpl_polynomial_fit(fit2d, samppos2d, NULL, fitvals, NULL, CPL_FALSE,
                         NULL, maxdeg2d);
cpl_polynomial  * fit3d      = cpl_polynomial_new(3);
cpl_matrix      * samppos3d  = my_sampling_points_3d(); // 3-row matrix
cpl_vector      * fitvals    = my_sampling_values();
const cpl_size    maxdeg3d[] = {2, 1, 2}; // Fit 18 coefficients
cpl_error_code    error3d
    = cpl_polynomial_fit(fit3d, samppos3d, NULL, fitvals, NULL, CPL_FALSE,
                         NULL, maxdeg3d);
```

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if a mindeg value is negative, or if a maxdeg value is less than the corresponding mindeg value.
- CPL_ERROR_DATA_NOT_FOUND if the number of columns in samppos is less than the number of coefficients to be determined.
- CPL_ERROR_INCOMPATIBLE_INPUT if samppos, fitvals or fitsigm have incompatible sizes, or if samppos, self or sampsym have incompatible sizes.
- CPL_ERROR_SINGULAR_MATRIX if samppos contains too few distinct values
- CPL_ERROR_DIVISION_BY_ZERO if an element in fitsigm is zero
- CPL_ERROR_UNSUPPORTED_MODE if the polynomial dimension exceeds three

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NULL_INPUT, cpl_matrix↩ _get_ncol_(), cpl_polynomial_get_dimension(), CPL_SIZE_FORMAT, and cpl_vector_get_size().

4.38.2.15 cpl_polynomial_fit_1d_create()

```
cpl_polynomial* cpl_polynomial_fit_1d_create (
             const cpl_vector * x_pos,
             const cpl_vector * values,
             cpl_size degree,
             double * pmse )
```

Fit a 1D-polynomial to a 1D-signal in a least squares sense.

Parameters

x_pos	Vector of positions of the signal to fit.
values	Vector of values of the signal to fit.
degree	Non-negative polynomial degree.
pmse	Iff pmse is not null, the mean squared error on success

Returns

The fitted polynomial or NULL on error

See also

cpl_polynomial_fit()

Deprecated Replace this call with cpl_polynomial_fit() and optionally cpl_vector_fill_polynomial_fit_residual().

4.38.2.16 cpl_polynomial_fit_2d_create()

```
cpl_polynomial* cpl_polynomial_fit_2d_create (
            const cpl_bivector * xy_pos,
            const cpl_vector * values,
            cpl_size degree,
            double * pmse )
```

Fit a 2D-polynomial to a 2D-surface in a least squares sense.

Parameters

xy_pos	Bivector positions of the surface to fit.
values	Vector of values of the surface to fit.
degree	Non-negative polynomial degree.
pmse	Iff pmse is not null, the mean squared error on success

Returns

The fitted polynomial or NULL on error

See also

cpl_polynomial_fit()

Deprecated Replace this call with cpl_polynomial_fit() and optionally cpl_vector_fill_polynomial_fit_residual().

4.38.2.17 cpl_polynomial_get_coeff()

```
double cpl_polynomial_get_coeff (
            const cpl_polynomial * self,
            const cpl_size * pows )
```

Get a coefficient of the polynomial.

Parameters

self	The polynomial to process
pows	The non-negative power(s) of the variable(s)

Returns

The coefficient or undefined on error

Note

For an N-dimensional polynomial the complexity is O(N)

See also

cpl_polynomial_set_coeff

Requesting the value of a coefficient that has not been set is allowed, in this case zero is returned.

Example of usage:

```
const cpl_size power       = 3;
double         coefficient = cpl_polynomial_get_coeff(poly1d, &power);
```

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if pows contains negative values

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, and CPL_SIZE_FORMAT.

Referenced by cpl_polynomial_extract().

4.38.2.18 cpl_polynomial_get_degree()

```
cpl_size cpl_polynomial_get_degree (
            const cpl_polynomial * self )
```

The degree of the polynomial.

Parameters

self	The polynomial to access

Returns

The degree or negative on error

The degree is the highest sum of exponents (with a non-zero coefficient).

If there are no non-zero coefficients the degree is zero.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_polynomial_extract().

4.38.2.19 cpl_polynomial_get_dimension()

```
cpl_size cpl_polynomial_get_dimension (
            const cpl_polynomial * self )
```

The dimension of the polynomial.

Parameters

self	The polynomial to access

Returns

The dimension or negative on error

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_image_fill_jacobian_polynomial(), cpl_image_fill_polynomial(), cpl_image_warp_polynomial(), cpl_polynomial_add(), cpl_polynomial_fit(), cpl_polynomial_multiply(), cpl_polynomial_multiply_scalar(), cpl_polynomial_shift_1d(), cpl_polynomial_subtract(), cpl_vector_fill_polynomial_fit_residual(), and cpl_wlcalib_find_best_1d().

4.38.2.20 cpl_polynomial_multiply()

```
cpl_error_code cpl_polynomial_multiply (
            cpl_polynomial * self,
            const cpl_polynomial * first,
            const cpl_polynomial * second )
```

Multiply two polynomials of the same dimension.

Parameters

self	The polynomial to hold the result
first	The polynomial to multiply
second	The polynomial to multiply

Returns

CPL_ERROR_NONE or the relevant CPL error code

Note

self may be passed also as first and/or second

See also

cpl_polynomial_add()

Possible CPL error code set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if the polynomials do not have identical dimensions

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NULL_INPUT, and cpl_↩ polynomial_get_dimension().

4.38.2.21 cpl_polynomial_multiply_scalar()

```
cpl_error_code cpl_polynomial_multiply_scalar (
            cpl_polynomial * self,
            const cpl_polynomial * other,
            double factor )
```

Multiply a polynomial with a scalar.

Parameters

self	The polynomial to hold the result
other	The polynomial to scale of same dimension, may equal self
factor	The factor to multiply with

Returns

CPL_ERROR_NONE or the relevant CPL error code

Note

If factor is zero all coefficients are reset, if it is 1 all are copied

Possible CPL error code set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if the two dimensions do not match

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NULL_INPUT, and cpl_↩ polynomial_get_dimension().

4.38.2.22 cpl_polynomial_set_coeff()

```
cpl_error_code cpl_polynomial_set_coeff (
            cpl_polynomial * self,
            const cpl_size * pows,
            double value )
```

Set a coefficient of the polynomial.

Parameters

self	The polynomial to modify
pows	The non-negative power(s) of the variable(s)
value	The coefficient

Returns

CPL_ERROR_NONE on success or else the relevant _cpl_error_code_

Note

For an N-dimensional polynomial the complexity is O(N)

See also

cpl_polynomial_get_coeff

The array pows is assumed to have the size of the polynomial dimension.

If the coefficient is already there, it is overwritten, if not, a new coefficient is added to the polynomial. This may cause the degree of the polynomial to be increased, or if the new coefficient is zero, to decrease.

Setting the coefficient of x1^4 * x3^2 in the 4-dimensional polynomial poly4d to 12.3 would be performed by:

```
const cpl_size pows[] = {4, 0, 2, 0};
cpl_error_code error  = cpl_polynomial_set_coeff(poly4d, pows, 12.3);
```

Setting the coefficient of x^3 in the 1-dimensional polynomial poly1d to 12.3 would be performed by:

```
const cpl_size power = 3;
cpl_error_code error = cpl_polynomial_set_coeff(poly1d, &power, 12.3);
```

When setting multiple coefficients there is a small efficiency gain when setting the highest powers first.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if pows contains negative values

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, and CPL_SIZE_FORMAT.

4.38.2.23 cpl_polynomial_shift_1d()

```
cpl_error_code cpl_polynomial_shift_1d (
            cpl_polynomial * p,
            cpl_size i,
            double u )
```

Modify p, p(x0, x1, ..., xi, ...) := (x0, x1, ..., xi+u, ...)

Parameters

p	The polynomial to be modified in place
i	The dimension to shift (0 for first)
u	The shift

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_

Shifting the polynomial p(x) = x^n with u = 1 will generate the binomial coefficients for n.

Shifting the coordinate system to (x,y) for the 2D-polynomium poly2d:

```
cpl_polynomial_shift_1d(poly2d, 0, x);
cpl_polynomial_shift_1d(poly2d, 1, y);
```

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if i is negative
- CPL_ERROR_ACCESS_OUT_OF_RANGE if i exceeds the dimension of p

References cpl_ensure_code, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, and cpl_polynomial_get_dimension().

4.38.2.24 cpl_polynomial_solve_1d()

```
cpl_error_code cpl_polynomial_solve_1d (
            const cpl_polynomial * p,
            double x0,
            double * px,
            cpl_size mul )
```

A real solution to p(x) = 0 using Newton-Raphsons method.

Parameters

p	The 1D-polynomial
x0	First guess of the solution
px	The solution, on error see below
mul	The root multiplicity (or 1 if unknown)

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_

Even if a real solution exists, it may not be found if the first guess is too far from the solution. But a solution is guaranteed to be found if all roots of p are real (except in the case where the derivative at the first guess happens to be zero, see below - for an n-degree polynomial there are up to n-1 such guesses). If the constant term is zero, the solution 0 will be returned regardless of the first guess.

No solution is found when the iterative process stops because: 1) It can not proceed because p`(x) = 0 (CPL_ER↩ ROR_DIVISION_BY_ZERO). 2) Only a finite number of iterations are allowed (CPL_ERROR_CONTINUE). Either can happen due to an an actual lack of a real solution or due to an insufficiently good first guess. In these two cases ∗px is set to the value where the error occurred. In case of other errors ∗px is unmodified.

The accuracy and robustness deteriorates with increasing multiplicity of the solution. This is also the case with numerical multiplicity, i.e. when multiple solutions are located close together.

mul is assumed to be the multiplicity of the solution. Knowledge of the root multiplicity often improves the robustness and accuracy. If there is no knowledge of the root multiplicity mul should be 1. Setting mul to a too high value should be avoided.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INVALID_TYPE if the polynomial has the wrong dimension
- CPL_ERROR_ILLEGAL_INPUT if the multiplicity is non-positive
- CPL_ERROR_DIVISION_BY_ZERO if a division by zero occurs
- CPL_ERROR_CONTINUE if the algorithm does not converge

4.38.2.25 cpl_polynomial_subtract()

```
cpl_error_code cpl_polynomial_subtract (
            cpl_polynomial * self,
            const cpl_polynomial * first,
            const cpl_polynomial * second )
```

Subtract two polynomials of the same dimension.

Parameters

self	The polynomial to hold the result
first	The polynomial to subtract from, or NULL
second	The polynomial to subtract, or NULL

Returns

CPL_ERROR_NONE or the relevant CPL error code

Note

self may be passed also as first and/or second

See also

cpl_polynomial_add()

Possible CPL error code set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if the polynomials do not have identical dimensions

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NULL_INPUT, and cpl_↩ polynomial_get_dimension().

4.38.2.26 cpl_vector_fill_polynomial()

```
cpl_error_code cpl_vector_fill_polynomial (
            cpl_vector * v,
            const cpl_polynomial * p,
            double x0,
            double d )
```

Evaluate a 1D-polynomial on equidistant points using Horner's rule.

Parameters

v	Preallocated vector to contain the result
p	The 1D-polynomial
x0	The first point of evaluation
d	The increment between points of evaluation

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_

See also

cpl_vector_fill

The evaluation points are x_i = x0 + i ∗ d, i=0, 1, ..., n-1, where n is the length of the vector.

If d is zero it is preferable to simply use cpl_vector_fill(v, cpl_polynomial_eval_1d(p, x0, NULL)).

The call requires about 2nm FLOPs, where m+1 is the number of coefficients in p.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INVALID_TYPE if the polynomial has the wrong dimension

References cpl_ensure_code, CPL_ERROR_INVALID_TYPE, and CPL_ERROR_NULL_INPUT.

4.38.2.27 cpl_vector_fill_polynomial_fit_residual()

```
cpl_error_code cpl_vector_fill_polynomial_fit_residual (
             cpl_vector * self,
             const cpl_vector * fitvals,
             const cpl_vector * fitsigm,
             const cpl_polynomial * fit,
             const cpl_matrix * samppos,
             double * rechisq )
```

Compute the residual of a polynomial fit.

Parameters

self	Vector to hold the fitting residuals, fitvals may be used
fitvals	Vector of the p fitted values
fitsigm	Uncertainties of the sampled values or NULL for a uniform uncertainty
fit	The fitted polynomial
samppos	Matrix of p sample positions, with d rows and p columns
rechisq	If non-NULL, the reduced chi square of the fit

Returns

CPL_ERROR_NONE on success, else the relevant _cpl_error_code_

Note

If necessary, self is resized to the length of fitvals.

See also

cpl_polynomial_fit()

It is allowed to pass the same vector as both fitvals and as self, in which case fitvals is overwritten with the residuals.

If the relative uncertainties of the sampled values are known, they may be passed via fitsigm. NULL means that all uncertainties equal one. The uncertainties are taken into account when computing the reduced chi square value.

If rechisq is non-NULL, the reduced chi square of the fit is computed as well.

The mean square error, which was computed directly by the former CPL functions cpl_polynomial_fit_1d_create() and cpl_polynomial_fit_2d_create() can be computed from the fitting residual like this:

```
const double mse = cpl_vector_product(fitresidual, fitresidual)
                 / cpl_vector_get_size(fitresidual);
```

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer (other than fitsigm) is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if samppos, fitvals, fitsigm or fit have incompatible sizes
- CPL_ERROR_DIVISION_BY_ZERO if an element in fitsigm is zero
- CPL_ERROR_DATA_NOT_FOUND if the number of columns in samppos is less than the number of coefficients in the fitted polynomial.

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NULL_INPUT, cpl_matrix↩
_get_ncol_(), cpl_polynomial_get_dimension(), and cpl_vector_get_size().

4.39 Properties

Typedefs

- typedef struct _cpl_property_ cpl_property

  The opaque property data type.

Functions

- void cpl_property_delete (cpl_property *self)

  Destroy a property.
- void cpl_property_dump (const cpl_property *, FILE *)

  Print a property.
- cpl_property * cpl_property_duplicate (const cpl_property *other)

  Create a copy of a property.
- int cpl_property_get_bool (const cpl_property *self)

  Get the value of a boolean property.
- char cpl_property_get_char (const cpl_property *self)

  Get the value of a character property.
- const char * cpl_property_get_comment (const cpl_property *self)

  Get the property comment.
- double cpl_property_get_double (const cpl_property *self)

  Get the value of a double property.
- double complex cpl_property_get_double_complex (const cpl_property *self)

  Get the value of a double complex property.
- float cpl_property_get_float (const cpl_property *self)

  Get the value of a float property.
- float complex cpl_property_get_float_complex (const cpl_property *self)

  Get the value of a float complex property.
- int cpl_property_get_int (const cpl_property *self)

  Get the value of an integer property.
- long cpl_property_get_long (const cpl_property *self)

  Get the value of a long property.
- long long cpl_property_get_long_long (const cpl_property *self)

  Get the value of a long long property.
- const char * cpl_property_get_name (const cpl_property *self)

  Get the property name.
- cpl_size cpl_property_get_size (const cpl_property *self)

  Get the current number of elements a property contains.
- const char * cpl_property_get_string (const cpl_property *self)

  Get the value of a string property.
- cpl_type cpl_property_get_type (const cpl_property *self)

  Get the type of a property.
- cpl_property * cpl_property_new (const char *name, cpl_type type)

  Create an empty property of a given type.
- cpl_property * cpl_property_new_array (const char *name, cpl_type type, cpl_size size)

  Create an empty property of a given type and size.
- cpl_error_code cpl_property_set_bool (cpl_property *self, int value)

  Set the value of a boolean property.
- cpl_error_code cpl_property_set_char (cpl_property *self, char value)

Set the value of a character property.

- cpl_error_code cpl_property_set_comment (cpl_property *self, const char *comment)
  Modify a property's comment.
- cpl_error_code cpl_property_set_double (cpl_property *self, double value)
  Set the value of a double property.
- cpl_error_code cpl_property_set_double_complex (cpl_property *self, double complex value)
  Set the value of a double complex property.
- cpl_error_code cpl_property_set_float (cpl_property *self, float value)
  Set the value of a float property.
- cpl_error_code cpl_property_set_float_complex (cpl_property *self, float complex value)
  Set the value of a complex float property.
- cpl_error_code cpl_property_set_int (cpl_property *self, int value)
  Set the value of an integer property.
- cpl_error_code cpl_property_set_long (cpl_property *self, long value)
  Set the value of a long property.
- cpl_error_code cpl_property_set_long_long (cpl_property *self, long long value)
  Set the value of a long long property.
- cpl_error_code cpl_property_set_name (cpl_property *self, const char *name)
  Modify the name of a property.
- cpl_error_code cpl_property_set_string (cpl_property *self, const char *value)
  Set the value of a string property.

4.39.1 Detailed Description

This module implements the property type. The type `cpl_property` is basically a variable container which consists of a name, a type identifier and a specific value of that type. The type identifier always determines the type of the associated value. A property is similar to an ordinary variable and its current value can be set or retrieved through its name. In addition a property may have a descriptive comment associated. A property can be created for the basic types char, bool (int), int, long, float and double. Also C strings are supported. Support for arrays in general is currently not available.

Synopsis:

```
#include <cpl_property.h>
```

4.39.2 Typedef Documentation

4.39.2.1 cpl_property

```
typedef struct _cpl_property_ cpl_property
```

The opaque property data type.

4.39.3 Function Documentation

4.39.3.1 cpl_property_delete()

```
void cpl_property_delete (
            cpl_property * self )
```

Destroy a property.

Parameters

self	The property.

Returns

Nothing.

The function destroys a property of any kind. All property members including their values are properly deallocated. If the property *self* is `NULL`, nothing is done and no error is set.

References cpl_msg_warning().

Referenced by cpl_propertylist_delete(), and cpl_propertylist_empty().

4.39.3.2 cpl_property_dump()

```
void cpl_property_dump (
            const cpl_property * property,
            FILE * stream )
```

Print a property.

Parameters

property	Pointer to property
stream	The output stream

Returns

Nothing.

This function is mainly intended for debug purposes. If the specified stream is `NULL`, it is set to *stdout*. The function used for printing is the standard C `fprintf()`.

References cpl_property_get_bool(), cpl_property_get_char(), cpl_property_get_comment(), cpl_property_get_↩ double(), cpl_property_get_float(), cpl_property_get_int(), cpl_property_get_long(), cpl_property_get_long_long(), cpl_property_get_name(), cpl_property_get_size(), cpl_property_get_string(), cpl_property_get_type(), CPL_TY↩ PE_BOOL, CPL_TYPE_CHAR, CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT, CPL_TYPE_INT, CPL_TYPE_LONG, CPL_TYPE_LONG_LONG, and CPL_TYPE_STRING.

4.39.3.3 cpl_property_duplicate()

```
cpl_property* cpl_property_duplicate (
            const cpl_property * other )
```

Create a copy of a property.

Parameters

other	The property to copy.

Returns

The copy of the given property, or NULL in case of an error. In the latter case an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function returns a copy of the property *self*. The copy is a deep copy, i.e. all property members are copied.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_propertylist_append(), cpl_propertylist_append_property(), cpl_propertylist_duplicate(), and cpl_propertylist_prepend_property().

4.39.3.4 cpl_property_get_bool()

```
int cpl_property_get_bool (
            const cpl_property * self )
```

Get the value of a boolean property.

Parameters

self	The property.

Returns

The current property value. If an error occurs the function returns 0 and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_TYPE_MISMATCH	The property *self* is not of type CPL_TYPE_BOOL.

The function retrieves the boolean value currently stored in the property *self*.

References CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, CPL_TYPE_BOOL, and cpl_type_↩ get_name().

Referenced by cpl_property_dump().

4.39.3.5 cpl_property_get_char()

```
char cpl_property_get_char (
            const cpl_property * self )
```

Get the value of a character property.

Parameters

self	The property.

Returns

The current property value. If an error occurs the function returns '\0' and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The property *self* is not of type `CPL_TYPE_CHAR`.

The function retrieves the character value currently stored in the property *self*.

References CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, CPL_TYPE_CHAR, and cpl_type_↩get_name().

Referenced by cpl_property_dump().

4.39.3.6 cpl_property_get_comment()

```
const char* cpl_property_get_comment (
            const cpl_property * self )
```

Get the property comment.

Parameters

self	The property.

Returns

The comment of the property if it is present or NULL. If an error occurs the function returns NULL and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function returns a handle for the read-only comment string of *self*.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_property_dump(), and cpl_propertylist_dump().

4.39.3.7 cpl_property_get_double()

```
double cpl_property_get_double (
            const cpl_property * self )
```

Get the value of a double property.

Parameters

self	The property.

Returns

The current property value. If an error occurs the function returns 0. and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_TYPE_MISMATCH	The property *self* is not of type CPL_TYPE_DOUBLE or CPL_TYPE_FLOAT.

The function retrieves the value currently stored in the property *self* as a double. The function accepts properties of a floating-point type with a rank less or equal than the function's return type.

References CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT, cpl_type_get_name(), and cpl_type_get_sizeof().

Referenced by cpl_property_dump().

4.39.3.8 cpl_property_get_double_complex()

```
double complex cpl_property_get_double_complex (
            const cpl_property * self )
```

Get the value of a double complex property.

Parameters

self	The property.

Returns

The current property value. If an error occurs the function returns 0. and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The property *self* is not of type `CPL_TYPE_DOUBLE_COMPLEX` or `CPL_TYPE_FLOAT_COMPLEX`.

The function retrieves the value currently stored in the property *self* as a double complex. The function accepts properties of a complex type with a rank less or equal than the function's return type.

References CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, CPL_TYPE_DOUBLE_COMPLEX, CPL_TYPE_FLOAT_COMPLEX, cpl_type_get_name(), and cpl_type_get_sizeof().

4.39.3.9 cpl_property_get_float()

```
float cpl_property_get_float (
            const cpl_property * self )
```

Get the value of a float property.

Parameters

self	The property.

Returns

The current property value. If an error occurs the function returns 0. and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The property *self* is not of type `CPL_TYPE_FLOAT` or `CPL_TYPE_DOUBLE`.

The function retrieves the value currently stored in the property *self* as a float. The function also accepts properties of type double and returns the property value as float.

Note

Calling the function for a property of type double may lead to truncation errors!

References CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, CPL_TYPE_DOUBLE, CPL_TYPE↩_FLOAT, cpl_type_get_name(), and cpl_type_get_sizeof().

Referenced by cpl_property_dump().

4.39.3.10 cpl_property_get_float_complex()

```
float complex cpl_property_get_float_complex (
            const cpl_property * self )
```

Get the value of a float complex property.

Parameters

self	The property.

Returns

The current property value. If an error occurs the function returns 0. and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The property *self* is not of type `CPL_TYPE_FLOAT_COMPLEX` or `CPL_TYPE_DOUBLE_COMPLEX`.

The function retrieves the value currently stored in the property *self* as a float complex. The function also accepts properties of type double complex and returns the property value as float complex.

Note

Calling the function for a property of type double complex may lead to truncation errors!

References CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, CPL_TYPE_DOUBLE_COMPLEX, CPL_TYPE_FLOAT_COMPLEX, cpl_type_get_name(), and cpl_type_get_sizeof().

4.39.3.11 cpl_property_get_int()

```
int cpl_property_get_int (
            const cpl_property * self )
```

Get the value of an integer property.

Parameters

self	The property.

Returns

The current property value. If an error occurs the function returns 0 and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The property *self* is not of type `CPL_TYPE_INT`.

The function retrieves the integer value currently stored in the property *self*.

References CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, cpl_type_get_name(), and CPL_T↩YPE_INT.

Referenced by cpl_property_dump().

4.39.3.12 cpl_property_get_long()

```
long cpl_property_get_long (
            const cpl_property * self )
```

Get the value of a long property.

Parameters

self	The property.

Returns

The current property value. If an error occurs the function returns 0 and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The property *self* is not of type `CPL_TYPE_LONG` or `CPL_TYP`↩`E_INT`.

The function retrieves the value currently stored in the property *self* as a long integer. The function accepts properties of integer type with a rank less or equal than the function's return type.

References CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, cpl_type_get_name(), CPL_TYPE↩_INT, and CPL_TYPE_LONG.

Referenced by cpl_property_dump().

4.39.3.13 cpl_property_get_long_long()

```
long long cpl_property_get_long_long (
            const cpl_property * self )
```

Get the value of a long long property.

Parameters

self	The property.

Returns

The current property value. If an error occurs the function returns 0 and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The property *self* is not of type `CPL_TYPE_LONG_LONG`, `CPL_↩TYPE_LONG`, or `CPL_TYPE_INT`.

The function retrieves the value currently stored in the property *self* as a long long integer. The function accepts properties of integer type with a rank less or equal than the function's return type.

References CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, cpl_type_get_name(), CPL_TYPE↩_INT, CPL_TYPE_LONG, and CPL_TYPE_LONG_LONG.

Referenced by cpl_property_dump().

4.39.3.14 cpl_property_get_name()

```
const char* cpl_property_get_name (
            const cpl_property * self )
```

Get the property name.

Parameters

self	The property.

Returns

The name of the property or `NULL` if an error occurred. In the latter case an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function returns a handle for the read-only identifier string of *self*.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_property_dump(), and cpl_propertylist_dump().

4.39.3.15 cpl_property_get_size()

```
cpl_size cpl_property_get_size (
            const cpl_property * self )
```

Get the current number of elements a property contains.

Parameters

self	The property.

Returns

The current number of elements or -1 in case of an error. If an error occurred an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function returns the current number of elements the property *self* contains. This is the array size of the property's value and in particular, for a property of the type `CPL_TYPE_STRING`, it is the length of the string as given by the **strlen()** function plus 1.

Testing whether the value of a string property equals a given string can be done like this:

```
if (cpl_property_get_size(property) == 8 &&
    memcmp(cpl_property_get_string(property), "EXAMPLE", 7) == 0) {
    my_strings_are_equal();
} else {
    my_strings_are_not_equal();
}
```

When the length of the string to compare to is known at compile time, the call to memcmp() will typically be inlined (and no calls to strlen() et al are needed). Further, if the length of the property string value is greater than 1, then it is not necessary to test whether its type is CPL_TYPE_STRING, since all properties of a numerical type has size 1.

See also

cpl_property_get_string()

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_property_dump(), and cpl_propertylist_dump().

4.39.3.16 cpl_property_get_string()

```
const char* cpl_property_get_string (
            const cpl_property * self )
```

Get the value of a string property.

Parameters

self	The property.

Returns

The current property value. If an error occurs the function returns `NULL` and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The property *self* is not of type `CPL_TYPE_STRING`.

The function retrieves the string value currently stored in the property *self*.

See also

cpl_property_get_size()

References CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, cpl_type_get_name(), and CPL_TYPE_STRING.

Referenced by cpl_property_dump().

4.39.3.17 cpl_property_get_type()

```
cpl_type cpl_property_get_type (
            const cpl_property * self )
```

Get the type of a property.

Parameters

self	The property.

Returns

The type code of this property. In case of an error the returned type code is `CPL_TYPE_INVALID` and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

This function returns the type of the data value stored in the property *self*.

References CPL_ERROR_NULL_INPUT, and CPL_TYPE_INVALID.

Referenced by cpl_property_dump().

4.39.3.18 cpl_property_new()

```
cpl_property* cpl_property_new (
            const char * name,
            cpl_type type )
```

Create an empty property of a given type.

Parameters

name	Property name.
type	Property type flag.

Returns

The newly created property, or `NULL` if it could not be created. In the latter case an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *name* is a NULL pointer.
CPL_ERROR_INVALID_TYPE	The requested property type *type* is not supported.

The function allocates memory for a property of type *type* and assigns the identifier string *name* to the newly created property.

The returned property must be destroyed using the property destructor **cpl_property_delete()**.

See also

cpl_property_delete()

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_propertylist_append_bool(), cpl_propertylist_append_char(), cpl_propertylist_append_double(), cpl_propertylist_append_double_complex(), cpl_propertylist_append_float(), cpl_propertylist_append_float_complex(), cpl_propertylist_append_int(), cpl_propertylist_append_long(), cpl_propertylist_append_long_long(), cpl_propertylist_append_string(), cpl_propertylist_prepend_bool(), cpl_propertylist_prepend_char(), cpl_propertylist_prepend_double(), cpl_propertylist_prepend_double_complex(), cpl_propertylist_prepend_float(), cpl_propertylist_prepend_float_complex(), cpl_propertylist_prepend_int(), cpl_propertylist_prepend_long(), cpl_propertylist_prepend_long_long(), and cpl_propertylist_prepend_string().

4.39.3.19 cpl_property_new_array()

```
cpl_property* cpl_property_new_array (
            const char * name,
            cpl_type type,
            cpl_size size )
```

Create an empty property of a given type and size.

Parameters

name	Property name.
type	Property type flag.
size	Size of the property value.

Returns

The newly created property, or NULL if it could not be created. in the latter case an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *name* is a NULL pointer.
CPL_ERROR_INVALID_TYPE	The requested property type *type* is not supported.
CPL_ERROR_ILLEGAL_INPUT	The requested *size* is non-positive.

The function allocates memory for a property of type *type* and assigns the identifier string *name* to the newly created property. The property value is created such that *size* elements of type *type* can be stored.

The returned property must be destroyed using the property destructor **cpl_property_delete()**.

See also

cpl_property_delete()

References CPL_ERROR_NULL_INPUT.

4.39.3.20 cpl_property_set_bool()

```
cpl_error_code cpl_property_set_bool (
            cpl_property * self,
            int value )
```

Set the value of a boolean property.

Parameters

self	The property.
value	New value.

Returns

The function returns `CPL_ERROR_NONE` on success and an appropriate error code if an error occurred. In the latter case the error code is also set in the error handling system.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The property *self* is not of type `CPL_TYPE_BOOL`.

The function replaces the current boolean value of *self* with a a 0, if *value* is equal to 0. If *value* is different from 0 any previous value of *self* is replaced by a 1.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, CPL_TYPE↩_BOOL, cpl_type_get_name(), and cpl_type_get_sizeof().

Referenced by cpl_propertylist_append_bool(), and cpl_propertylist_prepend_bool().

4.39.3.21 cpl_property_set_char()

```
cpl_error_code cpl_property_set_char (
            cpl_property * self,
            char value )
```

Set the value of a character property.

Parameters

self	The property.
value	New value.

Returns

The function returns CPL_ERROR_NONE on success and an appropriate error code if an error occurred. In the latter case the error code is also set in the error handling system.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_TYPE_MISMATCH	The property *self* is not of type CPL_TYPE_CHAR.

The function replaces the current character value of *self* with a copy of the value *value*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, CPL_TYPE↩_CHAR, cpl_type_get_name(), and cpl_type_get_sizeof().

Referenced by cpl_propertylist_append_char(), and cpl_propertylist_prepend_char().

4.39.3.22 cpl_property_set_comment()

```
cpl_error_code cpl_property_set_comment (
            cpl_property * self,
            const char * comment )
```

Modify a property's comment.

Parameters

self	The property.
comment	New comment.

Returns

The function returns CPL_ERROR_NONE on success and an appropriate error code if an error occurred. In the latter case the error code is also set in the error handling system.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *comment* is a NULL pointer.

The function replaces the current comment field of *self* with a copy of the string *comment*. The new comment may be NULL. In this case the function effectively deletes the current comment.

References CPL_ERROR_NULL_INPUT.

4.39.3.23 cpl_property_set_double()

```
cpl_error_code cpl_property_set_double (
            cpl_property * self,
            double value )
```

Set the value of a double property.

Parameters

self	The property.
value	New value.

Returns

The function returns CPL_ERROR_NONE on success and an appropriate error code if an error occurred. In the latter case the error code is also set in the error handling system.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_TYPE_MISMATCH	The property *self* is not of type CPL_TYPE_DOUBLE.

The function replaces the current double value of *self* with a copy of the value *value*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, CPL_TYPE↩_DOUBLE, cpl_type_get_name(), and cpl_type_get_sizeof().

Referenced by cpl_propertylist_append_double(), and cpl_propertylist_prepend_double().

4.39.3.24 cpl_property_set_double_complex()

```
cpl_error_code cpl_property_set_double_complex (
            cpl_property * self,
            double complex value )
```

Set the value of a double complex property.

Parameters

self	The property.
value	New value.

Returns

The function returns `CPL_ERROR_NONE` on success and an appropriate error code if an error occurred. In the latter case the error code is also set in the error handling system.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The property *self* is not of type `CPL_TYPE_DOUBLE_COMPLEX`.

The function replaces the current double value of *self* with a copy of the value *value*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, CPL_TYPE↩_DOUBLE_COMPLEX, cpl_type_get_name(), and cpl_type_get_sizeof().

Referenced by cpl_propertylist_append_double_complex(), and cpl_propertylist_prepend_double_complex().

4.39.3.25 cpl_property_set_float()

```
cpl_error_code cpl_property_set_float (
            cpl_property * self,
            float value )
```

Set the value of a float property.

Parameters

self	The property.
value	New value.

Returns

The function returns CPL_ERROR_NONE on success and an appropriate error code if an error occurred. In the latter case the error code is also set in the error handling system.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_TYPE_MISMATCH	The property *self* is not of type CPL_TYPE_FLOAT.

The function replaces the current float value of *self* with a copy of the value *value*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, CPL_TYPE↩_FLOAT, cpl_type_get_name(), and cpl_type_get_sizeof().

Referenced by cpl_propertylist_append_float(), and cpl_propertylist_prepend_float().

4.39.3.26 cpl_property_set_float_complex()

```
cpl_error_code cpl_property_set_float_complex (
            cpl_property * self,
            float complex value )
```

Set the value of a complex float property.

Parameters

self	The property.
value	New value.

Returns

The function returns CPL_ERROR_NONE on success and an appropriate error code if an error occurred. In the latter case the error code is also set in the error handling system.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_TYPE_MISMATCH	The property *self* is not of type CPL_TYPE_COMPLEX_FLOAT.

The function replaces the current complex float value of *self* with a copy of the value *value*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, CPL_TYPE↩_FLOAT_COMPLEX, cpl_type_get_name(), and cpl_type_get_sizeof().

Referenced by cpl_propertylist_append_float_complex(), and cpl_propertylist_prepend_float_complex().

4.39.3.27 cpl_property_set_int()

```
cpl_error_code cpl_property_set_int (
            cpl_property * self,
            int value )
```

Set the value of an integer property.

Parameters

self	The property.
value	New value.

Returns

The function returns CPL_ERROR_NONE on success and an appropriate error code if an error occurred. In the latter case the error code is also set in the error handling system.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_TYPE_MISMATCH	The property *self* is not of type CPL_TYPE_INT.

The function replaces the current integer value of *self* with a copy of the value *value*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, cpl_type_↩ get_name(), cpl_type_get_sizeof(), and CPL_TYPE_INT.

Referenced by cpl_propertylist_append_int(), and cpl_propertylist_prepend_int().

4.39.3.28 cpl_property_set_long()

```
cpl_error_code cpl_property_set_long (
            cpl_property * self,
            long value )
```

Set the value of a long property.

Parameters

self	The property.
value	New value.

Returns

The function returns CPL_ERROR_NONE on success and an appropriate error code if an error occurred. In the latter case the error code is also set in the error handling system.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_TYPE_MISMATCH	The property *self* is not of type CPL_TYPE_LONG.

The function replaces the current long value of *self* with a copy of the value *value*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, cpl_type_get_name(), cpl_type_get_sizeof(), and CPL_TYPE_LONG.

Referenced by cpl_propertylist_append_long(), and cpl_propertylist_prepend_long().

4.39.3.29 cpl_property_set_long_long()

```
cpl_error_code cpl_property_set_long_long (
            cpl_property * self,
            long long value )
```

Set the value of a long long property.

Parameters

self	The property.
value	New value.

Returns

The function returns CPL_ERROR_NONE on success and an appropriate error code if an error occurred. In the latter case the error code is also set in the error handling system.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.
CPL_ERROR_TYPE_MISMATCH	The property *self* is not of type CPL_TYPE_LONG.

The function replaces the current long long value of *self* with a copy of the value *value*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, cpl_type_get_name(), cpl_type_get_sizeof(), and CPL_TYPE_LONG_LONG.

Referenced by cpl_propertylist_append_long_long(), and cpl_propertylist_prepend_long_long().

4.39.3.30 cpl_property_set_name()

```
cpl_error_code cpl_property_set_name (
            cpl_property * self,
            const char * name )
```

Modify the name of a property.

Parameters

self	The property.
name	New name.

Returns

The function returns CPL_ERROR_NONE on success and an appropriate error code if an error occurred. In the latter case the error code is also set in the error handling system.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a NULL pointer.

The function replaces the current name of *self* with a copy of the string *name*. The function returns an error if *name* is NULL.

References CPL_ERROR_NULL_INPUT.

4.39.3.31 cpl_property_set_string()

```
cpl_error_code cpl_property_set_string (
            cpl_property * self,
            const char * value )
```

Set the value of a string property.

Parameters

self	The property.
value	New value.

Returns

The function returns `CPL_ERROR_NONE` on success and an appropriate error code if an error occurred. In the latter case the error code is also set in the error handling system.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *value* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The property *self* is not of type `CPL_TYPE_STRING`.

The function replaces the current string value of *self* with a copy of the value *value*.

References CPL_ERROR_NULL_INPUT, CPL_ERROR_TYPE_MISMATCH, cpl_type_get_name(), and CPL_T↩YPE_STRING.

Referenced by cpl_propertylist_append_string(), and cpl_propertylist_prepend_string().

4.40 Property Lists

Typedefs

- typedef struct _cpl_propertylist_ cpl_propertylist

  The opaque property list data type.
- typedef int(∗ cpl_propertylist_compare_func) (const cpl_property ∗first, const cpl_property ∗second)

  The property comparison function data type.

Functions

- cpl_error_code cpl_propertylist_append (cpl_propertylist ∗self, const cpl_propertylist ∗other)

  Append a property list..
- cpl_error_code cpl_propertylist_append_bool (cpl_propertylist ∗self, const char ∗name, int value)

  Append a boolean value to a property list.
- cpl_error_code cpl_propertylist_append_char (cpl_propertylist ∗self, const char ∗name, char value)

  Append a character value to a property list.
- cpl_error_code cpl_propertylist_append_double (cpl_propertylist ∗self, const char ∗name, double value)

  Append a double value to a property list.
- cpl_error_code cpl_propertylist_append_double_complex (cpl_propertylist ∗self, const char ∗name, double complex value)

  Append a double complex value to a property list.
- cpl_error_code cpl_propertylist_append_float (cpl_propertylist ∗self, const char ∗name, float value)

  Append a float value to a property list.
- cpl_error_code cpl_propertylist_append_float_complex (cpl_propertylist ∗self, const char ∗name, float complex value)

  Append a float complex value to a property list.
- cpl_error_code cpl_propertylist_append_int (cpl_propertylist ∗self, const char ∗name, int value)

  Append an integer value to a property list.
- cpl_error_code cpl_propertylist_append_long (cpl_propertylist ∗self, const char ∗name, long value)

  Append a long value to a property list.
- cpl_error_code cpl_propertylist_append_long_long (cpl_propertylist ∗self, const char ∗name, long long value)

  Append a long long value to a property list.
- cpl_error_code cpl_propertylist_append_property (cpl_propertylist ∗self, const cpl_property ∗property)

  Append a duplicate of a property to a property list.
- cpl_error_code cpl_propertylist_append_string (cpl_propertylist ∗self, const char ∗name, const char ∗value)

  Append a string value to a property list.
- cpl_error_code cpl_propertylist_copy_property (cpl_propertylist ∗self, const cpl_propertylist ∗other, const char ∗name)

  Copy a property from another property list.
- cpl_error_code cpl_propertylist_copy_property_regexp (cpl_propertylist ∗self, const cpl_propertylist ∗other, const char ∗regexp, int invert)

  Copy matching properties from another property list.
- void cpl_propertylist_delete (cpl_propertylist ∗self)

  Destroy a property list.
- void cpl_propertylist_dump (const cpl_propertylist ∗self, FILE ∗stream)

  Print a property list.
- cpl_propertylist ∗ cpl_propertylist_duplicate (const cpl_propertylist ∗self)

  Create a copy of the given property list.
- void cpl_propertylist_empty (cpl_propertylist ∗self)

Remove all properties from a property list.

- int cpl_propertylist_erase (cpl_propertylist *self, const char *name)

  Erase the given property from a property list.
- int cpl_propertylist_erase_regexp (cpl_propertylist *self, const char *regexp, int invert)

  Erase all properties with name matching a given regular expression.
- cpl_property * cpl_propertylist_get (cpl_propertylist *self, long position)

  Access property list elements by index.
- int cpl_propertylist_get_bool (const cpl_propertylist *self, const char *name)

  Get the boolean value of the given property list entry.
- char cpl_propertylist_get_char (const cpl_propertylist *self, const char *name)

  Get the character value of the given property list entry.
- const char * cpl_propertylist_get_comment (const cpl_propertylist *self, const char *name)

  Get the comment of the given property list entry.
- const cpl_property * cpl_propertylist_get_const (const cpl_propertylist *self, long position)

  Access property list elements by index.
- double cpl_propertylist_get_double (const cpl_propertylist *self, const char *name)

  Get the double value of the given property list entry.
- double complex cpl_propertylist_get_double_complex (const cpl_propertylist *self, const char *name)

  Get the double complex value of the given property list entry.
- float cpl_propertylist_get_float (const cpl_propertylist *self, const char *name)

  Get the float value of the given property list entry.
- float complex cpl_propertylist_get_float_complex (const cpl_propertylist *self, const char *name)

  Get the float complex value of the given property list entry.
- int cpl_propertylist_get_int (const cpl_propertylist *self, const char *name)

  Get the integer value of the given property list entry.
- long cpl_propertylist_get_long (const cpl_propertylist *self, const char *name)

  Get the long value of the given property list entry.
- long long cpl_propertylist_get_long_long (const cpl_propertylist *self, const char *name)

  Get the long long value of the given property list entry.
- cpl_property * cpl_propertylist_get_property (cpl_propertylist *self, const char *name)

  Access property list elements by property name.
- const cpl_property * cpl_propertylist_get_property_const (const cpl_propertylist *self, const char *name)

  Access property list elements by property name.
- cpl_size cpl_propertylist_get_size (const cpl_propertylist *self)

  Get the current size of a property list.
- const char * cpl_propertylist_get_string (const cpl_propertylist *self, const char *name)

  Get the string value of the given property list entry.
- cpl_type cpl_propertylist_get_type (const cpl_propertylist *self, const char *name)

  Get the the type of a property list entry.
- int cpl_propertylist_has (const cpl_propertylist *self, const char *name)

  Check whether a property is present in a property list.
- cpl_error_code cpl_propertylist_insert_after_bool (cpl_propertylist *self, const char *after, const char *name, int value)

  Insert a boolean value into a property list after the given position.
- cpl_error_code cpl_propertylist_insert_after_char (cpl_propertylist *self, const char *after, const char *name, char value)

  Insert a character value into a property list after the given position.
- cpl_error_code cpl_propertylist_insert_after_double (cpl_propertylist *self, const char *after, const char *name, double value)

  Insert a double value into a property list after the given position.

- cpl_error_code cpl_propertylist_insert_after_double_complex (cpl_propertylist *self, const char *after, const char *name, double complex value)

  Insert a double complex value into a property list after the given position.
- cpl_error_code cpl_propertylist_insert_after_float (cpl_propertylist *self, const char *after, const char *name, float value)

  Insert a float value into a property list after the given position.
- cpl_error_code cpl_propertylist_insert_after_float_complex (cpl_propertylist *self, const char *after, const char *name, float complex value)

  Insert a float complex value into a property list after the given position.
- cpl_error_code cpl_propertylist_insert_after_int (cpl_propertylist *self, const char *after, const char *name, int value)

  Insert a integer value into a property list after the given position.
- cpl_error_code cpl_propertylist_insert_after_long (cpl_propertylist *self, const char *after, const char *name, long value)

  Insert a long value into a property list after the given position.
- cpl_error_code cpl_propertylist_insert_after_long_long (cpl_propertylist *self, const char *after, const char *name, long long value)

  Insert a long long value into a property list after the given position.
- cpl_error_code cpl_propertylist_insert_after_property (cpl_propertylist *self, const char *after, const cpl_property *property)

  Insert a property into a property list after the given position.
- cpl_error_code cpl_propertylist_insert_after_string (cpl_propertylist *self, const char *after, const char *name, const char *value)

  Insert a string value into a property list after the given position.
- cpl_error_code cpl_propertylist_insert_bool (cpl_propertylist *self, const char *here, const char *name, int value)

  Insert a boolean value into a property list at the given position.
- cpl_error_code cpl_propertylist_insert_char (cpl_propertylist *self, const char *here, const char *name, char value)

  Insert a character value into a property list at the given position.
- cpl_error_code cpl_propertylist_insert_double (cpl_propertylist *self, const char *here, const char *name, double value)

  Insert a double value into a property list at the given position.
- cpl_error_code cpl_propertylist_insert_double_complex (cpl_propertylist *self, const char *here, const char *name, double complex value)

  Insert a double complex value into a property list at the given position.
- cpl_error_code cpl_propertylist_insert_float (cpl_propertylist *self, const char *here, const char *name, float value)

  Insert a float value into a property list at the given position.
- cpl_error_code cpl_propertylist_insert_float_complex (cpl_propertylist *self, const char *here, const char *name, float complex value)

  Insert a float complex value into a property list at the given position.
- cpl_error_code cpl_propertylist_insert_int (cpl_propertylist *self, const char *here, const char *name, int value)

  Insert a integer value into a property list at the given position.
- cpl_error_code cpl_propertylist_insert_long (cpl_propertylist *self, const char *here, const char *name, long value)

  Insert a long value into a property list at the given position.
- cpl_error_code cpl_propertylist_insert_long_long (cpl_propertylist *self, const char *here, const char *name, long long value)

  Insert a long long value into a property list at the given position.
- cpl_error_code cpl_propertylist_insert_property (cpl_propertylist *self, const char *here, const cpl_property *property)

Insert a property into a property list at the given position.

- cpl_error_code cpl_propertylist_insert_string (cpl_propertylist *self, const char *here, const char *name, const char *value)

  Insert a string value into a property list at the given position.
- int cpl_propertylist_is_empty (const cpl_propertylist *self)

  Check whether a property list is empty.
- cpl_propertylist * cpl_propertylist_load (const char *name, cpl_size position)

  Create a property list from a file.
- cpl_propertylist * cpl_propertylist_load_regexp (const char *name, cpl_size position, const char *regexp, int invert)

  Create a filtered property list from a file.
- cpl_propertylist * cpl_propertylist_new (void)

  Create an empty property list.
- cpl_error_code cpl_propertylist_prepend_bool (cpl_propertylist *self, const char *name, int value)

  Prepend a boolean value to a property list.
- cpl_error_code cpl_propertylist_prepend_char (cpl_propertylist *self, const char *name, char value)

  Prepend a character value to a property list.
- cpl_error_code cpl_propertylist_prepend_double (cpl_propertylist *self, const char *name, double value)

  Prepend a double value to a property list.
- cpl_error_code cpl_propertylist_prepend_double_complex (cpl_propertylist *self, const char *name, double complex value)

  Prepend a double complex value to a property list.
- cpl_error_code cpl_propertylist_prepend_float (cpl_propertylist *self, const char *name, float value)

  Prepend a float value to a property list.
- cpl_error_code cpl_propertylist_prepend_float_complex (cpl_propertylist *self, const char *name, float complex value)

  Prepend a float complex value to a property list.
- cpl_error_code cpl_propertylist_prepend_int (cpl_propertylist *self, const char *name, int value)

  Prepend a integer value to a property list.
- cpl_error_code cpl_propertylist_prepend_long (cpl_propertylist *self, const char *name, long value)

  Prepend a long value to a property list.
- cpl_error_code cpl_propertylist_prepend_long_long (cpl_propertylist *self, const char *name, long long value)

  Prepend a long long value to a property list.
- cpl_error_code cpl_propertylist_prepend_property (cpl_propertylist *self, const cpl_property *property)

  Prepend a property to a property list.
- cpl_error_code cpl_propertylist_prepend_string (cpl_propertylist *self, const char *name, const char *value)

  Prepend a string value to a property list.
- cpl_error_code cpl_propertylist_save (const cpl_propertylist *self, const char *filename, unsigned mode)

  Save a property list to a FITS file.
- cpl_error_code cpl_propertylist_set_bool (cpl_propertylist *self, const char *name, int value)

  Set the value of the given boolean property list entry.
- cpl_error_code cpl_propertylist_set_char (cpl_propertylist *self, const char *name, char value)

  Set the value of the given character property list entry.
- cpl_error_code cpl_propertylist_set_comment (cpl_propertylist *self, const char *name, const char *comment)

  Modify the comment field of the given property list entry.
- cpl_error_code cpl_propertylist_set_double (cpl_propertylist *self, const char *name, double value)

  Set the value of the given double property list entry.
- cpl_error_code cpl_propertylist_set_double_complex (cpl_propertylist *self, const char *name, double complex value)

  Set the value of the given double complex property list entry.

- cpl_error_code cpl_propertylist_set_float (cpl_propertylist *self, const char *name, float value)

  Set the value of the given float property list entry.
- cpl_error_code cpl_propertylist_set_float_complex (cpl_propertylist *self, const char *name, float complex value)

  Set the value of the given float complex property list entry.
- cpl_error_code cpl_propertylist_set_int (cpl_propertylist *self, const char *name, int value)

  Set the value of the given integer property list entry.
- cpl_error_code cpl_propertylist_set_long (cpl_propertylist *self, const char *name, long value)

  Set the value of the given long property list entry.
- cpl_error_code cpl_propertylist_set_long_long (cpl_propertylist *self, const char *name, long long value)

  Set the value of the given long long property list entry.
- cpl_error_code cpl_propertylist_set_string (cpl_propertylist *self, const char *name, const char *value)

  Set the value of the given string property list entry.
- cpl_error_code cpl_propertylist_sort (cpl_propertylist *self, cpl_propertylist_compare_func compare)

  Sort a property list.
- cpl_error_code cpl_propertylist_update_bool (cpl_propertylist *self, const char *name, int value)

  Update a property list with a boolean value.
- cpl_error_code cpl_propertylist_update_char (cpl_propertylist *self, const char *name, char value)

  Update a property list with a character value.
- cpl_error_code cpl_propertylist_update_double (cpl_propertylist *self, const char *name, double value)

  Update a property list with a double value.
- cpl_error_code cpl_propertylist_update_double_complex (cpl_propertylist *self, const char *name, double complex value)

  Update a property list with a double complex value.
- cpl_error_code cpl_propertylist_update_float (cpl_propertylist *self, const char *name, float value)

  Update a property list with a float value.
- cpl_error_code cpl_propertylist_update_float_complex (cpl_propertylist *self, const char *name, float complex value)

  Update a property list with a float complex value.
- cpl_error_code cpl_propertylist_update_int (cpl_propertylist *self, const char *name, int value)

  Update a property list with a integer value.
- cpl_error_code cpl_propertylist_update_long (cpl_propertylist *self, const char *name, long value)

  Update a property list with a long value.
- cpl_error_code cpl_propertylist_update_long_long (cpl_propertylist *self, const char *name, long long value)

  Update a property list with a long long value.
- cpl_error_code cpl_propertylist_update_string (cpl_propertylist *self, const char *name, const char *value)

  Update a property list with a string value.

4.40.1 Detailed Description

This module implements a container for *properties* (see Properties) which can be used to store auxiliary values related to another data object, an image or a table for instance. The property values can be set and retrieved by their associated name and properties can be added and removed from the list. The property list container is an ordered sequence of properties.

Synopsis:

```
#include <cpl_propertylist.h>
```

4.40.2 Typedef Documentation

4.40.2.1 cpl_propertylist

```
typedef struct _cpl_propertylist_ cpl_propertylist
```

The opaque property list data type.

4.40.2.2 cpl_propertylist_compare_func

```
typedef int(* cpl_propertylist_compare_func) (const cpl_property *first, const cpl_property
*second)
```

The property comparison function data type.

4.40.3 Function Documentation

4.40.3.1 cpl_propertylist_append()

```
cpl_error_code cpl_propertylist_append (
            cpl_propertylist * self,
            const cpl_propertylist * other )
```

Append a property list..

Parameters

self	A property list.
other	The property list to append.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function appends the property list *other* to the property list *self*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl_property_duplicate().

4.40.3.2 cpl_propertylist_append_bool()

```
cpl_error_code cpl_propertylist_append_bool (
            cpl_propertylist * self,
            const char * name,
            int value )
```

Append a boolean value to a property list.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The boolean value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.

The function creates a new boolean property with name *name* and value *value*. The property is appended to the property list *self*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_property_new(), cpl_property_set_bool(), and CPL_TYPE_BOOL.

4.40.3.3 cpl_propertylist_append_char()

```
cpl_error_code cpl_propertylist_append_char (
            cpl_propertylist * self,
            const char * name,
            char value )
```

Append a character value to a property list.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The character value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.

The function creates a new character property with name *name* and value *value*. The property is appended to the property list *self*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_property_new(), cpl_property_set_char(), and CPL_TYPE_CHAR.

4.40.3.4 cpl_propertylist_append_double()

```
cpl_error_code cpl_propertylist_append_double (
            cpl_propertylist * self,
            const char * name,
            double value )
```

Append a double value to a property list.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The double value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.

The function creates a new double property with name *name* and value *value*. The property is appended to the property list *self*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_property_new(), cpl_property_set_double(), and CPL_TYPE_DOUBLE.

4.40.3.5 cpl_propertylist_append_double_complex()

```
cpl_error_code cpl_propertylist_append_double_complex (
            cpl_propertylist * self,
            const char * name,
            double complex value )
```

Append a double complex value to a property list.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The double complex value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.

The function creates a new double complex property with name *name* and value *value*. The property is appended to the property list *self*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_property_new(), cpl_property_set_double_↩complex(), and CPL_TYPE_DOUBLE_COMPLEX.

4.40.3.6 cpl_propertylist_append_float()

```
cpl_error_code cpl_propertylist_append_float (
            cpl_propertylist * self,
            const char * name,
            float value )
```

Append a float value to a property list.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The float value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.

The function creates a new float property with name *name* and value *value*. The property is appended to the property list *self*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_property_new(), cpl_property_set_float(), and CPL_TYPE_FLOAT.

4.40.3.7 cpl_propertylist_append_float_complex()

```
cpl_error_code cpl_propertylist_append_float_complex (
            cpl_propertylist * self,
            const char * name,
            float complex value )
```

Append a float complex value to a property list.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The float complex value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.

The function creates a new float complex property with name *name* and value *value*. The property is appended to the property list *self*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_property_new(), cpl_property_set_float_↩ complex(), and CPL_TYPE_FLOAT_COMPLEX.

4.40.3.8 cpl_propertylist_append_int()

```
cpl_error_code cpl_propertylist_append_int (
            cpl_propertylist * self,
            const char * name,
            int value )
```

Append an integer value to a property list.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The integer value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.

The function creates a new integer property with name *name* and value *value*. The property is appended to the property list *self*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_property_new(), cpl_property_set_int(), and CPL_TYPE_INT.

4.40.3.9 cpl_propertylist_append_long()

```
cpl_error_code cpl_propertylist_append_long (
            cpl_propertylist * self,
            const char * name,
            long value )
```

Append a long value to a property list.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The long value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.

The function creates a new long property with name *name* and value *value*. The property is appended to the property list *self*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_property_new(), cpl_property_set_long(), and CPL_TYPE_LONG.

4.40.3.10 cpl_propertylist_append_long_long()

```
cpl_error_code cpl_propertylist_append_long_long (
            cpl_propertylist * self,
            const char * name,
            long long value )
```

Append a long long value to a property list.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The long long value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.

The function creates a new long long property with name *name* and value *value*. The property is appended to the property list *self*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_property_new(), cpl_property_set_long_↩ long(), and CPL_TYPE_LONG_LONG.

4.40.3.11 cpl_propertylist_append_property()

```
cpl_error_code cpl_propertylist_append_property (
            cpl_propertylist * self,
            const cpl_property * property )
```

Append a duplicate of a property to a property list.

Parameters

self	Property list to append to
property	The property to duplicate and append

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *property* is a NULL pointer.

This function creates a new property and appends it to the end of a property list. It will not check if the property already exists.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl_property_duplicate().

4.40.3.12 cpl_propertylist_append_string()

```
cpl_error_code cpl_propertylist_append_string (
            cpl_propertylist * self,
            const char * name,
            const char * value )
```

Append a string value to a property list.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The string value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *name* or *value* is a `NULL` pointer.

The function creates a new string property with name *name* and value *value*. The property is appended to the property list *self*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_property_new(), cpl_property_set_string(), and CPL_TYPE_STRING.

4.40.3.13 cpl_propertylist_copy_property()

```
cpl_error_code cpl_propertylist_copy_property (
            cpl_propertylist * self,
            const cpl_propertylist * other,
            const char * name )
```

Copy a property from another property list.

Parameters

self	A property list.
other	The property list from which a property is copied.
name	The name of the property to copy.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *other* or *name* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *other* does not contain a property with the name *name*.
CPL_ERROR_TYPE_MISMATCH	The property list *self* contains a property with the name *name* which is not of the same type as the property which should be copied from *other*.

The function copies the property *name* from the property list *other* to the property list *self*. If the property list *self* does not already contain a property *name* the property is appended to *self*. If a property *name* exists already in *self*

the function overwrites the contents of this property if and only if this property is of the same type as the property to be copied from *other*.

References CPL_ERROR_NULL_INPUT.

4.40.3.14 cpl_propertylist_copy_property_regexp()

```
cpl_error_code cpl_propertylist_copy_property_regexp (
            cpl_propertylist * self,
            const cpl_propertylist * other,
            const char * regexp,
            int invert )
```

Copy matching properties from another property list.

Parameters

self	A property list.
other	The property list from which a property is copied.
regexp	The regular expression used to select properties.
invert	Flag inverting the sense of matching.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *other* or *regexp* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	The parameter *regexp* is an invalid regular expression, including the empty stringr.
CPL_ERROR_TYPE_MISMATCH	The property list *self* contains a property with the name *name* which is not of the same type as the property which should be copied from *other*.

The function copies all properties with matching names from the property list *other* to the property list *self*. If the flag *invert* is zero, all properties whose names match the regular expression *regexp* are copied. If *invert* is set to a non-zero value, all properties with names not matching *regexp* are copied rather. The function expects POSIX 1003.2 compliant extended regular expressions.

If the property list *self* does not already contain one of the properties to be copied this property is appended to *self*. If a property to be copied exists already in *self* the function overwrites the contents of this property.

Before properties are copied from the property list *other* to *self* the types of the properties are checked and if any type mismatch is detected the function stops processing immediately. The property list *self* is not at all modified in this case.

See also

cpl_propertylist_copy_property(), cpl_propertylist_append()

References CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_wcs_new_from_propertylist().

4.40.3.15 cpl_propertylist_delete()

```
void cpl_propertylist_delete (
            cpl_propertylist * self )
```

Destroy a property list.

Parameters

self	The property list to .

Returns

Nothing.

The function destroys the property list *self* and its whole contents. If *self* is NULL, nothing is done and no error is set.

References cpl_property_delete().

Referenced by cpl_wcs_new_from_propertylist().

4.40.3.16 cpl_propertylist_dump()

```
void cpl_propertylist_dump (
            const cpl_propertylist * self,
            FILE * stream )
```

Print a property list.

Parameters

self	Pointer to property list
stream	The output stream

Returns

Nothing.

This function is mainly intended for debug purposes. If the specified stream is NULL, it is set to *stdout*. The function used for printing is the standard C fprintf().

References cpl_property_get_comment(), cpl_property_get_name(), cpl_property_get_size(), cpl_propertylist_get_const(), and cpl_propertylist_get_size().

4.40.3.17 cpl_propertylist_duplicate()

```
cpl_propertylist* cpl_propertylist_duplicate (
            const cpl_propertylist * self )
```

Create a copy of the given property list.

Parameters

self	The property list to be copied.

Returns

The created copy or NULL in case an error occurred.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function creates a deep copy of the given property list *self*, i.e the created copy and the original property list do not share any resources.

References CPL_ERROR_NULL_INPUT, cpl_property_duplicate(), and cpl_propertylist_new().

4.40.3.18 cpl_propertylist_empty()

```
void cpl_propertylist_empty (
            cpl_propertylist * self )
```

Remove all properties from a property list.

Parameters

self	A property list.

Returns

Nothing.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function removes all properties from *self*. Each property is properly deallocated. After calling this function *self* is empty.

References CPL_ERROR_NULL_INPUT, and cpl_property_delete().

4.40.3.19 cpl_propertylist_erase()

```
int cpl_propertylist_erase (
            cpl_propertylist * self,
            const char * name )
```

Erase the given property from a property list.

Parameters

self	A property list.
name	Name of the property to erase.

Returns

On success the function returns the number of erased entries. If an error occurs the function returns 0 and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.

The function searches the property with the name *name* in the property list *self* and removes it. The property is destroyed. If *self* contains multiple duplicates of a property named *name*, only the first one is erased.

References CPL_ERROR_NULL_INPUT.

4.40.3.20 cpl_propertylist_erase_regexp()

```
int cpl_propertylist_erase_regexp (
            cpl_propertylist * self,
            const char * regexp,
            int invert )
```

Erase all properties with name matching a given regular expression.

Parameters

self	A property list.
regexp	Regular expression.
invert	Flag inverting the sense of matching.

Returns

On success the function returns the number of erased entries or 0 if no entries are erased. If an error occurs the function returns -1 and an appropriate error code is set. In CPL versions earlier than 5.0, the return value in case of error is 0.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *regexp* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	The parameter *regexp* is the empty string.

The function searches for all the properties matching in the list *self* and removes them. Whether a property matches or not depends on the given regular expression *regexp*, and the flag *invert*. If *invert* is `0`, all properties matching *regexp* are removed from the list. If *invert* is set to a non-zero value, all properties which do not match *regexp* are erased. The removed properties are destroyed.

The function expects POSIX 1003.2 compliant extended regular expressions.

References CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_INPUT.

4.40.3.21 cpl_propertylist_get()

```
cpl_property* cpl_propertylist_get (
            cpl_propertylist * self,
            long position )
```

Access property list elements by index.

Parameters

self	The property list to query.
position	Index of the element to retrieve.

Returns

The function returns the property with index *position*, or NULL if *position* is out of range.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function returns a handle for the property list element, the property, with the index *position*. Numbering of property list elements extends from 0 to **cpl_propertylist_get_size()** - 1. If *position* is less than 0 or greater equal than **cpl_propertylist_get_size()** the function returns NULL.

References cpl_errorstate_get(), cpl_errorstate_is_equal(), and cpl_propertylist_get_const().

4.40.3.22 cpl_propertylist_get_bool()

```
int cpl_propertylist_get_bool (
            const cpl_propertylist * self,
            const char * name )
```

Get the boolean value of the given property list entry.

Parameters

self	A property list.
name	The property name to look up.

Returns

The integer representation of the boolean value stored in the list entry. TRUE is represented as non-zero value while 0 indicates FALSE. The function returns 0 if an error occurs and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.
CPL_ERROR_TYPE_MISMATCH	The sought-after property *name* is not of type CPL_TYPE_BOOL.

The function searches the property list *self* for a property named *name*. If it is present in the list, its boolean value is returned. If there is more than one property with the same *name*, it takes the first one from the list.

References CPL_ERROR_NULL_INPUT.

4.40.3.23 cpl_propertylist_get_char()

```
char cpl_propertylist_get_char (
             const cpl_propertylist * self,
             const char * name )
```

Get the character value of the given property list entry.

Parameters

self	A property list.
name	The property name to look up.

Returns

The character value stored in the list entry. The function returns '\0' if an error occurs and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.
CPL_ERROR_TYPE_MISMATCH	The sought-after property *name* is not of type `CPL_TYPE_CHAR`.

The function searches the property list *self* for a property named *name*. If it is present in the list, its character value is returned. If there is more than one property with the same *name*, it takes the first one from the list.

References CPL_ERROR_NULL_INPUT.

4.40.3.24 cpl_propertylist_get_comment()

```
const char* cpl_propertylist_get_comment (
             const cpl_propertylist * self,
             const char * name )
```

Get the comment of the given property list entry.

Parameters

self	A property list.
name	The property name to look up.

Returns

The comment of the property list entry, or NULL.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.

The function searches the property list *self* for a property named *name*. If it is present in the list, its comment string is returned. If an entry with the name *name* is not found, or if the entry has no comment the function returns NULL. If there is more than one property with the same *name*, it takes the first one from the list.

References CPL_ERROR_NULL_INPUT.

4.40.3.25 cpl_propertylist_get_const()

```
const cpl_property* cpl_propertylist_get_const (
            const cpl_propertylist * self,
            long position )
```

Access property list elements by index.

Parameters

self	The property list to query.
position	Index of the element to retrieve.

Returns

The function returns the property with index *position*, or NULL if *position* is out of range.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a NULL pointer.

The function returns a handle for the property list element, the property, with the index *position*. Numbering of property list elements extends from 0 to **cpl_propertylist_get_size()** - 1. If *position* is less than 0 or greater equal than **cpl_propertylist_get_size()** the function returns NULL.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_propertylist_dump(), cpl_propertylist_get(), and cpl_wcs_new_from_propertylist().

4.40.3.26 cpl_propertylist_get_double()

```
double cpl_propertylist_get_double (
            const cpl_propertylist * self,
            const char * name )
```

Get the double value of the given property list entry.

Parameters

self	A property list.
name	The property name to look up.

Returns

The double value stored in the list entry. The function returns 0 if an error occurs and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.
CPL_ERROR_TYPE_MISMATCH	The sought-after property *name* is not of type `CPL_TYPE_DOUBLE` or `CPL_TYPE_FLOAT`.

The function searches the property list *self* for a property named *name*. If it is present in the list, its double value is returned. If there is more than one property with the same *name*, it takes the first one from the list. If the value is of type float, the function casts it to double before returning it.

The function may be used to access the value of all floating-point type properties whose floating-point value has a rank less or equal to the functions return type. If the value of a compatible property is retrieved, it is promoted to the return type of the function.

References CPL_ERROR_NULL_INPUT.

4.40.3.27 cpl_propertylist_get_double_complex()

```
double complex cpl_propertylist_get_double_complex (
            const cpl_propertylist * self,
            const char * name )
```

Get the double complex value of the given property list entry.

Parameters

self	A property list.
name	The property name to look up.

Returns

The double complex value stored in the list entry. The function returns 0 if an error occurs and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.
CPL_ERROR_TYPE_MISMATCH	The sought-after property *name* is not of type `CPL_TYPE_DOUBLE_COMPLEX` or `CPL_TYPE_FLOAT_COMPLEX`.

The function searches the property list *self* for a property named *name*. If it is present in the list, its double complex value is returned. If there is more than one property with the same *name*, it takes the first one from the list. If the value is of type float, the function casts it to double complex before returning it.

The function may be used to access the value of all complex type properties whose complex value has a rank less or equal to the functions return type. If the value of a compatible property is retrieved, it is promoted to the return type of the function.

References CPL_ERROR_NULL_INPUT.

4.40.3.28 cpl_propertylist_get_float()

```
float cpl_propertylist_get_float (
            const cpl_propertylist * self,
            const char * name )
```

Get the float value of the given property list entry.

Parameters

self	A property list.
name	The property name to look up.

Returns

The float value stored in the list entry. The function returns 0 if an error occurs and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.
CPL_ERROR_TYPE_MISMATCH	The sought-after property *name* is not of type `CPL_TYPE_FLOAT` or `CPL_TYPE_DOUBLE`.

The function searches the property list *self* for a property named *name*. If it is present in the list, its float value is returned. If there is more than one property with the same *name*, it takes the first one from the list. If the value is of type double, the function casts it to float before returning it.

References CPL_ERROR_NULL_INPUT.

4.40.3.29 cpl_propertylist_get_float_complex()

```
float complex cpl_propertylist_get_float_complex (
            const cpl_propertylist * self,
            const char * name )
```

Get the float complex value of the given property list entry.

Parameters

self	A property list.
name	The property name to look up.

Returns

The float complex value stored in the list entry. The function returns 0 if an error occurs and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.
CPL_ERROR_TYPE_MISMATCH	The sought-after property *name* is not of type `CPL_TYPE_FL↩OAT_COMPLEX` or `CPL_TYPE_DOUBLE_COMPLEX`.

The function searches the property list *self* for a property named *name*. If it is present in the list, its float complex value is returned. If there is more than one property with the same *name*, it takes the first one from the list. If the value is of type double, the function casts it to float complex before returning it.

References CPL_ERROR_NULL_INPUT.

4.40.3.30 cpl_propertylist_get_int()

```
int cpl_propertylist_get_int (
            const cpl_propertylist * self,
            const char * name )
```

Get the integer value of the given property list entry.

Parameters

self	A property list.
name	The property name to look up.

Returns

The integer value stored in the list entry. The function returns 0 if an error occurs and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.
CPL_ERROR_TYPE_MISMATCH	The sought-after property *name* is not of type `CPL_TYPE_INT`.

The function searches the property list *self* for a property named *name*. If it is present in the list, its integer value is returned. If there is more than one property with the same *name*, it takes the first one from the list.

References CPL_ERROR_NULL_INPUT.

4.40.3.31 cpl_propertylist_get_long()

```
long cpl_propertylist_get_long (
            const cpl_propertylist * self,
            const char * name )
```

Get the long value of the given property list entry.

Parameters

self	A property list.
name	The property name to look up.

Returns

The long value stored in the list entry. The function returns 0 if an error occurs and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.
CPL_ERROR_TYPE_MISMATCH	The sought-after property *name* is not of type `CPL_TYPE_LONG` or `CPL_TYPE_INT`.

The function searches the property list *self* for a property named *name*. If it is present in the list, its long value is returned. If there is more than one property with the same *name*, it takes the first one from the list.

The function may be used to access the value of all integer type properties whose integer value has a rank less or equal to the functions return type. If the value of a compatible property is retrieved, it is promoted to the return type of the function.

References CPL_ERROR_NULL_INPUT.

4.40.3.32 cpl_propertylist_get_long_long()

```
long long cpl_propertylist_get_long_long (
            const cpl_propertylist * self,
            const char * name )
```

Get the long long value of the given property list entry.

Parameters

self	A property list.
name	The property name to look up.

Returns

The long value stored in the list entry. The function returns 0 if an error occurs and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.
CPL_ERROR_TYPE_MISMATCH	The sought-after property *name* is not of type `CPL_TYPE_LONG_LONG`, `CPL_TYPE_LONG`, or `CPL_TYPE_INT`

The function searches the property list *self* for a property named *name*. If it is present in the list, its long long value is returned. If there is more than one property with the same *name*, it takes the first one from the list.

The function may be used to access the value of all integer type properties whose integer value has a rank less or equal to the functions return type. If the value of a compatible property is retrieved, it is promoted to the return type of the function.

References CPL_ERROR_NULL_INPUT.

4.40.3.33 cpl_propertylist_get_property()

```
cpl_property* cpl_propertylist_get_property (
            cpl_propertylist * self,
            const char * name )
```

Access property list elements by property name.

Parameters

self	The property list to query.
name	The name of the property to retrieve.

Returns

The function returns the property with name *name*, or NULL if it does not exist.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or the *name* is a NULL pointer.

The function returns a handle to the property list element, the property, with the name *name*. If more than one property exist with the same *name*, then the first one found will be returned.

References CPL_ERROR_NULL_INPUT.

4.40.3.34 cpl_propertylist_get_property_const()

```
const cpl_property* cpl_propertylist_get_property_const (
            const cpl_propertylist * self,
            const char * name )
```

Access property list elements by property name.

Parameters

self	The property list to query.
name	The name of the property to retrieve.

Returns

The function returns the property with name *name*, or NULL if it does not exist.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or the *name* is a `NULL` pointer.

The function returns a handle to the property list element, the property, with the name *name*. If more than one property exist with the same *name*, then the first one found will be returned.

References CPL_ERROR_NULL_INPUT.

4.40.3.35 cpl_propertylist_get_size()

```
cpl_size cpl_propertylist_get_size (
            const cpl_propertylist * self )
```

Get the current size of a property list.

Parameters

self	A property list.

Returns

The property list's current size, or 0 if the list is empty. If an error occurs the function returns 0 and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function reports the current number of elements stored in the property list *self*.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_propertylist_dump(), cpl_table_sort(), and cpl_wcs_new_from_propertylist().

4.40.3.36 cpl_propertylist_get_string()

```
const char* cpl_propertylist_get_string (
            const cpl_propertylist * self,
            const char * name )
```

Get the string value of the given property list entry.

Parameters

self	A property list.
name	The property name to look up.

Returns

A handle to the string value stored in the list entry. The function returns `NULL` if an error occurs and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.
CPL_ERROR_TYPE_MISMATCH	The sought-after property *name* is not of type `CPL_TYPE_ST↩RING`.

The function searches the property list *self* for a property named *name*. If it is present in the list, a handle to its string value is returned. If there is more than one property with the same *name*, it takes the first one from the list.

References CPL_ERROR_NULL_INPUT.

4.40.3.37 cpl_propertylist_get_type()

```
cpl_type cpl_propertylist_get_type (
            const cpl_propertylist * self,
            const char * name )
```

Get the the type of a property list entry.

Parameters

self	A property list.
name	The property name to look up.

Returns

The type of the stored value. If an error occurs the function returns `CPL_TYPE_INVALID` and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.

The function returns the type of the value stored in *self* with the name *name*. If there is more than one property with the same *name*, it takes the first one from the list.

References CPL_ERROR_NULL_INPUT, and CPL_TYPE_INVALID.

4.40.3.38 cpl_propertylist_has()

```
int cpl_propertylist_has (
            const cpl_propertylist * self,
            const char * name )
```

Check whether a property is present in a property list.

Parameters

self	A property list.
name	The property name to look up.

Returns

The function returns 1 if the property is present, or 0 otherwise. If an error occurs the function returns 0 and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.

The function searches the property list *self* for a property with the name *name* and reports whether it was found or not.

References CPL_ERROR_NULL_INPUT.

4.40.3.39 cpl_propertylist_insert_after_bool()

```
cpl_error_code cpl_propertylist_insert_after_bool (
            cpl_propertylist * self,
            const char * after,
            const char * name,
            int value )
```

Insert a boolean value into a property list after the given position.

Parameters

self	A property list.
after	Name of the property after which the value is inserted.
name	The property name to be assigned to the value.
value	The boolean value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *after* or *name* is a `NULL` pointer.
CPL_ERROR_UNSPECIFIED	A property with the name *name* could not be inserted into *self*.

The function creates a new boolean property with name *name* and value *value*. The property is inserted into the property list *self* after the property named *after*.

References CPL_ERROR_NULL_INPUT.

4.40.3.40 cpl_propertylist_insert_after_char()

```
cpl_error_code cpl_propertylist_insert_after_char (
            cpl_propertylist * self,
            const char * after,
            const char * name,
            char value )
```

Insert a character value into a property list after the given position.

Parameters

self	A property list.
after	Name of the property after which the value is inserted.
name	The property name to be assigned to the value.
value	The character value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *after* or *name* is a `NULL` pointer.
CPL_ERROR_UNSPECIFIED	A property with the name *name* could not be inserted into *self*.

The function creates a new character property with name *name* and value *value*. The property is inserted into the property list *self* after the property named *after*.

References CPL_ERROR_NULL_INPUT.

4.40.3.41 cpl_propertylist_insert_after_double()

```
cpl_error_code cpl_propertylist_insert_after_double (
            cpl_propertylist * self,
            const char * after,
            const char * name,
            double value )
```

Insert a double value into a property list after the given position.

Parameters

self	A property list.
after	Name of the property after which the value is inserted.
name	The property name to be assigned to the value.
value	The double value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *after* or *name* is a `NULL` pointer.
CPL_ERROR_UNSPECIFIED	A property with the name *name* could not be inserted into *self*.

The function creates a new double property with name *name* and value *value*. The property is inserted into the property list *self* after the property named *after*.

References CPL_ERROR_NULL_INPUT.

4.40.3.42 cpl_propertylist_insert_after_double_complex()

```
cpl_error_code cpl_propertylist_insert_after_double_complex (
            cpl_propertylist * self,
            const char * after,
```

```
const char * name,
double complex value )
```

Insert a double complex value into a property list after the given position.

Parameters

self	A property list.
after	Name of the property after which the value is inserted.
name	The property name to be assigned to the value.
value	The double complex value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *after* or *name* is a `NULL` pointer.
CPL_ERROR_UNSPECIFIED	A property with the name *name* could not be inserted into *self*.

The function creates a new double complex property with name *name* and value *value*. The property is inserted into the property list *self* after the property named *after*.

References CPL_ERROR_NULL_INPUT.

4.40.3.43 cpl_propertylist_insert_after_float()

```
cpl_error_code cpl_propertylist_insert_after_float (
            cpl_propertylist * self,
            const char * after,
            const char * name,
            float value )
```

Insert a float value into a property list after the given position.

Parameters

self	A property list.
after	Name of the property after which the value is inserted.
name	The property name to be assigned to the value.
value	The float value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *after* or *name* is a `NULL` pointer.
CPL_ERROR_UNSPECIFIED	A property with the name *name* could not be inserted into *self*.

The function creates a new float property with name *name* and value *value*. The property is inserted into the property list *self* after the property named *after*.

References CPL_ERROR_NULL_INPUT.

4.40.3.44 cpl_propertylist_insert_after_float_complex()

```
cpl_error_code cpl_propertylist_insert_after_float_complex (
            cpl_propertylist * self,
            const char * after,
            const char * name,
            float complex value )
```

Insert a float complex value into a property list after the given position.

Parameters

self	A property list.
after	Name of the property after which the value is inserted.
name	The property name to be assigned to the value.
value	The float complex value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *after* or *name* is a `NULL` pointer.
CPL_ERROR_UNSPECIFIED	A property with the name *name* could not be inserted into *self*.

The function creates a new float complex property with name *name* and value *value*. The property is inserted into the property list *self* after the property named *after*.

References CPL_ERROR_NULL_INPUT.

4.40.3.45 cpl_propertylist_insert_after_int()

```
cpl_error_code cpl_propertylist_insert_after_int (
            cpl_propertylist * self,
            const char * after,
```

```
            const char * name,
            int value )
```

Insert a integer value into a property list after the given position.

Parameters

self	A property list.
after	Name of the property after which the value is inserted.
name	The property name to be assigned to the value.
value	The integer value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *after* or *name* is a `NULL` pointer.
CPL_ERROR_UNSPECIFIED	A property with the name *name* could not be inserted into *self*.

The function creates a new integer property with name *name* and value *value*. The property is inserted into the property list *self* after the property named *after*.

References CPL_ERROR_NULL_INPUT.

4.40.3.46 cpl_propertylist_insert_after_long()

```
cpl_error_code cpl_propertylist_insert_after_long (
            cpl_propertylist * self,
            const char * after,
            const char * name,
            long value )
```

Insert a long value into a property list after the given position.

Parameters

self	A property list.
after	Name of the property after which the value is inserted.
name	The property name to be assigned to the value.
value	The long value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *after* or *name* is a `NULL` pointer.
CPL_ERROR_UNSPECIFIED	A property with the name *name* could not be inserted into *self*.

The function creates a new long property with name *name* and value *value*. The property is inserted into the property list *self* after the property named *after*.

References CPL_ERROR_NULL_INPUT.

4.40.3.47 cpl_propertylist_insert_after_long_long()

```
cpl_error_code cpl_propertylist_insert_after_long_long (
            cpl_propertylist * self,
            const char * after,
            const char * name,
            long long value )
```

Insert a long long value into a property list after the given position.

Parameters

self	A property list.
after	Name of the property after which the value is inserted.
name	The property name to be assigned to the value.
value	The long long value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *after* or *name* is a `NULL` pointer.
CPL_ERROR_UNSPECIFIED	A property with the name *name* could not be inserted into *self*.

The function creates a new long long property with name *name* and value *value*. The property is inserted into the property list *self* after the property named *after*.

References CPL_ERROR_NULL_INPUT.

4.40.3.48 cpl_propertylist_insert_after_property()

```
cpl_error_code cpl_propertylist_insert_after_property (
            cpl_propertylist * self,
            const char * after,
            const cpl_property * property )
```

Insert a property into a property list after the given position.

Parameters

self	Property list
after	Name of the property after which to insert the property
property	The property to insert

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *property* or *after* is a `NULL` pointer.

The function creates a new property and inserts it into the property list *self* after the position of the property named *after*.

References CPL_ERROR_NULL_INPUT.

4.40.3.49 cpl_propertylist_insert_after_string()

```
cpl_error_code cpl_propertylist_insert_after_string (
            cpl_propertylist * self,
            const char * after,
            const char * name,
            const char * value )
```

Insert a string value into a property list after the given position.

Parameters

self	A property list.
after	Name of the property after which the value is inserted.
name	The property name to be assigned to the value.
value	The string value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *after*, *name* or *value* or *name* is a `NULL` pointer.
CPL_ERROR_UNSPECIFIED	A property with the name *name* could not be inserted into *self*.

The function creates a new string property with name *name* and value *value*. The property is inserted into the property list *self* after the property named *after*.

References CPL_ERROR_NULL_INPUT.

4.40.3.50 cpl_propertylist_insert_bool()

```
cpl_error_code cpl_propertylist_insert_bool (
            cpl_propertylist * self,
            const char * here,
            const char * name,
            int value )
```

Insert a boolean value into a property list at the given position.

Parameters

self	A property list.
here	Name indicating the position at which the value is inserted.
name	The property name to be assigned to the value.
value	The boolean value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *here* or *name* is a `NULL` pointer.
CPL_ERROR_UNSPECIFIED	A property with the name *name* could not be inserted into *self*.

The function creates a new boolean property with name *name* and value *value*. The property is inserted into the property list *self* at the position of the property named *here*.

References CPL_ERROR_NULL_INPUT.

4.40.3.51 cpl_propertylist_insert_char()

```
cpl_error_code cpl_propertylist_insert_char (
            cpl_propertylist * self,
            const char * here,
```

```
            const char * name,
            char value )
```

Insert a character value into a property list at the given position.

Parameters

self	A property list.
here	Name indicating the position at which the value is inserted.
name	The property name to be assigned to the value.
value	The character value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *here* or *name* is a `NULL` pointer.
CPL_ERROR_UNSPECIFIED	A property with the name *name* could not be inserted into *self*.

The function creates a new character property with name *name* and value *value*. The property is inserted into the property list *self* at the position of the property named *here*.

References CPL_ERROR_NULL_INPUT.

4.40.3.52 cpl_propertylist_insert_double()

```
cpl_error_code cpl_propertylist_insert_double (
            cpl_propertylist * self,
            const char * here,
            const char * name,
            double value )
```

Insert a double value into a property list at the given position.

Parameters

self	A property list.
here	Name indicating the position at which the value is inserted.
name	The property name to be assigned to the value.
value	The double value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *here* or *name* is a `NULL` pointer.
CPL_ERROR_UNSPECIFIED	A property with the name *name* could not be inserted into *self*.

The function creates a new double property with name *name* and value *value*. The property is inserted into the property list *self* at the position of the property named *here*.

References CPL_ERROR_NULL_INPUT.

4.40.3.53 cpl_propertylist_insert_double_complex()

```
cpl_error_code cpl_propertylist_insert_double_complex (
            cpl_propertylist * self,
            const char * here,
            const char * name,
            double complex value )
```

Insert a double complex value into a property list at the given position.

Parameters

self	A property list.
here	Name indicating the position at which the value is inserted.
name	The property name to be assigned to the value.
value	The double complex value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *here* or *name* is a `NULL` pointer.
CPL_ERROR_UNSPECIFIED	A property with the name *name* could not be inserted into *self*.

The function creates a new double complex property with name *name* and value *value*. The property is inserted into the property list *self* at the position of the property named *here*.

References CPL_ERROR_NULL_INPUT.

4.40.3.54 cpl_propertylist_insert_float()

```
cpl_error_code cpl_propertylist_insert_float (
            cpl_propertylist * self,
            const char * here,
```

```
            const char * name,
            float value )
```

Insert a float value into a property list at the given position.

Parameters

self	A property list.
here	Name indicating the position at which the value is inserted.
name	The property name to be assigned to the value.
value	The float value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *here* or *name* is a `NULL` pointer.
CPL_ERROR_UNSPECIFIED	A property with the name *name* could not be inserted into *self*.

The function creates a new float property with name *name* and value *value*. The property is inserted into the property list *self* at the position of the property named *here*.

References CPL_ERROR_NULL_INPUT.

4.40.3.55 cpl_propertylist_insert_float_complex()

```
cpl_error_code cpl_propertylist_insert_float_complex (
            cpl_propertylist * self,
            const char * here,
            const char * name,
            float complex value )
```

Insert a float complex value into a property list at the given position.

Parameters

self	A property list.
here	Name indicating the position at which the value is inserted.
name	The property name to be assigned to the value.
value	The float complex value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *here* or *name* is a `NULL` pointer.
CPL_ERROR_UNSPECIFIED	A property with the name *name* could not be inserted into *self*.

The function creates a new float complex property with name *name* and value *value*. The property is inserted into the property list *self* at the position of the property named *here*.

References CPL_ERROR_NULL_INPUT.

4.40.3.56 cpl_propertylist_insert_int()

```
cpl_error_code cpl_propertylist_insert_int (
            cpl_propertylist * self,
            const char * here,
            const char * name,
            int value )
```

Insert a integer value into a property list at the given position.

Parameters

self	A property list.
here	Name indicating the position at which the value is inserted.
name	The property name to be assigned to the value.
value	The integer value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *here* or *name* is a `NULL` pointer.
CPL_ERROR_UNSPECIFIED	A property with the name *name* could not be inserted into *self*.

The function creates a new integer property with name *name* and value *value*. The property is inserted into the property list *self* at the position of the property named *here*.

References CPL_ERROR_NULL_INPUT.

4.40.3.57 cpl_propertylist_insert_long()

```
cpl_error_code cpl_propertylist_insert_long (
            cpl_propertylist * self,
            const char * here,
```

```
            const char * name,
            long value )
```

Insert a long value into a property list at the given position.

Parameters

self	A property list.
here	Name indicating the position at which the value is inserted.
name	The property name to be assigned to the value.
value	The long value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *here* or *name* is a `NULL` pointer.
CPL_ERROR_UNSPECIFIED	A property with the name *name* could not be inserted into *self*.

The function creates a new long property with name *name* and value *value*. The property is inserted into the property list *self* at the position of the property named *here*.

References CPL_ERROR_NULL_INPUT.

4.40.3.58 cpl_propertylist_insert_long_long()

```
cpl_error_code cpl_propertylist_insert_long_long (
            cpl_propertylist * self,
            const char * here,
            const char * name,
            long long value )
```

Insert a long long value into a property list at the given position.

Parameters

self	A property list.
here	Name indicating the position at which the value is inserted.
name	The property name to be assigned to the value.
value	The long long value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *here* or *name* is a `NULL` pointer.
CPL_ERROR_UNSPECIFIED	A property with the name *name* could not be inserted into *self*.

The function creates a new long long property with name *name* and value *value*. The property is inserted into the property list *self* at the position of the property named *here*.

References CPL_ERROR_NULL_INPUT.

4.40.3.59 cpl_propertylist_insert_property()

```
cpl_error_code cpl_propertylist_insert_property (
            cpl_propertylist * self,
            const char * here,
            const cpl_property * property )
```

Insert a property into a property list at the given position.

Parameters

self	Property list
here	Name indicating the position where to insert the property
property	The property to insert

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *property* or *here* is a `NULL` pointer.

The function creates a new property and inserts it into the property list *self* at the position of the property named *here*.

References CPL_ERROR_NULL_INPUT.

4.40.3.60 cpl_propertylist_insert_string()

```
cpl_error_code cpl_propertylist_insert_string (
            cpl_propertylist * self,
            const char * here,
            const char * name,
            const char * value )
```

Insert a string value into a property list at the given position.

Parameters

self	A property list.
here	Name indicating the position at which the value is inserted.
name	The property name to be assigned to the value.
value	The string value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *here*, *name* or *value* or *name* is a `NULL` pointer.
CPL_ERROR_UNSPECIFIED	A property with the name *name* could not be inserted into *self*.

The function creates a new string property with name *name* and value *value*. The property is inserted into the property list *self* at the position of the property named *here*.

References CPL_ERROR_NULL_INPUT.

4.40.3.61 cpl_propertylist_is_empty()

```
int cpl_propertylist_is_empty (
            const cpl_propertylist * self )
```

Check whether a property list is empty.

Parameters

self	A property list.

Returns

The function returns 1 if the list is empty, and 0 otherwise. In case an error occurs the function returns -1 and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function checks if *self* contains any properties.

References CPL_ERROR_NULL_INPUT.

4.40.3.62 cpl_propertylist_load()

```
cpl_propertylist* cpl_propertylist_load (
            const char * name,
            cpl_size position )
```

Create a property list from a file.

Parameters

name	Name of the input file.
position	Index of the data set to read.

Returns

The function returns the newly created property list or `NULL` if an error occurred.

Errors

CPL_ERROR_NULL_INPUT	The parameter *name* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	The *position* is less than 0 or the properties cannot be read from the file *name*.
CPL_ERROR_FILE_IO	The file *name* does not exist.
CPL_ERROR_BAD_FILE_FORMAT	The file *name* is not a valid FITS file.
CPL_ERROR_DATA_NOT_FOUND	The requested data set at index *position* does not exist.

The function reads the properties of the data set with index *position* from the file *name*.

Currently only the FITS file format is supported. The property list is created by reading the FITS keywords from extension *position*. The numbering of the data sections starts from 0. When creating the property list from a FITS header, any keyword without a value such as undefined keywords, are not transformed into a property. In the case of float or double (complex) keywords, there is no way to identify the type returned by CFITSIO, therefore this function will always load them as double (complex).

See also

cpl_propertylist_load_regexp()

References CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_INPUT.

4.40.3.63 cpl_propertylist_load_regexp()

```
cpl_propertylist* cpl_propertylist_load_regexp (
            const char * name,
            cpl_size position,
            const char * regexp,
            int invert )
```

Create a filtered property list from a file.

Parameters

name	Name of the input file.
position	Index of the data set to read.
regexp	Regular expression used to filter properties.
invert	Flag inverting the sense of matching property names.

Returns

The function returns the newly created property list or NULL if an error occurred.

Errors

CPL_ERROR_NULL_INPUT	The parameter *name* or the parameter *regexp* is a NULL pointer.
CPL_ERROR_ILLEGAL_INPUT	The *position* is less than 0, the properties cannot be read from the file *name*, or *regexp* is not a valid extended regular expression, including the empty string..
CPL_ERROR_FILE_IO	The file *name* does not exist.
CPL_ERROR_BAD_FILE_FORMAT	The file *name* is not a valid FITS file.
CPL_ERROR_DATA_NOT_FOUND	The requested data set at index *position* does not exist.

The function reads all properties of the data set with index *position* with matching names from the file *name*. If the flag *invert* is zero, all properties whose names match the regular expression *regexp* are read. If *invert* is set to a non-zero value, all properties with names not matching *regexp* are read rather. The function expects POSIX 1003.2 compliant extended regular expressions.

Currently only the FITS file format is supported. The property list is created by reading the FITS keywords from extension *position*. The numbering of the data sections starts from 0.

When creating the property list from a FITS header, any keyword without a value such as undefined keywords, are not transformed into a property. In the case of float or double (complex) keywords, there is no way to identify the type returned by CFITSIO, therefore this function will always load them as double (complex).

FITS format specific keyword prefixes (e.g. HIERARCH) must not be part of the given pattern string *regexp*, but only the actual FITS keyword name may be given.

See also

cpl_propertylist_load()

References CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_INPUT.

4.40.3.64 cpl_propertylist_new()

```
cpl_propertylist* cpl_propertylist_new (
            void  )
```

Create an empty property list.

Returns

The newly created property list.

The function creates a new property list and returns a handle for it. To destroy the returned property list object use the property list destructor **cpl_propertylist_delete()**.

See also

cpl_propertylist_delete()

Referenced by cpl_propertylist_duplicate(), and cpl_wcs_new_from_propertylist().

4.40.3.65 cpl_propertylist_prepend_bool()

```
cpl_error_code cpl_propertylist_prepend_bool (
            cpl_propertylist * self,
            const char * name,
            int value )
```

Prepend a boolean value to a property list.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The boolean value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.

The function creates a new boolean property with name *name* and value *value*. The property is prepended to the property list *self*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_property_new(), cpl_property_set_bool(), and CPL_TYPE_BOOL.

4.40.3.66 cpl_propertylist_prepend_char()

```
cpl_error_code cpl_propertylist_prepend_char (
            cpl_propertylist * self,
            const char * name,
            char value )
```

Prepend a character value to a property list.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The character value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.

The function creates a new character property with name *name* and value *value*. The property is prepended to the property list *self*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_property_new(), cpl_property_set_char(), and CPL_TYPE_CHAR.

4.40.3.67 cpl_propertylist_prepend_double()

```
cpl_error_code cpl_propertylist_prepend_double (
            cpl_propertylist * self,
            const char * name,
            double value )
```

Prepend a double value to a property list.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The double value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.

The function creates a new double property with name *name* and value *value*. The property is prepended to the property list *self*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_property_new(), cpl_property_set_double(), and CPL_TYPE_DOUBLE.

4.40.3.68 cpl_propertylist_prepend_double_complex()

```
cpl_error_code cpl_propertylist_prepend_double_complex (
            cpl_propertylist * self,
            const char * name,
            double complex value )
```

Prepend a double complex value to a property list.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The double complex value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.

The function creates a new double complex property with name *name* and value *value*. The property is prepended to the property list *self*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_property_new(), cpl_property_set_double_↩complex(), and CPL_TYPE_DOUBLE_COMPLEX.

4.40.3.69 cpl_propertylist_prepend_float()

```
cpl_error_code cpl_propertylist_prepend_float (
            cpl_propertylist * self,
            const char * name,
            float value )
```

Prepend a float value to a property list.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The float value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.

The function creates a new float property with name *name* and value *value*. The property is prepended to the property list *self*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_property_new(), cpl_property_set_float(), and CPL_TYPE_FLOAT.

4.40.3.70 cpl_propertylist_prepend_float_complex()

```
cpl_error_code cpl_propertylist_prepend_float_complex (
            cpl_propertylist * self,
            const char * name,
            float complex value )
```

Prepend a float complex value to a property list.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The float complex value to store.

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a NULL pointer.

The function creates a new float complex property with name *name* and value *value*. The property is prepended to the property list *self*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_property_new(), cpl_property_set_float_↩ complex(), and CPL_TYPE_FLOAT_COMPLEX.

4.40.3.71 cpl_propertylist_prepend_int()

```
cpl_error_code cpl_propertylist_prepend_int (
            cpl_propertylist * self,
            const char * name,
            int value )
```

Prepend a integer value to a property list.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The integer value to store.

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a NULL pointer.

The function creates a new integer property with name *name* and value *value*. The property is prepended to the property list *self*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_property_new(), cpl_property_set_int(), and CPL_TYPE_INT.

4.40.3.72 cpl_propertylist_prepend_long()

```
cpl_error_code cpl_propertylist_prepend_long (
            cpl_propertylist * self,
            const char * name,
            long value )
```

Prepend a long value to a property list.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The long value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.

The function creates a new long property with name *name* and value *value*. The property is prepended to the property list *self*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_property_new(), cpl_property_set_long(), and CPL_TYPE_LONG.

4.40.3.73 cpl_propertylist_prepend_long_long()

```
cpl_error_code cpl_propertylist_prepend_long_long (
            cpl_propertylist * self,
            const char * name,
            long long value )
```

Prepend a long long value to a property list.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The long long value to store.

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a NULL pointer.

The function creates a new long long property with name *name* and value *value*. The property is prepended to the property list *self*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_property_new(), cpl_property_set_long_↩ long(), and CPL_TYPE_LONG_LONG.

4.40.3.74 cpl_propertylist_prepend_property()

```
cpl_error_code cpl_propertylist_prepend_property (
            cpl_propertylist * self,
            const cpl_property * property )
```

Prepend a property to a property list.

Parameters

self	Property list to prepend to
property	The property to prepend

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *property* is a NULL pointer.

This function creates a new property and prepends it to the beginning of a property list. It will not check if the property already exists.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl_property_duplicate().

4.40.3.75 cpl_propertylist_prepend_string()

```
cpl_error_code cpl_propertylist_prepend_string (
            cpl_propertylist * self,
            const char * name,
            const char * value )
```

Prepend a string value to a property list.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The string value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *name* or *value* is a `NULL` pointer.

The function creates a new string property with name *name* and value *value*. The property is prepended to the property list *self*.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_property_new(), cpl_property_set_string(), and CPL_TYPE_STRING.

4.40.3.76 cpl_propertylist_save()

```
cpl_error_code cpl_propertylist_save (
            const cpl_propertylist * self,
            const char * filename,
            unsigned mode )
```

Save a property list to a FITS file.

Parameters

self	The property list to save or NULL if empty
filename	Name of the file to write
mode	The desired output options (combined with bitwise or)

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

Errors

CPL_ERROR_NULL_INPUT	The *filename* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	The parameter *mode* is invalid.
CPL_ERROR_FILE_IO	The file cannot be written or accessed.

This function saves a property list to a FITS file, using cfitsio. The data unit is empty.

Supported output modes are CPL_IO_CREATE (create a new file) and CPL_IO_EXTEND (append to an existing file)

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_IN↩PUT, CPL_IO_CREATE, and CPL_IO_EXTEND.

Referenced by cpl_image_save(), and cpl_vector_save().

4.40.3.77 cpl_propertylist_set_bool()

```
cpl_error_code cpl_propertylist_set_bool (
            cpl_propertylist * self,
            const char * name,
            int value )
```

Set the value of the given boolean property list entry.

Parameters

self	A property list.
name	The property name to look up.
value	New boolean value.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.

The function searches the property list *self* for a property named *name*. If it is present in the list, its boolean value is replaced with the boolean *value*. If there is more than one property with the same *name*, it takes the first one from the list.

References CPL_ERROR_NULL_INPUT.

4.40.3.78 cpl_propertylist_set_char()

```
cpl_error_code cpl_propertylist_set_char (
            cpl_propertylist * self,
            const char * name,
            char value )
```

Set the value of the given character property list entry.

Parameters

self	A property list.
name	The property name to look up.
value	New character value.

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.

The function searches the property list *self* for a property named *name*. If it is present in the list, its character value is replaced with the character *value*. If there is more than one property with the same *name*, it takes the first one from the list.

References CPL_ERROR_NULL_INPUT.

4.40.3.79 cpl_propertylist_set_comment()

```
cpl_error_code cpl_propertylist_set_comment (
            cpl_propertylist * self,
            const char * name,
            const char * comment )
```

Modify the comment field of the given property list entry.

Parameters

self	A property list.
name	The property name to look up.
comment	New comment string.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.

The function searches the property list *self* for a property named *name*. If it is present in the list, its comment is replaced by the string *comment*. The provided comment string may be `NULL`. In this case an already existing comment is deleted. If there is more than one property with the same *name*, it takes the first one from the list.

References CPL_ERROR_NULL_INPUT.

4.40.3.80 cpl_propertylist_set_double()

```
cpl_error_code cpl_propertylist_set_double (
            cpl_propertylist * self,
            const char * name,
            double value )
```

Set the value of the given double property list entry.

Parameters

self	A property list.
name	The property name to look up.
value	New double value.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.

The function searches the property list *self* for a property named *name*. If it is present in the list, its double value is replaced with the double *value*. If there is more than one property with the same *name*, it takes the first one from the list.

References CPL_ERROR_NULL_INPUT.

4.40.3.81 cpl_propertylist_set_double_complex()

```
cpl_error_code cpl_propertylist_set_double_complex (
            cpl_propertylist * self,
            const char * name,
            double complex value )
```

Set the value of the given double complex property list entry.

Parameters

self	A property list.
name	The property name to look up.
value	New double complex value.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.

The function searches the property list *self* for a property named *name*. If it is present in the list, its double complex value is replaced with the double complex *value*. If there is more than one property with the same *name*, it takes the first one from the list.

References CPL_ERROR_NULL_INPUT.

4.40.3.82 cpl_propertylist_set_float()

```
cpl_error_code cpl_propertylist_set_float (
            cpl_propertylist * self,
            const char * name,
            float value )
```

Set the value of the given float property list entry.

Parameters

self	A property list.
name	The property name to look up.
value	New float value.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.

The function searches the property list *self* for a property named *name*. If it is present in the list, its float value is replaced with the float *value*. If there is more than one property with the same *name*, it takes the first one from the list.

References CPL_ERROR_NULL_INPUT.

4.40.3.83 cpl_propertylist_set_float_complex()

```
cpl_error_code cpl_propertylist_set_float_complex (
            cpl_propertylist * self,
            const char * name,
            float complex value )
```

Set the value of the given float complex property list entry.

Parameters

self	A property list.
name	The property name to look up.
value	New float complex value.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.

The function searches the property list *self* for a property named *name*. If it is present in the list, its float complex value is replaced with the float complex *value*. If there is more than one property with the same *name*, it takes the first one from the list.

References CPL_ERROR_NULL_INPUT.

4.40.3.84 cpl_propertylist_set_int()

```
cpl_error_code cpl_propertylist_set_int (
            cpl_propertylist * self,
            const char * name,
            int value )
```

Set the value of the given integer property list entry.

Parameters

self	A property list.
name	The property name to look up.
value	New integer value.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.

The function searches the property list *self* for a property named *name*. If it is present in the list, its integer value is replaced with the integer *value*. If there is more than one property with the same *name*, it takes the first one from the list.

References CPL_ERROR_NULL_INPUT.

4.40.3.85 cpl_propertylist_set_long()

```
cpl_error_code cpl_propertylist_set_long (
            cpl_propertylist * self,
            const char * name,
            long value )
```

Set the value of the given long property list entry.

Parameters

self	A property list.
name	The property name to look up.
value	New long value.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.

The function searches the property list *self* for a property named *name*. If it is present in the list, its long value is replaced with the long *value*. If there is more than one property with the same *name*, it takes the first one from the list.

References CPL_ERROR_NULL_INPUT.

4.40.3.86 cpl_propertylist_set_long_long()

```
cpl_error_code cpl_propertylist_set_long_long (
            cpl_propertylist * self,
            const char * name,
            long long value )
```

Set the value of the given long long property list entry.

Parameters

self	A property list.
name	The property name to look up.
value	New long long value.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.

The function searches the property list *self* for a property named *name*. If it is present in the list, its long long value is replaced with *value*. If there is more than one property with the same *name*, it takes the first one from the list.

References CPL_ERROR_NULL_INPUT.

4.40.3.87 cpl_propertylist_set_string()

```
cpl_error_code cpl_propertylist_set_string (
            cpl_propertylist * self,
            const char * name,
            const char * value )
```

Set the value of the given string property list entry.

Parameters

self	A property list.
name	The property name to look up.
value	New string value.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *name* or *value* is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	The property list *self* does not contain a property with the name *name*.

The function searches the property list *self* for a property named *name*. If it is present in the list, its string value is replaced with the string *value*. If there is more than one property with the same *name*, it takes the first one from the list.

References CPL_ERROR_NULL_INPUT.

4.40.3.88 cpl_propertylist_sort()

```
cpl_error_code cpl_propertylist_sort (
            cpl_propertylist * self,
            cpl_propertylist_compare_func compare )
```

Sort a property list.

Parameters

self	The property list to sort.
compare	The function used to compare two properties.

Returns

The function returns `CPL_ERROR_NONE` on success, or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function sorts the property list *self* in place, using the function *compare* to determine whether a property is less, equal or greater than another one.

The function *compare* must be of the type cpl_propertylist_compare_func.

See also

cpl_propertylist_compare_func

References CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.40.3.89 cpl_propertylist_update_bool()

```
cpl_error_code cpl_propertylist_update_bool (
            cpl_propertylist * self,
            const char * name,
            int value )
```

Update a property list with a boolean value.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The boolean value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The property list *self* contains a property with the name *name* which is not of type `CPL_TYPE_BOOL`.

The function updates the property list *self* with the boolean value *value*. This means, if a property with the name *name* exists already its value is updated, otherwise a property with the name *name* is created and added to *self*. The update will fail if a property with the name *name* exists already which is not of type `CPL_TYPE_BOOL`.

References CPL_ERROR_NULL_INPUT.

4.40.3.90 cpl_propertylist_update_char()

```
cpl_error_code cpl_propertylist_update_char (
            cpl_propertylist * self,
            const char * name,
            char value )
```

Update a property list with a character value.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The character value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The property list *self* contains a property with the name *name* which is not of type `CPL_TYPE_CHAR`.

The function updates the property list *self* with the character value *value*. This means, if a property with the name *name* exists already its value is updated, otherwise a property with the name *name* is created and added to *self*. The update will fail if a property with the name *name* exists already which is not of type `CPL_TYPE_CHAR`.

References CPL_ERROR_NULL_INPUT.

4.40.3.91 cpl_propertylist_update_double()

```
cpl_error_code cpl_propertylist_update_double (
            cpl_propertylist * self,
            const char * name,
            double value )
```

Update a property list with a double value.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The double value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The property list *self* contains a property with the name *name* which is not of type `CPL_TYPE_DOUBLE`.

The function updates the property list *self* with the double value *value*. This means, if a property with the name *name* exists already its value is updated, otherwise a property with the name *name* is created and added to *self*. The update will fail if a property with the name *name* exists already which is not of type `CPL_TYPE_DOUBLE`.

References CPL_ERROR_NULL_INPUT.

4.40.3.92 cpl_propertylist_update_double_complex()

```
cpl_error_code cpl_propertylist_update_double_complex (
            cpl_propertylist * self,
            const char * name,
            double complex value )
```

Update a property list with a double complex value.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The double complex value to store.

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a NULL pointer.
CPL_ERROR_TYPE_MISMATCH	The property list *self* contains a property with the name *name* which is not of type CPL_TYPE_DOUBLE_COMPLEX.

The function updates the property list *self* with the double complex value *value*. This means, if a property with the name *name* exists already its value is updated, otherwise a property with the name *name* is created and added to *self*. The update will fail if a property with the name *name* exists already which is not of type CPL_TYPE_DOUB↩LE_COMPLEX.

References CPL_ERROR_NULL_INPUT.

4.40.3.93 cpl_propertylist_update_float()

```
cpl_error_code cpl_propertylist_update_float (
            cpl_propertylist * self,
            const char * name,
            float value )
```

Update a property list with a float value.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The float value to store.

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a NULL pointer.
CPL_ERROR_TYPE_MISMATCH	The property list *self* contains a property with the name *name* which is not of type CPL_TYPE_FLOAT.

The function updates the property list *self* with the float value *value*. This means, if a property with the name *name* exists already its value is updated, otherwise a property with the name *name* is created and added to *self*. The update will fail if a property with the name *name* exists already which is not of type CPL_TYPE_FLOAT.

References CPL_ERROR_NULL_INPUT.

4.40.3.94 cpl_propertylist_update_float_complex()

```
cpl_error_code cpl_propertylist_update_float_complex (
            cpl_propertylist * self,
            const char * name,
            float complex value )
```

Update a property list with a float complex value.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The float complex value to store.

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a NULL pointer.
CPL_ERROR_TYPE_MISMATCH	The property list *self* contains a property with the name *name* which is not of type CPL_TYPE_FLOAT_COMPLEX.

The function updates the property list *self* with the float complex value *value*. This means, if a property with the name *name* exists already its value is updated, otherwise a property with the name *name* is created and added to *self*. The update will fail if a property with the name *name* exists already which is not of type CPL_TYPE_FLOA↩T_COMPLEX.

References CPL_ERROR_NULL_INPUT.

4.40.3.95 cpl_propertylist_update_int()

```
cpl_error_code cpl_propertylist_update_int (
            cpl_propertylist * self,
            const char * name,
            int value )
```

Update a property list with a integer value.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The integer value to store.

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a NULL pointer.
CPL_ERROR_TYPE_MISMATCH	The property list *self* contains a property with the name *name* which is not of type CPL_TYPE_INT.

The function updates the property list *self* with the integer value *value*. This means, if a property with the name *name* exists already its value is updated, otherwise a property with the name *name* is created and added to *self*. The update will fail if a property with the name *name* exists already which is not of type CPL_TYPE_INT.

References CPL_ERROR_NULL_INPUT.

4.40.3.96 cpl_propertylist_update_long()

```
cpl_error_code cpl_propertylist_update_long (
            cpl_propertylist * self,
            const char * name,
            long value )
```

Update a property list with a long value.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The long value to store.

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a NULL pointer.
CPL_ERROR_TYPE_MISMATCH	The property list *self* contains a property with the name *name* which is not of type CPL_TYPE_LONG.

The function updates the property list *self* with the long value *value*. This means, if a property with the name *name* exists already its value is updated, otherwise a property with the name *name* is created and added to *self*. The update will fail if a property with the name *name* exists already which is not of type CPL_TYPE_LONG.

References CPL_ERROR_NULL_INPUT.

4.40.3.97 cpl_propertylist_update_long_long()

```
cpl_error_code cpl_propertylist_update_long_long (
            cpl_propertylist * self,
            const char * name,
            long long value )
```

Update a property list with a long long value.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The long long value to store.

Returns

The function returns CPL_ERROR_NONE on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *name* is a NULL pointer.
CPL_ERROR_TYPE_MISMATCH	The property list *self* contains a property with the name *name* which is not of type CPL_TYPE_LONG_LONG.

The function updates the property list *self* with the long long value *value*. This means, if a property with the name *name* exists already its value is updated, otherwise a property with the name *name* is created and added to *self*. The update will fail if a property with the name *name* exists already which is not of type CPL_TYPE_LONG_LONG.

References CPL_ERROR_NULL_INPUT.

4.40.3.98 cpl_propertylist_update_string()

```
cpl_error_code cpl_propertylist_update_string (
            cpl_propertylist * self,
            const char * name,
            const char * value )
```

Update a property list with a string value.

Parameters

self	A property list.
name	The property name to be assigned to the value.
value	The string value to store.

Returns

The function returns `CPL_ERROR_NONE` on success or a CPL error code otherwise.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *name* or *value* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	The property list *self* contains a property with the name *name* which is not of type `CPL_TYPE_STRING`.

The function updates the property list *self* with the string value *value*. This means, if a property with the name *name* exists already its value is updated, otherwise a property with the name *name* is created and added to *self*. The update will fail if a property with the name *name* exists already which is not of type `CPL_TYPE_STRING`.

References CPL_ERROR_NULL_INPUT.

4.41 Recipe Configurations

Functions

- void cpl_recipeconfig_clear (cpl_recipeconfig *self)

  Clear a recipe configuration object.
- void cpl_recipeconfig_delete (cpl_recipeconfig *self)

  Delete a recipe configuration object.
- char ** cpl_recipeconfig_get_inputs (const cpl_recipeconfig *self, const char *tag)

  Get the input configuration for a given tag.
- cpl_size cpl_recipeconfig_get_max_count (const cpl_recipeconfig *self, const char *tag, const char *input)

  Get the maximum number of frames for the given configuration and tag.
- cpl_size cpl_recipeconfig_get_min_count (const cpl_recipeconfig *self, const char *tag, const char *input)

  Get the minimum number of frames for the given configuration and tag.
- char ** cpl_recipeconfig_get_outputs (const cpl_recipeconfig *self, const char *tag)

  Get the output configuration for a given tag.
- char ** cpl_recipeconfig_get_tags (const cpl_recipeconfig *self)

  Get the list of supported configuration tags.
- int cpl_recipeconfig_is_required (const cpl_recipeconfig *self, const char *tag, const char *input)

  Check whether a frame with the given tag is required.
- cpl_recipeconfig * cpl_recipeconfig_new (void)

  Create a new recipe configuration object.
- int cpl_recipeconfig_set_input (cpl_recipeconfig *self, const char *tag, const char *input, cpl_size min_count, cpl_size max_count)

  Add the configuration for the given input and configuration tag.
- int cpl_recipeconfig_set_inputs (cpl_recipeconfig *self, const char *tag, const cpl_framedata *data)

  Set the input configuration for a given tag.
- int cpl_recipeconfig_set_output (cpl_recipeconfig *self, const char *tag, const char *output)

  Add an output frame tag for the given configuration tag.
- int cpl_recipeconfig_set_outputs (cpl_recipeconfig *self, const char *tag, const char **data)

  Set the output configuration for a given tag.
- int cpl_recipeconfig_set_tag (cpl_recipeconfig *self, const char *tag, cpl_size min_count, cpl_size max_count)

  Set a configuration tag.
- int cpl_recipeconfig_set_tags (cpl_recipeconfig *self, const cpl_framedata *data)

  Set the list of configuration tags.

4.41.1 Detailed Description

This module implements the support for recipe configurations. A recipe configuration stores information about the input data frames a recipe can process, which other input frame are needed in addition, and which output frames may be created by the recipe.

For each input frame extra information, for instance, whether a particular frame type is a required or optional recipe input, or how many frames of a certain type are at least needed, can also be stored.

The information for the individual recipe configurations and also for the individual frames can be accessed by means of a unique string identifier for the configuration and the frame respectively. This string identifier is called configuration tag in the former, and frame tag in the latter case. In particular, the configuration tag is a frame tag too, namely the frame tag of the recipe's "primary" input, or trigger frame.

The recipe configuration object stores a separate configuration for each of the different frame types, indicated by its tag, it is able to process. Each of these configurations can be retrieved, using the appropriate configuration tag as a key.

In the same way the information about individual frames can be retrieved from the selected configuration.

Synopsis:

```
#include <cpl_recipeconfig.h>
```

4.41.2 Function Documentation

4.41.2.1 cpl_recipeconfig_clear()

```
void cpl_recipeconfig_clear (
            cpl_recipeconfig * self )
```

Clear a recipe configuration object.

Parameters

self	The recipe configuration object.

Returns

Nothing.

The function clears the contents of the recipe configuration *self*. After the return from this call, *self* is empty.

See also

cpl_recipeconfig_new()

4.41.2.2 cpl_recipeconfig_delete()

```
void cpl_recipeconfig_delete (
            cpl_recipeconfig * self )
```

Delete a recipe configuration object.

Parameters

self	The recipe configuration object.

Returns

Nothing.

The function destroys the recipe configuration object *self*. Any resources used by *self* are released. If *self* is `NULL`, nothing is done and no error is set.

4.41.2.3 cpl_recipeconfig_get_inputs()

```
char** cpl_recipeconfig_get_inputs (
            const cpl_recipeconfig * self,
            const char * tag )
```

Get the input configuration for a given tag.

Parameters

self	The recipe configuration object.
tag	The tag for which the input configuration is requested.

Returns

On success, the function returns a NULL terminated array of strings, and NULL if an error occurred. In the latter case an appropriate error code is also set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *tag* is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	The configuration tag *tag* was not found.

The function retrieves the list of input frame tags stored in the recipe configuration for the configuration tag *tag* from the configuration object *self*.

In case the input configuration for the tag *tag* is empty, i.e. no input frame tag has been added the function still returns a C string array. In this case the first element is set to NULL.

The returned array and each of its elements must be deallocated using cpl_free() if they are no longer used. The array is NULL terminated, i.e. the last element of the array is set to NULL.

References CPL_ERROR_NULL_INPUT.

4.41.2.4 cpl_recipeconfig_get_max_count()

```
cpl_size cpl_recipeconfig_get_max_count (
            const cpl_recipeconfig * self,
            const char * tag,
            const char * input )
```

Get the maximum number of frames for the given configuration and tag.

Parameters

self	The recipe configuration object.
tag	The configuration tag to look up.
input	The frame tag to search for.

Returns

The function returns the maximum number of frames with the tag *input*, or `-1`, if an error occurred. In the latter case an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *tag* or *input* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	The configuration tag *tag* is `NULL`, or an invalid string, the empty string for instance.
CPL_ERROR_DATA_NOT_FOUND	No frame tag *tag* or *input* was found, or *self* was not properly initialized.

The function queries the recipe configuration *self* for the configuration tag *tag*, searches this configuration for the frame tag *input* and returns the maximum number of frames required for this frame type.

If the same string is passed as *tag* and *input*, the settings for the frame with tag *tag* are returned.

References CPL_ERROR_NULL_INPUT.

4.41.2.5 cpl_recipeconfig_get_min_count()

```
cpl_size cpl_recipeconfig_get_min_count (
            const cpl_recipeconfig * self,
            const char * tag,
            const char * input )
```

Get the minimum number of frames for the given configuration and tag.

Parameters

self	The recipe configuration object.
tag	The configuration tag to look up.
input	The frame tag to search for.

Returns

The function returns the minimum number of frames with the tag *input*, or `-1`, if an error occurred. In the latter case an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *tag* or *input* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	The configuration tag *tag* is `NULL`, or an invalid string, the empty string for instance.
CPL_ERROR_DATA_NOT_FOUND	No frame tag *tag* or *input* was found, or *self* was not properly initialized.

The function queries the recipe configuration *self* for the configuration tag *tag*, searches this configuration for the frame tag *input* and returns the minimum number of frames required for this frame type.

If the same string is passed as *tag* and *input*, the settings for the frame with tag *tag* are returned.

References CPL_ERROR_NULL_INPUT.

4.41.2.6 cpl_recipeconfig_get_outputs()

```
char** cpl_recipeconfig_get_outputs (
            const cpl_recipeconfig * self,
            const char * tag )
```

Get the output configuration for a given tag.

Parameters

self	The recipe configuration object.
tag	The tag for which the ouput configuration is requested.

Returns

On success, the function returns a NULL terminated array of strings, and NULL if an error occurred. In the latter case an appropriate error code is also set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *tag* is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	The configuration tag *tag* was not found.

The function retrieves the list of all possible output frame tags stored in the recipe configuration object *self* for the configuration tag *tag*.

In case the output configuration for the tag *tag* is empty, i.e. no output frame tag has been added the function still returns a C string array. In this case the first element is set to NULL.

The returned array and each of its elements must be deallocated using cpl_free() if they are no longer used. The array is NULL terminated, i.e. the last element of the array is set to NULL.

References CPL_ERROR_NULL_INPUT.

4.41.2.7 cpl_recipeconfig_get_tags()

```
char** cpl_recipeconfig_get_tags (
            const cpl_recipeconfig * self )
```

Get the list of supported configuration tags.

Parameters

self	The recipe configuration object.

Returns

On success, the function returns a `NULL` terminated array of strings, and `NULL` if an error occurred. In the latter case an appropriate error code is also set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.

The function retrieves the list of configuration tags stored in the recipe configuration object *self*. The frame tags are returned as the elements of an array of C strings. The last element of the array is a `NULL` pointer indicating the end of the list.

In case the recipe configuration object is empty, i.e. no configuration tag has been added, or cpl_recipeconfig_clear() has been called for this object, the function still returns the C string array. In this case the first element is set to `NULL`.

If the returned list is not used any more each element, and the array itself must be deallocated using cpl_free().

References CPL_ERROR_NULL_INPUT.

4.41.2.8 cpl_recipeconfig_is_required()

```
int cpl_recipeconfig_is_required (
             const cpl_recipeconfig * self,
             const char * tag,
             const char * input )
```

Check whether a frame with the given tag is required.

Parameters

self	A recipe configuration object.
tag	The configuration tag to look up.
input	The frame tag to search for.

Returns

The function returns `1` if the frame with the tag *input* is required, and `0` if the frame is not a required input. If an error occurred `-1` is returned and an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *tag* or *input* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	The frame tag *tag* is `NULL`, an invalid string, the empty string for instance.
CPL_ERROR_DATA_NOT_FOUND	No frame tag *tag* or *input* was found, or *self* was not properly initialized.

The function queries the recipe configuration *self* for the configuration tag *tag* and searches this configuration for the frame tag *input*. It returns `1` if one or more frames of type *input* are a mandatory input for the recipe configuration selected by *tag*.

If the same string is passed as *tag* and *input*, the settings for the frame with tag *tag* are returned.

References CPL_ERROR_NULL_INPUT.

4.41.2.9 cpl_recipeconfig_new()

```
cpl_recipeconfig* cpl_recipeconfig_new (
            void  )
```

Create a new recipe configuration object.

Returns

The function returns a pointer to the newly created recipe configuration.

The function creates a new, empty recipe configuration object.

4.41.2.10 cpl_recipeconfig_set_input()

```
int cpl_recipeconfig_set_input (
            cpl_recipeconfig * self,
            const char * tag,
            const char * input,
            cpl_size min_count,
            cpl_size max_count )
```

Add the configuration for the given input and configuration tag.

Parameters

self	The recipe configuration object.
tag	The configuration tag.
input	The input frame tag.
min_count	The value to set as the minimum number of frames.
max_count	The value to set as the maximum number of frames.

Returns

The function returns `0` on success and a non-zero value if an error occurred. The return value is `-1` if *self*, *tag* or *input* is `NULL`, or, if *tag* or *input* is an invalid string. The function returns `1` if *self* was not properly initialized using cpl_recipeconfig_set_tag() or cpl_recipeconfig_set_tags(). If no tag *tag* is found in *self* the function returns `2`. If an error occurs an appropriate error code is also set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *tag* or *input* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	The frame tag *tag* or *input* is an empty string, or *tag* and *input* are equal.
CPL_ERROR_DATA_NOT_FOUND	No configuration for the tag *tag* was found, or *self* was not properly initialized.

The function sets the configuration for the input frame tag *input* of the configuration associated with the tag *tag* in the recipe configuration object *self*. The minimum and maximum number of frames of this input frame tag are given using the *min_count* and *max_count* arguments. Using a value of `-1` for the minimum and maximum number of frames, means that these two numbers are unspecified. Using a minimum number *min_count* greater then `0` makes the frame a required input.

Before an input configuration can be set using this function, the configuration tag *tag* must have been added to *self* previously using cpl_recipeconfig_set_tag() or cpl_recipeconfig_set_tags().

See also

cpl_recipeconfig_set_tag(), cpl_recipeconfig_set_tags()

References CPL_ERROR_DATA_NOT_FOUND, CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_IN↩PUT.

4.41.2.11 cpl_recipeconfig_set_inputs()

```
int cpl_recipeconfig_set_inputs (
            cpl_recipeconfig * self,
            const char * tag,
            const cpl_framedata * data )
```

Set the input configuration for a given tag.

Parameters

self	The recipe configuration object.
tag	The tag for which the input configuration is set.
data	An array containing the configuration informations.

Returns

The function returns 0 on success, or a non-zero value otherwise. If *self* or *tag* is NULL, or if *tag* is invalid, i.e. the empty string the function returns −1 and sets an appropriate error code. If no configuration tags were previously configured using cpl_recipeconfig_set_tag() or cpl_recipeconfig_set_tags() the function returns 1. If no configuration was found for the given tag *tag* the return value is 2. Finally, if the frame tag to add to the configuration is invalid, the function returns 3.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *tag* is a NULL pointer.
CPL_ERROR_ILLEGAL_INPUT	The frame tag *tag* is an invalid tag, i.e. the empty string or the input frame tag read from *data* is invalid, or the same as *tag*.
CPL_ERROR_DATA_NOT_FOUND	No configuration for the tag *tag* was found, or *self* was not properly initialized.

The function sets the input configuration for the tag *tag* in the recipe configuration object *self*. The minimum and maximum number of frames of this tag can be given using the arguments *min_count* and *max_count*. Using a value of −1 for the minimum and maximum number of frames, means that these two numbers are unspecified. Using a value greater than 0 for the minimum number of frames makes the input frame a required frame.

The function sets the configuration data for each input tag specified in the array *data*, until a tag set to NULL is reached. The array *data* must contain such an entry as last element, in order to indicate the end of the array.

Before an input configuration can be set using this function, the configuration tag *tag* must have been added to *self* previously using cpl_recipeconfig_set_tag() or cpl_recipeconfig_set_tags().

See also

cpl_recipeconfig_set_tag(), cpl_recipeconfig_set_tags()

References CPL_ERROR_DATA_NOT_FOUND, CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_IN↩PUT.

4.41.2.12 cpl_recipeconfig_set_output()

```
int cpl_recipeconfig_set_output (
            cpl_recipeconfig * self,
            const char * tag,
            const char * output )
```

Add an output frame tag for the given configuration tag.

Parameters

self	The recipe configuration object.
tag	The configuration tag.
output	The output frame tag to add.

Returns

The function returns 0 on success and a non-zero value if an error occurred. The return value is −1 if *self*, *tag* or *output* is NULL, or if *tag* or *input* is an invalid string. The function returns 1 if *self* was not properly initialized using cpl_recipeconfig_set_tag() or cpl_recipeconfig_set_tags(). If no tag *tag* is found in *self* the function returns 2. If an error occurs an appropriate error code is also set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self*, *tag* or *output* is a NULL pointer.
CPL_ERROR_ILLEGAL_INPUT	The frame tag *tag* or *input* is an invalid string, i.e. the empty string.
CPL_ERROR_DATA_NOT_FOUND	No configuration for the tag *tag* was found, or *self* was not properly initialized.

The function adds the output frame tag *ouput* to the configuration associated with the tag *tag* in the recipe configuration object *self*.

Before an output frame tag can be set using this function, the configuration tag *tag* must have been added to *self* previously, using cpl_recipeconfig_set_tag() or cpl_recipeconfig_set_tags().

See also

cpl_recipeconfig_set_tag(), cpl_recipeconfig_set_tags()

References CPL_ERROR_DATA_NOT_FOUND, CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_IN↩PUT.

4.41.2.13 cpl_recipeconfig_set_outputs()

```
int cpl_recipeconfig_set_outputs (
             cpl_recipeconfig * self,
             const char * tag,
             const char ** data )
```

Set the output configuration for a given tag.

Parameters

self	The recipe configuration object.
tag	The tag for which the output configuration is set.
data	An array of strings containing the output frame tags to set.

Returns

The function returns 0 on success, or a non-zero value otherwise. If *self* or *tag* is NULL, or if *tag* is invalid, i.e. the empty string the function returns −1 and sets an appropriate error code. If no configuration tags were previously configured using cpl_recipeconfig_set_tag() or cpl_recipeconfig_set_tags() the function returns 1. If no configuration was found for the given tag *tag* the return value is 2.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *tag* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	The frame tag *tag* is an invalid tag, i.e. the empty string.
CPL_ERROR_DATA_NOT_FOUND	No configuration for the tag *tag* was found, or *self* was not properly initialized.

The function sets the output configuration for the tag *tag* in the recipe configuration object *self*. The output configuration is a list of all possible frame tags which could result from the execution of the corresponding recipe.

The function stores each output frame tag found in the array *data*, until an array element set to `NULL` is reached. The array *data* must contain such an entry as last element, in order to indicate the end of the array.

Before an output configuration can be set using this function, the configuration tag *tag* must have been added to *self* previously using cpl_recipeconfig_set_tag() or cpl_recipeconfig_set_tags().

See also

cpl_recipeconfig_set_tag(), cpl_recipeconfig_set_tags()

References CPL_ERROR_DATA_NOT_FOUND, CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_IN↩PUT.

4.41.2.14 cpl_recipeconfig_set_tag()

```
int cpl_recipeconfig_set_tag (
              cpl_recipeconfig * self,
              const char * tag,
              cpl_size min_count,
              cpl_size max_count )
```

Set a configuration tag.

Parameters

self	The recipe configuration object.
tag	The configuration tag.
min_count	The value to set as the minimum number of frames.
max_count	The value to set as the maximum number of frames.

Returns

The function returns `0` on success and a non-zero value if an error occurred. The return value is `-1` if *self* or *tag* is `NULL`, or if *tag* is an invalid string. If an error occurs an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* or *tag* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	The frame tag *tag* is an invalid tag, i.e. the empty string.

The function creates a configuration for the configuration tag *tag* and adds it to the recipe configuration object *self*. The minimum and maximum number of frames of this tag can be given using the arguments *min_count* and *max_count*. Using a value of `-1` for the minimum and maximum number of frames, means that these two numbers are unspecified. If the minimum number of frames is greater than `0`, the frame is considered to be required, otherwise it is optional.

References CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_INPUT.

4.41.2.15 cpl_recipeconfig_set_tags()

```
int cpl_recipeconfig_set_tags (
            cpl_recipeconfig * self,
            const cpl_framedata * data )
```

Set the list of configuration tags.

Parameters

self	The recipe configuration object.
data	A frame data array from which the tags are set.

Returns

The function returns `0` on success, or a non-zero value otherwise. Inparticular, if the *self* is `NULL` the function returns `-1` and sets the appropriate error code. If any configuration tag in the input array *data* is invalid, an empty string for instance, the function returns `1` and sets an appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *self* is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	The frame data array *data* contains an invalid tag.

The function is a convenience function to allow an initialization of a recipe configuration from static data. The configuration tags to be stored are taken from the frame data array *data* and are added to the recipe configuration *self*. The configuration tag is copied to *self*.

In addition the tags can be configured using the remaining members of the frame data structures of *data*.

The function adds each configuration tag found in the array *data* to the configuration *self*, until a configuration tag set to *NULL* is reached. The array *data* must be terminated by such an entry, which indicates the end of the array.

References CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_INPUT.

4.42 Recipe Definition

Macros

- #define cpl_get_license(PACKAGE_NAME, YEAR)

  Generate the recipe copyright and license text (GPL v.2)
- #define cpl_recipe_define(RECIPE_NAME, RECIPE_VERSION, RECIPE_AUTHOR, RECIPE_EMAIL, RECIPE_YEAR, RECIPE_SYNOPSIS, RECIPE_DESCRIPTION)

  Define a standard CPL recipe.
- #define CPL_RECIPE_DEFINE(RECIPE_NAME, RECIPE_VERSION, RECIPE_FILL_PARAMS, RECIPE_AUTHOR, RECIPE_AUTHOR_EMAIL, RECIPE_YEAR, RECIPE_SYNOPSIS, RECIPE_DESCRIPTION)

  Define a standard CPL recipe.

4.42.1 Detailed Description

This module implements the support for recipe defition.

Synopsis:

```
#include <cpl_recipedefine.h>
```

4.42.2 Macro Definition Documentation

4.42.2.1 cpl_get_license

```
#define cpl_get_license(
              PACKAGE_NAME,
              YEAR )
```

Generate the recipe copyright and license text (GPL v.2)

Parameters

PACKAGE_NAME	The name as a string literal, e.g. from config.h
YEAR	The year(s) as a string literal

Returns

The recipe copyright and license text as a string literal

Example:

```
const char * eso_gpl_license = cpl_get_license(PACKAGE_NAME, "2005, 2008");
```

4.42.2.2 cpl_recipe_define

```
#define cpl_recipe_define(
                RECIPE_NAME,
                RECIPE_VERSION,
                RECIPE_AUTHOR,
                RECIPE_EMAIL,
                RECIPE_YEAR,
                RECIPE_SYNOPSIS,
                RECIPE_DESCRIPTION )
```

Define a standard CPL recipe.

Parameters

RECIPE_NAME	The name as an identifier
RECIPE_VERSION	The binary version number
RECIPE_AUTHOR	The author as a string literal
RECIPE_EMAIL	The contact email as a string literal
RECIPE_YEAR	The copyright year as a string literal
RECIPE_SYNOPSIS	The synopsis as a string literal
RECIPE_DESCRIPTION	The man-page as a string literal

A CPL-based recipe may use this macro to define its four mandatory functions: cpl_plugin_get_info(), <recipe>_create(), <recipe>_exec() and <recipe>_destroy(), as well as declaring the actual data reduction function, <recipe>() as

```
static int <recipe>(cpl_frameset *, const cpl_parameterlist *);
```

The macro also declares the recipe-specific function that fills the recipe parameterlist with the supported parameters as

```
static cpl_error_code <recipe>_fill_parameterlist(cpl_parameterlist *self);
```

A recipe that invokes cpl_recipe_define() must define this function.

The macro cpl_recipe_define() may be used by defining a macro, e.g. in my_recipe.h:

```
#define MY_RECIPE_DEFINE(NAME, SYNOPSIS, DESCRIPTION)                   \
  cpl_recipe_define(NAME, MY_BINARY_VERSION,                            \
  "Firstname Lastname", "2006, 2008", SYNOPSIS, DESCRIPTION)
```

- and then by invoking this macro in each recipe:

```
#include "my_recipe.h"
MY_RECIPE_DEFINE(instrume_img_dark,
                 "Dark recipe",
                 "instrume_img_dark -- imaging dark recipe.\n"
                 " ... recipe man-page\n");
static
cpl_error_code instrume_img_dark_fill_parameterlist(cpl_parameterlist *self);
{
   // Fill the parameterlist with the parameters supported by the recipe.
   retun CPL_ERROR_NONE;
}
```

4.42.2.3 CPL_RECIPE_DEFINE

```
#define CPL_RECIPE_DEFINE(
              RECIPE_NAME,
              RECIPE_VERSION,
              RECIPE_FILL_PARAMS,
              RECIPE_AUTHOR,
              RECIPE_AUTHOR_EMAIL,
              RECIPE_YEAR,
              RECIPE_SYNOPSIS,
              RECIPE_DESCRIPTION )
```

Define a standard CPL recipe.

Parameters

RECIPE_NAME	The name as an identifier
RECIPE_VERSION	The binary version number
RECIPE_FILL_PARAMS	A function call to fill the recipe parameterlist. Must evaluate to zero if and only if successful
RECIPE_AUTHOR	The author as a string literal
RECIPE_AUTHOR_EMAIL	The author email as a string literal
RECIPE_YEAR	The copyright year as a string literal
RECIPE_SYNOPSIS	The synopsis as a string literal
RECIPE_DESCRIPTION	The man-page as a string literal

Deprecated Use cpl_recipe_define()

4.43 Recipes

Recipe plugin interface definition.

Classes

- struct _cpl_recipe_

  The type representation of the recipe plugin interface.

Typedefs

- typedef struct _cpl_recipe_ cpl_recipe

  The recipe plugin data type.

4.43.1 Detailed Description

Recipe plugin interface definition.

This defines the interface in order to implement recipes as Pluggable Data Reduction Modules (PDRMs). It extends the generic plugin interface with a parameter list and a frame set containing the recipe's setup information (parameters to run with) and the data frames to process.

This interface is constructed in such a way, that a pointer to an object of type `cpl_recipe` can be cast into a pointer to `cpl_plugin` (see Plugin Interface).

Synopsis:

```
#include <cpl_recipe.h>
```

4.43.2 Typedef Documentation

4.43.2.1 cpl_recipe

```
typedef struct _cpl_recipe_ cpl_recipe
```

The recipe plugin data type.

4.44 Regular Expression Filter

Typedefs

- typedef struct _cpl_regex_ cpl_regex

  The opaque regular expression filter data type.
- typedef enum _cpl_regex_syntax_option_ cpl_regex_syntax_option

  Regular expression syntax options.

Enumerations

- enum _cpl_regex_syntax_option_ {
  CPL_REGEX_ICASE,
  CPL_REGEX_NOSUBS,
  CPL_REGEX_BASIC,
  CPL_REGEX_EXTENDED }

Functions

- int cpl_regex_apply (const cpl_regex ∗self, const char ∗string)

  Compare a regular expression with a given character string.
- void cpl_regex_delete (cpl_regex ∗self)

  Destroys a regular expression filter object.
- int cpl_regex_is_negated (const cpl_regex ∗self)

  Test whether a regular expression filter is negated.
- void cpl_regex_negate (cpl_regex ∗self)

  Toggle the negation state of a regular expression filter.
- cpl_regex ∗ cpl_regex_new (const char ∗expression, int negated, flag_type flags)

  Create a new regular expression filter.

4.44.1 Detailed Description

The module implements a regular expression filter type. The type `cpl_regex` is a compiled regular expression created with a given set of regular expression syntax options, and an optional negation of the result when it is applied to an input string.

4.44.2 Typedef Documentation

4.44.2.1 cpl_regex

```
typedef struct _cpl_regex_ cpl_regex
```

The opaque regular expression filter data type.

4.44.2.2 cpl_regex_syntax_option

```
typedef enum _cpl_regex_syntax_option_ cpl_regex_syntax_option
```

Regular expression syntax options.

4.44.3 Enumeration Type Documentation

4.44.3.1 _cpl_regex_syntax_option_

```
enum _cpl_regex_syntax_option_
```

Regular expressions syntax options

Enumerator

CPL_REGEX_ICASE	Case insensitive searches.
CPL_REGEX_NOSUBS	No sub-expressions.
CPL_REGEX_BASIC	Basic POSIX grammer.
CPL_REGEX_EXTENDED	Extended POSIX grammer.

4.44.4 Function Documentation

4.44.4.1 cpl_regex_apply()

```
int cpl_regex_apply (
            const cpl_regex * self,
            const char * string )
```

Compare a regular expression with a given character string.

Parameters

self	The regular expression filter object to apply.
string	The string to be tested.

Returns

The function returns 0 if the filter object does not match the input string, and a non-zero value otherwise.

The function compares the input string *string* with the regular expression of the filter object *self*. The function returns a non-zero value for positive matches, i.e. the regular expression matches the input string *string* and the filter is not

negated, or the regular expression does not match the input string but the filter is negated. Otherwise the function reports a negative match, i.e. returns `0`.

4.44.4.2 cpl_regex_delete()

```
void cpl_regex_delete (
             cpl_regex * self )
```

Destroys a regular expression filter object.

Parameters

self	The regular expression filter object.

Returns

Nothing.

The function destroys the given regular expression filter object *self*, and deallocates the memory used. If the filter object *self* is `NULL`, nothing is done and no error is set.

4.44.4.3 cpl_regex_is_negated()

```
int cpl_regex_is_negated (
             const cpl_regex * self )
```

Test whether a regular expression filter is negated.

Parameters

self	The regular expression filter object to test.

Returns

The function returns `0` if the filter *self* is not negated, and `1` otherwise.

The function reports whether the filter *self* is negated or not.

4.44.4.4 cpl_regex_negate()

```
void cpl_regex_negate (
             cpl_regex * self )
```

Toggle the negation state of a regular expression filter.

Parameters

self	The regular expression filter object to update.

Returns

Nothing.

The function toggles the negation state of the given regular expression filter object *self*. If *self* is negated, it is not negated after this function has been called, and vice versa.

4.44.4.5 cpl_regex_new()

```
cpl_regex* cpl_regex_new (
             const char * expression,
             int negated,
             flag_type flags )
```

Create a new regular expression filter.

Parameters

expression	Regular expression.
negated	Negate the result when applying the filter.
flags	Regular expression syntax options.

Returns

The function returns a newly allocated regular expression filter object, or `NULL` in case an error occurred.

The function allocates a regular expression filter object and initializes it with the compiled regular expression *expression*. If the flag *negated* is set the result when applying the filter to an input string is negated. The argument *flags* allows to specify regular expression syntax options for the compilation of the regular expression.

The returned regular expression filter object must be destroyed using the destructor cpl_regex_delete().

Note that the syntax option `CPL_REGEX_NOSUBS` is always set implicitly, since the interface does not allow to retrieve this information.

References CPL_ERROR_ILLEGAL_INPUT, CPL_REGEX_BASIC, CPL_REGEX_EXTENDED, CPL_REGEX_I↩CASE, and CPL_REGEX_NOSUBS.

4.45 Statistics

Typedefs

- typedef struct _cpl_stats_ cpl_stats

  The opaque CPL stats data type.
- typedef enum _cpl_stats_mode_ cpl_stats_mode

  The CPL stats mode. It is a bit field.

Enumerations

- enum _cpl_stats_mode_ {
  CPL_STATS_MIN,
  CPL_STATS_MAX,
  CPL_STATS_MEAN,
  CPL_STATS_MEDIAN,
  CPL_STATS_STDEV,
  CPL_STATS_FLUX,
  CPL_STATS_ABSFLUX,
  CPL_STATS_SQFLUX,
  CPL_STATS_MINPOS,
  CPL_STATS_MAXPOS,
  CPL_STATS_CENTROID,
  CPL_STATS_MEDIAN_DEV,
  CPL_STATS_MAD,
  CPL_STATS_ALL }

  The values of the CPL stats mode. The values can be combined with bitwise or.

Functions

- void cpl_stats_delete (cpl_stats *stats)

  Free memory associated to an cpl_stats object.
- cpl_error_code cpl_stats_dump (const cpl_stats *self, cpl_stats_mode mode, FILE *stream)

  Dump a cpl_stats object.
- double cpl_stats_get_absflux (const cpl_stats *in)

  Get the absolute flux from a cpl_stats object.
- double cpl_stats_get_centroid_x (const cpl_stats *in)

  Get the x centroid position from a cpl_stats object.
- double cpl_stats_get_centroid_y (const cpl_stats *in)

  Get the y centroid position from a cpl_stats object.
- double cpl_stats_get_flux (const cpl_stats *in)

  Get the flux from a cpl_stats object.
- double cpl_stats_get_mad (const cpl_stats *in)

  Get the median of the absolute median deviation.
- double cpl_stats_get_max (const cpl_stats *in)

  Get the maximum from a cpl_stats object.
- cpl_size cpl_stats_get_max_x (const cpl_stats *in)

  Get the maximum x position from a cpl_stats object.
- cpl_size cpl_stats_get_max_y (const cpl_stats *in)

  Get the maximum y position from a cpl_stats object.
- double cpl_stats_get_mean (const cpl_stats *in)

Get the mean from a cpl_stats object.

- double cpl_stats_get_median (const cpl_stats *in)

  Get the median from a cpl_stats object.
- double cpl_stats_get_median_dev (const cpl_stats *in)

  Get the mean of the absolute median deviation from a cpl_stats object.
- double cpl_stats_get_min (const cpl_stats *in)

  Get the minimum from a cpl_stats object.
- cpl_size cpl_stats_get_min_x (const cpl_stats *in)

  Get the minimum x position from a cpl_stats object.
- cpl_size cpl_stats_get_min_y (const cpl_stats *in)

  Get the minimum y position from a cpl_stats object.
- cpl_size cpl_stats_get_npix (const cpl_stats *in)

  Get the number of pixels from a cpl_stats object.
- double cpl_stats_get_sqflux (const cpl_stats *in)

  Get the sum of the squared values from a cpl_stats object.
- double cpl_stats_get_stdev (const cpl_stats *in)

  Get the std. dev. from a cpl_stats object.
- cpl_stats * cpl_stats_new_from_image (const cpl_image *image, cpl_stats_mode mode)

  Compute various statistics of an image.
- cpl_stats * cpl_stats_new_from_image_window (const cpl_image *image, cpl_stats_mode mode, cpl_size llx, cpl_size lly, cpl_size urx, cpl_size ury)

  Compute various statistics of an image sub-window.

4.45.1 Detailed Description

This module provides functions to handle the cpl_stats object. This object can contain the statistics that have been computed from different CPL objects. Currently, only the function that computes statistics on images (or images windows) is provided.

Synopsis:

```
#include "cpl_stats.h"
```

4.45.2 Typedef Documentation

4.45.2.1 cpl_stats

```
typedef struct _cpl_stats_ cpl_stats
```

The opaque CPL stats data type.

4.45.2.2 cpl_stats_mode

```
typedef enum _cpl_stats_mode_ cpl_stats_mode
```

The CPL stats mode. It is a bit field.

4.45.3 Enumeration Type Documentation

4.45.3.1 _cpl_stats_mode_

```
enum _cpl_stats_mode_
```

The values of the CPL stats mode. The values can be combined with bitwise or.

Enumerator

CPL_STATS_MIN	The minimum
CPL_STATS_MAX	The maximum
CPL_STATS_MEAN	The mean
CPL_STATS_MEDIAN	The median
CPL_STATS_STDEV	The standard deviation
CPL_STATS_FLUX	The flux
CPL_STATS_ABSFLUX	The absolute flux
CPL_STATS_SQFLUX	The square flux
CPL_STATS_MINPOS	The position of the minimum
CPL_STATS_MAXPOS	The position of the maximum
CPL_STATS_CENTROID	The centroid position
CPL_STATS_MEDIAN_DEV	The mean of the absolute median deviation
CPL_STATS_MAD	The median of the absolute median deviation
CPL_STATS_ALL	All of the above

4.45.4 Function Documentation

4.45.4.1 cpl_stats_delete()

```
void cpl_stats_delete (
            cpl_stats * stats )
```

Free memory associated to an cpl_stats object.

Parameters

stats	the object to delete

Returns

void

Frees all memory associated to a cpl_stats object. If the object *stats* is `NULL`, nothing is done and no error is set.

References cpl_free().

4.45.4.2 cpl_stats_dump()

```
cpl_error_code cpl_stats_dump (
            const cpl_stats * self,
            cpl_stats_mode mode,
            FILE * stream )
```

Dump a cpl_stats object.

Parameters

self	cpl_stats object to dump
mode	Bit field specifying which statistics to dump
stream	The output stream

Returns

CPL_ERROR_NONE or the relevant the _cpl_error_code_

See also

cpl_stats_new_from_image_window()

It is an error to request parameters that have not been set.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if mode specifies statistics that have not been computed
- CPL_ERROR_FILE_IO if the write fails

References cpl_ensure_code, CPL_ERROR_FILE_IO, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, CPL_ERROR_UNSUPPORTED_MODE, CPL_SIZE_FORMAT, CPL_STATS_A↩BSFLUX, CPL_STATS_CENTROID, CPL_STATS_FLUX, CPL_STATS_MAD, CPL_STATS_MAX, CPL_ST↩ATS_MAXPOS, CPL_STATS_MEAN, CPL_STATS_MEDIAN, CPL_STATS_MEDIAN_DEV, CPL_STATS_MIN, CPL_STATS_MINPOS, CPL_STATS_SQFLUX, and CPL_STATS_STDEV.

4.45.4.3 cpl_stats_get_absflux()

```
double cpl_stats_get_absflux (
            const cpl_stats * in )
```

Get the absolute flux from a cpl_stats object.

Parameters

in	the cpl_stats object

Returns

The absolute flux, or a negative number on error

See also

cpl_stats_get_min()

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, and CPL_STATS_ABSF↩LUX.

4.45.4.4 cpl_stats_get_centroid_x()

```
double cpl_stats_get_centroid_x (
            const cpl_stats * in )
```

Get the x centroid position from a cpl_stats object.

Parameters

in	the cpl_stats object

Returns

the x centroid

See also

cpl_stats_get_min()

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, and CPL_STATS_CENT↩ROID.

4.45.4.5 cpl_stats_get_centroid_y()

```
double cpl_stats_get_centroid_y (
            const cpl_stats * in )
```

Get the y centroid position from a cpl_stats object.

Parameters

in	the cpl_stats object

Returns

the y centroid

See also

cpl_stats_get_min()

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, and CPL_STATS_CENTROID.

4.45.4.6 cpl_stats_get_flux()

```
double cpl_stats_get_flux (
            const cpl_stats * in )
```

Get the flux from a cpl_stats object.

Parameters

in	the cpl_stats object

Returns

the flux

See also

cpl_stats_get_min()

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, and CPL_STATS_FLUX.

4.45.4.7 cpl_stats_get_mad()

```
double cpl_stats_get_mad (
            const cpl_stats * in )
```

Get the median of the absolute median deviation.

Parameters

in	the cpl_stats object

Returns

The median of the absolute median deviation, or undefined on error

See also

cpl_stats_get_min()

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, and CPL_STATS_MAD.

4.45.4.8 cpl_stats_get_max()

```
double cpl_stats_get_max (
            const cpl_stats * in )
```

Get the maximum from a cpl_stats object.

Parameters

in	the cpl_stats object

Returns

the maximum value

See also

cpl_stats_get_min()

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, CPL_STATS_MAX, and CPL_STATS_MAXPOS.

4.45.4.9 cpl_stats_get_max_x()

```
cpl_size cpl_stats_get_max_x (
            const cpl_stats * in )
```

Get the maximum x position from a cpl_stats object.

Parameters

in	the cpl_stats object

Returns

the x position (1 for the first pixel), non-positive on error.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, CPL_STATS_MAX, and CPL_STATS_MAXPOS.

4.45.4.10 cpl_stats_get_max_y()

```
cpl_size cpl_stats_get_max_y (
            const cpl_stats * in )
```

Get the maximum y position from a cpl_stats object.

Parameters

in	the cpl_stats object

Returns

the y position (1 for the first pixel), non-positive on error.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, CPL_STATS_MAX, and CPL_STATS_MAXPOS.

4.45.4.11 cpl_stats_get_mean()

```
double cpl_stats_get_mean (
            const cpl_stats * in )
```

Get the mean from a cpl_stats object.

Parameters

in	the cpl_stats object

Returns

the mean value

See also

cpl_stats_get_min()

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, and CPL_STATS_MEAN.

4.45.4.12 cpl_stats_get_median()

```
double cpl_stats_get_median (
            const cpl_stats * in )
```

Get the median from a cpl_stats object.

Parameters

in	the cpl_stats object

Returns

the median value

See also

cpl_stats_get_min()

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, and CPL_STATS_MEDIAN.

4.45.4.13 cpl_stats_get_median_dev()

```
double cpl_stats_get_median_dev (
            const cpl_stats * in )
```

Get the mean of the absolute median deviation from a cpl_stats object.

Parameters

in	the cpl_stats object

Returns

The mean of the absolute median deviation, or undefined on error

See also

cpl_stats_get_min()

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, and CPL_STATS_MEDI↩AN_DEV.

4.45.4.14 cpl_stats_get_min()

```
double cpl_stats_get_min (
            const cpl_stats * in )
```

Get the minimum from a cpl_stats object.

Parameters

in	the cpl_stats object

Returns

the minimum value

The call that created the cpl_stats object must have determined the minimum value.

In case of error, the _cpl_error_code_ code is set, and the returned double is undefined.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the requested stat has not been computed in in

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, CPL_STATS_MIN, and CPL_STATS_MINPOS.

4.45.4.15 cpl_stats_get_min_x()

```
cpl_size cpl_stats_get_min_x (
            const cpl_stats * in )
```

Get the minimum x position from a cpl_stats object.

Parameters

in	the cpl_stats object

Returns

the x position (1 for the first pixel), non-positive on error.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, CPL_STATS_MIN, and CPL_STATS_MINPOS.

4.45.4.16 cpl_stats_get_min_y()

```
cpl_size cpl_stats_get_min_y (
            const cpl_stats * in )
```

Get the minimum y position from a cpl_stats object.

Parameters

in	the cpl_stats object

Returns

the y position (1 for the first pixel), non-positive on error.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, CPL_STATS_MIN, and CPL_STATS_MINPOS.

4.45.4.17 cpl_stats_get_npix()

```
cpl_size cpl_stats_get_npix (
            const cpl_stats * in )
```

Get the number of pixels from a cpl_stats object.

Parameters

in	the cpl_stats object

Returns

the number of pixels, -1 in error case.

The creation of a cpl_stats object always causes the number of pixels to be determined.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

4.45.4.18 cpl_stats_get_sqflux()

```
double cpl_stats_get_sqflux (
            const cpl_stats * in )
```

Get the sum of the squared values from a cpl_stats object.

Parameters

in	the cpl_stats object

Returns

the square flux, or a negative number on error

See also

cpl_stats_get_min()

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, and CPL_STATS_SQFL↩UX.

4.45.4.19 cpl_stats_get_stdev()

```
double cpl_stats_get_stdev (
            const cpl_stats * in )
```

Get the std. dev. from a cpl_stats object.

Parameters

in	the cpl_stats object

Returns

the standard deviation

See also

cpl_stats_get_min()

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, and CPL_STATS_STDEV.

4.45.4.20 cpl_stats_new_from_image()

```
cpl_stats* cpl_stats_new_from_image (
             const cpl_image * image,
             cpl_stats_mode mode )
```

Compute various statistics of an image.

Parameters

image	input image.
mode	Bit field specifying which statistics to compute

Returns

1 newly allocated cpl_stats structure or NULL in error case

See also

cpl_stats_new_from_image_window()

References cpl_ensure, CPL_ERROR_NULL_INPUT, and cpl_stats_new_from_image_window().

4.45.4.21 cpl_stats_new_from_image_window()

```
cpl_stats* cpl_stats_new_from_image_window (
             const cpl_image * image,
             cpl_stats_mode mode,
             cpl_size llx,
             cpl_size lly,
             cpl_size urx,
             cpl_size ury )
```

Compute various statistics of an image sub-window.

Parameters

image	Input image.
mode	Bit field specifying which statistics to compute
llx	Lower left x position (FITS convention)
lly	Lower left y position (FITS convention)
urx	Upper right x position (FITS convention)
ury	Upper right y position (FITS convention)

Returns

1 newly allocated cpl_stats structure or NULL in error case

Compute various image statistics.

The specified bounds are included in the specified region.

The statistics to compute is specified with a bit field, that may be set to any of these values

- CPL_STATS_MIN
- CPL_STATS_MAX
- CPL_STATS_MEAN
- CPL_STATS_MEDIAN
- CPL_STATS_MEDIAN_DEV
- CPL_STATS_MAD
- CPL_STATS_STDEV
- CPL_STATS_FLUX
- CPL_STATS_ABSFLUX
- CPL_STATS_SQFLUX
- CPL_STATS_CENTROID
- CPL_STATS_MINPOS
- CPL_STATS_MAXPOS or any bitwise or (|) of these. For convenience the special value CPL_STATS_ALL may also be used, it is the conbination of all of the above values.

E.g. the mode CPL_STATS_MIN | CPL_STATS_MEDIAN specifies the minimum and the median of the image.

In the case of CPL_STATS_MIN and CPL_STATS_MAX where more than one set of coordinates share the extremum it is undefined which of those coordinates will be returned.

On i386 platforms there can be significant differences in the round-off of the computation of single statistics and statistics computed via CPL_STATS_ALL. This is especially true for squared quantities such as the CPL_STATS↩ _SQFLUX and CPL_STATS_STDEV.

Images can be CPL_TYPE_DOUBLE, CPL_TYPE_FLOAT, CPL_TYPE_INT.

For the CPL_STATS_CENTROID computation, if there are negative pixels, the minimum value is added to all the pixels in order to have all pixels with positive values for computation.

The returned object must be deallocated using cpl_stats_delete().

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ACCESS_OUT_OF_RANGE if the defined window is not in the image
- CPL_ERROR_ILLEGAL_INPUT if the window definition is wrong (e.g llx $>$ urx)
- CPL_ERROR_DATA_NOT_FOUND if all the pixels are bad
- CPL_ERROR_INVALID_TYPE if the passed image type is not supported
- CPL_ERROR_INVALID_TYPE if mode is 1, e.g. due to a logical or ($||$) of the allowed options.
- CPL_ERROR_UNSUPPORTED_MODE if mode is otherwise different from the allowed options.

References cpl_malloc().

Referenced by cpl_stats_new_from_image().

4.46 Tables

Enumerations

- enum { ,
  CPL_DFS_SIGNATURE_NONE,
  CPL_DFS_SIGNATURE_DATAMD5,
  CPL_DFS_SIGNATURE_CHECKSUM }

Functions

- cpl_error_code cpl_table_abs_column (cpl_table *table, const char *name)

  Compute the absolute value of column values.
- cpl_error_code cpl_table_add_columns (cpl_table *table, const char *to_name, const char *from_name)

  Add the values of two numeric or complex table columns.
- cpl_error_code cpl_table_add_scalar (cpl_table *table, const char *name, double value)

  Add a constant value to a numerical or complex column.
- cpl_error_code cpl_table_add_scalar_complex (cpl_table *table, const char *name, double complex value)

  Add a constant complex value to a numerical or complex column.
- cpl_size cpl_table_and_selected (cpl_table *table, const char *name1, cpl_table_select_operator operator, const char *name2)

  Select from selected table rows, by comparing the values of two numerical columns.
- cpl_size cpl_table_and_selected_double (cpl_table *table, const char *name, cpl_table_select_operator operator, double value)

  Select from selected table rows, by comparing double column values with a constant.
- cpl_size cpl_table_and_selected_double_complex (cpl_table *table, const char *name, cpl_table_select_↩ operator operator, double complex value)

  Select from selected table rows, by comparing double complex column values with a complex constant.
- cpl_size cpl_table_and_selected_float (cpl_table *table, const char *name, cpl_table_select_operator operator, float value)

  Select from selected table rows, by comparing float column values with a constant.
- cpl_size cpl_table_and_selected_float_complex (cpl_table *table, const char *name, cpl_table_select_↩ operator operator, float complex value)

  Select from selected table rows, by comparing float complex column values with a complex constant.
- cpl_size cpl_table_and_selected_int (cpl_table *table, const char *name, cpl_table_select_operator operator, int value)

  Select from selected table rows, by comparing integer column values with a constant.
- cpl_size cpl_table_and_selected_invalid (cpl_table *table, const char *name)

  Select from selected table rows all rows with an invalid value in a specified column.
- cpl_size cpl_table_and_selected_long (cpl_table *table, const char *name, cpl_table_select_operator operator, long value)

  Select from selected table rows, by comparing long column values with a constant.
- cpl_size cpl_table_and_selected_long_long (cpl_table *table, const char *name, cpl_table_select_operator operator, long long value)

  Select from selected table rows, by comparing long long column values with a constant.
- cpl_size cpl_table_and_selected_string (cpl_table *table, const char *name, cpl_table_select_operator operator, const char *string)

  Select from selected table rows, by comparing string column values with a character string.
- cpl_size cpl_table_and_selected_window (cpl_table *table, cpl_size start, cpl_size count)

  Select from selected rows only those within a table segment.
- cpl_error_code cpl_table_arg_column (cpl_table *table, const char *name)

Compute the phase angle value of table column elements.

- cpl_error_code cpl_table_cast_column (cpl_table *table, const char *from_name, const char *to_name, cpl_type type)

  Cast a numeric or complex column to a new numeric or complex type column.
- int cpl_table_compare_structure (const cpl_table *table1, const cpl_table *table2)

  Compare the structure of two tables.
- cpl_error_code cpl_table_conjugate_column (cpl_table *table, const char *name)

  Compute the complex conjugate of column values.
- cpl_error_code cpl_table_copy_data_double (cpl_table *table, const char *name, const double *data)

  Copy existing data to a table double column.
- cpl_error_code cpl_table_copy_data_double_complex (cpl_table *table, const char *name, const double complex *data)

  Copy existing data to a table double complex column.
- cpl_error_code cpl_table_copy_data_float (cpl_table *table, const char *name, const float *data)

  Copy existing data to a table float column.
- cpl_error_code cpl_table_copy_data_float_complex (cpl_table *table, const char *name, const float complex *data)

  Copy existing data to a table float complex column.
- cpl_error_code cpl_table_copy_data_int (cpl_table *table, const char *name, const int *data)

  Copy existing data to a table integer column.
- cpl_error_code cpl_table_copy_data_long (cpl_table *table, const char *name, const long *data)

  Copy existing data to a table long column.
- cpl_error_code cpl_table_copy_data_long_long (cpl_table *table, const char *name, const long long *data)

  Copy existing data to a table long long column.
- cpl_error_code cpl_table_copy_data_string (cpl_table *table, const char *name, const char **data)

  Copy existing data to a table string column.
- cpl_error_code cpl_table_copy_structure (cpl_table *table, const cpl_table *mtable)

  Give to a table the same structure of another table.
- cpl_size cpl_table_count_invalid (const cpl_table *table, const char *name)

  Count number of invalid values in a table column.
- cpl_size cpl_table_count_selected (const cpl_table *table)

  Get number of selected rows in given table.
- void cpl_table_delete (cpl_table *table)

  Delete a table.
- cpl_error_code cpl_table_divide_columns (cpl_table *table, const char *to_name, const char *from_name)

  Divide two numeric or complex table columns.
- cpl_error_code cpl_table_divide_scalar (cpl_table *table, const char *name, double value)

  Divide a numerical or complex column by a constant.
- cpl_error_code cpl_table_divide_scalar_complex (cpl_table *table, const char *name, double complex value)

  Divide a numerical or complex column by a complex constant.
- void cpl_table_dump (const cpl_table *table, cpl_size start, cpl_size count, FILE *stream)

  Print a table.
- void cpl_table_dump_structure (const cpl_table *table, FILE *stream)

  Describe the structure and the contents of a table.
- cpl_table * cpl_table_duplicate (const cpl_table *table)

  Make a copy of a table.
- cpl_error_code cpl_table_duplicate_column (cpl_table *to_table, const char *to_name, const cpl_table *from_table, const char *from_name)

  Copy a column from a table to another.
- cpl_error_code cpl_table_erase_column (cpl_table *table, const char *name)

  Delete a column from a table.

- cpl_error_code cpl_table_erase_invalid (cpl_table *table)

  Remove from a table all columns just containing invalid elements, and then all rows containing at least one invalid element.
- cpl_error_code cpl_table_erase_invalid_rows (cpl_table *table)

  Remove from a table columns and rows just containing invalid elements.
- cpl_error_code cpl_table_erase_selected (cpl_table *table)

  Delete the selected rows of a table.
- cpl_error_code cpl_table_erase_window (cpl_table *table, cpl_size start, cpl_size count)

  Delete a table segment.
- cpl_error_code cpl_table_exponential_column (cpl_table *table, const char *name, double base)

  Compute the exponential of column values.
- cpl_table * cpl_table_extract (const cpl_table *table, cpl_size start, cpl_size count)

  Create a table from a section of another table.
- cpl_table * cpl_table_extract_selected (const cpl_table *table)

  Create a new table from the selected rows of another table.
- cpl_error_code cpl_table_fill_column_window (cpl_table *table, const char *name, cpl_size start, cpl_size count, double value)

  Write a value to a numerical column segment.
- cpl_error_code cpl_table_fill_column_window_array (cpl_table *table, const char *name, cpl_size start, cpl_size count, const cpl_array *array)

  Write an array to an array column segment.
- cpl_error_code cpl_table_fill_column_window_complex (cpl_table *table, const char *name, cpl_size start, cpl_size count, double complex value)

  Write a value to a complex column segment.
- cpl_error_code cpl_table_fill_column_window_double (cpl_table *table, const char *name, cpl_size start, cpl_size count, double value)

  Write a value to a double column segment.
- cpl_error_code cpl_table_fill_column_window_double_complex (cpl_table *table, const char *name, cpl_size start, cpl_size count, double complex value)

  Write a value to a double complex column segment.
- cpl_error_code cpl_table_fill_column_window_float (cpl_table *table, const char *name, cpl_size start, cpl_size count, float value)

  Write a value to a float column segment.
- cpl_error_code cpl_table_fill_column_window_float_complex (cpl_table *table, const char *name, cpl_size start, cpl_size count, float complex value)

  Write a value to a float complex column segment.
- cpl_error_code cpl_table_fill_column_window_int (cpl_table *table, const char *name, cpl_size start, cpl_size count, int value)

  Write a value to an integer column segment.
- cpl_error_code cpl_table_fill_column_window_long (cpl_table *table, const char *name, cpl_size start, cpl_size count, long value)

  Write a value to an long column segment.
- cpl_error_code cpl_table_fill_column_window_long_long (cpl_table *table, const char *name, cpl_size start, cpl_size count, long long value)

  Write a value to an long long column segment.
- cpl_error_code cpl_table_fill_column_window_string (cpl_table *table, const char *name, cpl_size start, cpl_size count, const char *value)

  Write a character string to a string column segment.
- cpl_error_code cpl_table_fill_invalid_double (cpl_table *table, const char *name, double code)

  Write a numerical value to invalid double column elements.
- cpl_error_code cpl_table_fill_invalid_double_complex (cpl_table *table, const char *name, double complex code)

Write a numerical value to invalid double complex column elements.

- cpl_error_code cpl_table_fill_invalid_float (cpl_table *table, const char *name, float code)

  Write a numerical value to invalid float column elements.
- cpl_error_code cpl_table_fill_invalid_float_complex (cpl_table *table, const char *name, float complex code)

  Write a numerical value to invalid float complex column elements.
- cpl_error_code cpl_table_fill_invalid_int (cpl_table *table, const char *name, int code)

  Write a numerical value to invalid integer column elements.
- cpl_error_code cpl_table_fill_invalid_long (cpl_table *table, const char *name, long code)

  Write a numerical value to invalid long column elements.
- cpl_error_code cpl_table_fill_invalid_long_long (cpl_table *table, const char *name, long long code)

  Write a numerical value to invalid long long column elements.
- double cpl_table_get (const cpl_table *table, const char *name, cpl_size row, int *null)

  Read a value from a numerical column.
- const cpl_array * cpl_table_get_array (const cpl_table *table, const char *name, cpl_size row)

  Read an array from an array column.
- cpl_size cpl_table_get_column_depth (const cpl_table *table, const char *name)

  Get the depth of a table column.
- cpl_size cpl_table_get_column_dimension (const cpl_table *table, const char *name, cpl_size indx)

  Get size of one dimension of a table column of arrays.
- cpl_size cpl_table_get_column_dimensions (const cpl_table *table, const char *name)

  Get the number of dimensions of a table column of arrays.
- const char * cpl_table_get_column_format (const cpl_table *table, const char *name)

  Get the format of a table column.
- double cpl_table_get_column_max (const cpl_table *table, const char *name)

  Get maximum value in a numerical column.
- cpl_error_code cpl_table_get_column_maxpos (const cpl_table *table, const char *name, cpl_size *row)

  Get position of maximum in a numerical column.
- double cpl_table_get_column_mean (const cpl_table *table, const char *name)

  Compute the mean value of a numerical column.
- double complex cpl_table_get_column_mean_complex (const cpl_table *table, const char *name)

  Compute the mean value of a numerical or complex column.
- double cpl_table_get_column_median (const cpl_table *table, const char *name)

  Compute the median value of a numerical column.
- double cpl_table_get_column_min (const cpl_table *table, const char *name)

  Get minimum value in a numerical column.
- cpl_error_code cpl_table_get_column_minpos (const cpl_table *table, const char *name, cpl_size *row)

  Get position of minimum in a numerical column.
- const char * cpl_table_get_column_name (const cpl_table *table)

  Get table columns names.
- cpl_array * cpl_table_get_column_names (const cpl_table *table)

  Get table columns names.
- double cpl_table_get_column_stdev (const cpl_table *table, const char *name)

  Find the standard deviation of a table column.
- cpl_type cpl_table_get_column_type (const cpl_table *table, const char *name)

  Get the type of a table column.
- const char * cpl_table_get_column_unit (const cpl_table *table, const char *name)

  Get the unit of a table column.
- double complex cpl_table_get_complex (const cpl_table *table, const char *name, cpl_size row, int *null)

  Read a value from a complex column.
- cpl_array ** cpl_table_get_data_array (cpl_table *table, const char *name)

  Get a pointer to array column data.

- const cpl_array ** cpl_table_get_data_array_const (const cpl_table *table, const char *name)

  Get a pointer to array column data.
- double * cpl_table_get_data_double (cpl_table *table, const char *name)

  Get a pointer to double column data.
- double complex * cpl_table_get_data_double_complex (cpl_table *table, const char *name)

  Get a pointer to double complex column data.
- const double complex * cpl_table_get_data_double_complex_const (const cpl_table *table, const char *name)

  Get a pointer to constant double complex column data.
- const double * cpl_table_get_data_double_const (const cpl_table *table, const char *name)

  Get a pointer to constant double column data.
- float * cpl_table_get_data_float (cpl_table *table, const char *name)

  Get a pointer to float column data.
- float complex * cpl_table_get_data_float_complex (cpl_table *table, const char *name)

  Get a pointer to float complex column data.
- const float complex * cpl_table_get_data_float_complex_const (const cpl_table *table, const char *name)

  Get a pointer to constant float complex column data.
- const float * cpl_table_get_data_float_const (const cpl_table *table, const char *name)

  Get a pointer to constant float column data.
- int * cpl_table_get_data_int (cpl_table *table, const char *name)

  Get a pointer to integer column data.
- const int * cpl_table_get_data_int_const (const cpl_table *table, const char *name)

  Get a pointer to constant integer column data.
- long * cpl_table_get_data_long (cpl_table *table, const char *name)

  Get a pointer to long column data.
- const long * cpl_table_get_data_long_const (const cpl_table *table, const char *name)

  Get a pointer to constant long column data.
- long long * cpl_table_get_data_long_long (cpl_table *table, const char *name)

  Get a pointer to long long column data.
- const long long * cpl_table_get_data_long_long_const (const cpl_table *table, const char *name)

  Get a pointer to constant long long column data.
- char ** cpl_table_get_data_string (cpl_table *table, const char *name)

  Get a pointer to string column data.
- const char ** cpl_table_get_data_string_const (const cpl_table *table, const char *name)

  Get a pointer to constant string column data.
- double cpl_table_get_double (const cpl_table *table, const char *name, cpl_size row, int *null)

  Read a value from a double column.
- double complex cpl_table_get_double_complex (const cpl_table *table, const char *name, cpl_size row, int *null)

  Read a value from a double complex column.
- float cpl_table_get_float (const cpl_table *table, const char *name, cpl_size row, int *null)

  Read a value from a float column.
- float complex cpl_table_get_float_complex (const cpl_table *table, const char *name, cpl_size row, int *null)

  Read a value from a float complex column.
- int cpl_table_get_int (const cpl_table *table, const char *name, cpl_size row, int *null)

  Read a value from an integer column.
- long cpl_table_get_long (const cpl_table *table, const char *name, cpl_size row, int *null)

  Read a value from a long column.
- long long cpl_table_get_long_long (const cpl_table *table, const char *name, cpl_size row, int *null)

  Read a value from a long long column.
- cpl_size cpl_table_get_ncol (const cpl_table *table)

Get the number of columns in a table.

- cpl_size cpl_table_get_nrow (const cpl_table *table)

  Get the number of rows in a table.

- const char * cpl_table_get_string (const cpl_table *table, const char *name, cpl_size row)

  Read a value from a string column.

- int cpl_table_has_column (const cpl_table *table, const char *name)

  Check if a column with a given name exists.

- int cpl_table_has_invalid (const cpl_table *table, const char *name)

  Check if a column contains at least one invalid value.

- int cpl_table_has_valid (const cpl_table *table, const char *name)

  Check if a column contains at least one valid value.

- cpl_error_code cpl_table_imag_column (cpl_table *table, const char *name)

  Compute the imaginary part value of table column elements.

- cpl_error_code cpl_table_insert (cpl_table *target_table, const cpl_table *insert_table, cpl_size row)

  Merge two tables.

- cpl_error_code cpl_table_insert_window (cpl_table *table, cpl_size start, cpl_size count)

  Insert a segment of rows into table data.

- int cpl_table_is_selected (const cpl_table *table, cpl_size row)

  Determine whether a table row is selected or not.

- int cpl_table_is_valid (const cpl_table *table, const char *name, cpl_size row)

  Check if a column element is valid.

- cpl_table * cpl_table_load (const char *filename, int xtnum, int check_nulls)

  Load a FITS table extension into a new cpl_table.

- cpl_table * cpl_table_load_window (const char *filename, int xtnum, int check_nulls, const cpl_array *selcol, cpl_size firstrow, cpl_size nrow)

  Load part of a FITS table extension into a new cpl_table.

- cpl_error_code cpl_table_logarithm_column (cpl_table *table, const char *name, double base)

  Compute the logarithm of column values.

- cpl_error_code cpl_table_move_column (cpl_table *to_table, const char *name, cpl_table *from_table)

  Move a column from a table to another.

- cpl_error_code cpl_table_multiply_columns (cpl_table *table, const char *to_name, const char *from_name)

  Multiply two numeric or complex table columns.

- cpl_error_code cpl_table_multiply_scalar (cpl_table *table, const char *name, double value)

  Multiply a numerical or complex column by a constant.

- cpl_error_code cpl_table_multiply_scalar_complex (cpl_table *table, const char *name, double complex value)

  Multiply a numerical or complex column by a complex constant.

- cpl_error_code cpl_table_name_column (cpl_table *table, const char *from_name, const char *to_name)

  Rename a table column.

- cpl_table * cpl_table_new (cpl_size length)

  Create an empty table structure.

- cpl_error_code cpl_table_new_column (cpl_table *table, const char *name, cpl_type type)

  Create an empty column in a table.

- cpl_error_code cpl_table_new_column_array (cpl_table *table, const char *name, cpl_type type, cpl_size depth)

  Create an empty column of arrays in a table.

- cpl_size cpl_table_not_selected (cpl_table *table)

  Select unselected table rows, and unselect selected ones.

- cpl_size cpl_table_or_selected (cpl_table *table, const char *name1, cpl_table_select_operator operator, const char *name2)

  Select from unselected table rows, by comparing the values of two numerical columns.

- cpl_size cpl_table_or_selected_double (cpl_table *table, const char *name, cpl_table_select_operator operator, double value)

  Select from unselected table rows, by comparing double column values with a constant.
- cpl_size cpl_table_or_selected_double_complex (cpl_table *table, const char *name, cpl_table_select_↩ operator operator, double complex value)

  Select from unselected table rows, by comparing double complex column values with a complex constant.
- cpl_size cpl_table_or_selected_float (cpl_table *table, const char *name, cpl_table_select_operator operator, float value)

  Select from unselected table rows, by comparing float column values with a constant.
- cpl_size cpl_table_or_selected_float_complex (cpl_table *table, const char *name, cpl_table_select_↩ operator operator, float complex value)

  Select from unselected table rows, by comparing float complex column values with a complex constant.
- cpl_size cpl_table_or_selected_int (cpl_table *table, const char *name, cpl_table_select_operator operator, int value)

  Select from unselected table rows, by comparing integer column values with a constant.
- cpl_size cpl_table_or_selected_invalid (cpl_table *table, const char *name)

  Select from unselected table rows all rows with an invalid value in a specified column.
- cpl_size cpl_table_or_selected_long (cpl_table *table, const char *name, cpl_table_select_operator operator, long value)

  Select from unselected table rows, by comparing long column values with a constant.
- cpl_size cpl_table_or_selected_long_long (cpl_table *table, const char *name, cpl_table_select_operator operator, long long value)

  Select from unselected table rows, by comparing long long column values with a constant.
- cpl_size cpl_table_or_selected_string (cpl_table *table, const char *name, cpl_table_select_operator operator, const char *string)

  Select from unselected table rows, by comparing column values with a constant.
- cpl_size cpl_table_or_selected_window (cpl_table *table, cpl_size start, cpl_size count)

  Select from unselected rows only those within a table segment.
- cpl_error_code cpl_table_power_column (cpl_table *table, const char *name, double exponent)

  Compute the power of numerical column values.
- cpl_error_code cpl_table_real_column (cpl_table *table, const char *name)

  Compute the real part value of table column elements.
- cpl_error_code cpl_table_save (const cpl_table *table, const cpl_propertylist *pheader, const cpl_propertylist *header, const char *filename, unsigned mode)

  Save a cpl_table to a FITS file.
- cpl_error_code cpl_table_select_all (cpl_table *table)

  Select all table rows.
- cpl_error_code cpl_table_select_row (cpl_table *table, cpl_size row)

  Flag a table row as selected.
- cpl_error_code cpl_table_set (cpl_table *table, const char *name, cpl_size row, double value)

  Write a value to a numerical table column element.
- cpl_error_code cpl_table_set_array (cpl_table *table, const char *name, cpl_size row, const cpl_array *array)

  Write an array to an array table column element.
- cpl_error_code cpl_table_set_column_depth (cpl_table *table, const char *name, cpl_size depth)

  Modify depth of a column of arrays.
- cpl_error_code cpl_table_set_column_dimensions (cpl_table *table, const char *name, const cpl_array *dimensions)

  Set the dimensions of a table column of arrays.
- cpl_error_code cpl_table_set_column_format (cpl_table *table, const char *name, const char *format)

  Give a new format to a table column.
- cpl_error_code cpl_table_set_column_invalid (cpl_table *table, const char *name, cpl_size start, cpl_size count)

Invalidate a column segment.

- cpl_error_code cpl_table_set_column_unit (cpl_table *table, const char *name, const char *unit)

  Give a new unit to a table column.
- cpl_error_code cpl_table_set_complex (cpl_table *table, const char *name, cpl_size row, double complex value)

  Write a complex value to a complex numerical table column element.
- cpl_error_code cpl_table_set_double (cpl_table *table, const char *name, cpl_size row, double value)

  Write a value to a double table column element.
- cpl_error_code cpl_table_set_double_complex (cpl_table *table, const char *name, cpl_size row, double complex value)

  Write a value to a double complex table column element.
- cpl_error_code cpl_table_set_float (cpl_table *table, const char *name, cpl_size row, float value)

  Write a value to a float table column element.
- cpl_error_code cpl_table_set_float_complex (cpl_table *table, const char *name, cpl_size row, float complex value)

  Write a value to a float complex table column element.
- cpl_error_code cpl_table_set_int (cpl_table *table, const char *name, cpl_size row, int value)

  Write a value to an integer table column element.
- cpl_error_code cpl_table_set_invalid (cpl_table *table, const char *name, cpl_size row)

  Flag a column element as invalid.
- cpl_error_code cpl_table_set_long (cpl_table *table, const char *name, cpl_size row, long value)

  Write a value to an long table column element.
- cpl_error_code cpl_table_set_long_long (cpl_table *table, const char *name, cpl_size row, long long value)

  Write a value to an long long table column element.
- cpl_error_code cpl_table_set_size (cpl_table *table, cpl_size new_length)

  Resize a table to a new number of rows.
- cpl_error_code cpl_table_set_string (cpl_table *table, const char *name, cpl_size row, const char *value)

  Write a character string to a string table column element.
- cpl_error_code cpl_table_shift_column (cpl_table *table, const char *name, cpl_size shift)

  Shift the position of numeric or complex column values.
- cpl_error_code cpl_table_sort (cpl_table *table, const cpl_propertylist *reflist)

  Sort table rows according to columns values.
- cpl_error_code cpl_table_subtract_columns (cpl_table *table, const char *to_name, const char *from_name)

  Subtract two numeric or complex table columns.
- cpl_error_code cpl_table_subtract_scalar (cpl_table *table, const char *name, double value)

  Subtract a constant value from a numerical or complex column.
- cpl_error_code cpl_table_subtract_scalar_complex (cpl_table *table, const char *name, double complex value)

  Subtract a constant complex value from a numerical or complex column.
- cpl_error_code cpl_table_unselect_all (cpl_table *table)

  Unselect all table rows.
- cpl_error_code cpl_table_unselect_row (cpl_table *table, cpl_size row)

  Flag a table row as unselected.
- void * cpl_table_unwrap (cpl_table *table, const char *name)

  Unwrap a table column.
- cpl_array * cpl_table_where_selected (const cpl_table *table)

  Get array of indexes to selected table rows.
- cpl_error_code cpl_table_wrap_double (cpl_table *table, double *data, const char *name)

  Create in table a new double column obtained from existing data.
- cpl_error_code cpl_table_wrap_double_complex (cpl_table *table, double complex *data, const char *name)

  Create in table a new double complex column from existing data.

- cpl_error_code cpl_table_wrap_float (cpl_table *table, float *data, const char *name)

  Create in table a new float column obtained from existing data.
- cpl_error_code cpl_table_wrap_float_complex (cpl_table *table, float complex *data, const char *name)

  Create in table a new float complex column obtained from existing data.
- cpl_error_code cpl_table_wrap_int (cpl_table *table, int *data, const char *name)

  Create in table a new integer column obtained from existing data.
- cpl_error_code cpl_table_wrap_long (cpl_table *table, long *data, const char *name)

  Create in table a new long column obtained from existing data.
- cpl_error_code cpl_table_wrap_long_long (cpl_table *table, long long *data, const char *name)

  Create in table a new long long column obtained from existing data.
- cpl_error_code cpl_table_wrap_string (cpl_table *table, char **data, const char *name)

  Create in table a new string column obtained from existing data.

4.46.1 Detailed Description

This module provides functions to create, use, and destroy a *cpl_table*. A *cpl_table* is made of columns, and a column consists of an array of elements of a given type. Currently three numerical types are supported, `CPL_TYPE_INT`, `CPL_TYPE_FLOAT`, and `CPL_TYPE_DOUBLE`, plus a type indicating columns containing character strings, `CPL_TYPE_STRING`. Moreover, it is possible to define columns of arrays, i.e. columns whose elements are arrays of all the basic types listed above. Within the same column all arrays must have the same type and the same length.

A table column is accessed by specifying its name. The ordering of the columns within a table is undefined: a *cpl_table* is not an n-tuple of columns, but just a set of columns. The N elements of a column are counted from 0 to N-1, with element 0 on top. The set of all the table columns elements with the same index constitutes a table row, and table rows are counted according to the same convention. It is possible to flag each *cpl_table* row as "selected" or "unselected", and each column element as "valid" or "invalid". Selecting table rows is mainly a way to extract just those table parts fulfilling any given condition, while invalidating column elements is a way to exclude such elements from any computation. A *cpl_table* is created with all rows selected, and a column is created with all elements invalidated.

Note

The *cpl_table* pointers specified in the argument list of all the *cpl_table* functions must point to valid objects: these functions do not perform any check in this sense. Only in the particular case of a `NULL` pointer the functions will set a `CPL_ERROR_NULL_INPUT` error code, unless differently specified.

Synopsis:

```
#include <cpl_table.h>
```

4.46.2 Enumeration Type Documentation

4.46.2.1 anonymous enum

```
anonymous enum
```

Enumerator

CPL_DFS_SIGNATURE_NONE	Do not compute any signatures
CPL_DFS_SIGNATURE_DATAMD5	Compute the DATAMD5 data hash
CPL_DFS_SIGNATURE_CHECKSUM	Compute FITS standard CHECKSUM and DATASUM

4.46.3 Function Documentation

4.46.3.1 cpl_table_abs_column()

```
cpl_error_code cpl_table_abs_column (
            cpl_table * table,
            const char * name )
```

Compute the absolute value of column values.

Parameters

table	Pointer to table.
name	Table column name.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is not numerical, or has type array.

Each column element is replaced by its absolute value. Invalid elements are not modified by this operation. If the column is complex, its type will be turned to real (CPL_TYPE_FLOAT_COMPLEX will be changed into CPL_TYPE_FLOAT, and CPL_TYPE_DOUBLE_COMPLEX will be changed into CPL_TYPE_DOUBLE), and any pointer retrieved by calling `cpl_table_get_data_float_complex()` and `cpl_array_get_data_double_complex()` should be discarded.

4.46.3.2 cpl_table_add_columns()

```
cpl_error_code cpl_table_add_columns (
            cpl_table * table,
            const char * to_name,
            const char * from_name )
```

Add the values of two numeric or complex table columns.

Parameters

table	Pointer to table.
to_name	Name of target column.
from_name	Name of source column.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or any column name are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with any specified name is not found in *table*.
CPL_ERROR_INVALID_TYPE	Any specified column is neither numerical nor complex, or it is an array column.

The columns are summed element by element, and the result of the sum is stored in the target column. The columns' types may differ, and in that case the operation would be performed using the standard C upcasting rules, with a final cast of the result to the target column type. Invalid elements are propagated consistently: if either or both members of the sum are invalid, the result will be invalid too. Underflows and overflows are ignored.

4.46.3.3 cpl_table_add_scalar()

```
cpl_error_code cpl_table_add_scalar (
            cpl_table * table,
            const char * name,
            double value )
```

Add a constant value to a numerical or complex column.

Parameters

table	Pointer to table.
name	Column name.
value	Value to add.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is neither numerical nor complex, or it is an array column.

The operation is always performed in double precision, with a final cast of the result to the target column type. Invalid elements are are not modified by this operation.

4.46.3.4 cpl_table_add_scalar_complex()

```
cpl_error_code cpl_table_add_scalar_complex (
            cpl_table * table,
            const char * name,
            double complex value )
```

Add a constant complex value to a numerical or complex column.

Parameters

table	Pointer to table.
name	Column name.
value	Value to add.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is neither numerical nor complex, or it is an array column.

The operation is always performed in double precision, with a final cast of the result to the target column type. Invalid elements are are not modified by this operation.

4.46.3.5 cpl_table_and_selected()

```
cpl_size cpl_table_and_selected (
            cpl_table * table,
            const char * name1,
            cpl_table_select_operator operator,
            const char * name2 )
```

Select from selected table rows, by comparing the values of two numerical columns.

Parameters

table	Pointer to table.
name1	Name of first table column.
operator	Relational operator.
name2	Name of second table column.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column names are NULL pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with any of the specified names is not found in *table*.
CPL_ERROR_INVALID_TYPE	Invalid types for comparison.

Either both columns must be numerical, or they both must be strings. The comparison between strings is lexicographical. Comparison between complex types and array types are not supported.

For all the already selected table rows, the values of the specified columns are compared. The table rows not fulfilling the comparison are unselected. Invalid elements from either columns never fulfill any comparison by definition. Allowed relational operators are CPL_EQUAL_TO, CPL_NOT_EQUAL_TO, CPL_GREATER_THAN, CPL_NOT_GREATER_THAN, CPL_LESS_THAN, CPL_NOT_LESS_THAN. See also the function cpl_table_or_selected().

4.46.3.6 cpl_table_and_selected_double()

```
cpl_size cpl_table_and_selected_double (
            cpl_table * table,
            const char * name,
            cpl_table_select_operator operator,
            double value )
```

Select from selected table rows, by comparing *double* column values with a constant.

Parameters

table	Pointer to table.
name	Column name.
operator	Relational operator.
value	Reference value.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are NULL pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_DOUBLE.

For all the already selected table rows, the values of the specified column are compared with the reference value. All

table rows not fulfilling the comparison are unselected. An invalid element never fulfills any comparison by definition. Allowed relational operators are CPL_EQUAL_TO, CPL_NOT_EQUAL_TO, CPL_GREATER_THAN, CPL_NO↩T_GREATER_THAN, CPL_LESS_THAN, and CPL_NOT_LESS_THAN. If the table has no rows, no error is set, and 0 is returned. See also the function cpl_table_or_selected_double().

4.46.3.7 cpl_table_and_selected_double_complex()

```
cpl_size cpl_table_and_selected_double_complex (
            cpl_table * table,
            const char * name,
            cpl_table_select_operator operator,
            double complex value )
```

Select from selected table rows, by comparing *double* complex column values with a complex constant.

Parameters

table	Pointer to table.
name	Column name.
operator	Relational operator.
value	Reference value.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are NULL pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_DOUBLE_CO↩MPLEX.
CPL_ERROR_ILLEGAL_INPUT	Operator other than CPL_EQUAL_TO or CPL_NOT_EQUAL_TO was specified.

For all the already selected table rows, the values of the specified column are compared with the reference value. All table rows not fulfilling the comparison are unselected. An invalid element never fulfills any comparison by definition. Allowed relational operators are CPL_EQUAL_TO and CPL_NOT_EQUAL_TO. If the table has no rows, no error is set, and 0 is returned. See also the function cpl_table_or_selected_double_complex().

4.46.3.8 cpl_table_and_selected_float()

```
cpl_size cpl_table_and_selected_float (
            cpl_table * table,
            const char * name,
            cpl_table_select_operator operator,
            float value )
```

Select from selected table rows, by comparing *float* column values with a constant.

Parameters

table	Pointer to table.
name	Column name.
operator	Relational operator.
value	Reference value.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_FLOAT`.

For all the already selected table rows, the values of the specified column are compared with the reference value. All table rows not fulfilling the comparison are unselected. An invalid element never fulfills any comparison by definition. Allowed relational operators are `CPL_EQUAL_TO`, `CPL_NOT_EQUAL_TO`, `CPL_GREATER_THAN`, `CPL_NOT_GREATER_THAN`, `CPL_LESS_THAN`, and `CPL_NOT_LESS_THAN`. If the table has no rows, no error is set, and 0 is returned. See also the function `cpl_table_or_selected_float()`.

4.46.3.9 cpl_table_and_selected_float_complex()

```
cpl_size cpl_table_and_selected_float_complex (
            cpl_table * table,
            const char * name,
            cpl_table_select_operator operator,
            float complex value )
```

Select from selected table rows, by comparing *float* complex column values with a complex constant.

Parameters

table	Pointer to table.
name	Column name.
operator	Relational operator.
value	Reference value.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are NULL pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_FLOAT_COMPLEX.
CPL_ERROR_ILLEGAL_INPUT	Operator other than CPL_EQUAL_TO or CPL_NOT_EQUAL_TO was specified.

For all the already selected table rows, the values of the specified column are compared with the reference value. All table rows not fulfilling the comparison are unselected. An invalid element never fulfills any comparison by definition. Allowed relational operators are CPL_EQUAL_TO and CPL_NOT_EQUAL_TO. If the table has no rows, no error is set, and 0 is returned. See also the function cpl_table_or_selected_float_complex().

4.46.3.10 cpl_table_and_selected_int()

```
cpl_size cpl_table_and_selected_int (
            cpl_table * table,
            const char * name,
            cpl_table_select_operator operator,
            int value )
```

Select from selected table rows, by comparing *integer* column values with a constant.

Parameters

table	Pointer to table.
name	Column name.
operator	Relational operator.
value	Reference value.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are NULL pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_INT.

For all the already selected table rows, the values of the specified column are compared with the reference value. All table rows not fulfilling the comparison are unselected. An invalid element never fulfills any comparison by definition. Allowed relational operators are CPL_EQUAL_TO, CPL_NOT_EQUAL_TO, CPL_GREATER_THAN, CPL_NOT_GREATER_THAN, CPL_LESS_THAN, and CPL_NOT_LESS_THAN. If the table has no rows, no error is set, and 0 is returned. See also the function cpl_table_or_selected_int().

4.46.3.11 cpl_table_and_selected_invalid()

```
cpl_size cpl_table_and_selected_invalid (
            cpl_table * table,
            const char * name )
```

Select from selected table rows all rows with an invalid value in a specified column.

Parameters

table	Pointer to table.
name	Column name.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.

For all the already selected table rows, all the rows containing valid values at the specified column are unselected. See also the function `cpl_table_or_selected_invalid()`.

4.46.3.12 cpl_table_and_selected_long()

```
cpl_size cpl_table_and_selected_long (
            cpl_table * table,
            const char * name,
            cpl_table_select_operator operator,
            long value )
```

Select from selected table rows, by comparing *long* column values with a constant.

Parameters

table	Pointer to table.
name	Column name.
operator	Relational operator.
value	Reference value.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are NULL pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_LONG.

For all the already selected table rows, the values of the specified column are compared with the reference value. All table rows not fulfilling the comparison are unselected. An invalid element never fulfills any comparison by definition. Allowed relational operators are CPL_EQUAL_TO, CPL_NOT_EQUAL_TO, CPL_GREATER_THAN, CPL_NO↩T_GREATER_THAN, CPL_LESS_THAN, and CPL_NOT_LESS_THAN. If the table has no rows, no error is set, and 0 is returned. See also the function cpl_table_or_selected_long().

4.46.3.13 cpl_table_and_selected_long_long()

```
cpl_size cpl_table_and_selected_long_long (
            cpl_table * table,
            const char * name,
            cpl_table_select_operator operator,
            long long value )
```

Select from selected table rows, by comparing *long long* column values with a constant.

Parameters

table	Pointer to table.
name	Column name.
operator	Relational operator.
value	Reference value.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are NULL pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_LONG_LONG.

For all the already selected table rows, the values of the specified column are compared with the reference value. All table rows not fulfilling the comparison are unselected. An invalid element never fulfills any comparison by definition. Allowed relational operators are CPL_EQUAL_TO, CPL_NOT_EQUAL_TO, CPL_GREATER_THAN, CPL_NO↩T_GREATER_THAN, CPL_LESS_THAN, and CPL_NOT_LESS_THAN. If the table has no rows, no error is set, and 0 is returned. See also the function cpl_table_or_selected_long_long().

4.46.3.14 cpl_table_and_selected_string()

```
cpl_size cpl_table_and_selected_string (
```

```
            cpl_table * table,
            const char * name,
            cpl_table_select_operator operator,
            const char * string )
```

Select from selected table rows, by comparing *string* column values with a character string.

Parameters

table	Pointer to table.
name	Column name.
operator	Relational operator.
string	Reference character string.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_STRING`.
CPL_ERROR_ILLEGAL_INPUT	Invalid regular expression.

For all the already selected table rows, the values of the specified column are compared with the reference string. The comparison function used is the C standard `strcmp()`, but in case the relational operators `CPL_EQUAL_TO` or `CPL_NOT_EQUAL_TO` are specified, the comparison string is treated as a regular expression. All table rows not fulfilling the comparison are unselected. An invalid element never fulfills any comparison by definition. Allowed relational operators are `CPL_EQUAL_TO`, `CPL_NOT_EQUAL_TO`, `CPL_GREATER_THAN`, `CPL_NOT_GREATER_THAN`, `CPL_LESS_THAN`, and `CPL_NOT_LESS_THAN`. If the table has no rows, no error is set, and 0 is returned. See also the function `cpl_table_or_selected_string()`.

4.46.3.15 cpl_table_and_selected_window()

```
cpl_size cpl_table_and_selected_window (
            cpl_table * table,
            cpl_size start,
            cpl_size count )
```

Select from selected rows only those within a table segment.

Parameters

table	Pointer to table.
start	First row of table segment.
count	Length of segment.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* is a `NULL` pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *start* is outside the table boundaries.
CPL_ERROR_ILLEGAL_INPUT	*count* is negative.

All the selected table rows that are outside the specified interval are unselected. If the sum of *start* and *count* goes beyond the end of the input table, rows are checked up to the end of the table. See also the function `cpl_table_or_selected_window()`.

References cpl_calloc(), CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, CPL_ER↩ ROR_NULL_INPUT, and cpl_free().

4.46.3.16 cpl_table_arg_column()

```
cpl_error_code cpl_table_arg_column (
            cpl_table * table,
            const char * name )
```

Compute the phase angle value of table column elements.

Parameters

table	Pointer to table.
name	Column name.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is neither numerical nor complex.

Each column element is replaced by its phase angle value. The phase angle will be in the range of [-pi,pi]. Invalid elements are not modified by this operation. If the column is complex, its type will be

turned to real (CPL_TYPE_FLOAT_COMPLEX will be changed into CPL_TYPE_FLOAT, and CPL_TYP↩ E_DOUBLE_COMPLEX will be changed into CPL_TYPE_DOUBLE), and any pointer retrieved by calling cpl_table_get_data_float_complex(), cpl_table_get_data_double_complex(), etc., should be discarded.

4.46.3.17 cpl_table_cast_column()

```
cpl_error_code cpl_table_cast_column (
            cpl_table * table,
            const char * from_name,
            const char * to_name,
            cpl_type type )
```

Cast a numeric or complex column to a new numeric or complex type column.

Parameters

table	Pointer to table.
from_name	Name of table column to cast.
to_name	Name of new table column.
type	Type of new table column.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any of *table* or *from_name* is a NULL pointer.
CPL_ERROR_ILLEGAL_OUTPUT	A column with the specified *to_name* already exists in *table*. Note however that *to_name* equal to *from_name* is legal (in-place cast).
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *from_name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is neither numerical nor complex.
CPL_ERROR_ILLEGAL_INPUT	The specified *type* is neither numerical nor complex.

A new column of the specified type is created, and the content of the given numeric column is cast to the new type. If the input column type is identical to the specified type the column is duplicated as is done by the function cpl_table_duplicate_column(). Note that a column of arrays is always cast to another column of arrays of the specified type, unless it has depth 1. Consistently, a column of numbers can be cast to a column of arrays of depth 1. Here is a complete summary of how any (legal) *type* specification would be interpreted, depending on the type of the input column:

```
from_name type = CPL_TYPE_XXX
specified type = CPL_TYPE_XXX
to_name   type = CPL_TYPE_XXX
from_name type = CPL_TYPE_XXX | CPL_TYPE_POINTER
specified type = CPL_TYPE_XXX | CPL_TYPE_POINTER
to_name   type = CPL_TYPE_XXX | CPL_TYPE_POINTER
from_name type = CPL_TYPE_XXX | CPL_TYPE_POINTER (depth > 1)
specified type = CPL_TYPE_XXX
to_name   type = CPL_TYPE_XXX | CPL_TYPE_POINTER
from_name type = CPL_TYPE_XXX | CPL_TYPE_POINTER (depth = 1)
specified type = CPL_TYPE_XXX
to_name   type = CPL_TYPE_XXX
from_name type = CPL_TYPE_XXX
```

```
specified type = CPL_TYPE_XXX | CPL_TYPE_POINTER
to_name   type = CPL_TYPE_XXX | CPL_TYPE_POINTER (depth = 1)
from_name type = CPL_TYPE_XXX
specified type = CPL_TYPE_POINTER
to_name   type = CPL_TYPE_XXX | CPL_TYPE_POINTER (depth = 1)
from_name type = CPL_TYPE_XXX
specified type = CPL_TYPE_YYY
to_name   type = CPL_TYPE_YYY
from_name type = CPL_TYPE_XXX | CPL_TYPE_POINTER
specified type = CPL_TYPE_YYY | CPL_TYPE_POINTER
to_name   type = CPL_TYPE_YYY | CPL_TYPE_POINTER
from_name type = CPL_TYPE_XXX | CPL_TYPE_POINTER (depth > 1)
specified type = CPL_TYPE_YYY
to_name   type = CPL_TYPE_YYY | CPL_TYPE_POINTER
from_name type = CPL_TYPE_XXX | CPL_TYPE_POINTER (depth = 1)
specified type = CPL_TYPE_YYY
to_name   type = CPL_TYPE_YYY
from_name type = CPL_TYPE_XXX
specified type = CPL_TYPE_YYY | CPL_TYPE_POINTER
to_name   type = CPL_TYPE_YYY | CPL_TYPE_POINTER (depth = 1)
```

Note

If *to_name* is a NULL pointer, or it is equal to *from_name*, the cast is done in-place. The pointers to data will change, therefore pointers previously retrieved by `cpl_table_get_data_xxx()`, should be discarded.

Referenced by cpl_plot_column(), and cpl_plot_columns().

4.46.3.18 cpl_table_compare_structure()

```
int cpl_table_compare_structure (
              const cpl_table * table1,
              const cpl_table * table2 )
```

Compare the structure of two tables.

Parameters

table1	Pointer to a table.
table2	Pointer to another table.

Returns

0 if the tables have the same structure, 1 otherwise. In case of error, -1 is returned.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a `NULL` pointer.

Two tables have the same structure if they have the same number of columns, with the same names, the same types, and the same units. The order of the columns is not relevant.

References cpl_array_delete(), cpl_array_get_string(), cpl_table_get_column_depth(), cpl_table_get_column_↩ names(), cpl_table_get_column_type(), cpl_table_get_column_unit(), cpl_table_get_ncol(), and cpl_table_has_↩ column().

Referenced by cpl_table_insert().

4.46.3.19 cpl_table_conjugate_column()

```
cpl_error_code cpl_table_conjugate_column (
            cpl_table * table,
            const char * name )
```

Compute the complex conjugate of column values.

Parameters

table	Pointer to table.
name	Column name.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is neither numerical nor complex, or it is integer, or it is an array column.
CPL_ERROR_ILLEGAL_INPUT	The input *base* is not positive.

Each column element is replaced by its complex conjugate. The operation is always performed in double precision, with a final cast of the result to the target column type. Invalid elements are not modified by this operation.

4.46.3.20 cpl_table_copy_data_double()

```
cpl_error_code cpl_table_copy_data_double (
            cpl_table * table,
            const char * name,
            const double * data )
```

Copy existing data to a table *double* column.

Parameters

table	Pointer to table.
name	Name of the column.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_DOUBLE.

See the description of cpl_table_copy_data_int() for details.

4.46.3.21 cpl_table_copy_data_double_complex()

```
cpl_error_code cpl_table_copy_data_double_complex (
            cpl_table * table,
            const char * name,
            const double complex * data )
```

Copy existing data to a table *double* complex column.

Parameters

table	Pointer to table.
name	Name of the column.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_DOUBLE_COMPLEX.

See the description of cpl_table_copy_data_int() for details.

4.46.3.22 cpl_table_copy_data_float()

```
cpl_error_code cpl_table_copy_data_float (
            cpl_table * table,
```

```
            const char * name,
            const float * data )
```

Copy existing data to a table *float* column.

Parameters

table	Pointer to table.
name	Name of the column.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_FLOAT.

See the description of cpl_table_copy_data_int() for details.

4.46.3.23 cpl_table_copy_data_float_complex()

```
cpl_error_code cpl_table_copy_data_float_complex (
            cpl_table * table,
            const char * name,
            const float complex * data )
```

Copy existing data to a table *float* complex column.

Parameters

table	Pointer to table.
name	Name of the column.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_FLOAT_COM↩ PLEX.

See the description of cpl_table_copy_data_int() for details.

4.46.3.24 cpl_table_copy_data_int()

```
cpl_error_code cpl_table_copy_data_int (
            cpl_table * table,
            const char * name,
            const int * data )
```

Copy existing data to a table *integer* column.

Parameters

table	Pointer to table.
name	Name of the column.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_INT.

The input data values are copied to the specified column. The size of the input array is not checked in any way, and it is expected to be compatible with the number of rows in the given table. The copied data values are all taken as valid: invalid values should be marked using the functions cpl_table_set_invalid() and cpl_table_set_column_invalid().

4.46.3.25 cpl_table_copy_data_long()

```
cpl_error_code cpl_table_copy_data_long (
            cpl_table * table,
            const char * name,
            const long * data )
```

Copy existing data to a table *long* column.

Parameters

table	Pointer to table.
name	Name of the column.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_LONG.

See the description of cpl_table_copy_data_int() for details.

4.46.3.26 cpl_table_copy_data_long_long()

```
cpl_error_code cpl_table_copy_data_long_long (
            cpl_table * table,
            const char * name,
            const long long * data )
```

Copy existing data to a table *long long* column.

Parameters

table	Pointer to table.
name	Name of the column.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_LONG_LONG.

See the description of cpl_table_copy_data_int() for details.

4.46.3.27 cpl_table_copy_data_string()

```
cpl_error_code cpl_table_copy_data_string (
            cpl_table * table,
```

```
            const char * name,
            const char ** data )
```

Copy existing data to a table *string* column.

Parameters

table	Pointer to table.
name	Name of the column.
data	Existing data buffer.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_STRING`.

See the description of `cpl_table_copy_data_int()` for details. In the particular case of a string column, it should be noted that the data are copied in-depth, i.e., also the pointed strings are duplicated. Strings contained in the existing table column are deallocated before being replaced by the new ones.

4.46.3.28 cpl_table_copy_structure()

```
cpl_error_code cpl_table_copy_structure (
            cpl_table * table,
            const cpl_table * mtable )
```

Give to a table the same structure of another table.

Parameters

table	Pointer to empty table.
mtable	Pointer to model table.

Returns

`CPL_ERROR_NONE` in case of success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a `NULL` pointer.
CPL_ERROR_ILLEGAL_INPUT	*table* contains columns.

This function assignes to a columnless table the same column structure (names, types, units) of a given model table. All columns are physically created in the new table, and they are initialised to contain just invalid elements.

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_table_get↩ _ncol(), cpl_table_new_column(), cpl_table_new_column_array(), cpl_table_set_column_format(), cpl_table_set↩ _column_unit(), and CPL_TYPE_POINTER.

Referenced by cpl_table_extract_selected().

4.46.3.29 cpl_table_count_invalid()

```
cpl_size cpl_table_count_invalid (
            const cpl_table * table,
            const char * name )
```

Count number of invalid values in a table column.

Parameters

table	Pointer to table.
name	Name of table column to examine.

Returns

Number of invalid elements in a table column, or -1 in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.

Count number of invalid elements in a table column.

Referenced by cpl_plot_column().

4.46.3.30 cpl_table_count_selected()

```
cpl_size cpl_table_count_selected (
            const cpl_table * table )
```

Get number of selected rows in given table.

Parameters

table	Pointer to table.

Returns

Number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* is a `NULL` pointer.

Get number of selected rows in given table.

References CPL_ERROR_NULL_INPUT.

4.46.3.31 cpl_table_delete()

```
void cpl_table_delete (
            cpl_table * table )
```

Delete a table.

Parameters

table	Pointer to table to be deleted.

Returns

Nothing.

This function deletes a table, releasing all the memory associated to it, including any existing column. If *table* is `NULL`, nothing is done, and no error is set.

References cpl_free(), and cpl_table_get_ncol().

Referenced by cpl_plot_column().

4.46.3.32 cpl_table_divide_columns()

```
cpl_error_code cpl_table_divide_columns (
            cpl_table * table,
            const char * to_name,
            const char * from_name )
```

Divide two numeric or complex table columns.

Parameters

table	Pointer to table.
to_name	Name of target column.
from_name	Name of column dividing the target column.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or any column name are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with any specified name is not found in *table*.
CPL_ERROR_INVALID_TYPE	Any specified column is neither numerical nor complex, or it is an array column.

The columns are divided element by element, and the result of the division is stored in the target column. The columns' types may differ, and in that case the operation would be performed using the standard C upcasting rules, with a final cast of the result to the target column type. Invalid elements are propagated consistently: if either or both members of the division are invalid, the result will be invalid too. Underflows and overflows are ignored, but a division by exactly zero will set an invalid column element.

4.46.3.33 cpl_table_divide_scalar()

```
cpl_error_code cpl_table_divide_scalar (
            cpl_table * table,
            const char * name,
            double value )
```

Divide a numerical or complex column by a constant.

Parameters

table	Pointer to table.
name	Column name.
value	Divisor value.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is neither numerical nor complex, or it is an array column.
CPL_ERROR_DIVISION_BY_ZERO	The input *value* is 0.0.

The operation is always performed in double precision, with a final cast of the result to the target column type. Invalid elements are not modified by this operation.

4.46.3.34 cpl_table_divide_scalar_complex()

```
cpl_error_code cpl_table_divide_scalar_complex (
            cpl_table * table,
            const char * name,
            double complex value )
```

Divide a numerical or complex column by a complex constant.

Parameters

table	Pointer to table.
name	Column name.
value	Divisor value.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are NULL pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is neither numerical nor complex, or it is an array column.
CPL_ERROR_DIVISION_BY_ZERO	The input *value* is 0.0.

The operation is always performed in double precision, with a final cast of the result to the target column type. Invalid elements are not modified by this operation.

4.46.3.35 cpl_table_dump()

```
void cpl_table_dump (
            const cpl_table * table,
            cpl_size start,
            cpl_size count,
            FILE * stream )
```

Print a table.

Parameters

table	Pointer to table
start	First row to print
count	Number of rows to print
stream	The output stream

Returns

Nothing.

This function is mainly intended for debug purposes. All column elements are printed according to the column formats, that may be specified for each table column with the function `cpl_table_set_column_format()`. The default column formats have been chosen to provide a reasonable printout in most cases. Table rows are counted from 0, and their sequence number is printed at the left of each row. Invalid table elements are represented as a sequence of "-" as wide as the field occupied by the column to which they belong. Array elements are not resolved, and are represented by a sequence of "+" as wide as the field occupied by the column to which they belong. It is not shown whether a table row is selected or not. Specifying a *start* beyond the table boundaries, or a non-positive *count*, would generate a warning message, but no error would be set. The specified number of rows to print may exceed the table end, and in that case the table would be printed up to its last row. If the specified stream is `NULL`, it is set to *stdout*. The function used for printing is the standard C `fprintf()`.

References cpl_calloc(), cpl_malloc(), CPL_SIZE_FORMAT, and cpl_table_get_ncol().

4.46.3.36 cpl_table_dump_structure()

```
void cpl_table_dump_structure (
             const cpl_table * table,
             FILE * stream )
```

Describe the structure and the contents of a table.

Parameters

table	Pointer to table.
stream	The output stream

Returns

Nothing.

This function is mainly intended for debug purposes. Some information about the structure of a table and its contents is printed to terminal:

- Number of columns, with their names and types
- Number of invalid elements for each column
- Number of rows and of selected rows

If the specified stream is NULL, it is set to *stdout*. The function used for printing is the standard C fprintf().

References CPL_SIZE_FORMAT, and cpl_table_get_ncol().

4.46.3.37 cpl_table_duplicate()

```
cpl_table* cpl_table_duplicate (
            const cpl_table * table )
```

Make a copy of a table.

Parameters

table	Pointer to table.

Returns

Pointer to the new table, or NULL in case of NULL input, or in case of error.

The copy operation is done "in depth": columns data are duplicated too, not just their pointers. Also the selection flags of the original table are transferred to the new table.

References cpl_malloc(), cpl_table_get_ncol(), cpl_table_get_nrow(), and cpl_table_new().

Referenced by cpl_table_extract_selected().

4.46.3.38 cpl_table_duplicate_column()

```
cpl_error_code cpl_table_duplicate_column (
            cpl_table * to_table,
            const char * to_name,
            const cpl_table * from_table,
            const char * from_name )
```

Copy a column from a table to another.

Parameters

to_table	Target table.
to_name	New name of copied column.
from_table	Source table.
from_name	Name of column to copy.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a `NULL` pointer.
CPL_ERROR_INCOMPATIBLE_INPUT	The input tables do not have the same number of rows.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *from_name* is not found in the source table.
CPL_ERROR_ILLEGAL_OUTPUT	A column with the specified *to_name* already exists in the target table.

Copy a column from a table to another. The column is duplicated. A column may be duplicated also within the same table.

References CPL_ERROR_INCOMPATIBLE_INPUT, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_plot_column(), and cpl_plot_columns().

4.46.3.39 cpl_table_erase_column()

```
cpl_error_code cpl_table_erase_column (
            cpl_table * table,
            const char * name )
```

Delete a column from a table.

Parameters

table	Pointer to table.
name	Name of table column to delete.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.

Delete a column from a table. If the table is left without columns, also the selection flags are lost.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_table_erase_invalid(), and cpl_table_erase_invalid_rows().

4.46.3.40 cpl_table_erase_invalid()

```
cpl_error_code cpl_table_erase_invalid (
            cpl_table * table )
```

Remove from a table all columns just containing invalid elements, and then all rows containing at least one invalid element.

Parameters

table	Pointer to table.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* is a `NULL` pointer.

Firstly, all columns consisting just of invalid elements are deleted from the table. Next, the remaining table rows containing at least one invalid element are also deleted from the table. The selection flags are set back to "all selected" even if no rows or columns are erased. The pointers to data may change, therefore pointers previously retrieved by calling `cpl_table_get_data_int()`, etc., should be discarded.

The function is similar to the function cpl_table_erase_invalid_rows(), except for the criteria to remove rows containing invalid elements after all invalid columns have been removed. While cpl_table_erase_invalid_rows() requires all elements to be invalid in order to remove a row from the table, this function requires only one (or more) elements to be invalid.

Note

If the input table just contains invalid elements, all columns are deleted.

See also

cpl_table_erase_invalid_rows()

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_table_erase_column(), cpl_table_erase_↩ window(), cpl_table_get_ncol(), cpl_table_get_nrow(), and cpl_table_select_all().

Referenced by cpl_plot_column().

4.46.3.41 cpl_table_erase_invalid_rows()

```
cpl_error_code cpl_table_erase_invalid_rows (
            cpl_table * table )
```

Remove from a table columns and rows just containing invalid elements.

Parameters

table	Pointer to table.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* is a `NULL` pointer.

Table columns and table rows just containing invalid elements are deleted from the table, i.e. a column or a row is deleted only if all of its elements are invalid. The selection flags are set back to "all selected" even if no rows or columns are removed. The pointers to data may change, therefore pointers previously retrieved by `cpl_table_get_data_int()`, `cpl_table_get_data_string()`, etc., should be discarded.

Note

If the input table just contains invalid elements, all columns are deleted.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_table_erase_column(), cpl_table_erase_↩ window(), cpl_table_get_ncol(), cpl_table_get_nrow(), and cpl_table_select_all().

4.46.3.42 cpl_table_erase_selected()

```
cpl_error_code cpl_table_erase_selected (
            cpl_table * table )
```

Delete the selected rows of a table.

Parameters

table	Pointer to table

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	*table* is a `NULL` pointer.

A portion of the table data is physically removed. The pointer to column data may change, therefore pointers previously retrieved by calling cpl_table_get_data_int(), cpl_table_get_data_string(), etc., should be discarded. The table selection flags are set back to "all selected".

References CPL_ERROR_NULL_INPUT, cpl_table_get_ncol(), cpl_table_get_nrow(), cpl_table_select_all(), and cpl_table_set_size().

4.46.3.43 cpl_table_erase_window()

```
cpl_error_code cpl_table_erase_window (
            cpl_table * table,
            cpl_size start,
            cpl_size count )
```

Delete a table segment.

Parameters

table	Pointer to table.
start	First row to delete.
count	Number of rows to delete.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	*table* is a NULL pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input table has length zero, or *start* is outside the table range.
CPL_ERROR_ILLEGAL_INPUT	*count* is negative.

A portion of the table data is physically removed. The pointers to column data may change, therefore pointers previously retrieved by calling cpl_table_get_data_int(), cpl_table_get_data_string(), etc., should be discarded. The table selection flags are set back to "all selected". The specified segment can extend beyond the end of the table, and in that case rows will be removed up to the end of the table.

References CPL_ERROR_NULL_INPUT, cpl_table_get_ncol(), and cpl_table_select_all().

Referenced by cpl_table_erase_invalid(), and cpl_table_erase_invalid_rows().

4.46.3.44 cpl_table_exponential_column()

```
cpl_error_code cpl_table_exponential_column (
            cpl_table * table,
            const char * name,
            double base )
```

Compute the exponential of column values.

Parameters

table	Pointer to table.
name	Column name.
base	Exponential base.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is neither numerical nor complex, or it is an array column.
CPL_ERROR_ILLEGAL_INPUT	The input *base* is not positive.

Each column element is replaced by its exponential in the specified base. The operation is always performed in double precision, with a final cast of the result to the target column type. Invalid elements are not modified by this operation.

4.46.3.45 cpl_table_extract()

```
cpl_table* cpl_table_extract (
            const cpl_table * table,
            cpl_size start,
            cpl_size count )
```

Create a table from a section of another table.

Parameters

table	Pointer to table.
start	First row to be copied to new table.
count	Number of rows to be copied.

Returns

Pointer to the new table, or NULL in case or error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* is a NULL pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *start* is outside the table boundaries.
CPL_ERROR_ILLEGAL_INPUT	*count* is negative.

A number of consecutive rows are copied from an input table to a newly created table. The new table will have the same structure of the original table (see function cpl_table_compare_structure()). If the sum of *start* and *count* goes beyond the end of the input table, rows are copied up to the end. All the rows of the new table are selected, i.e., existing selection flags are not transferred from the old table to the new one.

References CPL_ERROR_NULL_INPUT, cpl_table_get_ncol(), cpl_table_get_nrow(), and cpl_table_new().

4.46.3.46 cpl_table_extract_selected()

```
cpl_table* cpl_table_extract_selected (
             const cpl_table * table )
```

Create a new table from the selected rows of another table.

Parameters

table	Pointer to table.

Returns

Pointer to new table, or NULL in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* is a NULL pointer.

A new table is created, containing a copy of all the selected rows of the input table. In the output table all rows are selected.

References CPL_ERROR_NULL_INPUT, cpl_table_copy_structure(), cpl_table_duplicate(), and cpl_table_new().

4.46.3.47 cpl_table_fill_column_window()

```
cpl_error_code cpl_table_fill_column_window (
            cpl_table * table,
            const char * name,
            cpl_size start,
            cpl_size count,
            double value )
```

Write a value to a numerical column segment.

Parameters

table	Pointer to table.
name	Name of table column to access.
start	Position where to begin to write the value.
count	Number of values to write.
value	Value to write.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are `NULL` pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *start* is outside the table boundaries.
CPL_ERROR_ILLEGAL_INPUT	*count* is negative.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is not numerical, or is of type *array*.

Write the same value to a numerical column segment. The value is cast to the type of the accessed column according to the C casting rules. The written values are automatically marked as valid. To invalidate a column interval use `cpl_table_set_column_invalid()` instead. If the sum of *start* and *count* exceeds the number of table rows, the column is filled up to its end.

4.46.3.48 cpl_table_fill_column_window_array()

```
cpl_error_code cpl_table_fill_column_window_array (
            cpl_table * table,
            const char * name,
            cpl_size start,
            cpl_size count,
            const cpl_array * array )
```

Write an array to an *array* column segment.

Parameters

table	Pointer to table.
name	Name of table column to access.
start	Position where to begin to write the array.
count	Number of arrays to write.
array	Array to write.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are `NULL` pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *start* is outside the table boundaries.
CPL_ERROR_ILLEGAL_INPUT	*count* is negative.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column does not match the type of the input *array*, or it is not made of arrays.
CPL_ERROR_INCOMPATIBLE_INPUT	The size of the input *array* is different from the depth of the specified column.

Write the same array to a segment of an array column. If the input array is not a `NULL` pointer, it is duplicated for each accessed column element. If the input array is a `NULL` pointer, this call is equivalent to a call to `cpl_table_set_column_invalid()`. If the sum of *start* and *count* exceeds the number of rows in the table, the column is filled up to its end.

4.46.3.49 cpl_table_fill_column_window_complex()

```
cpl_error_code cpl_table_fill_column_window_complex (
            cpl_table * table,
            const char * name,
            cpl_size start,
            cpl_size count,
            double complex value )
```

Write a value to a complex column segment.

Parameters

table	Pointer to table.
name	Name of table column to access.
start	Position where to begin to write the value.
count	Number of values to write.
value	Value to write.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are NULL pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *start* is outside the table boundaries.
CPL_ERROR_ILLEGAL_INPUT	*count* is negative.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is not complex, or is of type *array*.

Write the same value to a complex column segment. The value is cast to the type of the accessed column according to the C casting rules. The written values are automatically marked as valid. To invalidate a column interval use cpl_table_set_column_invalid() instead. If the sum of *start* and *count* exceeds the number of table rows, the column is filled up to its end.

4.46.3.50 cpl_table_fill_column_window_double()

```
cpl_error_code cpl_table_fill_column_window_double (
            cpl_table * table,
            const char * name,
            cpl_size start,
            cpl_size count,
            double value )
```

Write a value to a *double* column segment.

Parameters

table	Pointer to table.
name	Name of table column to access.
start	Position where to begin to write the value.
count	Number of values to write.
value	Value to write.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are NULL pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *start* is outside the table boundaries.
CPL_ERROR_ILLEGAL_INPUT	*count* is negative.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_DOUBLE.

Write the same value to a *double* column segment. The written values are automatically marked as valid. To invalidate a column interval use cpl_table_set_column_invalid() instead. If the sum of *start* and *count* exceeds the number of table rows, the column is filled up to its end.

4.46.3.51 cpl_table_fill_column_window_double_complex()

```
cpl_error_code cpl_table_fill_column_window_double_complex (
            cpl_table * table,
            const char * name,
            cpl_size start,
            cpl_size count,
            double complex value )
```

Write a value to a *double* complex column segment.

Parameters

table	Pointer to table.
name	Name of table column to access.
start	Position where to begin to write the value.
count	Number of values to write.
value	Value to write.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are NULL pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *start* is outside the table boundaries.
CPL_ERROR_ILLEGAL_INPUT	*count* is negative.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_DOUBLE_COMPLEX.

Write the same value to a *double* complex column segment. The written values are automatically marked as valid. To invalidate a column interval use cpl_table_set_column_invalid() instead. If the sum of *start* and *count* exceeds the number of table rows, the column is filled up to its end.

4.46.3.52 cpl_table_fill_column_window_float()

```
cpl_error_code cpl_table_fill_column_window_float (
            cpl_table * table,
```

```
            const char * name,
            cpl_size start,
            cpl_size count,
            float value )
```

Write a value to a *float* column segment.

Parameters

table	Pointer to table.
name	Name of table column to access.
start	Position where to begin to write the value.
count	Number of values to write.
value	Value to write.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are `NULL` pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *start* is outside the table boundaries.
CPL_ERROR_ILLEGAL_INPUT	*count* is negative.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_FLOAT`.

Write the same value to a *float* column segment. The written values are automatically marked as valid. To invalidate a column interval use `cpl_table_set_column_invalid()` instead. If the sum of *start* and *count* exceeds the number of table rows, the column is filled up to its end.

4.46.3.53 cpl_table_fill_column_window_float_complex()

```
cpl_error_code cpl_table_fill_column_window_float_complex (
            cpl_table * table,
            const char * name,
            cpl_size start,
            cpl_size count,
            float complex value )
```

Write a value to a *float* complex column segment.

Parameters

table	Pointer to table.
name	Name of table column to access.
start	Position where to begin to write the value.
count	Number of values to write.
value	Value to write.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are NULL pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *start* is outside the table boundaries.
CPL_ERROR_ILLEGAL_INPUT	*count* is negative.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_FLOA↩ T_COMPLEX.

Write the same value to a *float* complex column segment. The written values are automatically marked as valid. To invalidate a column interval use cpl_table_set_column_invalid() instead. If the sum of *start* and *count* exceeds the number of table rows, the column is filled up to its end.

4.46.3.54 cpl_table_fill_column_window_int()

```
cpl_error_code cpl_table_fill_column_window_int (
            cpl_table * table,
            const char * name,
            cpl_size start,
            cpl_size count,
            int value )
```

Write a value to an *integer* column segment.

Parameters

table	Pointer to table.
name	Name of table column to access.
start	Position where to begin to write the value.
count	Number of values to write.
value	Value to write.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are NULL pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *start* is outside the table boundaries.
CPL_ERROR_ILLEGAL_INPUT	*count* is negative.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_INT.

Write the same value to an *integer* column segment. The written values are automatically marked as valid. To invalidate a column interval use cpl_table_set_column_invalid() instead. If the sum of *start* and *count* exceeds the number of table rows, the column is filled up to its end.

Note

For automatic conversion to the accessed column type use the function cpl_table_fill_column_window().

4.46.3.55 cpl_table_fill_column_window_long()

```
cpl_error_code cpl_table_fill_column_window_long (
            cpl_table * table,
            const char * name,
            cpl_size start,
            cpl_size count,
            long value )
```

Write a value to an *long* column segment.

Parameters

table	Pointer to table.
name	Name of table column to access.
start	Position where to begin to write the value.
count	Number of values to write.
value	Value to write.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are NULL pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *start* is outside the table boundaries.
CPL_ERROR_ILLEGAL_INPUT	*count* is negative.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_LONG.

Write the same value to an *long* column segment. The written values are automatically marked as valid. To invalidate a column interval use cpl_table_set_column_invalid() instead. If the sum of *start* and *count* exceeds the number of table rows, the column is filled up to its end.

Note

For automatic conversion to the accessed column type use the function cpl_table_fill_column_window().

4.46.3.56 cpl_table_fill_column_window_long_long()

```
cpl_error_code cpl_table_fill_column_window_long_long (
            cpl_table * table,
            const char * name,
            cpl_size start,
            cpl_size count,
            long long value )
```

Write a value to an *long long* column segment.

Parameters

table	Pointer to table.
name	Name of table column to access.
start	Position where to begin to write the value.
count	Number of values to write.
value	Value to write.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are NULL pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *start* is outside the table boundaries.
CPL_ERROR_ILLEGAL_INPUT	*count* is negative.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_LONG↩ _LONG.

Write the same value to an *long long* column segment. The written values are automatically marked as valid. To invalidate a column interval use cpl_table_set_column_invalid() instead. If the sum of *start* and *count* exceeds the number of table rows, the column is filled up to its end.

Note

For automatic conversion to the accessed column type use the function cpl_table_fill_column_window().

4.46.3.57 cpl_table_fill_column_window_string()

```
cpl_error_code cpl_table_fill_column_window_string (
            cpl_table * table,
            const char * name,
            cpl_size start,
            cpl_size count,
            const char * value )
```

Write a character string to a *string* column segment.

Parameters

table	Pointer to table.
name	Name of table column to access.
start	Position where to begin to write the character string.
count	Number of strings to write.
value	Character string to write.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are `NULL` pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *start* is outside the table boundaries.
CPL_ERROR_ILLEGAL_INPUT	*count* is negative.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_STRING`.

Write the same value to a *string* column segment. If the input string is not a `NULL` pointer, it is duplicated for each accessed column element. If the input string is a `NULL` pointer, this call is equivalent to a call to `cpl_table_set_column_invalid()`. If the sum of *start* and *count* exceeds the number of rows in the table, the column is filled up to its end.

4.46.3.58 cpl_table_fill_invalid_double()

```
cpl_error_code cpl_table_fill_invalid_double (
            cpl_table * table,
            const char * name,
            double code )
```

Write a numerical value to invalid *double* column elements.

Parameters

table	Pointer to table containing the column.
name	Column name.
code	Value to write to invalid column elements.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_DOUBLE`, or of type `CPL_TYPE_DOUBLE | CPL_TYPE_POINTER`.

See also

cpl_table_fill_invalid_int()

Note

Assigning a value to an invalid numerical element will not make it valid, but assigning a value to an element consisting of an array of numbers will make the array element valid.

4.46.3.59 cpl_table_fill_invalid_double_complex()

```
cpl_error_code cpl_table_fill_invalid_double_complex (
            cpl_table * table,
            const char * name,
            double complex code )
```

Write a numerical value to invalid *double* complex column elements.

Parameters

table	Pointer to table containing the column.
name	Column name.
code	Value to write to invalid column elements.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_DOUBLE_COMPLEX`, or of type `CPL_TYPE_DOUBLE_COMPLEX | CPL_TYPE_POINTER`.

See also

cpl_table_fill_invalid_int()

Note

Assigning a value to an invalid numerical element will not make it valid, but assigning a value to an element consisting of an array of numbers will make the array element valid.

4.46.3.60 cpl_table_fill_invalid_float()

```
cpl_error_code cpl_table_fill_invalid_float (
            cpl_table * table,
            const char * name,
            float code )
```

Write a numerical value to invalid *float* column elements.

Parameters

table	Pointer to table containing the column.
name	Column name.
code	Value to write to invalid column elements.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_FLOAT`, or of type `CPL_TYPE_FLOAT | CPL_TYPE_POINTER`.

See also

cpl_table_fill_invalid_int()

Note

Assigning a value to an invalid numerical element will not make it valid, but assigning a value to an element consisting of an array of numbers will make the array element valid.

4.46.3.61 cpl_table_fill_invalid_float_complex()

```
cpl_error_code cpl_table_fill_invalid_float_complex (
            cpl_table * table,
            const char * name,
            float complex code )
```

Write a numerical value to invalid *float* complex column elements.

Parameters

table	Pointer to table containing the column.
name	Column name.
code	Value to write to invalid column elements.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_FLOAT_COMPLEX`, or of type `CPL_TYPE_FLOAT_COMPLEX | CPL_TYPE_POINTER`.

See also

cpl_table_fill_invalid_int()

Note

Assigning a value to an invalid numerical element will not make it valid, but assigning a value to an element consisting of an array of numbers will make the array element valid.

4.46.3.62 cpl_table_fill_invalid_int()

```
cpl_error_code cpl_table_fill_invalid_int (
            cpl_table * table,
            const char * name,
            int code )
```

Write a numerical value to invalid *integer* column elements.

Parameters

table	Pointer to table containing the column.
name	Column name.
code	Value to write to invalid column elements.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_INT`, or of type `CPL_TYPE_INT | CPL_TYPE_POINTER`.

In general, a numeric column element that is flagged as invalid is undefined and should not be read. It is however sometimes convenient to read such values, f.ex. via calls to `cpl_table_get_data_int()` and memcpy(). In order to avoid that such usage causes uninitialized memory to be read, the invalid elements may be set to a value specified by a call to this function. Note that only existing invalid elements will be filled as indicated: new invalid column elements would still have their actual values left undefined. Also, any further processing of the column would not take care of maintaining the assigned value to a given invalid column element: therefore the value should be applied just before it is actually needed. An invalid numerical column element remains invalid after this call, and the usual method of checking whether the element is invalid or not should still be used. This function can be applied also to columns of arrays of integers. In this case the call will cause the array element to be flagged as valid.

Note

Assigning a value to an invalid numerical element will not make it valid, but assigning a value to an element consisting of an array of numbers will make the array element valid.

4.46.3.63 cpl_table_fill_invalid_long()

```
cpl_error_code cpl_table_fill_invalid_long (
            cpl_table * table,
            const char * name,
            long code )
```

Write a numerical value to invalid *long* column elements.

Parameters

table	Pointer to table containing the column.
name	Column name.
code	Value to write to invalid column elements.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_LONG`, or of type `CPL_TYPE_LONG | CPL_TYPE_POINTER`.

See also

cpl_table_fill_invalid_int()

Note

Assigning a value to an invalid numerical element will not make it valid, but assigning a value to an element consisting of an array of numbers will make the array element valid.

4.46.3.64 cpl_table_fill_invalid_long_long()

```
cpl_error_code cpl_table_fill_invalid_long_long (
            cpl_table * table,
            const char * name,
            long long code )
```

Write a numerical value to invalid *long long* column elements.

Parameters

table	Pointer to table containing the column.
name	Column name.
code	Value to write to invalid column elements.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_LONG_LONG`, or of type `CPL_TYPE_LONG_LONG | CPL_TYPE_POINTER`.

See also

cpl_table_fill_invalid_int()

Note

Assigning a value to an invalid numerical element will not make it valid, but assigning a value to an element consisting of an array of numbers will make the array element valid.

4.46.3.65 cpl_table_get()

```
double cpl_table_get (
            const cpl_table * table,
            const char * name,
            cpl_size row,
            int * null )
```

Read a value from a numerical column.

Parameters

table	Pointer to table.
name	Name of table column to be accessed.
row	Position of element to be read.
null	Flag indicating *null* values, or error condition.

Returns

Value read. In case of invalid table element, or in case of error, 0.0 is returned.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are `NULL` pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is not numerical, or is a column of arrays.

Rows are counted starting from 0. The *null* flag is used to indicate whether the accessed table element is valid (0) or invalid (1). The *null* flag also signals an error condition (-1). The *null* argument can be left to NULL.

4.46.3.66 cpl_table_get_array()

```
const cpl_array* cpl_table_get_array (
            const cpl_table * table,
            const char * name,
            cpl_size row )
```

Read an array from an *array* column.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Position of element to be read.

Returns

Pointer to array. In case of an invalid column element, or in case of error, a NULL pointer is always returned.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are NULL pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type *array*.

Read a value from a column of any array type. Rows are counted starting from 0.

Note

The returned array is a pointer to a table element, not its copy. Its manipulation will directly affect that element, while changing that element using cpl_table_set_array() will turn it into garbage. Therefore, if a real copy of an array column element is required, this function should be called as an argument of the function cpl_array_duplicate().

4.46.3.67 cpl_table_get_column_depth()

```
cpl_size cpl_table_get_column_depth (
            const cpl_table * table,
            const char * name )
```

Get the depth of a table column.

Parameters

table	Pointer to table.
name	Column name.

Returns

Column depth, or -1 in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.

Get the depth of a column. Columns of type *array* always have positive depth, while columns listing numbers or character strings have depth 0.

Referenced by cpl_table_compare_structure().

4.46.3.68 cpl_table_get_column_dimension()

```
cpl_size cpl_table_get_column_dimension (
            const cpl_table * table,
            const char * name,
            cpl_size indx )
```

Get size of one dimension of a table column of arrays.

Parameters

table	Pointer to table.
name	Column name.
indx	Indicate dimension to query (0 = x, 1 = y, 2 = z, etc.).

Returns

Size of queried dimension of the column, or zero in case of error.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_UNSUPPORTED_MODE	The specified column is not of type *array*.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The specified *indx* array is not compatible with the column dimensions.
CPL_ERROR_INCOMPATIBLE_INPUT	The specified dimensions are incompatible with the total number of elements in the column arrays.

Get the size of one dimension of a column. If a column is not an array column, or if it has no dimensions, 1 is returned.

4.46.3.69 cpl_table_get_column_dimensions()

```
cpl_size cpl_table_get_column_dimensions (
            const cpl_table * table,
            const char * name )
```

Get the number of dimensions of a table column of arrays.

Parameters

table	Pointer to table.
name	Column name.

Returns

Column number of dimensions, or 0 in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.

Get the number of dimensions of a column. If a column is not an array column, or if it has no dimensions, 1 is returned.

4.46.3.70 cpl_table_get_column_format()

```
const char* cpl_table_get_column_format (
            const cpl_table * table,
            const char * name )
```

Get the format of a table column.

Parameters

table	Pointer to table.
name	Column name.

Returns

Format of column, or `NULL` in case of error.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.

Return the format of a column. Note that the returned string is a pointer to the column format, not its copy. Its manipulation will directly affect the column format, while changing the column format using `cpl_column_set_format()` will turn it into garbage. Therefore it should be considered read-only, and if a real copy of a column format is required, this function should be called as an argument of the function `strdup()`.

4.46.3.71 cpl_table_get_column_max()

```
double cpl_table_get_column_max (
            const cpl_table * table,
            const char * name )
```

Get maximum value in a numerical column.

Parameters

table	Pointer to table.
name	Column name.

Returns

Maximum value. See documentation of `cpl_table_get_column_mean()`.

See the description of the function `cpl_table_get_column_mean()`.

References cpl_errorstate_get().

4.46.3.72 cpl_table_get_column_maxpos()

```
cpl_error_code cpl_table_get_column_maxpos (
            const cpl_table * table,
            const char * name,
            cpl_size * row )
```

Get position of maximum in a numerical column.

Parameters

table	Pointer to table.
name	Column name.
row	Returned position of maximum value.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are NULL pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*, or it just contains invalid elements, or the table has length zero.
CPL_ERROR_INVALID_TYPE	The specified column is not numerical.

Invalid column values are excluded from the search. The *row* argument will be assigned the position of the maximum value, where rows are counted starting from 0. If more than one column element correspond to the max value, the position with the lowest row number is returned. In case of error, *row* is left untouched. The table selection flags have no influence on the result.

4.46.3.73 cpl_table_get_column_mean()

```
double cpl_table_get_column_mean (
             const cpl_table * table,
             const char * name )
```

Compute the mean value of a numerical column.

Parameters

table	Pointer to table.
name	Column name.

Returns

Mean value. In case of error 0.0 is returned.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are NULL pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*, or it just contains invalid elements, or the table has length zero.
CPL_ERROR_INVALID_TYPE	The specified column is not numerical.

Invalid column values are excluded from the computation. The table selection flags have no influence on the result.

References cpl_errorstate_get().

4.46.3.74 cpl_table_get_column_mean_complex()

```
double complex cpl_table_get_column_mean_complex (
            const cpl_table * table,
            const char * name )
```

Compute the mean value of a numerical or complex column.

Parameters

table	Pointer to table.
name	Column name.

Returns

Mean value. In case of error 0.0 is returned.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*, or it just contains invalid elements, or the table has length zero.
CPL_ERROR_INVALID_TYPE	The specified column is neither numerical nor complex.

Invalid column values are excluded from the computation. The table selection flags have no influence on the result.

References cpl_errorstate_get().

4.46.3.75 cpl_table_get_column_median()

```
double cpl_table_get_column_median (
            const cpl_table * table,
            const char * name )
```

Compute the median value of a numerical column.

Parameters

table	Pointer to table.
name	Column name.

Returns

Median value. See documentation of `cpl_table_get_column_mean()`.

See the description of the function `cpl_table_get_column_mean()`.

4.46.3.76 cpl_table_get_column_min()

```
double cpl_table_get_column_min (
            const cpl_table * table,
            const char * name )
```

Get minimum value in a numerical column.

Parameters

table	Pointer to table.
name	Column name.

Returns

Minimum value. See documentation of cpl_table_get_column_mean().

See the description of the function cpl_table_get_column_mean().

References cpl_errorstate_get().

4.46.3.77 cpl_table_get_column_minpos()

```
cpl_error_code cpl_table_get_column_minpos (
            const cpl_table * table,
            const char * name,
            cpl_size * row )
```

Get position of minimum in a numerical column.

Parameters

table	Pointer to table.
name	Column name.
row	Returned position of minimum value.

Returns

See function cpl_table_get_column_maxpos().

See the description of the function cpl_table_get_column_maxpos().

4.46.3.78 cpl_table_get_column_name()

```
const char* cpl_table_get_column_name (
            const cpl_table * table )
```

Get table columns names.

Parameters

table	Pointer to table.

Returns

Name of a table column.

If this function is not called with a `NULL` pointer the name of the first table column will be returned. Further calls made with a `NULL` pointer would return the next columns names, till the end of the list of columns when a `NULL` would be returned. This function only guarantees that all the table column names would be returned by subsequent calls to this function, but the order in which the column names are returned is undefined. The table structure must not be modified (e.g. by deleting, creating, moving, or renaming columns) between a sequence of calls to `cpl_table_get_column_name()` related to the same table, or this function behaviour will be undetermined. This function returns a pointer to the table column name, and not to its copy, therefore the pointed string shouldn't be deallocated or manipulated in any way. Its manipulation would directly affect the column name, while changing the column name using `cpl_table_name_column()` would turn it into garbage. Therefore, if a real copy of a column name is required, this function should be called as an argument of the function `strdup()`.

Deprecated This function is deprecated, because its usage could create serious problems in case it is attempted to get names from different tables simultaneously. For instance, a programmer may call cpl_table_get_column_name() in a loop, and in the same loop call a CPL function that calls as well the same function. The behaviour in this case would be unpredictable. The function cpl_table_get_column_names() should be used instead.

References cpl_table_get_ncol().

4.46.3.79 cpl_table_get_column_names()

```
cpl_array* cpl_table_get_column_names (
             const cpl_table * table )
```

Get table columns names.

Parameters

table	Pointer to table.

Returns

Array of table columns names.

The returned CPL array of strings should be finally destroyed using cpl_array_delete().

References cpl_array_new(), cpl_array_set_string(), CPL_ERROR_NULL_INPUT, and CPL_TYPE_STRING.

Referenced by cpl_table_compare_structure().

4.46.3.80 cpl_table_get_column_stdev()

```
double cpl_table_get_column_stdev (
            const cpl_table * table,
            const char * name )
```

Find the standard deviation of a table column.

Parameters

table	Pointer to table.
name	Column name.

Returns

Standard deviation. See documentation of `cpl_table_get_column_mean()`.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*, or it just contains invalid elements, or the table has length zero.
CPL_ERROR_INVALID_TYPE	The specified column is not numerical.

Invalid column values are excluded from the computation of the standard deviation. If just one valid element is found, 0.0 is returned but no error is set. The table selection flags have no influence on the result.

References cpl_errorstate_get().

4.46.3.81 cpl_table_get_column_type()

```
cpl_type cpl_table_get_column_type (
            const cpl_table * table,
            const char * name )
```

Get the type of a table column.

Parameters

table	Pointer to table.
name	Column name.

Returns

Column type, or `CPL_TYPE_INVALID` in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.

Get the type of a column.

Referenced by cpl_plot_column(), cpl_plot_columns(), cpl_table_compare_structure(), and cpl_table_unwrap().

4.46.3.82 cpl_table_get_column_unit()

```
const char* cpl_table_get_column_unit (
             const cpl_table * table,
             const char * name )
```

Get the unit of a table column.

Parameters

table	Pointer to table.
name	Column name.

Returns

Unit of column, or `NULL` if no unit can be returned.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.

Return the unit of a column if present, otherwise a NULL pointer is returned. Note that the returned string is a pointer to the column unit, not its copy. Its manipulation will directly affect the column unit, while changing the column unit using `cpl_column_set_unit()` will turn it into garbage. Therefore it should be considered read-only, and if a real copy of a column unit is required, this function should be called as an argument of the function `strdup()`.

Referenced by cpl_table_compare_structure().

4.46.3.83 cpl_table_get_complex()

```
double complex cpl_table_get_complex (
             const cpl_table * table,
```

```
            const char * name,
            cpl_size row,
            int * null )
```

Read a value from a complex column.

Parameters

table	Pointer to table.
name	Name of table column to be accessed.
row	Position of element to be read.
null	Flag indicating *null* values, or error condition.

Returns

Value read. In case of invalid table element, or in case of error, 0.0 is returned.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are `NULL` pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is not complex, or is a column of arrays.

Rows are counted starting from 0. The *null* flag is used to indicate whether the accessed table element is valid (0) or invalid (1). The *null* flag also signals an error condition (-1). The *null* argument can be left to `NULL`.

4.46.3.84 cpl_table_get_data_array()

```
cpl_array** cpl_table_get_data_array (
            cpl_table * table,
            const char * name )
```

Get a pointer to *array* column data.

Parameters

table	Pointer to table.
name	Column name.

Returns

Pointer to column data, or `NULL` if the column has zero length, or in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type *array*.

A table column of type *array* includes an array of values of type *cpl_array*∗. This function returns a pointer to this array.

Note

Use at your own risk: direct manipulation of column data rules out any check performed by the table object interface, and may introduce inconsistencies between the information maintained internally, and the actual column data and structure.

4.46.3.85 cpl_table_get_data_array_const()

```
const cpl_array** cpl_table_get_data_array_const (
            const cpl_table * table,
            const char * name )
```

Get a pointer to *array* column data.

Parameters

table	Pointer to table.
name	Column name.

Returns

Pointer to column data, or NULL if the column has zero length, or in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type *array*.

A table column of type *array* includes an array of values of type *cpl_array*∗. This function returns a pointer to this array.

4.46.3.86 cpl_table_get_data_double()

```
double* cpl_table_get_data_double (
```

```
            cpl_table * table,
            const char * name )
```

Get a pointer to *double* column data.

Parameters

table	Pointer to table.
name	Column name.

Returns

Pointer to column data, or `NULL` if the column has zero length, or in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_DOUBLE`.

A *cpl_table* column of type `CPL_TYPE_DOUBLE` includes an array of values of type *double*. This function returns a pointer to this array. The data buffer elements corresponding to invalid column elements would in general contain garbage. To avoid this, `cpl_table_fill_invalid_double()` should be called just before this function, assigning to all the invalid column elements an *ad hoc* numerical value. See the description of function `cpl_table_fill_invalid_double()` for further details.

Note

Use at your own risk: direct manipulation of column data rules out any check performed by the table object interface, and may introduce inconsistencies between the information maintained internally, and the actual column data and structure.

Referenced by cpl_plot_column().

4.46.3.87 cpl_table_get_data_double_complex()

```
double complex* cpl_table_get_data_double_complex (
            cpl_table * table,
            const char * name )
```

Get a pointer to *double* complex column data.

Parameters

table	Pointer to table.
name	Column name.

Returns

Pointer to column data, or NULL if the column has zero length, or in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_DOUBLE_COMPLEX.

A *cpl_table* column of type CPL_TYPE_DOUBLE_COMPLEX includes an array of values of type *double* complex. This function returns a pointer to this array. The data buffer elements corresponding to invalid column elements would in general contain garbage. To avoid this, cpl_table_fill_invalid_double_complex() should be called just before this function, assigning to all the invalid column elements an *ad hoc* numerical value. See the description of function cpl_table_fill_invalid_double_complex() for further details.

Note

Use at your own risk: direct manipulation of column data rules out any check performed by the table object interface, and may introduce inconsistencies between the information maintained internally, and the actual column data and structure.

4.46.3.88 cpl_table_get_data_double_complex_const()

```
const double complex* cpl_table_get_data_double_complex_const (
             const cpl_table * table,
             const char * name )
```

Get a pointer to constant *double* complex column data.

Parameters

table	Pointer to constant table.
name	Column name.

Returns

Pointer to constant column data, or NULL if the column has zero length, or in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_DOUBLE_COMPLEX.

A *cpl_table* column of type CPL_TYPE_DOUBLE_COMPLEX includes an array of values of type *double* complex. This function returns a pointer to this array. The data buffer elements corresponding to invalid column elements would in general contain garbage. To avoid this, cpl_table_fill_invalid_double_complex() should be called just before this function, assigning to all the invalid column elements an *ad hoc* numerical value. See the description of function cpl_table_fill_invalid_double_complex() for further details.

4.46.3.89 cpl_table_get_data_double_const()

```
const double* cpl_table_get_data_double_const (
            const cpl_table * table,
            const char * name )
```

Get a pointer to constant *double* column data.

Parameters

table	Pointer to constant table.
name	Column name.

Returns

Pointer to constant column data, or NULL if the column has zero length, or in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_DOUBLE.

A *cpl_table* column of type CPL_TYPE_DOUBLE includes an array of values of type *double*. This function returns a pointer to this array. The data buffer elements corresponding to invalid column elements would in general contain garbage. To avoid this, cpl_table_fill_invalid_double() should be called just before this function, assigning to all the invalid column elements an *ad hoc* numerical value. See the description of function cpl_table_fill_invalid_double() for further details.

Referenced by cpl_plot_column(), and cpl_plot_columns().

4.46.3.90 cpl_table_get_data_float()

```
float* cpl_table_get_data_float (
            cpl_table * table,
            const char * name )
```

Get a pointer to *float* column data.

Parameters

table	Pointer to table.
name	Column name.

Returns

Pointer to column data, or NULL if the column has zero length, or in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_FLOAT.

A *cpl_table* column of type CPL_TYPE_FLOAT includes an array of values of type *float*. This function returns a pointer to this array. The data buffer elements corresponding to invalid column elements would in general contain garbage. To avoid this, cpl_table_fill_invalid_float() should be called just before this function, assigning to all the invalid column elements an *ad hoc* numerical value. See the description of function cpl_table_fill_invalid_float() for further details.

Note

Use at your own risk: direct manipulation of column data rules out any check performed by the table object interface, and may introduce inconsistencies between the information maintained internally, and the actual column data and structure.

4.46.3.91 cpl_table_get_data_float_complex()

```
float complex* cpl_table_get_data_float_complex (
             cpl_table * table,
             const char * name )
```

Get a pointer to *float* complex column data.

Parameters

table	Pointer to table.
name	Column name.

Returns

Pointer to column data, or NULL if the column has zero length, or in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_FLOAT_COM↩ PLEX.

A *cpl_table* column of type CPL_TYPE_FLOAT_COMPLEX includes an array of values of type *float* complex. This function returns a pointer to this array. The data buffer elements corresponding to invalid column elements would in general contain garbage. To avoid this, cpl_table_fill_invalid_float_complex() should be called just before this function, assigning to all the invalid column elements an *ad hoc* numerical value. See the description of function cpl_table_fill_invalid_float_complex() for further details.

Note

Use at your own risk: direct manipulation of column data rules out any check performed by the table object interface, and may introduce inconsistencies between the information maintained internally, and the actual column data and structure.

4.46.3.92 cpl_table_get_data_float_complex_const()

```
const float complex* cpl_table_get_data_float_complex_const (
             const cpl_table * table,
             const char * name )
```

Get a pointer to constant *float* complex column data.

Parameters

table	Pointer to constant table.
name	Column name.

Returns

Pointer to constant column data, or NULL if the column has zero length, or in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_FLOAT_COM↩ PLEX.

A *cpl_table* column of type CPL_TYPE_FLOAT_COMPLEX includes an array of values of type *float* complex. This function returns a pointer to this array. The data buffer elements corresponding to invalid column elements would in

general contain garbage. To avoid this, cpl_table_fill_invalid_float_complex() should be called just before this function, assigning to all the invalid column elements an *ad hoc* numerical value. See the description of function cpl_table_fill_invalid_float_complex() for further details.

4.46.3.93 cpl_table_get_data_float_const()

```
const float* cpl_table_get_data_float_const (
             const cpl_table * table,
             const char * name )
```

Get a pointer to constant *float* column data.

Parameters

table	Pointer to constant table.
name	Column name.

Returns

Pointer to constant column data, or NULL if the column has zero length, or in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_FLOAT.

A *cpl_table* column of type CPL_TYPE_FLOAT includes an array of values of type *float*. This function returns a pointer to this array. The data buffer elements corresponding to invalid column elements would in general contain garbage. To avoid this, cpl_table_fill_invalid_float() should be called just before this function, assigning to all the invalid column elements an *ad hoc* numerical value. See the description of function cpl_table_fill_invalid_float() for further details.

4.46.3.94 cpl_table_get_data_int()

```
int* cpl_table_get_data_int (
             cpl_table * table,
             const char * name )
```

Get a pointer to *integer* column data.

Parameters

table	Pointer to table.
name	Column name.

Returns

Pointer to column data, or NULL if the column has zero length, or in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_INT.

A *cpl_table* column of type CPL_TYPE_INT includes an array of values of type *int*. This function returns a pointer to this array. The data buffer elements corresponding to invalid column elements would in general contain garbage. To avoid this, cpl_table_fill_invalid_int() should be called just before this function, assigning to all the invalid column elements an *ad hoc* numerical value. See the description of function cpl_table_fill_invalid_int() for further details.

Note

Use at your own risk: direct manipulation of column data rules out any check performed by the table object interface, and may introduce inconsistencies between the information maintained internally, and the actual column data and structure.

4.46.3.95 cpl_table_get_data_int_const()

```
const int* cpl_table_get_data_int_const (
             const cpl_table * table,
             const char * name )
```

Get a pointer to constant *integer* column data.

Parameters

table	Pointer to constant table.
name	Column name.

Returns

Pointer to constant column data, or NULL if the column has zero length, or in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_INT.

A *cpl_table* column of type CPL_TYPE_INT includes an array of values of type *int*. This function returns a pointer to this array. The data buffer elements corresponding to invalid column elements would in general contain garbage. To avoid this, cpl_table_fill_invalid_int() should be called just before this function, assigning to all the invalid column elements an *ad hoc* numerical value. See the description of function cpl_table_fill_invalid_int() for further details.

4.46.3.96 cpl_table_get_data_long()

```
long* cpl_table_get_data_long (
            cpl_table * table,
            const char * name )
```

Get a pointer to *long* column data.

Parameters

table	Pointer to table.
name	Column name.

Returns

Pointer to column data, or NULL if the column has zero length, or in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_LONG.

A *cpl_table* column of type CPL_TYPE_LONG includes an array of values of type *long*. This function returns a pointer to this array. The data buffer elements corresponding to invalid column elements would in general contain garbage. To avoid this, cpl_table_fill_invalid_long() should be called just before this function, assigning to all the invalid column elements an *ad hoc* numerical value. See the description of function cpl_table_fill_invalid_long() for further details.

Note

Use at your own risk: direct manipulation of column data rules out any check performed by the table object interface, and may introduce inconsistencies between the information maintained internally, and the actual column data and structure.

4.46.3.97 cpl_table_get_data_long_const()

```
const long* cpl_table_get_data_long_const (
            const cpl_table * table,
            const char * name )
```

Get a pointer to constant *long* column data.

Parameters

table	Pointer to constant table.
name	Column name.

Returns

Pointer to constant column data, or NULL if the column has zero length, or in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_LONG.

A *cpl_table* column of type CPL_TYPE_LONG includes an array of values of type *long*. This function returns a pointer to this array. The data buffer elements corresponding to invalid column elements would in general contain garbage. To avoid this, cpl_table_fill_invalid_long() should be called just before this function, assigning to all the invalid column elements an *ad hoc* numerical value. See the description of function cpl_table_fill_invalid_long() for further details.

4.46.3.98 cpl_table_get_data_long_long()

```
long long* cpl_table_get_data_long_long (
             cpl_table * table,
             const char * name )
```

Get a pointer to *long long* column data.

Parameters

table	Pointer to table.
name	Column name.

Returns

Pointer to column data, or NULL if the column has zero length, or in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_LONG_LONG.

A *cpl_table* column of type CPL_TYPE_LONG_LONG includes an array of values of type *long* long. This function

returns a pointer to this array. The data buffer elements corresponding to invalid column elements would in general contain garbage. To avoid this, cpl_table_fill_invalid_long_long() should be called just before this function, assigning to all the invalid column elements an *ad hoc* numerical value. See the description of function cpl_table_fill_invalid_long_long() for further details.

Note

Use at your own risk: direct manipulation of column data rules out any check performed by the table object interface, and may introduce inconsistencies between the information maintained internally, and the actual column data and structure.

4.46.3.99 cpl_table_get_data_long_long_const()

```
const long long* cpl_table_get_data_long_long_const (
            const cpl_table * table,
            const char * name )
```

Get a pointer to constant *long* long column data.

Parameters

table	Pointer to constant table.
name	Column name.

Returns

Pointer to constant column data, or NULL if the column has zero length, or in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_LONG_LONG.

A *cpl_table* column of type CPL_TYPE_LONG_LONG includes an array of values of type *long* long. This function returns a pointer to this array. The data buffer elements corresponding to invalid column elements would in general contain garbage. To avoid this, cpl_table_fill_invalid_long_long() should be called just before this function, assigning to all the invalid column elements an *ad hoc* numerical value. See the description of function cpl_table_fill_invalid_long_long() for further details.

4.46.3.100 cpl_table_get_data_string()

```
char** cpl_table_get_data_string (
            cpl_table * table,
            const char * name )
```

Get a pointer to *string* column data.

Parameters

table	Pointer to table.
name	Column name.

Returns

Pointer to column data, or NULL if the column has zero length, or in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_STRING.

A table column of type CPL_TYPE_STRING includes an array of values of type *char**. This function returns a pointer to this array.

Note

Use at your own risk: direct manipulation of column data rules out any check performed by the table object interface, and may introduce inconsistencies between the information maintained internally, and the actual column data and structure.

4.46.3.101 cpl_table_get_data_string_const()

```
const char** cpl_table_get_data_string_const (
             const cpl_table * table,
             const char * name )
```

Get a pointer to constant *string* column data.

Parameters

table	Pointer to constant table.
name	Column name.

Returns

Pointer to constant column data, or NULL if the column has zero length, or in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_STRING.

A table column of type CPL_TYPE_STRING includes an array of values of type *char*∗. This function returns a pointer to this array.

4.46.3.102 cpl_table_get_double()

```
double cpl_table_get_double (
            const cpl_table * table,
            const char * name,
            cpl_size row,
            int * null )
```

Read a value from a *double* column.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Position of element to be read.
null	Flag indicating *null* values, or error condition.

Returns

Double value read. In case of an invalid table element, or in case of error, 0.0 is always returned.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are NULL pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_DOUB↩ LE.

Read a value from a column of type CPL_TYPE_DOUBLE. See the documentation of function cpl_table_get_int().

4.46.3.103 cpl_table_get_double_complex()

```
double complex cpl_table_get_double_complex (
            const cpl_table * table,
            const char * name,
            cpl_size row,
            int * null )
```

Read a value from a *double* complex column.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Position of element to be read.
null	Flag indicating *null* values, or error condition.

Returns

Double complex value read. In case of an invalid table element, or in case of error, 0.0 is always returned.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are `NULL` pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_DOUB↩ LE_COMPLEX`.

Read a value from a column of type `CPL_TYPE_DOUBLE_COMPLEX`. See the documentation of function cpl_table_get_int().

4.46.3.104 cpl_table_get_float()

```
float cpl_table_get_float (
            const cpl_table * table,
            const char * name,
            cpl_size row,
            int * null )
```

Read a value from a *float* column.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Position of element to be read.
null	Flag indicating *null* values, or error condition.

Returns

Float value read. In case of an invalid table element, or in case of error, 0.0 is always returned.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are `NULL` pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_FLOAT`.

Read a value from a column of type `CPL_TYPE_FLOAT`. See the documentation of function cpl_table_get_int().

4.46.3.105 cpl_table_get_float_complex()

```
float complex cpl_table_get_float_complex (
            const cpl_table * table,
            const char * name,
            cpl_size row,
            int * null )
```

Read a value from a *float* complex column.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Position of element to be read.
null	Flag indicating *null* values, or error condition.

Returns

Float complex value read. In case of an invalid table element, or in case of error, 0.0 is always returned.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are `NULL` pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_FLOA↩ T_COMPLEX`.

Read a value from a column of type `CPL_TYPE_FLOAT_COMPLEX`. See the documentation of function cpl_table_get_int().

4.46.3.106 cpl_table_get_int()

```
int cpl_table_get_int (
            const cpl_table * table,
```

```
              const char * name,
              cpl_size row,
              int * null )
```

Read a value from an *integer* column.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Position of element to be read.
null	Flag indicating *null* values, or error condition.

Returns

Integer value read. In case of an invalid table element, or in case of error, 0 is always returned.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are `NULL` pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_INT`.

Read a value from a column of type `CPL_TYPE_INT`. If the *null* flag is a valid pointer, it is used to indicate whether the accessed column element is valid (0) or invalid (1). The *null* flag also signals an error condition (-1). The *null* flag pointer can also be `NULL`, and in that case this option will be disabled. Rows are counted starting from 0.

Note

For automatic conversion (always to type *double*), use the function `cpl_table_get()`.

References CPL_ERROR_NULL_INPUT.

4.46.3.107 cpl_table_get_long()

```
long cpl_table_get_long (
              const cpl_table * table,
              const char * name,
              cpl_size row,
              int * null )
```

Read a value from a *long* column.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Position of element to be read.
null	Flag indicating *null* values, or error condition.

Returns

Long integer value read. In case of an invalid table element, or in case of error, 0 is always returned.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are `NULL` pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_LONG`.

Read a value from a column of type `CPL_TYPE_LONG`. See the documentation of function cpl_table_get_int().

4.46.3.108 cpl_table_get_long_long()

```
long long cpl_table_get_long_long (
             const cpl_table * table,
             const char * name,
             cpl_size row,
             int * null )
```

Read a value from a *long long* column.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Position of element to be read.
null	Flag indicating *null* values, or error condition.

Returns

Long long integer value read. In case of an invalid table element, or in case of error, 0 is always returned.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are `NULL` pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.

CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_LONG_LONG`.

Read a value from a column of type `CPL_TYPE_LONG_LONG`. See the documentation of function cpl_table_get_int().

4.46.3.109 cpl_table_get_ncol()

```
cpl_size cpl_table_get_ncol (
            const cpl_table * table )
```

Get the number of columns in a table.

Parameters

table	Pointer to table to examine.

Returns

Number of columns in the table. If a `NULL` table pointer is passed, -1 is returned.

Errors

CPL_ERROR_NULL_INPUT	*table* is a `NULL` pointer.

Get the number of columns in a table.

References CPL_ERROR_NULL_INPUT, and cpl_error_set.

Referenced by cpl_table_compare_structure(), cpl_table_copy_structure(), cpl_table_delete(), cpl_table_dump(), cpl_table_dump_structure(), cpl_table_duplicate(), cpl_table_erase_invalid(), cpl_table_erase_invalid_rows(), cpl_table_erase_selected(), cpl_table_erase_window(), cpl_table_extract(), cpl_table_get_column_name(), cpl_table_insert(), cpl_table_insert_window(), and cpl_table_set_size().

4.46.3.110 cpl_table_get_nrow()

```
cpl_size cpl_table_get_nrow (
            const cpl_table * table )
```

Get the number of rows in a table.

Parameters

table	Pointer to table to examine.

Returns

Number of rows in the table. If a NULL table pointer is passed, -1 is returned.

Errors

CPL_ERROR_NULL_INPUT	*table* is a NULL pointer.

Get the number of rows in a table.

References CPL_ERROR_NULL_INPUT, and cpl_error_set.

Referenced by cpl_plot_column(), cpl_plot_columns(), cpl_table_duplicate(), cpl_table_erase_invalid(), cpl_table_erase_invalid_rows(), cpl_table_erase_selected(), and cpl_table_extract().

4.46.3.111 cpl_table_get_string()

```
const char* cpl_table_get_string (
             const cpl_table * table,
             const char * name,
             cpl_size row )
```

Read a value from a *string* column.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Position of element to be read.

Returns

Pointer to string. In case of an invalid column element, or in case of error, a NULL pointer is always returned.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are NULL pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_STRING.

Read a value from a column of type CPL_TYPE_STRING. Rows are counted starting from 0.

Note

The returned string is a pointer to a table element, not its copy. Its manipulation will directly affect that element, while changing that element using cpl_table_set_string() will turn it into garbage. Therefore, if a real copy of a string column element is required, this function should be called as an argument of the function strdup().

4.46.3.112 cpl_table_has_column()

```
int cpl_table_has_column (
            const cpl_table * table,
            const char * name )
```

Check if a column with a given name exists.

Parameters

table	Pointer to table.
name	Name of table column.

Returns

1 if column exists, 0 if column doesn't exist, -1 in case of error.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.

Check if a column with a given name exists in the specified table.

References CPL_ERROR_NULL_INPUT, and cpl_error_set.

Referenced by cpl_plot_column(), cpl_plot_columns(), and cpl_table_compare_structure().

4.46.3.113 cpl_table_has_invalid()

```
int cpl_table_has_invalid (
            const cpl_table * table,
            const char * name )
```

Check if a column contains at least one invalid value.

Parameters

table	Pointer to table.
name	Name of table column to access.

Returns

1 if the column contains at least one invalid element, 0 if not, -1 in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.

Check if there are invalid elements in a column. In case of columns of arrays, invalid values within an array are not considered: an invalid element here means that an array element is not allocated, i.e., it is a `NULL` pointer. In order to detect invalid elements within an array element, this element must be extracted using the function `cpl_table_get_array()`, and then use the function `cpl_array_has_invalid()`.

References cpl_errorstate_get().

4.46.3.114 cpl_table_has_valid()

```
int cpl_table_has_valid (
             const cpl_table * table,
             const char * name )
```

Check if a column contains at least one valid value.

Parameters

table	Pointer to table.
name	Name of table column to access.

Returns

1 if the column contains at least one valid value, 0 if not, -1 in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.

Check if there are valid elements in a column. In case of columns of arrays, invalid values within an array are not considered: an invalid element here means that an array element is not allocated, i.e., it is a NULL pointer. In order to detect valid elements within an array element, this element must be extracted using the function cpl_table_get_array(), and then use the function cpl_array_has_valid().

References cpl_errorstate_get().

4.46.3.115 cpl_table_imag_column()

```
cpl_error_code cpl_table_imag_column (
            cpl_table * table,
            const char * name )
```

Compute the imaginary part value of table column elements.

Parameters

table	Pointer to table.
name	Column name.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are NULL pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is neither numerical nor complex.

Each column element is replaced by its imaginary party value only. Invalid elements are not modified by this operation. If the column is complex, its type will be turned to real (CPL_TYPE_FLOAT_COMPLEX will be changed into CPL_TYPE_FLOAT, and CPL_TYPE_DOUBLE_COMPLEX will be changed into CP↩L_TYPE_DOUBLE), and any pointer retrieved by calling cpl_table_get_data_float_complex(), cpl_table_get_data_double_complex(), etc., should be discarded.

4.46.3.116 cpl_table_insert()

```
cpl_error_code cpl_table_insert (
            cpl_table * target_table,
            const cpl_table * insert_table,
            cpl_size row )
```

Merge two tables.

Parameters

target_table	Target table.
insert_table	Table to be inserted in the target table.
row	Row where to insert the insert table.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any input table is a NULL pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	*row* is negative.
CPL_ERROR_INCOMPATIBLE_INPUT	The input tables do not have the same structure.

The input tables must have the same structure, as defined by the function cpl_table_compare_structure() . Data from the *insert_table* are duplicated and inserted at the specified position of the *target_table*. If the specified *row* is not less than the target table length, the second table will be appended to the target table. The selection flags of the target table are always set back to "all selected". The pointers to column data in the target table may change, therefore pointers previously retrieved by calling cpl_table_get_data_int(), cpl_table_get_data_string(), etc., should be discarded.

References CPL_ERROR_INCOMPATIBLE_INPUT, cpl_table_compare_structure(), and cpl_table_get_ncol().

4.46.3.117 cpl_table_insert_window()

```
cpl_error_code cpl_table_insert_window (
            cpl_table * table,
            cpl_size start,
            cpl_size count )
```

Insert a segment of rows into table data.

Parameters

table	Pointer to table
start	Row where to insert the segment.
count	Length of the segment.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	*table* is a `NULL` pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	*start* is negative.
CPL_ERROR_ILLEGAL_INPUT	*count* is negative.

Insert a segment of empty rows, just containing invalid elements. Setting *start* to a number greater than the column length is legal, and has the effect of appending extra rows at the end of the table: this is equivalent to expanding the table using `cpl_table_set_size()`. The input *column* may also have zero length. The pointers to column data values may change, therefore pointers previously retrieved by calling `cpl_table_get_data_int()`, `cpl_table_get_data_string()`, etc., should be discarded. The table selection flags are set back to "all selected".

References CPL_ERROR_NULL_INPUT, cpl_table_get_ncol(), and cpl_table_select_all().

4.46.3.118 cpl_table_is_selected()

```
int cpl_table_is_selected (
            const cpl_table * table,
            cpl_size row )
```

Determine whether a table row is selected or not.

Parameters

table	Pointer to table.
row	Table row to check.

Returns

1 if row is selected, 0 if it is not selected, -1 in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* is a `NULL` pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.

Check if a table row is selected.

References CPL_ERROR_ACCESS_OUT_OF_RANGE, and CPL_ERROR_NULL_INPUT.

4.46.3.119 cpl_table_is_valid()

```
int cpl_table_is_valid (
            const cpl_table * table,
            const char * name,
            cpl_size row )
```

Check if a column element is valid.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Column element to examine.

Returns

1 if the column element is valid, 0 if invalid, -1 in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are `NULL` pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.

Check if a column element is valid.

References cpl_errorstate_get().

4.46.3.120 cpl_table_load()

```
cpl_table* cpl_table_load (
            const char * filename,
            int xtnum,
            int check_nulls )
```

Load a FITS table extension into a new *cpl_table*.

Parameters

filename	Name of FITS file with at least one table extension.
xtnum	Number of extension to read, starting from 1.
check_nulls	If set to 0, identified invalid values are not marked.

Returns

New table, or NULL in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Input *filename* is a NULL pointer.
CPL_ERROR_FILE_NOT_FOUND	A file named as specified in *filename* is not found.
CPL_ERROR_BAD_FILE_FORMAT	The input file is not in FITS format.
CPL_ERROR_ILLEGAL_INPUT	The specified FITS file extension is not a table, or, if it is a table, it has more than 9999 columns.
CPL_ERROR_ACCESS_OUT_OF_RANGE	*xtnum* is greater than the number of FITS extensions in the FITS file, or is less than 1.
CPL_ERROR_DATA_NOT_FOUND	The FITS table has no rows or no columns.
CPL_ERROR_UNSPECIFIED	Generic error condition, that should be reported to the CPL Team.

The selected FITS file table extension is just read and converted into the *cpl_table* conventions.

4.46.3.121 cpl_table_load_window()

```
cpl_table* cpl_table_load_window (
             const char * filename,
             int xtnum,
             int check_nulls,
             const cpl_array * selcol,
             cpl_size firstrow,
             cpl_size nrow )
```

Load part of a FITS table extension into a new *cpl_table*.

Parameters

filename	Name of FITS file with at least one table extension.
xtnum	Number of extension to read, starting from 1.
check_nulls	If set to 0, identified invalid values are not marked.
selcol	Array with the names of the columns to extract.
firstrow	First table row to extract.
nrow	Number of rows to extract.

Returns

New table, or NULL in case of failure.

Errors

CPL_ERROR_NULL_INPUT	Input *filename* is a NULL pointer.
CPL_ERROR_FILE_NOT_FOUND	A file named as specified in *filename* is not found.

CPL_ERROR_BAD_FILE_FORMAT	The input file is not in FITS format.
CPL_ERROR_ILLEGAL_INPUT	The specified FITS file extension is not a table. Or the specified number of rows to extract is less than zero. Or the array of column names to extract contains empty fields.
CPL_ERROR_ACCESS_OUT_OF_RANGE	*xtnum* is greater than the number of FITS extensions in the FITS file, or is less than 1. Or *firstrow* is either less than zero, or greater than the number of rows in the table.
CPL_ERROR_DATA_NOT_FOUND	The FITS table has no columns. Or *selcol* includes columns that are not found in table.
CPL_ERROR_UNSPECIFIED	Generic error condition, that should be reported to the CPL Team.

The selected FITS file table extension is just read in the specified columns and rows intervals, and converted into the *cpl_table* conventions. If *selcol* is NULL, all columns are selected.

References CPL_ERROR_ACCESS_OUT_OF_RANGE, and CPL_ERROR_ILLEGAL_INPUT.

4.46.3.122 cpl_table_logarithm_column()

```
cpl_error_code cpl_table_logarithm_column (
            cpl_table * table,
            const char * name,
            double base )
```

Compute the logarithm of column values.

Parameters

table	Pointer to table.
name	Table column name.
base	Logarithm base.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are NULL pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is neither numerical nor complex, or it is an array column.
CPL_ERROR_ILLEGAL_INPUT	The input *base* is not positive.

Each column element is replaced by its logarithm in the specified base. The operation is always performed in double

precision, with a final cast of the result to the target column type. Invalid elements are not modified by this operation, but zero or negative elements are invalidated by this operation. In case of complex numbers, values very close to the origin may cause an overflow. The imaginary part of the result is chosen in the interval [-pi/ln(base),pi/ln(base)], so it should be kept in mind that doing the logarithm of exponential of a complex number will not always express the phase angle with the same number. For instance, the exponential in base 2 of (5.00, 5.00) is (-30.33, -10.19), and the logarithm in base 2 of the latter will be expressed as (5.00, -4.06).

4.46.3.123 cpl_table_move_column()

```
cpl_error_code cpl_table_move_column (
            cpl_table * to_table,
            const char * name,
            cpl_table * from_table )
```

Move a column from a table to another.

Parameters

to_table	Target table.
name	Name of column to move.
from_table	Source table.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a `NULL` pointer.
CPL_ERROR_INCOMPATIBLE_INPUT	The input tables do not have the same number of rows.
CPL_ERROR_ILLEGAL_INPUT	Source and target tables are the same table.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in the source table.
CPL_ERROR_ILLEGAL_OUTPUT	A column with the same *name* already exists in the target table.

Move a column from a table to another.

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INCOMPATIBLE_INPUT, and CPL_ERROR_NULL↩ _INPUT.

4.46.3.124 cpl_table_multiply_columns()

```
cpl_error_code cpl_table_multiply_columns (
            cpl_table * table,
            const char * to_name,
            const char * from_name )
```

Multiply two numeric or complex table columns.

Parameters

table	Pointer to table.
to_name	Name of target column.
from_name	Name of column to be multiplied with target column.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or any column name are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with any specified name is not found in *table*.
CPL_ERROR_INVALID_TYPE	Any specified column is neither numerical nor complex, or it is an array column.

The columns are multiplied element by element, and the result of the multiplication is stored in the target column. See the documentation of the function `cpl_table_add_columns()` for further details.

4.46.3.125 cpl_table_multiply_scalar()

```
cpl_error_code cpl_table_multiply_scalar (
            cpl_table * table,
            const char * name,
            double value )
```

Multiply a numerical or complex column by a constant.

Parameters

table	Pointer to table.
name	Column name.
value	Multiplication factor.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is neither numerical nor complex, or it is an array column.

See the description of the function cpl_table_add_scalar().

4.46.3.126 cpl_table_multiply_scalar_complex()

```
cpl_error_code cpl_table_multiply_scalar_complex (
            cpl_table * table,
            const char * name,
            double complex value )
```

Multiply a numerical or complex column by a complex constant.

Parameters

table	Pointer to table.
name	Column name.
value	Multiplication factor.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are NULL pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is neither numerical nor complex, or it is an array column.

See the description of the function cpl_table_add_scalar_complex().

4.46.3.127 cpl_table_name_column()

```
cpl_error_code cpl_table_name_column (
            cpl_table * table,
            const char * from_name,
            const char * to_name )
```

Rename a table column.

Parameters

table	Pointer to table.
from_name	Name of table column to rename.
to_name	New name of column.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *from_name* is not found in *table*.
CPL_ERROR_ILLEGAL_OUTPUT	A column with the specified *to_name* already exists in *table*.

This function is used to change the name of a column.

References CPL_ERROR_NULL_INPUT.

4.46.3.128 cpl_table_new()

```
cpl_table* cpl_table_new (
            cpl_size length )
```

Create an empty table structure.

Parameters

length	Number of rows in table.

Returns

Pointer to new table, or NULL in case of error.

Errors

CPL_ERROR_ILLEGAL_INPUT	The specified *length* is negative.

This function allocates and initialises memory for a table data container. A new table is created with no columns, but the size of the columns that will be created is defined in advance, to ensure that all columns will be created with the same length. All table rows are marked a priori as selected. This should be considered the normal status of a table, as long as no row selection has been applied to it.

References cpl_calloc(), CPL_ERROR_ILLEGAL_INPUT, and cpl_error_set.

Referenced by cpl_plot_column(), cpl_plot_columns(), cpl_table_duplicate(), cpl_table_extract(), and cpl_table_↩ extract_selected().

4.46.3.129 cpl_table_new_column()

```
cpl_error_code cpl_table_new_column (
            cpl_table * table,
            const char * name,
            cpl_type type )
```

Create an empty column in a table.

Parameters

table	Pointer to table.
name	Name of the new column.
type	Type of the new column.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_INVALID_TYPE	The specified *type* is not supported.
CPL_ERROR_ILLEGAL_OUTPUT	A column with the same *name* already exists in *table*.

This function allocates memory for a new column of specified *type*, excluding *array* types (for creating a column of arrays use the function cpl_table_new_column_array(), where the column depth must also be specified). The new column name must be different from any other column name in the table. All the elements of the new column are marked as invalid.

References CPL_ERROR_NULL_INPUT.

Referenced by cpl_table_copy_structure().

4.46.3.130 cpl_table_new_column_array()

```
cpl_error_code cpl_table_new_column_array (
            cpl_table * table,
            const char * name,
            cpl_type type,
            cpl_size depth )
```

Create an empty column of arrays in a table.

Parameters

table	Pointer to table.
name	Name of the new column.
type	Type of the new column.
depth	Depth of the new column.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_INVALID_TYPE	The specified *type* is not supported.
CPL_ERROR_ILLEGAL_INPUT	The specified *depth* is negative.
CPL_ERROR_ILLEGAL_OUTPUT	A column with the same *name* already exists in *table*.

This function allocates memory for a new column of specified array *type*, (for creating a column of simple scalars or character strings use the function cpl_table_new_column() instead). It doesn't make any difference if a simple or an array *type* is specified, the corresponding array type will always be created (e.g., specifying a *type* CPL_TYPE_INT or a type CPL_TYPE_INT | CPL_TYPE_POINTER would always create a column of type CPL_TYPE_INT | CPL_TYPE_POINTER). The new column name must be different from any other column name in the table. All the elements of the new column are marked as invalid.

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NULL_INPUT, and CPL_TYPE_LONG.

Referenced by cpl_table_copy_structure().

4.46.3.131 cpl_table_not_selected()

```
cpl_size cpl_table_not_selected (
            cpl_table * table )
```

Select unselected table rows, and unselect selected ones.

Parameters

table	Pointer to table.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* is a NULL pointer.

Select unselected table rows, and unselect selected ones.

References CPL_ERROR_NULL_INPUT.

4.46.3.132 cpl_table_or_selected()

```
cpl_size cpl_table_or_selected (
            cpl_table * table,
            const char * name1,
            cpl_table_select_operator operator,
            const char * name2 )
```

Select from unselected table rows, by comparing the values of two numerical columns.

Parameters

table	Pointer to table.
name1	Name of first table column.
operator	Relational operator.
name2	Name of second table column.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column names are NULL pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with any of the specified names is not found in *table*.
CPL_ERROR_INVALID_TYPE	Invalid types for comparison.

Either both columns must be numerical, or they both must be strings. The comparison between strings is lexicographical. Comparison between complex types and array types are not supported.

Both columns must be numerical. For all the unselected table rows, the values of the specified columns are compared. The table rows fulfilling the comparison are selected. Invalid elements from either columns never fulfill any comparison by definition. Allowed relational operators are CPL_EQUAL_TO, CPL_NOT_EQUAL_TO, CPL_GREATER_THAN, CPL_NOT_GREATER_THAN, CPL_LESS_THAN, CPL_NOT_LESS_THAN. See also the function cpl_table_and_selected().

4.46.3.133 cpl_table_or_selected_double()

```
cpl_size cpl_table_or_selected_double (
            cpl_table * table,
            const char * name,
            cpl_table_select_operator operator,
            double value )
```

Select from unselected table rows, by comparing *double* column values with a constant.

Parameters

table	Pointer to table.
name	Column name.
operator	Relational operator.
value	Reference value.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_DOUBLE`.

For all the unselected table rows, the values of the specified column are compared with the reference value. The table rows fulfilling the comparison are selected. An invalid element never fulfills any comparison by definition. Allowed relational operators are `CPL_EQUAL_TO`, `CPL_NOT_EQUAL_TO`, `CPL_GREATER_THAN`, `CPL_NOT_GREATER_THAN`, `CPL_LESS_THAN`, and `CPL_NOT_LESS_THAN`. If the table has no rows, no error is set, and 0 is returned. See also the description of the function `cpl_table_and_selected_double()`.

4.46.3.134 cpl_table_or_selected_double_complex()

```
cpl_size cpl_table_or_selected_double_complex (
            cpl_table * table,
            const char * name,
            cpl_table_select_operator operator,
            double complex value )
```

Select from unselected table rows, by comparing *double* complex column values with a complex constant.

Parameters

table	Pointer to table.
name	Column name.
operator	Relational operator.
value	Reference value.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are NULL pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_DOUBLE_COMPLEX.
CPL_ERROR_ILLEGAL_INPUT	Operator other than CPL_EQUAL_TO or CPL_NOT_EQUAL_TO was specified.

For all the unselected table rows, the values of the specified column are compared with the reference value. The table rows fulfilling the comparison are selected. An invalid element never fulfills any comparison by definition. Allowed relational operators are CPL_EQUAL_TO and CPL_NOT_EQUAL_TO. If the table has no rows, no error is set, and 0 is returned. See also the description of the function cpl_table_and_selected_double_complex().

4.46.3.135 cpl_table_or_selected_float()

```
cpl_size cpl_table_or_selected_float (
            cpl_table * table,
            const char * name,
            cpl_table_select_operator operator,
            float value )
```

Select from unselected table rows, by comparing *float* column values with a constant.

Parameters

table	Pointer to table.
name	Column name.
operator	Relational operator.
value	Reference value.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are NULL pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_FLOAT.

For all the unselected table rows, the values of the specified column are compared with the reference value. The table rows fulfilling the comparison are selected. An invalid element never fulfills any comparison by definition. Allowed relational operators are CPL_EQUAL_TO, CPL_NOT_EQUAL_TO, CPL_GREATER_THAN, CPL_NOT_GREATER_THAN, CPL_LESS_THAN, and CPL_NOT_LESS_THAN. If the table has no rows, no error is set, and 0 is returned. See also the description of the function cpl_table_and_selected_float().

4.46.3.136 cpl_table_or_selected_float_complex()

```
cpl_size cpl_table_or_selected_float_complex (
            cpl_table * table,
            const char * name,
            cpl_table_select_operator operator,
            float complex value )
```

Select from unselected table rows, by comparing *float* complex column values with a complex constant.

Parameters

table	Pointer to table.
name	Column name.
operator	Relational operator.
value	Reference value.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_FLOAT_COMPLEX`.
CPL_ERROR_ILLEGAL_INPUT	Operator other than `CPL_EQUAL_TO` or `CPL_NOT_EQUAL_TO` was specified.

For all the unselected table rows, the values of the specified column are compared with the reference value. The table rows fulfilling the comparison are selected. An invalid element never fulfills any comparison by definition. Allowed relational operators are `CPL_EQUAL_TO` and `CPL_NOT_EQUAL_TO`. If the table has no rows, no error is set, and 0 is returned. See also the description of the function `cpl_table_and_selected_float_complex()`.

4.46.3.137 cpl_table_or_selected_int()

```
cpl_size cpl_table_or_selected_int (
            cpl_table * table,
            const char * name,
            cpl_table_select_operator operator,
            int value )
```

Select from unselected table rows, by comparing *integer* column values with a constant.

Parameters

table	Pointer to table.
name	Column name.
operator	Relational operator.
value	Reference value.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are NULL pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_INT.

For all the unselected table rows, the values of the specified column are compared with the reference value. The table rows fulfilling the comparison are selected. An invalid element never fulfills any comparison by definition. Allowed relational operators are CPL_EQUAL_TO, CPL_NOT_EQUAL_TO, CPL_GREATER_THAN, CPL_NOT_GREATER_THAN, CPL_LESS_THAN, and CPL_NOT_LESS_THAN. If the table has no rows, no error is set, and 0 is returned. See also the description of the function cpl_table_and_selected_int().

4.46.3.138 cpl_table_or_selected_invalid()

```
cpl_size cpl_table_or_selected_invalid (
            cpl_table * table,
            const char * name )
```

Select from unselected table rows all rows with an invalid value in a specified column.

Parameters

table	Pointer to table.
name	Column name.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are NULL pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.

For all the unselected table rows, all the rows containing invalid values at the specified column are selected. See also the function cpl_table_and_selected_invalid().

4.46.3.139 cpl_table_or_selected_long()

```
cpl_size cpl_table_or_selected_long (
            cpl_table * table,
```

```
            const char * name,
            cpl_table_select_operator operator,
            long value )
```

Select from unselected table rows, by comparing *long* column values with a constant.

Parameters

table	Pointer to table.
name	Column name.
operator	Relational operator.
value	Reference value.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_LONG`.

For all the unselected table rows, the values of the specified column are compared with the reference value. The table rows fulfilling the comparison are selected. An invalid element never fulfills any comparison by definition. Allowed relational operators are `CPL_EQUAL_TO`, `CPL_NOT_EQUAL_TO`, `CPL_GREATER_THAN`, `CPL_NOT_GREATER_THAN`, `CPL_LESS_THAN`, and `CPL_NOT_LESS_THAN`. If the table has no rows, no error is set, and 0 is returned. See also the description of the function `cpl_table_and_selected_long()`.

4.46.3.140 cpl_table_or_selected_long_long()

```
cpl_size cpl_table_or_selected_long_long (
            cpl_table * table,
            const char * name,
            cpl_table_select_operator operator,
            long long value )
```

Select from unselected table rows, by comparing *long long* column values with a constant.

Parameters

table	Pointer to table.
name	Column name.
operator	Relational operator.
value	Reference value.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_LONG_LONG`.

For all the unselected table rows, the values of the specified column are compared with the reference value. The table rows fulfilling the comparison are selected. An invalid element never fulfills any comparison by definition. Allowed relational operators are `CPL_EQUAL_TO`, `CPL_NOT_EQUAL_TO`, `CPL_GREATER_THAN`, `CPL_NOT_GREATER_THAN`, `CPL_LESS_THAN`, and `CPL_NOT_LESS_THAN`. If the table has no rows, no error is set, and 0 is returned. See also the description of the function `cpl_table_and_selected_long_long()`.

4.46.3.141 cpl_table_or_selected_string()

```
cpl_size cpl_table_or_selected_string (
            cpl_table * table,
            const char * name,
            cpl_table_select_operator operator,
            const char * string )
```

Select from unselected table rows, by comparing column values with a constant.

Parameters

table	Pointer to table.
name	Column name.
operator	Relational operator.
string	Reference value.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_STRING`.
CPL_ERROR_ILLEGAL_INPUT	Invalid regular expression.

For all the unselected table rows, the values of the specified column are compared with the reference value. The comparison function used is the C standard `strcmp()`, but in case the relational operators `CPL_EQUAL_`↩

TO or CPL_NOT_EQUAL_TO are specified, the comparison string is treated as a regular expression. The table rows fulfilling the comparison are selected. An invalid element never fulfills any comparison by definition. Allowed relational operators are CPL_EQUAL_TO, CPL_NOT_EQUAL_TO, CPL_GREATER_THAN, CPL_NOT_GREATER_THAN, CPL_LESS_THAN, and CPL_NOT_LESS_THAN. If the table has no rows, no error is set, and 0 is returned. See also the description of the function cpl_table_and_selected_string().

4.46.3.142 cpl_table_or_selected_window()

```
cpl_size cpl_table_or_selected_window (
            cpl_table * table,
            cpl_size start,
            cpl_size count )
```

Select from unselected rows only those within a table segment.

Parameters

table	Table to handle.
start	First row of table segment.
count	Length of segment.

Returns

Current number of selected rows, or a negative number in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* is a NULL pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *start* is outside the table boundaries.
CPL_ERROR_ILLEGAL_INPUT	*count* is negative.

All the unselected table rows that are within the specified interval are selected. If the sum of *start* and *count* goes beyond the end of the input table, rows are checked up to the end of the table. See also the function cpl_table_and_selected_window().

References cpl_calloc(), CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_free(), and cpl_table_select_all().

4.46.3.143 cpl_table_power_column()

```
cpl_error_code cpl_table_power_column (
            cpl_table * table,
            const char * name,
            double exponent )
```

Compute the power of numerical column values.

Parameters

table	Pointer to table.
name	Name of column of numerical type
exponent	Constant exponent.

Returns

`CPL_ERROR_NONE` on success.

See also

pow(), cpow()

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is not numerical.

Each column element is replaced by its power to the specified exponent. For float and float complex the operation is performed in single precision, otherwise it is performed in double precision and then rounded if the column is of an integer type. Results that would or do cause domain errors or overflow are marked as invalid.

4.46.3.144 cpl_table_real_column()

```
cpl_error_code cpl_table_real_column (
            cpl_table * table,
            const char * name )
```

Compute the real part value of table column elements.

Parameters

table	Pointer to table.
name	Column name.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is neither numerical nor complex.

Each column element is replaced by its real party value only. Invalid elements are not modified by this operation. If the column is complex, its type will be turned to real (CPL_TYPE_FLOAT_COMPLEX will be changed into CPL_TYPE_FLOAT, and CPL_TYPE_DOUBLE_COMPLEX will be changed into CPL_↩ TYPE_DOUBLE), and any pointer retrieved by calling cpl_table_get_data_float_complex(), cpl_table_get_data_double_complex(), etc., should be discarded.

4.46.3.145 cpl_table_save()

```
cpl_error_code cpl_table_save (
            const cpl_table * table,
            const cpl_propertylist * pheader,
            const cpl_propertylist * header,
            const char * filename,
            unsigned mode )
```

Save a *cpl_table* to a FITS file.

Parameters

table	Input table.
pheader	Primary header entries.
header	Table header entries.
filename	Name of output FITS file.
mode	Output mode.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *filename* are NULL pointers.
CPL_ERROR_UNSUPPORTED_MODE	A saving mode other than CPL_IO_CREATE and CPL_I↩ O_EXTEND was specified.
CPL_ERROR_FILE_NOT_FOUND	While *mode* is set to CPL_IO_EXTEND, a file named as specified in *filename* is not found.
CPL_ERROR_BAD_FILE_FORMAT	While *mode* is set to CPL_IO_EXTEND, the specified file is not in FITS format.
CPL_ERROR_FILE_NOT_CREATED	The FITS file could not be created.
CPL_ERROR_FILE_IO	The FITS file could only partially be created.
CPL_ERROR_UNSPECIFIED	Generic error condition, that should be reported to the CPL Team.

This function can be used to convert a CPL table into a binary FITS table extension. If the *mode* is set to CPL_I↩ O_CREATE, a new FITS file will be created containing an empty primary array, with just one FITS table extension.

An existing (and writable) FITS file with the same name would be overwritten. If the *mode* flag is set to CPL_IO↩_EXTEND, a new table extension would be appended to an existing FITS file. If *mode* is set to CPL_IO_APPEND it is possible to add rows to the last FITS table extension of the output FITS file.

Note that the modes CPL_IO_EXTEND and CPL_IO_APPEND require that the target file must be writable (and do not take for granted that a file is writable just because it was created by the same application, as this depends on the system *umask*).

When using the mode CPL_IO_APPEND additional requirements must be fulfilled, which are that the column properties like type, format, units, etc. must match as the properties of the FITS table extension to which the rows should be added exactly. In particular this means that both tables use the same null value representation for integral type columns!

Two property lists may be passed to this function, both optionally. The first property list, *pheader*, is just used if the *mode* is set to CPL_IO_CREATE, and it is assumed to contain entries for the FITS file primary header. In *pheader* any property name related to the FITS convention, as SIMPLE, BITPIX, NAXIS, EXTEND, BLOCKED, and END, are ignored: such entries would be written anyway to the primary header and set to some standard values.

If a NULL *pheader* is passed, the primary array would be created with just such entries, that are mandatory in any regular FITS file. The second property list, *header*, is assumed to contain entries for the FITS table extension header. In this property list any property name related to the FITS convention, as XTENSION, BITPIX, NAXIS, PCOUNT, GCOUNT, and END, and to the table structure, as TFIELDS, TTYPEi, TUNITi, TDISPi, TNULLi, TFORMi, would be ignored: such entries are always computed internally, to guarantee their consistency with the actual table structure. A DATE keyword containing the date of table creation in ISO8601 format is also added automatically.

Note

- Invalid strings in columns of type CPL_TYPE_STRING are written to FITS as blanks.
- Invalid values in columns of type CPL_TYPE_FLOAT or CPL_TYPE_DOUBLE will be written to the FITS file as a NaN.
- Invalid values in columns of type CPL_TYPE_INT and CPL_TYPE_LONG_LONG are the only ones that need a specific code to be explicitly assigned to them. This can be realised by calling the functions cpl_table_fill_invalid_int() and cpl_table_fill_invalid_long_long(), respectively, for each table column of type CPL_TYPE_INT and CPL_TYPE_LONG_LONG containing invalid values, just before saving the table to FITS. The numerical values identifying invalid integer column elements are written to the FITS keywords TNULLn (where n is the column sequence number). Beware that if valid column elements have the value identical to the chosen null-code, they will also be considered invalid in the FITS convention.
- Using the mode CPL_IO_APPEND requires that the column properties of the table to be appended are compared to the column properties of the target FITS extension for each call, which introduces a certain overhead. This means that appending a single table row at a time may not be efficient and is not recommended. Rather than writing one row at a time one should write table chunks containing a suitable number or rows.

References CPL_IO_APPEND.

4.46.3.146 cpl_table_select_all()

```
cpl_error_code cpl_table_select_all (
            cpl_table * table )
```

Select all table rows.

Parameters

table	Pointer to table.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* is a NULL pointer.

The table selection flags are reset, meaning that they are all marked as selected. This is the initial state of any table.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl_free().

Referenced by cpl_table_erase_invalid(), cpl_table_erase_invalid_rows(), cpl_table_erase_selected(), cpl_table_↩ erase_window(), cpl_table_insert_window(), cpl_table_or_selected_window(), and cpl_table_set_size().

4.46.3.147 cpl_table_select_row()

```
cpl_error_code cpl_table_select_row (
            cpl_table * table,
            cpl_size row )
```

Flag a table row as selected.

Parameters

table	Pointer to table.
row	Row to select.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* is a NULL pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.

Flag a table row as selected. Any previous selection is kept.

References cpl_calloc(), CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_NONE, CPL_ERROR_NU↩
LL_INPUT, and cpl_free().

4.46.3.148 cpl_table_set()

```
cpl_error_code cpl_table_set (
            cpl_table * table,
            const char * name,
            cpl_size row,
            double value )
```

Write a value to a numerical table column element.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Position where to write value.
value	Value to write.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are NULL pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is not numerical, or it is of type *array*.

Write a value to a numerical column element. The value is cast to the accessed column type according to the C casting rules. The written value is automatically marked as valid. To invalidate a column value use cpl_table_set_invalid(). Table rows are counted starting from 0.

4.46.3.149 cpl_table_set_array()

```
cpl_error_code cpl_table_set_array (
            cpl_table * table,
            const char * name,
            cpl_size row,
            const cpl_array * array )
```

Write an array to an *array* table column element.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Position where to write the array.
array	Array to write.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are NULL pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type *array*.
CPL_ERROR_INCOMPATIBLE_INPUT	The size of the input *array* is different from the depth of the specified column.

Write an array to a table column of type *array*. The written value can also be a NULL pointer, that is equivalent to a call to cpl_table_set_invalid(). Note that the array is copied, therefore the original can be modified without affecting the table element. To "plug" an array directly into a table element, use the function cpl_table_get_data_array(). Beware that the "plugged" array must have the same type and depth declared for the column.

4.46.3.150 cpl_table_set_column_depth()

```
cpl_error_code cpl_table_set_column_depth (
            cpl_table * table,
            const char * name,
            cpl_size depth )
```

Modify depth of a column of arrays.

Parameters

table	Pointer to table.
name	Column name.
depth	New column depth.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any input argument is a NULL pointer.
CPL_ERROR_ILLEGAL_INPUT	The specified new depth is negative.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The column with the specified *name* is not an *array* type.

This function is applicable just to columns of arrays. The contents of the arrays in the specified column will be unchanged up to the lesser of the new and old depths. If the depth is increased, the extra array elements would be flagged as invalid. The pointers to array data may change, therefore pointers previously retrieved by calling cpl_array_get_data_int(), cpl_array_get_data_string(), etc. should be discarded.

4.46.3.151 cpl_table_set_column_dimensions()

```
cpl_error_code cpl_table_set_column_dimensions (
            cpl_table * table,
            const char * name,
            const cpl_array * dimensions )
```

Set the dimensions of a table column of arrays.

Parameters

table	Pointer to table.
name	Column name.
dimensions	Integer array containing the sizes of the column dimensions

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_ILLEGAL_INPUT	Either the specified column is not of type *array*, or the *dimensions* array contains invalid elements.
CPL_ERROR_TYPE_MISMATCH	The *dimensions* array is not of type CPL_TYPE_INT.
CPL_ERROR_INCOMPATIBLE_INPUT	The specified dimensions are incompatible with the total number of elements in the column arrays.

Set the number of dimensions of a column. If the *dimensions* array has size less than 2, nothing is done and no error is returned.

4.46.3.152 cpl_table_set_column_format()

```
cpl_error_code cpl_table_set_column_format (
```

```
            cpl_table * table,
            const char * name,
            const char * format )
```

Give a new format to a table column.

Parameters

table	Pointer to table.
name	Column name.
format	New format.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.

The input format string is duplicated before being used as the column format. If *format* is a `NULL` pointer, "%%s" will be used if the column is of type `CPL_TYPE_STRING`, "% 1.5e" if the column is of type `CPL_TYPE_FLOAT` or `CPL_TYPE_DOUBLE`, and "% 7d" if it is of type `CPL_TYPE_INT`. The format associated to a column has no effect on any operation performed on columns, and it is used just in the `printf()` calls made while printing a table using the function `cpl_table_dump()`. This information is lost after saving the table in FITS format using `cpl_table_save()`.

Referenced by cpl_table_copy_structure().

4.46.3.153 cpl_table_set_column_invalid()

```
cpl_error_code cpl_table_set_column_invalid (
            cpl_table * table,
            const char * name,
            cpl_size start,
            cpl_size count )
```

Invalidate a column segment.

Parameters

table	Pointer to table.
name	Name of table column to access.
start	Position where to begin invalidation.
count	Number of column elements to invalidate.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are NULL pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *start* is outside the table boundaries.
CPL_ERROR_ILLEGAL_INPUT	*count* is negative.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.

All the column elements in the specified interval are invalidated. In the case of either a *string* or an *array* column, the corresponding strings or arrays are set free. If the sum of *start* and *count* exceeds the number of rows in the table, the column is invalidated up to its end.

4.46.3.154 cpl_table_set_column_unit()

```
cpl_error_code cpl_table_set_column_unit (
            cpl_table * table,
            const char * name,
            const char * unit )
```

Give a new unit to a table column.

Parameters

table	Pointer to table.
name	Column name.
unit	New unit.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are NULL pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.

The input unit string is duplicated before being used as the column unit. If *unit* is a NULL pointer, the column will be unitless. The unit associated to a column has no effect on any operation performed on columns, and it must be considered just an optional description of the content of a column. It is however saved to a FITS file when using cpl_table_save().

Referenced by cpl_table_copy_structure().

4.46.3.155 cpl_table_set_complex()

```
cpl_error_code cpl_table_set_complex (
            cpl_table * table,
            const char * name,
            cpl_size row,
            double complex value )
```

Write a complex value to a complex numerical table column element.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Position where to write value.
value	Value to write.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are NULL pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is not complex, or it is of type *array*.

Write a value to a complex column element. The value is cast to the accessed column type according to the C casting rules. The written value is automatically marked as valid. To invalidate a column value use cpl_table_set_invalid(). Table rows are counted starting from 0.

4.46.3.156 cpl_table_set_double()

```
cpl_error_code cpl_table_set_double (
            cpl_table * table,
            const char * name,
            cpl_size row,
            double value )
```

Write a value to a *double* table column element.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Position where to write value.
value	Value to write.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are NULL pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_DOUBLE.

Write a value to a table column of type CPL_TYPE_DOUBLE. The written value is automatically marked as valid. To invalidate a column value use cpl_table_set_invalid(). Table rows are counted starting from 0.

Note

For automatic conversion to the column type, use the function cpl_table_set().

4.46.3.157 cpl_table_set_double_complex()

```
cpl_error_code cpl_table_set_double_complex (
            cpl_table * table,
            const char * name,
            cpl_size row,
            double complex value )
```

Write a value to a *double* complex table column element.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Position where to write value.
value	Value to write.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are NULL pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.

CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_DOUBLE_COMPLEX.

Write a value to a table column of type CPL_TYPE_DOUBLE_COMPLEX. The written value is automatically marked as valid. To invalidate a column value use cpl_table_set_invalid(). Table rows are counted starting from 0.

Note

For automatic conversion to the column type, use the function cpl_table_set_complex().

4.46.3.158 cpl_table_set_float()

```
cpl_error_code cpl_table_set_float (
            cpl_table * table,
            const char * name,
            cpl_size row,
            float value )
```

Write a value to a *float* table column element.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Position where to write value.
value	Value to write.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are NULL pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_FLOAT.

Write a value to a table column of type CPL_TYPE_FLOAT. The written value is automatically marked as valid. To invalidate a column value use cpl_table_set_invalid(). Table rows are counted starting from 0.

Note

For automatic conversion to the column type, use the function cpl_table_set().

4.46.3.159 cpl_table_set_float_complex()

```
cpl_error_code cpl_table_set_float_complex (
            cpl_table * table,
            const char * name,
            cpl_size row,
            float complex value )
```

Write a value to a *float* complex table column element.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Position where to write value.
value	Value to write.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are NULL pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_FLOA↩ T_COMPLEX.

Write a value to a table column of type CPL_TYPE_FLOAT_COMPLEX. The written value is automatically marked as valid. To invalidate a column value use cpl_table_set_invalid(). Table rows are counted starting from 0.

Note

For automatic conversion to the column type, use the function cpl_table_set_complex().

4.46.3.160 cpl_table_set_int()

```
cpl_error_code cpl_table_set_int (
            cpl_table * table,
            const char * name,
            cpl_size row,
            int value )
```

Write a value to an *integer* table column element.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Position where to write value.
value	Value to write.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are `NULL` pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_INT`.

Write a value to a table column of type `CPL_TYPE_INT`. The written value is automatically marked as valid. To invalidate a column value use `cpl_table_set_invalid()`. Table rows are counted starting from 0.

Note

For automatic conversion to the column type, use the function `cpl_table_set()`.

4.46.3.161 cpl_table_set_invalid()

```
cpl_error_code cpl_table_set_invalid (
            cpl_table * table,
            const char * name,
            cpl_size row )
```

Flag a column element as invalid.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Row where to write a *null*.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are `NULL` pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.

In the case of either a *string* or an *array* column, the corresponding string or array is set free and its pointer is set to `NULL`. For other data types, the corresponding table element is flagged internally as invalid.

4.46.3.162 cpl_table_set_long()

```
cpl_error_code cpl_table_set_long (
            cpl_table * table,
            const char * name,
            cpl_size row,
            long value )
```

Write a value to an *long* table column element.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Position where to write value.
value	Value to write.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are `NULL` pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type `CPL_TYPE_LONG`.

Write a value to a table column of type CPL_TYPE_LONG. The written value is automatically marked as valid. To invalidate a column value use cpl_table_set_invalid(). Table rows are counted starting from 0.

Note

For automatic conversion to the column type, use the function cpl_table_set().

4.46.3.163 cpl_table_set_long_long()

```
cpl_error_code cpl_table_set_long_long (
            cpl_table * table,
            const char * name,
            cpl_size row,
            long long value )
```

Write a value to an *long long* table column element.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Position where to write value.
value	Value to write.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are NULL pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_LONG↩ _LONG.

Write a value to a table column of type CPL_TYPE_LONG_LONG. The written value is automatically marked as valid. To invalidate a column value use cpl_table_set_invalid(). Table rows are counted starting from 0.

Note

For automatic conversion to the column type, use the function cpl_table_set().

4.46.3.164 cpl_table_set_size()

```
cpl_error_code cpl_table_set_size (
            cpl_table * table,
            cpl_size new_length )
```

Resize a table to a new number of rows.

Parameters

table	Pointer to table.
new_length	New number of rows in table.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* is a NULL pointer.
CPL_ERROR_ILLEGAL_INPUT	The specified new length is negative.

The contents of the columns will be unchanged up to the lesser of the new and old sizes. If the table is expanded, the extra table rows would just contain invalid elements. The table selection flags are set back to "all selected". The pointer to column data may change, therefore pointers previously retrieved by calling cpl_table_get_data_int(), cpl_table_get_data_string(), etc. should be discarded.

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_table_get_ncol(), and cpl_table_↩ select_all().

Referenced by cpl_table_erase_selected().

4.46.3.165 cpl_table_set_string()

```
cpl_error_code cpl_table_set_string (
            cpl_table * table,
            const char * name,
            cpl_size row,
            const char * value )
```

Write a character string to a *string* table column element.

Parameters

table	Pointer to table.
name	Name of table column to access.
row	Position where to write the character string.
value	Character string to write.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *name* are NULL pointers.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_TYPE_MISMATCH	The specified column is not of type CPL_TYPE_STRING.

Write a string to a table column of type CPL_TYPE_STRING. The written value can also be a NULL pointer, that is equivalent to a call to cpl_table_set_invalid(). Note that the character string is copied, therefore the original can be modified without affecting the table element. To "plug" a character string directly into a table element, use the function cpl_table_get_data_string().

4.46.3.166 cpl_table_shift_column()

```
cpl_error_code cpl_table_shift_column (
            cpl_table * table,
            const char * name,
            cpl_size shift )
```

Shift the position of numeric or complex column values.

Parameters

table	Pointer to table.
name	Name of table column to shift.
shift	Shift column values by so many rows.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are NULL pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is neither numerical nor complex.
CPL_ERROR_ILLEGAL_INPUT	The absolute value of *shift* is greater than the column length.

The position of all column values is shifted by the specified amount. If *shift* is positive, all values will be moved toward the bottom of the column, otherwise toward its top. In either case as many column elements as the amount

of the *shift* will be left undefined, either at the top or at the bottom of the column according to the direction of the shift. These column elements will be marked as invalid. This function is applicable just to numeric and complex columns, and not to strings and or array types. The selection flags are always set back to "all selected" after this operation.

4.46.3.167 cpl_table_sort()

```
cpl_error_code cpl_table_sort (
            cpl_table * table,
            const cpl_propertylist * reflist )
```

Sort table rows according to columns values.

Parameters

table	Pointer to table.
reflist	Names of reference columns with corresponding sorting mode.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or *reflist* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	Any reference column specified in *reflist* is not found in *table*.
CPL_ERROR_ILLEGAL_INPUT	The size of *reflist* exceeds the total number of columns in *table*, or *reflist* is empty.
CPL_ERROR_TYPE_MISMATCH	The input *reflist* includes properties of type different from `CPL_TYPE_BOOL`.
CPL_ERROR_UNSUPPORTED_MODE	Any reference column specified in *reflist* is of type *array*.

The table rows are sorted according to the values of the specified reference columns. The reference column names are listed in the input *reflist*, that associates to each reference column a boolean value. If the associated value is `FALSE`, the table is sorted according to the ascending values of the specified column, otherwise if the associated value is `TRUE`, the table is sorted according to the descending values of that column. The sorting will be performed by comparing the values of the first reference column; if the compared values are equal, the values from the next column in the list are compared, and so on till the end of the reference columns list. An invalid table element is always treated as if it doesn't fulfill any comparison, i.e., sorting either by increasing or decreasing column values would accumulate invalid elements toward the top of the table. The sorting is made in-place, therefore pointers to data retrieved with calls to `cpl_table_get_data_<TYPE>()` remain valid.

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_malloc(), and cpl_propertylist_get_size().

4.46.3.168 cpl_table_subtract_columns()

```
cpl_error_code cpl_table_subtract_columns (
            cpl_table * table,
            const char * to_name,
            const char * from_name )
```

Subtract two numeric or complex table columns.

Parameters

table	Pointer to table.
to_name	Name of target column.
from_name	Name of column to be subtracted from target column.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or any column name are NULL pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with any specified name is not found in *table*.
CPL_ERROR_INVALID_TYPE	Any specified column is neither numerical non complex, or it is an array column.

The columns are subtracted element by element, and the result of the subtraction is stored in the target column. See the documentation of the function cpl_table_add_columns() for further details.

4.46.3.169 cpl_table_subtract_scalar()

```
cpl_error_code cpl_table_subtract_scalar (
            cpl_table * table,
            const char * name,
            double value )
```

Subtract a constant value from a numerical or complex column.

Parameters

table	Pointer to table.
name	Column name.
value	Value to subtract.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is neither numerical nor complex, or it is an array column.

See the description of the function `cpl_table_add_scalar()`.

4.46.3.170 cpl_table_subtract_scalar_complex()

```
cpl_error_code cpl_table_subtract_scalar_complex (
            cpl_table * table,
            const char * name,
            double complex value )
```

Subtract a constant complex value from a numerical or complex column.

Parameters

table	Pointer to table.
name	Column name.
value	Value to subtract.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* or column *name* are `NULL` pointers.
CPL_ERROR_DATA_NOT_FOUND	A column with the specified *name* is not found in *table*.
CPL_ERROR_INVALID_TYPE	The specified column is neither numerical nor complex, or it is an array column.

See the description of the function `cpl_table_add_scalar_complex()`.

4.46.3.171 cpl_table_unselect_all()

```
cpl_error_code cpl_table_unselect_all (
            cpl_table * table )
```

Unselect all table rows.

Parameters

table	Pointer to table.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* is a NULL pointer.

The table selection flags are all unset, meaning that no table rows are selected.

References CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl_free().

4.46.3.172 cpl_table_unselect_row()

```
cpl_error_code cpl_table_unselect_row (
            cpl_table * table,
            cpl_size row )
```

Flag a table row as unselected.

Parameters

table	Pointer to table.
row	Row to unselect.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Input *table* is a NULL pointer.
CPL_ERROR_ACCESS_OUT_OF_RANGE	The input *table* has zero length, or *row* is outside the table boundaries.

Flag a table row as unselected. Any previous selection is kept.

References CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_free(), and cpl_malloc().

4.46.3.173 cpl_table_unwrap()

```
void* cpl_table_unwrap (
            cpl_table * table,
            const char * name )
```

Unwrap a table column.

Parameters

table	Pointer to table.
name	Name of the column.

Returns

Pointer to internal data buffer, `NULL` in case of error.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a `NULL` pointer.
CPL_ERROR_DATA_NOT_FOUND	A column with the given *name* is not found in *table*.
CPL_ERROR_UNSUPPORTED_MODE	The column with the given *name* is a column of arrays.

This function deallocates all the memory associated to a table column, with the exception of its data buffer. This type of destructor should be used on columns created with the `cpl_table_wrap_<type>()` constructors, if the data buffer specified then was not allocated using the functions of the `cpl_memory` module. In such a case, the data buffer should be deallocated separately. See the documentation of the functions `cpl_table_wrap_<type>()`.

Note

Columns of arrays cannot be unwrapped. Use the function `cpl_table_get_data_array()` to directly access the column data buffer.

References CPL_ERROR_NULL_INPUT, CPL_ERROR_UNSUPPORTED_MODE, cpl_table_get_column_type(), and CPL_TYPE_POINTER.

4.46.3.174 cpl_table_where_selected()

```
cpl_array* cpl_table_where_selected (
            const cpl_table * table )
```

Get array of indexes to selected table rows.

Parameters

table	Pointer to table.

Returns

Indexes to selected table rows, or NULL in case of error.

Errors

CPL_ERROR_NULL_INPUT	Input *table* is a `NULL` pointer.

Get array of indexes to selected table rows. If no rows are selected, an array of size zero is returned. The returned array must be deleted using `cpl_array_delete()`.

References cpl_array_fill_window(), cpl_array_get_data_cplsize(), cpl_array_new(), CPL_ERROR_NULL_INPUT, and CPL_TYPE_SIZE.

4.46.3.175 cpl_table_wrap_double()

```
cpl_error_code cpl_table_wrap_double (
            cpl_table * table,
            double * data,
            const char * name )
```

Create in table a new *double* column obtained from existing data.

Parameters

table	Pointer to table.
name	Name of the new column.
data	Existing data buffer.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a `NULL` pointer.
CPL_ERROR_ILLEGAL_OUTPUT	A column with the same *name* already exists in *table*.

This function creates a new column of type `CPL_TYPE_DOUBLE` that will encapsulate the given data. See the description of `cpl_table_wrap_int()` for further details.

References CPL_ERROR_NULL_INPUT.

4.46.3.176 cpl_table_wrap_double_complex()

```
cpl_error_code cpl_table_wrap_double_complex (
            cpl_table * table,
            double complex * data,
            const char * name )
```

Create in table a new *double* complex column from existing data.

Parameters

table	Pointer to table.
name	Name of the new column.
data	Existing data buffer.

Returns

`CPL_ERROR_NONE` on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a `NULL` pointer.
CPL_ERROR_ILLEGAL_OUTPUT	A column with the same *name* already exists in *table*.

This function creates a new column of type `CPL_TYPE_FLOAT_COMPLEX` that will encapsulate the given data. See the description of `cpl_table_wrap_int()` for further details.

References CPL_ERROR_NULL_INPUT.

4.46.3.177 cpl_table_wrap_float()

```
cpl_error_code cpl_table_wrap_float (
            cpl_table * table,
            float * data,
            const char * name )
```

Create in table a new *float* column obtained from existing data.

Parameters

table	Pointer to table.
name	Name of the new column.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_ILLEGAL_OUTPUT	A column with the same *name* already exists in *table*.

This function creates a new column of type CPL_TYPE_FLOAT that will encapsulate the given data. See the description of cpl_table_wrap_int() for further details.

References CPL_ERROR_NULL_INPUT.

4.46.3.178 cpl_table_wrap_float_complex()

```
cpl_error_code cpl_table_wrap_float_complex (
            cpl_table * table,
            float complex * data,
            const char * name )
```

Create in table a new *float* complex column obtained from existing data.

Parameters

table	Pointer to table.
name	Name of the new column.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_ILLEGAL_OUTPUT	A column with the same *name* already exists in *table*.

This function creates a new column of type CPL_TYPE_FLOAT_COMPLEX that will encapsulate the given data. See the description of cpl_table_wrap_int() for further details.

References CPL_ERROR_NULL_INPUT.

4.46.3.179 cpl_table_wrap_int()

```
cpl_error_code cpl_table_wrap_int (
            cpl_table * table,
            int * data,
            const char * name )
```

Create in table a new *integer* column obtained from existing data.

Parameters

table	Pointer to table where to create the new column.
name	Name of the new column.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_ILLEGAL_OUTPUT	A column with the same *name* already exists in *table*.

This function creates a new column of type CPL_TYPE_INT that will encapsulate the given data. The size of the input data array is not checked in any way, and it is expected to match the number of rows assigned to the given table. The pointed data values are all taken as valid: invalid values should be marked using the functions cpl_table_set_invalid() and cpl_table_set_column_invalid(). The data buffer is not copied, so it should not be deallocated while the table column is still in use: the functions cpl_table_erase_column() or cpl_table_delete() would take care of deallocating it. To avoid problems with the memory managment, the specified data buffer should have been allocated using the functions of the *cpl_memory* module, and statically allocated data should be avoided too. If this is not possible, then the function cpl_table_unwrap() should be used on that column before destroying the table that contains it.

References CPL_ERROR_NULL_INPUT.

4.46.3.180 cpl_table_wrap_long()

```
cpl_error_code cpl_table_wrap_long (
            cpl_table * table,
            long * data,
            const char * name )
```

Create in table a new *long* column obtained from existing data.

Parameters

table	Pointer to table.
name	Name of the new column.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_ILLEGAL_OUTPUT	A column with the same *name* already exists in *table*.

This function creates a new column of type CPL_TYPE_LONG that will encapsulate the given data. See the description of cpl_table_wrap_int() for further details.

References CPL_ERROR_NULL_INPUT.

4.46.3.181 cpl_table_wrap_long_long()

```
cpl_error_code cpl_table_wrap_long_long (
            cpl_table * table,
            long long * data,
            const char * name )
```

Create in table a new *long long* column obtained from existing data.

Parameters

table	Pointer to table.
name	Name of the new column.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_ILLEGAL_OUTPUT	A column with the same *name* already exists in *table*.

This function creates a new column of type CPL_TYPE_LONG_LONG that will encapsulate the given data. See the description of cpl_table_wrap_int() for further details.

References CPL_ERROR_NULL_INPUT.

4.46.3.182 cpl_table_wrap_string()

```
cpl_error_code cpl_table_wrap_string (
            cpl_table * table,
            char ** data,
            const char * name )
```

Create in table a new *string* column obtained from existing data.

Parameters

table	Pointer to table.
name	Name of the new column.
data	Existing data buffer.

Returns

CPL_ERROR_NONE on success.

Errors

CPL_ERROR_NULL_INPUT	Any argument is a NULL pointer.
CPL_ERROR_ILLEGAL_OUTPUT	A column with the same *name* already exists in *table*.

This function creates a new column of type CPL_TYPE_STRING that will encapsulate the given data. See the description of cpl_table_wrap_int() for further details, especially with regard to memory management. In the specific case of *string* columns the described restrictions applies also to the single column elements (strings). To deallocate specific column elements the functions cpl_table_set_invalid() and cpl_table_set_column_invalid() should be used.

References CPL_ERROR_NULL_INPUT.

4.47 Type codes

Functions

- const char * cpl_type_get_name (cpl_type type)

  Get a string with the name of a type, e.g. "char", "int", "float".
- size_t cpl_type_get_sizeof (cpl_type type)

  Compute the size of a type.

- enum _cpl_type_ {
  CPL_TYPE_FLAG_ARRAY,
  CPL_TYPE_INVALID,
  CPL_TYPE_CHAR,
  CPL_TYPE_UCHAR,
  CPL_TYPE_BOOL,
  CPL_TYPE_SHORT,
  CPL_TYPE_USHORT,
  CPL_TYPE_INT,
  CPL_TYPE_UINT,
  CPL_TYPE_LONG,
  CPL_TYPE_ULONG,
  CPL_TYPE_LONG_LONG,
  CPL_TYPE_SIZE,
  CPL_TYPE_FLOAT,
  CPL_TYPE_DOUBLE,
  CPL_TYPE_POINTER,
  CPL_TYPE_COMPLEX,
  CPL_TYPE_UNSPECIFIED,
  CPL_TYPE_BITMASK,
  CPL_TYPE_STRING,
  CPL_TYPE_FLOAT_COMPLEX,
  CPL_TYPE_DOUBLE_COMPLEX }

  The CPL type codes and flags.
- typedef enum _cpl_type_ cpl_type

  The type code type.
- typedef long long cpl_size

  The type used for sizes and indices in CPL.
- typedef uint64_t cpl_bitmask

  The CPL bitmask type for bitmask operations.
- #define CPL_SIZE_MIN

  Minimum value a variable of type cpl_size can hold.
- #define CPL_SIZE_MAX

  Maximum value a variable of type cpl_size can hold.
- #define CPL_SIZE_FORMAT

  The format specifier for the type cpl_size.

4.47.1 Detailed Description

This module provides codes for the basic types (including `char`, `int`, `float`, etc.) used in CPL. These type codes may be used to indicate the type of a value stored in another object, the value of a property or the pixel of an image for instance. In addition, a utility function is provided to compute the size, which is required to store a value of the type indicated by a given type code.

The module

Synopsis

```
#include <cpl_type.h>
```

4.47.2 Macro Definition Documentation

4.47.2.1 CPL_SIZE_FORMAT

```
#define CPL_SIZE_FORMAT
```

The format specifier for the type cpl_size.

Note

It is "ld" when cpl_size is a `long` int and "d" when it is an `int`

See also

cpl_size

It can be used like this:

```
cpl_size i = my_index();
return cpl_sprintf("The index is %" CPL_SIZE_FORMAT "\n", i);
```

4.47.2.2 CPL_SIZE_MAX

```
#define CPL_SIZE_MAX
```

Maximum value a variable of type *cpl_size* can hold.

4.47.2.3 CPL_SIZE_MIN

```
#define CPL_SIZE_MIN
```

Minimum value a variable of type *cpl_size* can hold.

4.47.3 Typedef Documentation

4.47.3.1 cpl_bitmask

```
typedef uint64_t cpl_bitmask
```

The CPL bitmask type for bitmask operations.

Note

The CPL bitmask is currently used only for bit-wise operations on CPL images of integer pixel type, which are 32-bits wide. For forward compatibility the CPL bitmask is 64 bits wide, which is cast to 32 bits when used with a CPL_TYPE_INT.

4.47.3.2 cpl_size

```
static cpl_image cpl_size
```

The type used for sizes and indices in CPL.

4.47.3.3 cpl_type

```
typedef enum _cpl_type_ cpl_type
```

The type code type.

4.47.4 Enumeration Type Documentation

4.47.4.1 _cpl_type_

```
enum _cpl_type_
```

The CPL type codes and flags.

Enumerator

CPL_TYPE_FLAG_ARRAY	Flag indicating whether a type is an array or a basic type.
CPL_TYPE_INVALID	Invalid or undetermined type.
CPL_TYPE_CHAR	Type code corresponding to type `char`.
CPL_TYPE_UCHAR	Type code corresponding to type `unsigned` char.
CPL_TYPE_BOOL	Type code corresponding to the boolean type.
CPL_TYPE_SHORT	Type code corresponding to type `short`.
CPL_TYPE_USHORT	Type code corresponding to type `unsigned` short.
CPL_TYPE_INT	Type code corresponding to type `int`.

Enumerator

CPL_TYPE_UINT	Type code corresponding to type `unsigned` int.
CPL_TYPE_LONG	Type code corresponding to type `long`.
CPL_TYPE_ULONG	Type code corresponding to type `unsigned` long.
CPL_TYPE_LONG_LONG	Type code corresponding to type `long long`.
CPL_TYPE_SIZE	Type code corresponding to type `cpl_size`
CPL_TYPE_FLOAT	Type code corresponding to type `float`.
CPL_TYPE_DOUBLE	Type code corresponding to type `double`.
CPL_TYPE_POINTER	Type code corresponding to a pointer type.
CPL_TYPE_COMPLEX	Type code corresponding to a complex type.
CPL_TYPE_UNSPECIFIED	Type code to be used for inheritance of original FITS type.
CPL_TYPE_BITMASK	Type code corresponding to type `cpl_bitmask`
CPL_TYPE_STRING	Type code corresponding to a character array.
CPL_TYPE_FLOAT_COMPLEX	Type code corresponding to type `float` complex.
CPL_TYPE_DOUBLE_COMPLEX	Type code corresponding to type `double` complex.

4.47.5 Function Documentation

4.47.5.1 cpl_type_get_name()

```
const char* cpl_type_get_name (
            cpl_type type )
```

Get a string with the name of a type, e.g. "char", "int", "float".

Parameters

type	Type code to be evaluated.

Returns

A pointer to a string literal with the name or empty string on error.

References CPL_ERROR_INVALID_TYPE, CPL_TYPE_BOOL, CPL_TYPE_CHAR, CPL_TYPE_DOUBLE, CP↩L_TYPE_DOUBLE_COMPLEX, CPL_TYPE_FLAG_ARRAY, CPL_TYPE_FLOAT, CPL_TYPE_FLOAT_COMPL↩EX, CPL_TYPE_INT, CPL_TYPE_INVALID, CPL_TYPE_LONG, CPL_TYPE_LONG_LONG, CPL_TYPE_POIN↩TER, CPL_TYPE_SHORT, CPL_TYPE_SIZE, CPL_TYPE_STRING, CPL_TYPE_UCHAR, CPL_TYPE_UINT, C↩PL_TYPE_ULONG, CPL_TYPE_UNSPECIFIED, and CPL_TYPE_USHORT.

Referenced by cpl_array_new(), cpl_fit_image_gaussian(), cpl_image_and(), cpl_image_and_scalar(), cpl_↩image_cast(), cpl_image_dump_structure(), cpl_image_exponential(), cpl_image_hypot(), cpl_image_logarithm(), cpl_image_not(), cpl_image_or(), cpl_image_or_scalar(), cpl_image_power(), cpl_image_xor(), cpl_image_xor_↩scalar(), cpl_parameter_get_bool(), cpl_parameter_get_default_bool(), cpl_parameter_get_default_double(), cpl↩_parameter_get_default_int(), cpl_parameter_get_default_string(), cpl_parameter_get_double(), cpl_parameter_↩get_enum_double(), cpl_parameter_get_enum_int(), cpl_parameter_get_enum_size(), cpl_parameter_get_enum↩_string(), cpl_parameter_get_int(), cpl_parameter_get_range_max_double(), cpl_parameter_get_range_max_int(),

cpl_parameter_get_range_min_double(), cpl_parameter_get_range_min_int(), cpl_parameter_get_string(), cpl_↩
parameter_set_bool(), cpl_parameter_set_default_bool(), cpl_parameter_set_default_double(), cpl_parameter_↩
set_default_int(), cpl_parameter_set_default_string(), cpl_parameter_set_double(), cpl_parameter_set_int(), cpl↩
_parameter_set_string(), cpl_property_get_bool(), cpl_property_get_char(), cpl_property_get_double(), cpl_↩
property_get_double_complex(), cpl_property_get_float(), cpl_property_get_float_complex(), cpl_property_get_↩
int(), cpl_property_get_long(), cpl_property_get_long_long(), cpl_property_get_string(), cpl_property_set_bool(), cpl_property_set_char(), cpl_property_set_double(), cpl_property_set_double_complex(), cpl_property_set_float(), cpl_property_set_float_complex(), cpl_property_set_int(), cpl_property_set_long(), cpl_property_set_long_long(), and cpl_property_set_string().

4.47.5.2 cpl_type_get_sizeof()

```
size_t cpl_type_get_sizeof (
             cpl_type type )
```

Compute the size of a type.

Parameters

type	Type code to be evaluated.

Returns

The size of the fundamental type, or 0 in case an invalid type code was given.

The function computes the atomic size of the type *type*. The result for fundamental types like `CPL_TYPE_FLOAT` is what you would expect from the C **sizeof()** operator. For arrays, i.e. types having the `CPL_TYPE_FLAG_ARRAY` set the returned size is not the size of a pointer to `CPL_TYPE_FLOAT` for instance, but the size of its fundamental type, i.e. the returned size is same as for the type `CPL_TYPE_FLOAT`.

Especially for the type `CPL_TYPE_STRING`, which is explicitly defined for convenience reasons, the size returned by this function is the size of `CPL_TYPE_CHAR`!

References CPL_TYPE_BOOL, CPL_TYPE_CHAR, CPL_TYPE_FLAG_ARRAY, CPL_TYPE_INVALID, CPL_T↩
YPE_UCHAR, and CPL_TYPE_UNSPECIFIED.

Referenced by cpl_fit_imagelist_polynomial_window(), cpl_image_and_scalar(), cpl_image_cast(), cpl_image↩
_copy(), cpl_image_duplicate(), cpl_image_filter(), cpl_image_filter_mask(), cpl_image_not(), cpl_image_or↩
_scalar(), cpl_image_shift(), cpl_image_xor_scalar(), cpl_property_get_double(), cpl_property_get_double_↩
complex(), cpl_property_get_float(), cpl_property_get_float_complex(), cpl_property_set_bool(), cpl_property↩
_set_char(), cpl_property_set_double(), cpl_property_set_double_complex(), cpl_property_set_float(), cpl_↩
property_set_float_complex(), cpl_property_set_int(), cpl_property_set_long(), cpl_property_set_long_long(), and cpl_test_get_bytes_image().

4.48 Unit testing functions

Macros

- #define cpl_test(bool)

  Evaluate an expression and increment an internal counter if zero.
- #define cpl_test_abs(first, second, tolerance)

  Test if two numerical expressions are within a given absolute tolerance.
- #define cpl_test_abs_complex(first, second, tolerance)

  Test if two complex expressions are within a given absolute tolerance.
- #define cpl_test_array_abs(first, second, tolerance)

  Test if two numerical arrays are identical within a given (absolute) tolerance.
- #define cpl_test_assert(bool)

  Evaluate an expression and terminate the process if it fails.
- #define cpl_test_eq(first, second)

  Test if two integer expressions are equal.
- #define cpl_test_eq_error(first, second)

  Test if two error expressions are equal and reset the CPL error code.
- #define cpl_test_eq_mask(first, second)

  Test if two CPL masks are equal.
- #define cpl_test_eq_ptr(first, second)

  Test if two pointer expressions are equal.
- #define cpl_test_eq_string(first, second)

  Test if two strings are equal.
- #define cpl_test_error(error)

  Test and reset the CPL error code.
- #define cpl_test_errorstate(errorstate)

  Test and if necessary reset the CPL errorstate.
- #define cpl_test_fits(fitsfile)

  Test if a file is valid FITS using an external verification utility.
- #define cpl_test_image_abs(first, second, tolerance)

  Test if two images are identical within a given (absolute) tolerance.
- #define cpl_test_image_rel(first, second, tolerance)

  Test if two images are identical within a given (relative) tolerance.
- #define cpl_test_imagelist_abs(first, second, tolerance)

  Test if two imagelists are identical within a given (absolute) tolerance.
- #define cpl_test_init(REPORT, LEVEL)

  Initialize CPL + CPL messaging + unit test.
- #define cpl_test_leq(value, tolerance)

  Evaluate A <= B and increment an internal counter if it is not true.
- #define cpl_test_lt(value, tolerance)

  Evaluate A < B and increment an internal counter if it is not true.
- #define cpl_test_matrix_abs(first, second, tolerance)

  Test if two matrices are identical within a given (absolute) tolerance.
- #define cpl_test_memory_is_empty()

  Test if the memory system is empty.
- #define cpl_test_noneq(first, second)

  Test if two integer expressions are not equal.
- #define cpl_test_noneq_ptr(first, second)

  Test if two pointer expressions are not equal.

- #define cpl_test_noneq_string(first, second)
  Test if two strings are not equal.
- #define cpl_test_nonnull(pointer)
  Test if a pointer is non-NULL.
- #define cpl_test_null(pointer)
  Test if a pointer is NULL and update an internal counter on failure.
- #define cpl_test_polynomial_abs(first, second, tolerance)
  Test if two polynomials are identical within a given (absolute) tolerance.
- #define cpl_test_rel(first, second, tolerance)
  Test if two numerical expressions are within a given relative tolerance.
- #define cpl_test_vector_abs(first, second, tolerance)
  Test if two vectors are identical within a given (absolute) tolerance.
- #define cpl_test_zero(zero)
  Evaluate an expression and increment an internal counter if non-zero.

Functions

- int cpl_test_end (cpl_size nfail)
  Finalize CPL and unit-testing environment and report any failures.
- size_t cpl_test_get_bytes_image (const cpl_image *self)
  Get the amount of storage [bytes] for the CPL object.
- size_t cpl_test_get_bytes_imagelist (const cpl_imagelist *self)
  Get the amount of storage [bytes] for the CPL object.
- size_t cpl_test_get_bytes_matrix (const cpl_matrix *self)
  Get the amount of storage [bytes] for the CPL object.
- size_t cpl_test_get_bytes_vector (const cpl_vector *self)
  Get the amount of storage [bytes] for the CPL object.
- cpl_size cpl_test_get_failed (void)
  Get the number of failed CPL tests.
- cpl_size cpl_test_get_tested (void)
  Get the number of CPL tests performed.
- double cpl_test_get_walltime (void)
  Get the process wall-clock time, when available.

4.48.1 Detailed Description

This module provides various functions for unit testing.

Synopsis:

```
#include "cpl_test.h"
```

4.48.2 Macro Definition Documentation

4.48.2.1 cpl_test

```
#define cpl_test(
              bool )
```

Evaluate an expression and increment an internal counter if zero.

Parameters

bool	The expression to evaluate, side-effects are allowed

Note

A zero value of the expression is a failure, other values are not

Returns

void

See also

cpl_test_init()

Note

This macro should be used for unit tests

Example of usage:

```
cpl_test(myfunc()); // myfunc() is expected to return non-zero
```

4.48.2.2 cpl_test_abs

```
#define cpl_test_abs(
              first,
              second,
              tolerance )
```

Test if two numerical expressions are within a given absolute tolerance.

Parameters

first	The first value in the comparison, side-effects are allowed
second	The second value in the comparison, side-effects are allowed
tolerance	A non-negative tolerance

Note

If the tolerance is negative, the test will always fail

See also

cpl_test()

Example of usage:

```
cpl_test_abs(computed, expected, DBL_EPSILON);
```

4.48.2.3 cpl_test_abs_complex

```
#define cpl_test_abs_complex(
              first,
              second,
              tolerance )
```

Test if two complex expressions are within a given absolute tolerance.

Parameters

first	The first value in the comparison, side-effects are allowed
second	The second value in the comparison, side-effects are allowed
tolerance	A non-negative tolerance

Note

If the tolerance is negative, the test will always fail

See also

cpl_test()

Example of usage:

```
cpl_test_abs_complex(computed, expected, DBL_EPSILON);
```

4.48.2.4 cpl_test_array_abs

```
#define cpl_test_array_abs(
              first,
              second,
              tolerance )
```

Test if two numerical arrays are identical within a given (absolute) tolerance.

Parameters

first	The first array in the comparison
second	The second array of identical size in the comparison
tolerance	A non-negative tolerance

Returns

void

See also

cpl_test_abs()

Note

The test will fail if one or both the arrays are NULL or of a non-numerical type

4.48.2.5 cpl_test_assert

```
#define cpl_test_assert(
              bool )
```

Evaluate an expression and terminate the process if it fails.

Parameters

bool	The (boolean) expression to evaluate, side-effects are allowed

Note

A zero value of the expression is a failure, other values are not

Returns

void

See also

cpl_test()

Note

This macro should be used for unit tests that cannot continue after a failure.

Example of usage:

```
int main (void)
{
    cpl_test_init(CPL_MSG_WARNING);
    cpl_test(myfunc(&p));
    cpl_test_assert(p != NULL);
    cpl_test(*p);
    return cpl_test_end(0);
}
```

4.48.2.6 cpl_test_eq

```
#define cpl_test_eq(
              first,
              second )
```

Test if two integer expressions are equal.

Parameters

first	The first value in the comparison, side-effects are allowed
second	The second value in the comparison, side-effects are allowed

See also

cpl_test()

Returns

void

Example of usage:
```
cpl_test_eq(computed, expected);
```

For comparison of floating point values, see cpl_test_abs() and cpl_test_rel().

4.48.2.7 cpl_test_eq_error

```
#define cpl_test_eq_error(
              first,
              second )
```

Test if two error expressions are equal and reset the CPL error code.

Parameters

first	The first value in the comparison
second	The second value in the comparison

See also

cpl_test_error

Note

If the two CPL error expressions are equal they will also be tested against the current CPL error code. After the test(s), the CPL errorstate is reset.

Example of usage:
```
cpl_error_code error = my_func(NULL);
// my_func(NULL) is expected to return CPL_ERROR_NULL_INPUT
// and to set the same error code
cpl_test_eq_error(error, CPL_ERROR_NULL_INPUT);
// The errorstate has been reset.
error = my_func(p); // Successful call
cpl_test_eq_error(error, CPL_ERROR_NONE);
```

4.48.2.8 cpl_test_eq_mask

```
#define cpl_test_eq_mask(
              first,
              second )
```

Test if two CPL masks are equal.

Parameters

first	The first mask or NULL of the comparison
second	The second mask or NULL of the comparison

Note

One or two NULL pointer(s) is considered a failure.

Example of usage:
```
cpl_test_eq_mask(computed, expected);
```

4.48.2.9 cpl_test_eq_ptr

```
#define cpl_test_eq_ptr(
              first,
              second )
```

Test if two pointer expressions are equal.

Parameters

first	The first value in the comparison, side-effects are allowed
second	The second value in the comparison, side-effects are allowed

See also

cpl_test_eq()

Returns

void

Example of usage:
```
cpl_test_eq_ptr(computed, expected);
```

4.48.2.10 cpl_test_eq_string

```
#define cpl_test_eq_string(
              first,
              second )
```

Test if two strings are equal.

Parameters

first	The first string or NULL of the comparison
second	The second string or NULL of the comparison

Note

One or two NULL pointer(s) is considered a failure.

Example of usage:

```
cpl_test_eq_string(computed, expected);
```

4.48.2.11 cpl_test_error

```
#define cpl_test_error(
              error )
```

Test and reset the CPL error code.

Parameters

error	The expected CPL error code (incl. CPL_ERROR_NONE)

See also

cpl_test()

Note

After the test, the CPL errorstate is reset

Returns

void

Example of usage:

```
cpl_test( my_func(NULL) ); // my_func(NULL) is expected to return non-zero
cpl_test_error(CPL_ERROR_NULL_INPUT); // ... and to set this error code
// The errorstate has been reset.
cpl_test( !my_func(p) ); // my_func(p) is expected to return zero
```

4.48.2.12 cpl_test_errorstate

```
#define cpl_test_errorstate(
              errorstate )
```

Test and if necessary reset the CPL errorstate.

Parameters

errorstate	The expected CPL errorstate

See also

cpl_test()

Note

After the test, the CPL errorstate is set to the provided state

Returns

void

This function is useful for verifying that a successful call to a function does not modify any pre-existing errors.

Example of usage:

```
const cpl_error_code error = cpl_error_set(cpl_func, CPL_ERROR_EOL);
cpl_errorstate prestate = cpl_errorstate_get();
const cpl_error_code ok = my_func(); // Expected to succeed
cpl_test_errorstate(prestate); // Verify that no additional errors occurred
cpl_test_error(CPL_ERROR_EOL); // Reset error
cpl_test_eq(ok, CPL_ERROR_NONE); // Verify that my_func() succeeded
```

4.48.2.13 cpl_test_fits

```
#define cpl_test_fits(
              fitsfile )
```

Test if a file is valid FITS using an external verification utility.

Parameters

fitsfile	The file to verify, NULL causes failure

Note

The external verification utility is specified with the environemt variable CPL_TEST_FITS, if is not set the test will pass on any non-NULL file.

Example of usage:

```
export CPL_TEST_FITS=/usr/local/bin/fitsverify
cpl_test_fits(fitsfile);
```

4.48.2.14 cpl_test_image_abs

```
#define cpl_test_image_abs(
              first,
              second,
              tolerance )
```

Test if two images are identical within a given (absolute) tolerance.

Parameters

first	The first image in the comparison
second	The second image of identical size in the comparison
tolerance	A non-negative tolerance

Returns

void

See also

cpl_test_abs()

Note

The test will fail if one or both the images are NULL

4.48.2.15 cpl_test_image_rel

```
#define cpl_test_image_rel(
              first,
              second,
              tolerance )
```

Test if two images are identical within a given (relative) tolerance.

Parameters

first	The first image in the comparison
second	The second image of identical size in the comparison
tolerance	A non-negative tolerance

Returns

void

See also

cpl_test_rel()

Note

The test will fail if one or both the images are NULL

For each pixel position the two values x, y must pass the test: $|x - y| <= tol * min(|x|, |y|)$. This definition is chosen since it is commutative and meaningful also for zero-valued pixels.

4.48.2.16 cpl_test_imagelist_abs

```
#define cpl_test_imagelist_abs(
              first,
              second,
              tolerance )
```

Test if two imagelists are identical within a given (absolute) tolerance.

Parameters

first	The first imagelist in the comparison
second	The second imagelist of identical size in the comparison
tolerance	A non-negative tolerance

Returns

void

See also

cpl_test_image_abs()

Note

The test will fail if one or both the imagelists are NULL

4.48.2.17 cpl_test_init

```
#define cpl_test_init(
              REPORT,
              LEVEL )
```

Initialize CPL + CPL messaging + unit test.

Parameters

REPORT	The email address for the error message e.g. PACKAGE_BUGREPORT
LEVEL	The default messaging level, e.g. CPL_MSG_WARNING

Returns

void

See also

cpl_init()

Note

This macro should be used at the beginning of main() of a unit test instead of cpl_init() and before any other CPL function call.

4.48.2.18 cpl_test_leq

```
#define cpl_test_leq(
              value,
              tolerance )
```

Evaluate A <= B and increment an internal counter if it is not true.

Parameters

value	The number to test
tolerance	The upper limit to compare against

Returns

void

See also

cpl_test_init()

Note

This macro should be used for unit tests

Example of usage:

```
cpl_test_leq(fabs(myfunc(&p)), DBL_EPSILON);
cpl_test_nonnull(p);
```

4.48.2.19 cpl_test_lt

```
#define cpl_test_lt(
              value,
              tolerance )
```

Evaluate A $<$ B and increment an internal counter if it is not true.

Parameters

value	The number to test
tolerance	The upper limit to compare against

Returns

void

See also

cpl_test_init()

Note

This macro should be used for unit tests

Example of usage:

```
cpl_test_lt(0.0, myfunc());
```

4.48.2.20 cpl_test_matrix_abs

```
#define cpl_test_matrix_abs(
              first,
              second,
              tolerance )
```

Test if two matrices are identical within a given (absolute) tolerance.

Parameters

first	The first matrix in the comparison
second	The second matrix of identical size in the comparison
tolerance	A non-negative tolerance

Returns

void

See also

cpl_test_abs()

Note

The test will fail if one or both the matrices are NULL

4.48.2.21 cpl_test_memory_is_empty

```
#define cpl_test_memory_is_empty( )
```

Test if the memory system is empty.

See also

cpl_memory_is_empty()

Deprecated Called by cpl_test_end()

4.48.2.22 cpl_test_noneq

```
#define cpl_test_noneq(
              first,
              second )
```

Test if two integer expressions are not equal.

Parameters

first	The first value in the comparison, side-effects are allowed
second	The second value in the comparison, side-effects are allowed

See also

cpl_test_eq()

Returns

void

Example of usage:

```
cpl_test_noneq(computed, wrong);
```

4.48.2.23 cpl_test_noneq_ptr

```
#define cpl_test_noneq_ptr(
              first,
              second )
```

Test if two pointer expressions are not equal.

Parameters

first	The first value in the comparison, side-effects are allowed
second	The second value in the comparison, side-effects are allowed

See also

cpl_test_eq_ptr()

Returns

void

Example of usage:
```
cpl_test_noneq_ptr(computed, wrong);
```

4.48.2.24 cpl_test_noneq_string

```
#define cpl_test_noneq_string(
              first,
              second )
```

Test if two strings are not equal.

Parameters

first	The first string or NULL of the comparison
second	The second string or NULL of the comparison

Note

One or two NULL pointer(s) is considered a failure.

Example of usage:
```
cpl_test_noneq_string(computed, expected);
```

4.48.2.25 cpl_test_nonnull

```
#define cpl_test_nonnull(
              pointer )
```

Test if a pointer is non-NULL.

Parameters

pointer	The pointer to check, side-effects are allowed

See also

cpl_test_nonnull()

Returns

void

Example of usage:
```
cpl_test_nonnull(pointer); // pointer is expected to be non-NULL
```

4.48.2.26 cpl_test_null

```
#define cpl_test_null(
              pointer )
```

Test if a pointer is NULL and update an internal counter on failure.

Parameters

pointer	The NULL-pointer to check, side-effects are allowed

See also

cpl_test()

Returns

void

Example of usage:
```
cpl_test_null(pointer); // pointer is expected to be NULL
```

4.48.2.27 cpl_test_polynomial_abs

```
#define cpl_test_polynomial_abs(
              first,
              second,
              tolerance )
```

Test if two polynomials are identical within a given (absolute) tolerance.

Parameters

first	The first polynomial in the comparison
second	The second polynomial in the comparison
tolerance	A non-negative tolerance

Returns

void

See also

cpl_test_abs(), cpl_polynomial_compare()

Note

The test will fail if one or both the polynomials are NULL

4.48.2.28 cpl_test_rel

```
#define cpl_test_rel(
              first,
              second,
              tolerance )
```

Test if two numerical expressions are within a given relative tolerance.

Parameters

first	The first value in the comparison, side-effects are allowed
second	The second value in the comparison, side-effects are allowed
tolerance	A non-negative tolerance

Note

If the tolerance is negative or if one but not both of the two values is zero, the test will always fail. If both values are zero, the test will succeed for any non-negative tolerance. The test is commutative in the two values.

See also

cpl_test()

The test is carried out by comparing the absolute value of the difference abs (*first* - *second*) to the product of the tolerance and the minimum of the absolute value of the two values, tolerance ∗ min(abs(*first*), abs(*second*)) (The test is implemented like this to avoid division with a number that may be zero.

Example of usage:

```
cpl_test_rel(computed, expected, 0.001);
```

4.48.2.29 cpl_test_vector_abs

```
#define cpl_test_vector_abs(
              first,
              second,
              tolerance )
```

Test if two vectors are identical within a given (absolute) tolerance.

Parameters

first	The first vector in the comparison
second	The second vector of identical size in the comparison
tolerance	A non-negative tolerance

Returns

void

See also

cpl_test_abs()

Note

The test will fail if one or both the vectors are NULL

4.48.2.30 cpl_test_zero

```
#define cpl_test_zero(
              zero )
```

Evaluate an expression and increment an internal counter if non-zero.

Parameters

zero	The numerical expression to evaluate, side-effects are allowed

Note

A zero value of the expression is a success, other values are not

Returns

void

See also

cpl_test()

Note

This macro should be used for unit tests

Example of usage:

```
cpl_test_zero(myfunc()); // myfunc() is expected to return zero
```

4.48.3 Function Documentation

4.48.3.1 cpl_test_end()

```
int cpl_test_end (
            cpl_size nfail )
```

Finalize CPL and unit-testing environment and report any failures.

Parameters

nfail	The number of failures counted apart from cpl_test() et al.

Returns

EXIT_SUCCESS iff the CPL errorstate is clean

Note

This function should be used for the final return from a unit test

See also

cpl_test_init()

nfail should normally be zero, but may be set to a positive number when it is necessary to ensure a failure. nfail should only be negative in the unit test of the unit-test functions themselves.

Example of usage:

```
int main (void)
{
    cpl_test_init(PACKAGE_BUGREPORT, CPL_MSG_WARNING);
    cpl_test(myfunc(&p));
    cpl_test(p != NULL);
    return cpl_test_end(0);
}
```

4.48.3.2 cpl_test_get_bytes_image()

```
size_t cpl_test_get_bytes_image (
            const cpl_image * self )
```

Get the amount of storage [bytes] for the CPL object.

Parameters

self	The CPL object

Returns

The size in bytes

Note

Passing NULL is allowed and will return zero

See also

cpl_test_get_bytes_vector

References cpl_image_get_size_x(), cpl_image_get_size_y(), cpl_image_get_type(), and cpl_type_get_sizeof().

Referenced by cpl_test_get_bytes_imagelist().

4.48.3.3 cpl_test_get_bytes_imagelist()

```
size_t cpl_test_get_bytes_imagelist (
            const cpl_imagelist * self )
```

Get the amount of storage [bytes] for the CPL object.

Parameters

self	The CPL object

Returns

The size in bytes

Note

Passing NULL is allowed and will return zero

See also

cpl_test_get_bytes_vector

References cpl_imagelist_get_const(), cpl_imagelist_get_size(), and cpl_test_get_bytes_image().

4.48.3.4 cpl_test_get_bytes_matrix()

```
size_t cpl_test_get_bytes_matrix (
            const cpl_matrix * self )
```

Get the amount of storage [bytes] for the CPL object.

Parameters

self	The CPL object

Returns

The size in bytes

Note

Passing NULL is allowed and will return zero

See also

cpl_test_get_bytes_vector

References cpl_matrix_get_ncol(), and cpl_matrix_get_nrow().

4.48.3.5 cpl_test_get_bytes_vector()

```
size_t cpl_test_get_bytes_vector (
            const cpl_vector * self )
```

Get the amount of storage [bytes] for the CPL object.

Parameters

self	The CPL object

Returns

The size in bytes

Note

Passing NULL is allowed and will return zero

Example of usage:

```
int my_benchmark (void)
{
    const size_t storage = cpl_test_get_bytes_vector(mydata);
    double walltime, tstop;
    const double tstart = cpl_test_get_walltime();
    myfunc(mydata);
    tstop = cpl_test_get_walltime();
    walltime = tstop - tstart;
    if (walltime > 0.0) {
      cpl_msg_info(cpl_func, "Processing rate: %g",
                   (double)storage/walltime);
    }
}
```

References cpl_vector_get_size().

4.48.3.6 cpl_test_get_failed()

```
cpl_size cpl_test_get_failed (
             void  )
```

Get the number of failed CPL tests.

Returns

The count of failed tests

See also

cpl_test_get_tested()

Example of usage:

```
void my_tester (void)
{
    const cpl_size prefailed = cpl_test_get_failed();
    cpl_test(mytest());
    if (cpl_test_get_failed() > prefailed) {
      cpl_msg_info(cpl_func, "The function mytest() failed!");
    }
}
```

4.48.3.7 cpl_test_get_tested()

```
cpl_size cpl_test_get_tested (
            void  )
```

Get the number of CPL tests performed.

Returns

The test count

See also

cpl_test_get_failed()

4.48.3.8 cpl_test_get_walltime()

```
double cpl_test_get_walltime (
            void  )
```

Get the process wall-clock time, when available.

Returns

The process wall-clock time in seconds.

Note

Will always return 0 if clock_gettime() and gettimeofday() are unavailable or failing

See also

clock_gettime(), gettimeofday()

Example of usage:

```
int my_benchmark (void)
{
    double walltime, tstop;
    const double tstart = cpl_test_get_walltime();
    myfunc();
    tstop = cpl_test_get_walltime();
    walltime = tstop - tstart;
    cpl_msg_info(cpl_func, "The call took %g seconds of wall-clock time",
                 walltime);
}
```

4.49 Vector

Functions

- cpl_error_code cpl_vector_add (cpl_vector *v1, const cpl_vector *v2)

  Add a cpl_vector to another.
- cpl_error_code cpl_vector_add_scalar (cpl_vector *v, double addend)

  Elementwise addition of a scalar to a vector.
- cpl_error_code cpl_vector_convolve_symmetric (cpl_vector *smoothed, const cpl_vector *conv_kernel)

  Convolve a 1d-signal with a symmetric 1D-signal.
- cpl_error_code cpl_vector_copy (cpl_vector *destination, const cpl_vector *source)

  This function copies contents of a vector into another vector.
- cpl_size cpl_vector_correlate (cpl_vector *vxc, const cpl_vector *v1, const cpl_vector *v2)

  Cross-correlation of two vectors.
- cpl_error_code cpl_vector_cycle (cpl_vector *self, const cpl_vector *other, double shift)

  Perform a cyclic shift to the right of the elements of a vector.
- void cpl_vector_delete (cpl_vector *v)

  Delete a cpl_vector.
- cpl_error_code cpl_vector_divide (cpl_vector *v1, const cpl_vector *v2)

  Divide two vectors element-wise.
- cpl_error_code cpl_vector_divide_scalar (cpl_vector *v, double divisor)

  Elementwise division of a vector with a scalar.
- void cpl_vector_dump (const cpl_vector *v, FILE *stream)

  Dump a cpl_vector as ASCII to a stream.
- cpl_vector * cpl_vector_duplicate (const cpl_vector *v)

  This function duplicates an existing vector and allocates memory.
- cpl_error_code cpl_vector_exponential (cpl_vector *v, double base)

  Compute the exponential of all vector elements.
- cpl_vector * cpl_vector_extract (const cpl_vector *v, cpl_size istart, cpl_size istop, cpl_size istep)

  Extract a sub_vector from a vector.
- cpl_error_code cpl_vector_fill (cpl_vector *v, double val)

  Fill a cpl_vector.
- cpl_error_code cpl_vector_fill_kernel_profile (cpl_vector *profile, cpl_kernel type, double radius)

  Fill a vector with a kernel profile.
- cpl_vector * cpl_vector_filter_lowpass_create (const cpl_vector *v, cpl_lowpass filter_type, cpl_size hw)

  Apply a low-pass filter to a cpl_vector.
- cpl_vector * cpl_vector_filter_median_create (const cpl_vector *self, cpl_size hw)

  Apply a 1D median filter of given half-width to a cpl_vector.
- cpl_size cpl_vector_find (const cpl_vector *sorted, double key)

  In a sorted vector find the element closest to the given value.
- cpl_error_code cpl_vector_fit_gaussian (const cpl_vector *x, const cpl_vector *sigma_x, const cpl_vector *y, const cpl_vector *sigma_y, cpl_fit_mode fit_pars, double *x0, double *sigma, double *area, double *offset, double *mse, double *red_chisq, cpl_matrix **covariance)

  Apply a 1d gaussian fit.
- double cpl_vector_get (const cpl_vector *in, cpl_size idx)

  Get an element of the vector.
- double * cpl_vector_get_data (cpl_vector *in)

  Get a pointer to the data part of the vector.
- const double * cpl_vector_get_data_const (const cpl_vector *in)

  Get a pointer to the data part of the vector.
- double cpl_vector_get_max (const cpl_vector *v)

Get the maximum of the cpl_vector.

- cpl_size cpl_vector_get_maxpos (const cpl_vector *self)

  Get the index of the maximum element of the cpl_vector.
- double cpl_vector_get_mean (const cpl_vector *v)

  Compute the mean value of vector elements.
- double cpl_vector_get_median (cpl_vector *v)

  Compute the median of the elements of a vector.
- double cpl_vector_get_median_const (const cpl_vector *v)

  Compute the median of the elements of a vector.
- double cpl_vector_get_min (const cpl_vector *v)

  Get the minimum of the cpl_vector.
- cpl_size cpl_vector_get_minpos (const cpl_vector *self)

  Get the index of the minimum element of the cpl_vector.
- cpl_size cpl_vector_get_size (const cpl_vector *in)

  Get the size of the vector.
- double cpl_vector_get_stdev (const cpl_vector *v)

  Compute the bias-corrected standard deviation of a vectors elements.
- double cpl_vector_get_sum (const cpl_vector *v)

  Get the sum of the elements of the cpl_vector.
- cpl_vector * cpl_vector_load (const char *filename, cpl_size xtnum)

  Load a list of values from a FITS file.
- cpl_error_code cpl_vector_logarithm (cpl_vector *v, double base)

  Compute the element-wise logarithm.
- cpl_error_code cpl_vector_multiply (cpl_vector *v1, const cpl_vector *v2)

  Multiply two vectors component-wise.
- cpl_error_code cpl_vector_multiply_scalar (cpl_vector *v, double factor)

  Elementwise multiplication of a vector with a scalar.
- cpl_vector * cpl_vector_new (cpl_size n)

  Create a new cpl_vector.
- cpl_vector * cpl_vector_new_lss_kernel (double slitw, double fwhm)

  Create Right Half of a symmetric smoothing kernel for LSS.
- cpl_error_code cpl_vector_power (cpl_vector *v, double exponent)

  Compute the power of all vector elements.
- double cpl_vector_product (const cpl_vector *v1, const cpl_vector *v2)

  Compute the vector dot product.
- cpl_vector * cpl_vector_read (const char *filename)

  Read a list of values from an ASCII file and create a cpl_vector.
- cpl_error_code cpl_vector_save (const cpl_vector *self, const char *filename, cpl_type type, const cpl_propertylist *plist, unsigned mode)

  Save a vector to a FITS file.
- cpl_error_code cpl_vector_set (cpl_vector *in, cpl_size idx, double value)

  Set an element of the vector.
- cpl_error_code cpl_vector_set_size (cpl_vector *in, cpl_size newsize)

  Resize the vector.
- cpl_error_code cpl_vector_sort (cpl_vector *self, cpl_sort_direction dir)

  Sort a cpl_vector.
- cpl_error_code cpl_vector_sqrt (cpl_vector *v)

  Compute the sqrt of a cpl_vector.
- cpl_error_code cpl_vector_subtract (cpl_vector *v1, const cpl_vector *v2)

  Subtract a cpl_vector from another.
- cpl_error_code cpl_vector_subtract_scalar (cpl_vector *v, double subtrahend)

Elementwise subtraction of a scalar from a vector.

- void * cpl_vector_unwrap (cpl_vector *v)

  Delete a cpl_vector except the data array.

- cpl_vector * cpl_vector_wrap (cpl_size n, double *data)

  Create a cpl_vector from existing data.

4.49.1 Detailed Description

This module provides functions to handle *cpl_vector*.

A *cpl_vector* is an object containing a list of values (as doubles) and the (always positive) number of values. The functionalities provided here are simple ones like sorting, statistics, or simple operations. The *cpl_bivector* object is composed of two of these vectors. No special provisions are made to handle special values like NaN or Inf, for data with such elements, the *cpl_array* object may be preferable.

Synopsis:

```
#include "cpl_vector.h"
```

4.49.2 Function Documentation

4.49.2.1 cpl_vector_add()

```
cpl_error_code cpl_vector_add (
            cpl_vector * v1,
            const cpl_vector * v2 )
```

Add a cpl_vector to another.

Parameters

v1	First cpl_vector (modified)
v2	Second cpl_vector

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

The second vector is added to the first one. The input first vector is modified.

The input vectors must have the same size.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if v1 and v2 have different sizes

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NONE, and CPL_ERRO↩R_NULL_INPUT.

4.49.2.2 cpl_vector_add_scalar()

```
cpl_error_code cpl_vector_add_scalar (
            cpl_vector * v,
            double addend )
```

Elementwise addition of a scalar to a vector.

Parameters

v	cpl_vector to modify
addend	Number to add

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

Add a number to each element of the cpl_vector.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure_code, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.49.2.3 cpl_vector_convolve_symmetric()

```
cpl_error_code cpl_vector_convolve_symmetric (
            cpl_vector * smoothed,
            const cpl_vector * conv_kernel )
```

Convolve a 1d-signal with a symmetric 1D-signal.

Parameters

smoothed	Preallocated vector to be smoothed in place
conv_kernel	Vector with symmetric convolution function

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

Deprecated Unstable API, may change or disappear. Do not use in new code!

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_IN↩PUT, cpl_vector_delete(), cpl_vector_duplicate(), cpl_vector_get_data(), cpl_vector_get_data_const(), and cpl_↩vector_get_size().

4.49.2.4 cpl_vector_copy()

```
cpl_error_code cpl_vector_copy (
            cpl_vector * destination,
            const cpl_vector * source )
```

This function copies contents of a vector into another vector.

Parameters

destination	destination cpl_vector
source	source cpl_vector

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_vector_set_size() if source and destination have different sizes.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure_code, cpl_error_get_code(), CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, and cpl↩_vector_set_size().

Referenced by cpl_bivector_copy(), and cpl_vector_duplicate().

4.49.2.5 cpl_vector_correlate()

```
cpl_size cpl_vector_correlate (
            cpl_vector * vxc,
            const cpl_vector * v1,
            const cpl_vector * v2 )
```

Cross-correlation of two vectors.

Parameters

vxc	Odd-sized vector with the computed cross-correlations
v1	1st vector to correlate
v2	2nd vector to correlate

Returns

Index of maximum cross-correlation, or negative on error

vxc must have an odd number of elements, 2∗half_search+1, where half_search is the half-size of the search domain.

The length of v2 may not exceed that of v1. If the difference in length between v1 and v2 is less than half_search then this difference must be even (if the difference is odd resampling of v2 may be useful).

The cross-correlation is computed with shifts ranging from -half_search to half_search.

On succesful return element i (starting with 0) of vxc contains the cross- correlation at offset i-half_search. On error vxc is unmodified.

The cross-correlation is in fact the dot-product of two unit-vectors and ranges therefore from -1 to 1.

The cross-correlation is, in absence of rounding errors, commutative only for equal-sized vectors, i.e. changing the order of v1 and v2 will move element j in vxc to 2∗half_search - j and thus change the return value from i to 2∗half_search - i.

If, in absence of rounding errors, more than one shift would give the maximum cross-correlation, rounding errors may cause any one of those shifts to be returned. If rounding errors have no effect the index corresponding to the shift with the smallest absolute value is returned (with preference given to the smaller of two indices that correspond to the same absolute shift).

If v1 is longer than v2, the first element in v1 used for the resulting cross-correlation is max(0,shift + (cpl_vector_↩ get_size(v1)-cpl_vector_get_size(v2))/2).

Cross-correlation with half_search == 0 requires about 8n FLOPs, where n = cpl_vector_get_size(v2). Each increase of half_search by 1 requires about 4n FLOPs more, when all of v2's elements can be cross-correlated, otherwise the extra cost is about 4m, where m is the number of elements in v2 that can be cross-correlated, n - half_search <= m < n.

In case of error, the _cpl_error_code_ code is set, and the returned delta and cross-correlation is undefined.

Example of 1D-wavelength calibration (error handling omitted for brevity):

```
cpl_vector   * model = my_model(dispersion);
cpl_vector   * vxc   = cpl_vector_new(1+2*maxshift);
const cpl_size shift = cpl_vector_correlate(vxc, model, observed) - maxshift;
cpl_error_code error = cpl_polynomial_shift_1d(dispersion, 0, (double)shift);
cpl_msg_info(cpl_func, "Shifted dispersion relation by %" CPL_SIZE_FORMAT
             " pixels, shift);
```

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if v1 and v2 have different sizes or if vxc is not as requested

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_vector_get_size(), and cpl_vector_set().

4.49.2.6 cpl_vector_cycle()

```
cpl_error_code cpl_vector_cycle (
            cpl_vector * self,
            const cpl_vector * other,
            double shift )
```

Perform a cyclic shift to the right of the elements of a vector.

Parameters

self	Vector to hold the shifted result
other	Vector to read from, or NULL for in-place shift of self
shift	The number of positions to cyclic right-shift

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_fft_image()

Note

Passing two distinct vectors (partially) wrapped around the same buffer is not supported and will lead to undefined behaviour.

A shift of +1 will move the last element to the first, a shift of -1 will move the first element to the last, a zero-shift will perform a copy (or do nothing in case of an in-place operation).

A non-integer shift will perform the shift in the Fourier domain. Large discontinuities in the vector to shift will thus lead to FFT artifacts around each discontinuity.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if the self pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if self and other have different sizes
- CPL_ERROR_UNSUPPORTED_MODE if the shift is non-integer and FFTW is unavailable

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NONE, CPL_ERROR_N↩ ULL_INPUT, cpl_free(), cpl_image_unwrap(), cpl_image_wrap_double(), cpl_malloc(), cpl_vector_get_data(), and cpl_vector_get_size().

4.49.2.7 cpl_vector_delete()

```
void cpl_vector_delete (
            cpl_vector * v )
```

Delete a cpl_vector.

Parameters

v	cpl_vector to delete

Returns

void

If the vector *v* is NULL, nothing is done and no error is set.

References cpl_free().

Referenced by cpl_bivector_delete(), cpl_bivector_read(), cpl_geom_img_offset_combine(), cpl_geom_img_offset_saa(), cpl_image_fill_jacobian_polynomial(), cpl_image_get_fwhm(), cpl_image_warp_polynomial(), cpl_plot_column(), cpl_vector_convolve_symmetric(), cpl_vector_fit_gaussian(), cpl_vector_new_from_image_column(), cpl_vector_new_from_image_row(), cpl_vector_new_lss_kernel(), cpl_vector_read(), and cpl_wlcalib_slitmodel_delete().

4.49.2.8 cpl_vector_divide()

```
cpl_error_code cpl_vector_divide (
            cpl_vector * v1,
            const cpl_vector * v2 )
```

Divide two vectors element-wise.

Parameters

v1	First cpl_vector
v2	Second cpl_vector

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_vector_add()

If an element in v2 is zero v1 is not modified and CPL_ERROR_DIVISION_BY_ZERO is returned.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if v1 and v2 have different sizes
- CPL_ERROR_DIVISION_BY_ZERO if a division by 0 would occur

References cpl_ensure_code, CPL_ERROR_DIVISION_BY_ZERO, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.49.2.9 cpl_vector_divide_scalar()

```
cpl_error_code cpl_vector_divide_scalar (
            cpl_vector * v,
            double divisor )
```

Elementwise division of a vector with a scalar.

Parameters

v	cpl_vector to modify
divisor	Non-zero number to divide with

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

Divide each element of the cpl_vector with a number.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_DIVISION_BY_ZERO if divisor is 0.0

References cpl_ensure_code, CPL_ERROR_DIVISION_BY_ZERO, CPL_ERROR_NONE, and CPL_ERROR_N↩ ULL_INPUT.

4.49.2.10 cpl_vector_dump()

```
void cpl_vector_dump (
             const cpl_vector * v,
             FILE * stream )
```

Dump a cpl_vector as ASCII to a stream.

Parameters

v	Input cpl_vector to dump
stream	Output stream, accepts `stdout` or `stderr`

Returns

void

Each element is preceded by its index number (starting with 1!) and written on a single line.

Comment lines start with the hash character.

stream may be NULL in which case `stdout` is used.

Note

In principle a cpl_vector can be saved using cpl_vector_dump() and re-read using cpl_vector_read(). This will however introduce significant precision loss due to the limited accuracy of the ASCII representation.

References CPL_SIZE_FORMAT.

4.49.2.11 cpl_vector_duplicate()

```
cpl_vector* cpl_vector_duplicate (
            const cpl_vector * v )
```

This function duplicates an existing vector and allocates memory.

Parameters

v	the input cpl_vector

Returns

a newly allocated cpl_vector or NULL in case of an error

The returned object must be deallocated using cpl_vector_delete()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, CPL_ERROR_NULL_INPUT, cpl_vector_copy(), and cpl_vector_new().

Referenced by cpl_bivector_duplicate(), cpl_geom_img_offset_saa(), cpl_plot_column(), cpl_vector_convolve_↩symmetric(), cpl_vector_filter_median_create(), and cpl_vector_fit_gaussian().

4.49.2.12 cpl_vector_exponential()

```
cpl_error_code cpl_vector_exponential (
            cpl_vector * v,
            double base )
```

Compute the exponential of all vector elements.

Parameters

v	Target cpl_vector.
base	Exponential base.

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

If the base is zero all vector elements must be positive and if the base is negative all vector elements must be integer, otherwise a cpl_error_code is returned and the vector is unmodified.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT base and v are not as requested
- CPL_ERROR_DIVISION_BY_ZERO if one of the v values is negative or 0

References cpl_ensure_code, CPL_ERROR_DIVISION_BY_ZERO, CPL_ERROR_ILLEGAL_INPUT, CPL_ERR↩OR_NONE, and CPL_ERROR_NULL_INPUT.

4.49.2.13 cpl_vector_extract()

```
cpl_vector* cpl_vector_extract (
             const cpl_vector * v,
             cpl_size istart,
             cpl_size istop,
             cpl_size istep )
```

Extract a sub_vector from a vector.

Parameters

v	Input cpl_vector
istart	Start index (from 0 to number of elements - 1)
istop	Stop index (from 0 to number of elements - 1)
istep	Extract every step element

Returns

A newly allocated cpl_vector or NULL in case of an error

The returned object must be deallocated using cpl_vector_delete()

FIXME: Currently istop must be greater than istart. FIXME: Currently istep must equal 1.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if istart, istop, istep are not as requested

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, CPL_ER↩ROR_NULL_INPUT, and cpl_vector_new().

4.49.2.14 cpl_vector_fill()

```
cpl_error_code cpl_vector_fill (
             cpl_vector * v,
             double val )
```

Fill a cpl_vector.

Parameters

v	cpl_vector to be filled with the value val
val	Value used to fill the cpl_vector

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

Input vector is modified

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure_code, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.49.2.15 cpl_vector_fill_kernel_profile()

```
cpl_error_code cpl_vector_fill_kernel_profile (
            cpl_vector * profile,
            cpl_kernel type,
            double radius )
```

Fill a vector with a kernel profile.

Parameters

profile	cpl_vector to be filled
type	Type of kernel profile
radius	Radius of the profile in pixels

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_image_get_interpolated

A number of predefined kernel profiles are available:

- CPL_KERNEL_DEFAULT: default kernel, currently CPL_KERNEL_TANH
- CPL_KERNEL_TANH: Hyperbolic tangent
- CPL_KERNEL_SINC: Sinus cardinal

- CPL_KERNEL_SINC2: Square sinus cardinal
- CPL_KERNEL_LANCZOS: Lanczos2 kernel
- CPL_KERNEL_HAMMING: Hamming kernel
- CPL_KERNEL_HANN: Hann kernel
- CPL_KERNEL_NEAREST: Nearest neighbor kernel (1 when dist $<$ 0.5, else 0)

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if radius is non-positive, or in case of the CPL_KERNEL_TANH profile if the length of the profile exceeds 32768

References cpl_ensure_code, cpl_error_get_code(), CPL_ERROR_ILLEGAL_INPUT, and cpl_vector_get_size().

4.49.2.16 cpl_vector_filter_lowpass_create()

```
cpl_vector* cpl_vector_filter_lowpass_create (
             const cpl_vector * v,
             cpl_lowpass filter_type,
             cpl_size hw )
```

Apply a low-pass filter to a cpl_vector.

Parameters

v	cpl_vector
filter_type	Type of filter to use
hw	Filter half-width

Returns

Pointer to newly allocated cpl_vector or NULL in case of an error

This type of low-pass filtering consists in a convolution with a given kernel. The chosen filter type determines the kind of kernel to apply for convolution. Supported kernels are CPL_LOWPASS_LINEAR and CPL_LOWPASS_↩ GAUSSIAN.

In the case of CPL_LOWPASS_GAUSSIAN, the gaussian sigma used is 1/sqrt(2). As this function is not meant to be general and cover all possible cases, this sigma is hardcoded and cannot be changed.

The returned smooth cpl_vector must be deallocated using cpl_vector_delete(). The returned signal has exactly as many samples as the input signal.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

- CPL_ERROR_ILLEGAL_INPUT if filter_type is not supported or if hw is negative or bigger than half the vector v size

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_INPUT.

4.49.2.17 cpl_vector_filter_median_create()

```
cpl_vector* cpl_vector_filter_median_create (
            const cpl_vector * self,
            cpl_size hw )
```

Apply a 1D median filter of given half-width to a cpl_vector.

Parameters

self	Input vector to be filtered
hw	Filter half-width

Returns

Pointer to newly allocated cpl_vector or NULL in case of an error

See also

cpl_image_filter_mask()

This function applies a median smoothing to a cpl_vector and returns a newly allocated cpl_vector containing a median-smoothed version of the input. The returned cpl_vector must be deallocated using cpl_vector_delete().

The returned cpl_vector has exactly as many samples as the input one. The outermost hw values are copies of the input, each of the others is set to the median of its surrounding 1 + 2 ∗ hw values.

For historical reasons twice the half-width is allowed to equal the vector length, although in this case the returned vector is simply a duplicate of the input one.

If different processing of the outer values is needed or if a more general kernel is needed, then cpl_image_filter_mask() can be called instead with CPL_FILTER_MEDIAN and the 1D-image input wrapped around self.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if hw is negative or bigger than half the vector size

References CPL_BORDER_COPY, cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, CPL_E↩RROR_NULL_INPUT, CPL_FILTER_MEDIAN, cpl_image_filter_mask(), cpl_image_unwrap(), cpl_image_wrap_↩double(), cpl_mask_delete(), cpl_mask_new(), cpl_mask_not(), cpl_vector_duplicate(), and cpl_vector_new().

4.49.2.18 cpl_vector_find()

```
cpl_size cpl_vector_find (
            const cpl_vector * sorted,
            double key )
```

In a sorted vector find the element closest to the given value.

Parameters

sorted	CPL vector sorted using CPL_SORT_ASCENDING
key	Value to find

Returns

The index that minimizes fabs(sorted[index] - key) or negative on error

See also

cpl_vector_sort()

Bisection is used to find the element.

If two (neighboring) elements with different values both minimize fabs(sorted[index] - key) the index of the larger element is returned.

If the vector contains identical elements that minimize fabs(sorted[index] - key) then it is undefined which element has its index returned.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if two elements are found to not be sorted

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_bivector_interpolate_linear().

4.49.2.19 cpl_vector_fit_gaussian()

```
cpl_error_code cpl_vector_fit_gaussian (
             const cpl_vector * x,
             const cpl_vector * sigma_x,
             const cpl_vector * y,
             const cpl_vector * sigma_y,
             cpl_fit_mode fit_pars,
             double * x0,
             double * sigma,
             double * area,
             double * offset,
             double * mse,
             double * red_chisq,
             cpl_matrix ** covariance )
```

Apply a 1d gaussian fit.

Parameters

x	Positions to fit
sigma_x	Uncertainty (one sigma, gaussian errors assumed) assosiated with *x*. Taking into account the uncertainty of the independent variable is currently unsupported, and this parameter must therefore be set to NULL.
y	Values to fit
sigma_y	Uncertainty (one sigma, gaussian errors assumed) associated with *y*. If NULL, constant uncertainties are assumed.
fit_pars	Specifies which parameters participate in the fit (any other parameters will be held constant). Possible values are CPL_FIT_CENTROID, CPL_FIT_STDEV, CPL_FIT_AREA, CPL_FIT_OFFSET and any bitwise combination of these. As a shorthand for including all four parameters in the fit, use CPL_FIT_ALL.
x0	(output) Center of best fit gaussian. If CPL_FIT_CENTROID is not set, this is also an input parameter.
sigma	(output) Width of best fit gaussian. A positive number on success. If CPL_FIT_STDEV is not set, this is also an input parameter.
area	(output) Area of gaussian. A positive number on succes. If CPL_FIT_AREA is not set, this is also an input parameter.
offset	(output) Fitted background level. If CPL_FIT_OFFSET is not set, this is also an input parameter.
mse	(output) If non-NULL, the mean squared error of the best fit is returned.
red_chisq	(output) If non-NULL, the reduced chi square of the best fit is returned. This requires the noise vector to be specified.
covariance	(output) If non-NULL, the formal covariance matrix of the best fit is returned. This requires *sigma_y* to be specified. The order of fit parameters in the covariance matrix is defined as (*x0*, *sigma*, *area*, *offset*), for example the (3,3) element of the matrix (counting from zero) is the variance of the fitted *offset*. The matrix must be deallocated by calling `cpl_matrix_delete()` . On error, NULL is returned.

Returns

CPL_ERROR_NONE iff okay

This function fits to the input vectors a 1d gaussian function of the form

f(x) = *area* / sqrt(2 pi *sigma*^*2*) ∗ exp(-(*x* - *x0*)^2/(2 *sigma*^*2*)) + *offset*

(*area* > 0) by minimizing chi^2 using a Levenberg-Marquardt algorithm.

The values to fit are read from the input vector *x*. Optionally, a vector *sigma_x* (of same size as *x*) may be specified.

Optionally, the mean squared error, the reduced chi square and the covariance matrix of the best fit are computed . Set corresponding parameter to NULL to ignore.

If the covariance matrix is requested and successfully computed, the diagonal elements (the variances) are guaranteed to be positive.

Occasionally, the Levenberg-Marquardt algorithm fails to converge to a set of sensible parameters. In this case (and only in this case), a CPL_ERROR_CONTINUE is set. To allow the caller to recover from this particular error, the parameters *x0*, *sigma*, *area* and *offset* will on output contain estimates of the best fit parameters, specifically estimated as the median position, the median of the absolute residuals multiplied by 1.4828, the minimum flux value and the maximum flux difference multiplied by sqrt(2 pi sigma^2), respectively. In this case, *mse*, *red_chisq* and *covariance* are not computed. Note that the variance of *x0* (the (0,0) element of the covariance matrix) in this case can be estimated by *sigma*^*2* / *area* .

A CPL_ERROR_SINGULAR_MATRIX occurs if the covariance matrix cannot be computed. In that case all other output parameters are valid.

Current limitations

- Taking into account the uncertainties of the independent variable is not supported.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if *x*, *y*, *x0*, *sigma*, *area* or *offset* is NULL.
- CPL_ERROR_INVALID_TYPE if the specified *fit_pars* is not a bitwise combination of the allowed values (e.g. 0 or 1).
- CPL_ERROR_UNSUPPORTED_MODE *sigma_x* is non-NULL.
- CPL_ERROR_INCOMPATIBLE_INPUT if the sizes of any input vectors are different, or if the computation of reduced chi square or covariance is requested, but *sigma_y* is not provided.
- CPL_ERROR_ILLEGAL_INPUT if any input noise values, *sigma* or *area* is non-positive, or if chi square computation is requested and there are less than 5 data points to fit.
- CPL_ERROR_ILLEGAL_OUTPUT if memory allocation failed.
- CPL_ERROR_CONTINUE if the fitting algorithm failed.
- CPL_ERROR_SINGULAR_MATRIX if the covariance matrix could not be calculated.

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_ILLEGAL_OUTPUT, CPL_ERRO↩R_INCOMPATIBLE_INPUT, CPL_ERROR_INVALID_TYPE, CPL_ERROR_NULL_INPUT, CPL_ERROR_UNS↩UPPORTED_MODE, cpl_matrix_unwrap(), cpl_matrix_wrap(), cpl_vector_delete(), cpl_vector_duplicate(), cpl_↩vector_get_data(), cpl_vector_get_data_const(), cpl_vector_get_min(), and cpl_vector_get_size().

4.49.2.20 cpl_vector_get()

```
double cpl_vector_get (
            const cpl_vector * in,
            cpl_size idx )
```

Get an element of the vector.

Parameters

in	the input vector
idx	the index of the element (0 to nelem-1)

Returns

The element value

In case of error, the _cpl_error_code_ code is set, and the returned double is undefined.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if idx is negative

- CPL_ERROR_ACCESS_OUT_OF_RANGE if idx is out of the vector bounds

References cpl_ensure, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL↩ _ERROR_NULL_INPUT.

Referenced by cpl_apertures_extract(), and cpl_imagelist_erase().

4.49.2.21 cpl_vector_get_data()

```
double* cpl_vector_get_data (
            cpl_vector * in )
```

Get a pointer to the data part of the vector.

Parameters

in	the input vector

Returns

Pointer to the data or NULL in case of an error

The returned pointer refers to already allocated data.

Note

Use at your own risk: direct manipulation of vector data rules out any check performed by the vector object interface, and may introduce inconsistencies between the information maintained internally, and the actual vector data and structure.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_bivector_get_x_data(), cpl_bivector_get_y_data(), cpl_geom_img_offset_fine(), cpl_image↩ _fill_jacobian_polynomial(), cpl_image_warp_polynomial(), cpl_photom_fill_blackbody(), cpl_vector_convolve_↩ symmetric(), cpl_vector_cycle(), cpl_vector_fit_gaussian(), cpl_vector_new_from_image_column(), cpl_vector_↩ new_from_image_row(), and cpl_wlcalib_find_best_1d().

4.49.2.22 cpl_vector_get_data_const()

```
const double* cpl_vector_get_data_const (
            const cpl_vector * in )
```

Get a pointer to the data part of the vector.

Parameters

in	the input vector

Returns

Pointer to the data or NULL in case of an error

See also

cpl_vector_get_data

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_bivector_dump(), cpl_bivector_get_x_data_const(), cpl_bivector_get_y_data_const(), cpl_geom_img_offset_combine(), cpl_geom_img_offset_saa(), cpl_image_get_interpolated(), cpl_image_warp(), cpl_image_warp_polynomial(), cpl_photom_fill_blackbody(), cpl_plot_vector(), cpl_plot_vectors(), cpl_ppm_match_positions(), cpl_vector_convolve_symmetric(), cpl_vector_fit_gaussian(), cpl_vector_product(), and cpl_wlcalib_find_best_1d().

4.49.2.23 cpl_vector_get_max()

```
double cpl_vector_get_max (
            const cpl_vector * v )
```

Get the maximum of the cpl_vector.

Parameters

v	const cpl_vector

Returns

the maximum value of the vector or undefined on error

See also

cpl_vector_get_min()

References cpl_vector_get_maxpos().

Referenced by cpl_geom_img_offset_saa().

4.49.2.24 cpl_vector_get_maxpos()

```
cpl_size cpl_vector_get_maxpos (
            const cpl_vector * self )
```

Get the index of the maximum element of the cpl_vector.

Parameters

self	The vector to process

Returns

The index (0 for first) of the maximum value or negative on error

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_vector_get_max().

4.49.2.25 cpl_vector_get_mean()

```
double cpl_vector_get_mean (
            const cpl_vector * v )
```

Compute the mean value of vector elements.

Parameters

v	Input const cpl_vector

Returns

Mean value of vector elements or undefined on error.

See also

cpl_vector_get_min()

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

4.49.2.26 cpl_vector_get_median()

```
double cpl_vector_get_median (
            cpl_vector * v )
```

Compute the median of the elements of a vector.

Parameters

v	Input cpl_vector

Returns

Median value of the vector elements or undefined on error.

See also

cpl_vector_get_median_const()

Note

For efficiency reasons, this function modifies the order of the elements of the input vector.

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

4.49.2.27 cpl_vector_get_median_const()

```
double cpl_vector_get_median_const (
            const cpl_vector * v )
```

Compute the median of the elements of a vector.

Parameters

v	Input const cpl_vector

Returns

Median value of the vector elements or undefined on error.

See also

cpl_vector_get_min()

For a finite population or sample, the median is the middle value of an odd number of values (arranged in ascending order) or any value between the two middle values of an even number of values. The criteria used for an even number of values in the input array is to choose the mean between the two middle values. Note that in this case, the median might not be a value of the input array. Also, note that in the case of integer data types, the result will be converted to an integer. Consider to transform your int array to float if that is not the desired behavior.

References cpl_ensure, CPL_ERROR_NULL_INPUT, cpl_free(), and cpl_malloc().

4.49.2.28 cpl_vector_get_min()

```
double cpl_vector_get_min (
            const cpl_vector * v )
```

Get the minimum of the cpl_vector.

Parameters

v	const cpl_vector

Returns

The minimum value of the vector or undefined on error

In case of error, the _cpl_error_code_ code is set, and the returned double is undefined.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_vector_get_minpos().

Referenced by cpl_geom_img_offset_saa(), and cpl_vector_fit_gaussian().

4.49.2.29 cpl_vector_get_minpos()

```
cpl_size cpl_vector_get_minpos (
            const cpl_vector * self )
```

Get the index of the minimum element of the cpl_vector.

Parameters

self	The vector to process

Returns

The index (0 for first) of the minimum value or negative on error

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_vector_get_min().

4.49.2.30 cpl_vector_get_size()

```
cpl_size cpl_vector_get_size (
            const cpl_vector * in )
```

Get the size of the vector.

Parameters

in	the input vector

Returns

The size or -1 in case of an error

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_apertures_extract(), cpl_bivector_dump(), cpl_bivector_duplicate(), cpl_bivector_get_size(), cpl_bivector_wrap_vectors(), cpl_fit_imagelist_polynomial_window(), cpl_geom_img_offset_fine(), cpl_image_get_interpolated(), cpl_image_warp(), cpl_image_warp_polynomial(), cpl_imagelist_erase(), cpl_photom_fill_blackbody(), cpl_plot_column(), cpl_plot_vector(), cpl_plot_vectors(), cpl_polynomial_fit(), cpl_ppm_match_positions(), cpl_test_get_bytes_vector(), cpl_vector_convolve_symmetric(), cpl_vector_correlate(), cpl_vector_cycle(), cpl_vector_fill_kernel_profile(), cpl_vector_fill_polynomial_fit_residual(), cpl_vector_fit_gaussian(), cpl_vector_product(), and cpl_wlcalib_find_best_1d().

4.49.2.31 cpl_vector_get_stdev()

```
double cpl_vector_get_stdev (
            const cpl_vector * v )
```

Compute the bias-corrected standard deviation of a vectors elements.

Parameters

v	Input const cpl_vector

Returns

standard deviation of the elements or a negative number on error.

See also

cpl_vector_get_min()

S(n-1) = sqrt((1/n-1) sum((xi-mean)^2) (i=1 -> n)

The length of v must be at least 2.

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_INPUT.

4.49.2.32 cpl_vector_get_sum()

```
double cpl_vector_get_sum (
            const cpl_vector * v )
```

Get the sum of the elements of the cpl_vector.

Parameters

v	const cpl_vector

Returns

the sum of the elements of the vector or undefined on error

See also

cpl_vector_get_min()

References cpl_ensure, and CPL_ERROR_NULL_INPUT.

4.49.2.33 cpl_vector_load()

```
cpl_vector* cpl_vector_load (
            const char * filename,
            cpl_size xtnum )
```

Load a list of values from a FITS file.

Parameters

filename	Name of the input file
xtnum	Extension number in the file (0 for primary HDU)

Returns

1 newly allocated cpl_vector or NULL in case of an error

See also

cpl_vector_save

This function loads a vector from a FITS file (NAXIS=1), using cfitsio. The returned image has to be deallocated with cpl_vector_delete().

'xtnum' specifies from which extension the vector should be loaded. This could be 0 for the main data section or any number between 1 and N, where N is the number of extensions present in the file.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the extension is not valid
- CPL_ERROR_FILE_IO if the file cannot be read
- CPL_ERROR_UNSUPPORTOED_MODE if the file is too large to be read

References cpl_ensure, CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_INPUT.

4.49.2.34 cpl_vector_logarithm()

```
cpl_error_code cpl_vector_logarithm (
            cpl_vector * v,
            double base )
```

Compute the element-wise logarithm.

Parameters

v	cpl_vector to modify.
base	Logarithm base.

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

The base and all the vector elements must be positive and the base must be different from 1, or a cpl_error_code will be returned and the vector will be left unmodified.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if base is negative or zero or if one of the vector values is negative or zero
- CPL_ERROR_DIVISION_BY_ZERO if a division by zero occurs

References cpl_ensure_code, CPL_ERROR_DIVISION_BY_ZERO, CPL_ERROR_ILLEGAL_INPUT, CPL_ERR↩OR_NONE, and CPL_ERROR_NULL_INPUT.

4.49.2.35 cpl_vector_multiply()

```
cpl_error_code cpl_vector_multiply (
            cpl_vector * v1,
            const cpl_vector * v2 )
```

Multiply two vectors component-wise.

Parameters

v1	First cpl_vector
v2	Second cpl_vector

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_vector_add()

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NONE, and CPL_ERRO↩R_NULL_INPUT.

4.49.2.36 cpl_vector_multiply_scalar()

```
cpl_error_code cpl_vector_multiply_scalar (
            cpl_vector * v,
            double factor )
```

Elementwise multiplication of a vector with a scalar.

Parameters

v	cpl_vector to modify
factor	Number to multiply with

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

Multiply each element of the cpl_vector with a number.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure_code, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.49.2.37 cpl_vector_new()

```
cpl_vector* cpl_vector_new (
            cpl_size n )
```

Create a new cpl_vector.

Parameters

n	Number of element of the cpl_vector

Returns

1 newly allocated cpl_vector or NULL in case of an error

The returned object must be deallocated using cpl_vector_delete(). There is no default values assigned to the created object, they are undefined until they are set.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_ILLEGAL_INPUT if n is negative or zero

References CPL_ERROR_ILLEGAL_INPUT, cpl_malloc(), and CPL_SIZE_FORMAT.

Referenced by cpl_bivector_new(), cpl_bivector_read(), cpl_geom_img_offset_combine(), cpl_image_fill_↩ jacobian_polynomial(), cpl_image_warp_polynomial(), cpl_matrix_solve_svd(), cpl_matrix_solve_svd_threshold(), cpl_vector_duplicate(), cpl_vector_extract(), cpl_vector_filter_median_create(), cpl_vector_new_from_image_↩ column(), cpl_vector_new_from_image_row(), cpl_vector_new_lss_kernel(), cpl_vector_read(), and cpl_wlcalib_↩ find_best_1d().

4.49.2.38 cpl_vector_new_lss_kernel()

```
cpl_vector* cpl_vector_new_lss_kernel (
            double slitw,
            double fwhm )
```

Create Right Half of a symmetric smoothing kernel for LSS.

Parameters

slitw	The slit width [pixel]
fwhm	The spectral FWHM [pixel]

Returns

Right Half of (symmetric) smoothing vector

Deprecated Unstable API, may change or disappear. Do not use in new code!

References CPL_MATH_SIG_FWHM, cpl_vector_delete(), and cpl_vector_new().

4.49.2.39 cpl_vector_power()

```
cpl_error_code cpl_vector_power (
            cpl_vector * v,
            double exponent )
```

Compute the power of all vector elements.

Parameters

v	Target cpl_vector.
exponent	Constant exponent.

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

If the exponent is negative all vector elements must be non-zero and if the exponent is non-integer all vector elements must be non-negative, otherwise a cpl_error_code is returned and the vector is unmodified.

Following the behaviour of C99 pow() function, this function sets 0^0 = 1.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if v and exponent are not as requested
- CPL_ERROR_DIVISION_BY_ZERO if one of the v values is 0

References cpl_ensure_code, CPL_ERROR_DIVISION_BY_ZERO, CPL_ERROR_ILLEGAL_INPUT, CPL_ERR↩OR_NONE, and CPL_ERROR_NULL_INPUT.

4.49.2.40 cpl_vector_product()

```
double cpl_vector_product (
            const cpl_vector * v1,
            const cpl_vector * v2 )
```

Compute the vector dot product.

Parameters

v1	One vector
v2	Another vector of the same size

Returns

The (non-negative) product or negative on error.

The same vector may be passed twice, in which case the square of its 2-norm is computed.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_INCOMPATIBLE_INPUT if v1 and v2 have different sizes

References cpl_ensure, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NULL_INPUT, cpl_vector_get_↩data_const(), and cpl_vector_get_size().

4.49.2.41 cpl_vector_read()

```
cpl_vector* cpl_vector_read (
            const char * filename )
```

Read a list of values from an ASCII file and create a cpl_vector.

Parameters

filename	Name of the input ASCII file

Returns

1 newly allocated cpl_vector or NULL in case of an error

See also

cpl_vector_dump

Parse an input ASCII file values and create a cpl_vector from it Lines beginning with a hash are ignored, blank lines also. In valid lines the value is preceeded by an integer, which is ignored.

The returned object must be deallocated using cpl_vector_delete()

In addition to normal files, FIFO (see man mknod) are also supported.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_FILE_IO if the file cannot be read
- CPL_ERROR_BAD_FILE_FORMAT if the file contains no valid lines

References cpl_ensure, CPL_ERROR_BAD_FILE_FORMAT, CPL_ERROR_FILE_IO, CPL_ERROR_NULL_INP↩UT, cpl_vector_delete(), cpl_vector_new(), cpl_vector_set(), and cpl_vector_set_size().

4.49.2.42 cpl_vector_save()

```
cpl_error_code cpl_vector_save (
            const cpl_vector * self,
            const char * filename,
            cpl_type type,
            const cpl_propertylist * plist,
            unsigned mode )
```

Save a vector to a FITS file.

Parameters

self	Vector to write to disk or NULL
filename	Name of the file to write
type	The type used to represent the data in the file
plist	Property list for the output header or NULL
mode	The desired output options (combined with bitwise or)

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

This function saves a vector to a FITS file (NAXIS=1), using cfitsio. If a property list is provided, it is written to the named file before the pixels are written. If the image is not provided, the created file will only contain the primary header. This can be useful to create multiple extension files.

The type used in the file can be one of: CPL_TYPE_UCHAR (8 bit unsigned), CPL_TYPE_SHORT (16 bit signed), CPL_TYPE_USHORT (16 bit unsigned), CPL_TYPE_INT (32 bit signed), CPL_TYPE_FLOAT (32 bit floating point), or CPL_TYPE_DOUBLE (64 bit floating point). Use CPL_TYPE_DOUBLE when no loss of information is required.

Supported output modes are CPL_IO_CREATE (create a new file) and CPL_IO_EXTEND (append to an existing file)

If you are in append mode, make sure that the file has writing permissions. You may have problems if you create a file in your application and append something to it with the umask set to 222. In this case, the file created by your application would not be writable, and the append would fail.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if the type or the mode is not supported
- CPL_ERROR_FILE_NOT_CREATED if the output file cannot be created
- CPL_ERROR_FILE_IO if the data cannot be written to the file
- CPL_ERROR_UNSUPPORTOED_MODE if the file is too large to be saved

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_IN↩PUT, CPL_ERROR_UNSUPPORTED_MODE, CPL_IO_CREATE, CPL_IO_EXTEND, and cpl_propertylist_save().

4.49.2.43 cpl_vector_set()

```
cpl_error_code cpl_vector_set (
            cpl_vector * in,
            cpl_size idx,
            double value )
```

Set an element of the vector.

Parameters

in	the input vector
idx	the index of the element (0 to nelem-1)
value	the value to set in the vector

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if idx is negative
- CPL_ERROR_ACCESS_OUT_OF_RANGE if idx is out of the vector bounds

References cpl_ensure_code, CPL_ERROR_ACCESS_OUT_OF_RANGE, CPL_ERROR_ILLEGAL_INPUT, and CPL_ERROR_NULL_INPUT.

Referenced by cpl_bivector_read(), cpl_plot_column(), cpl_vector_correlate(), cpl_vector_read(), and cpl_wlcalib↩
_find_best_1d().

4.49.2.44 cpl_vector_set_size()

```
cpl_error_code cpl_vector_set_size (
            cpl_vector * in,
            cpl_size newsize )
```

Resize the vector.

Parameters

in	The vector to be resized
newsize	The new (positive) number of elements in the vector

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

Note

On succesful return the value of the elements of the vector is unchanged to the minimum of the old and new sizes; any newly allocated elements are undefined. The pointer to the vector data buffer may change, therefore pointers previously retrieved by calling `cpl_vector_get_data()` should be discarded. If the vector was created with cpl_vector_wrap() the argument pointer to that call should be discarded as well.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if newsize is negative or zero

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_IN↩PUT, and cpl_realloc().

Referenced by cpl_bivector_read(), cpl_geom_img_offset_combine(), cpl_vector_copy(), and cpl_vector_read().

4.49.2.45 cpl_vector_sort()

```
cpl_error_code cpl_vector_sort (
            cpl_vector * self,
            cpl_sort_direction dir )
```

Sort a cpl_vector.

Parameters

self	cpl_vector to sort in place
dir	CPL_SORT_ASCENDING or CPL_SORT_DESCENDING

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

The input cpl_vector is modified to sort its values in either ascending (CPL_SORT_ASCENDING) or descending (CPL_SORT_DESCENDING) order.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if dir is neither CPL_SORT_DESCENDING nor CPL_SORT_ASCENDING

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, and CPL_ERROR_NULL↩_INPUT.

Referenced by cpl_geom_img_offset_saa().

4.49.2.46 cpl_vector_sqrt()

```
cpl_error_code cpl_vector_sqrt (
            cpl_vector * v )
```

Compute the sqrt of a cpl_vector.

Parameters

v	cpl_vector

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

The sqrt of the data is computed. The input cpl_vector is modified

If an element in v is negative v is not modified and CPL_ERROR_ILLEGAL_INPUT is returned.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if one of the vector values is negative

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, and CPL_ERROR_NULL↩ _INPUT.

4.49.2.47 cpl_vector_subtract()

```
cpl_error_code cpl_vector_subtract (
            cpl_vector * v1,
            const cpl_vector * v2 )
```

Subtract a cpl_vector from another.

Parameters

v1	First cpl_vector (modified)
v2	Second cpl_vector

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_vector_add()

References cpl_ensure_code, CPL_ERROR_INCOMPATIBLE_INPUT, CPL_ERROR_NONE, and CPL_ERRO↩ R_NULL_INPUT.

4.49.2.48 cpl_vector_subtract_scalar()

```
cpl_error_code cpl_vector_subtract_scalar (
            cpl_vector * v,
            double subtrahend )
```

Elementwise subtraction of a scalar from a vector.

Parameters

v	cpl_vector to modify
subtrahend	Number to subtract

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

Subtract a number from each element of the cpl_vector.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL

References cpl_ensure_code, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.49.2.49 cpl_vector_unwrap()

```
void* cpl_vector_unwrap (
            cpl_vector * v )
```

Delete a cpl_vector except the data array.

Parameters

v	cpl_vector to delete

Returns

A pointer to the data array or NULL if the input is NULL.

Note

The data array must subsequently be deallocated. Failure to do so will result in a memory leak.

References cpl_free().

Referenced by cpl_plot_column().

4.49.2.50 cpl_vector_wrap()

```
cpl_vector* cpl_vector_wrap (
             cpl_size n,
             double * data )
```

Create a cpl_vector from existing data.

Parameters

n	Number of elements in the vector
data	Pointer to array of n doubles

Returns

1 newly allocated cpl_vector or NULL on error

Note

The returned object must be deallocated using e.g. cpl_vector_unwrap()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is NULL
- CPL_ERROR_ILLEGAL_INPUT if n is negative or zero

References CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, cpl_malloc(), and CPL_SIZE_FORMAT.

Referenced by cpl_fit_image_gaussian(), cpl_plot_column(), cpl_ppm_match_positions(), and cpl_wlcalib_find_↩ best_1d().

4.50 Wavelength calibration

Functions

- cpl_error_code cpl_wlcalib_fill_line_spectrum (cpl_vector ∗self, void ∗model, const cpl_polynomial ∗disp)

  Generate a 1D spectrum from a model and a dispersion relation.
- cpl_error_code cpl_wlcalib_fill_line_spectrum_fast (cpl_vector ∗self, void ∗model, const cpl_polynomial ∗disp)

  Generate a 1D spectrum from a model and a dispersion relation.
- cpl_error_code cpl_wlcalib_fill_logline_spectrum (cpl_vector ∗self, void ∗model, const cpl_polynomial ∗disp)

  Generate a 1D spectrum from a model and a dispersion relation.
- cpl_error_code cpl_wlcalib_fill_logline_spectrum_fast (cpl_vector ∗self, void ∗model, const cpl_polynomial ∗disp)

  Generate a 1D spectrum from a model and a dispersion relation.
- cpl_error_code cpl_wlcalib_find_best_1d (cpl_polynomial ∗self, const cpl_polynomial ∗guess, const cpl_↩ vector ∗spectrum, void ∗model, cpl_error_code(∗filler)(cpl_vector ∗, void ∗, const cpl_polynomial ∗), const cpl_vector ∗wl_search, cpl_size nsamples, cpl_size hsize, double ∗xcmax, cpl_vector ∗xcorrs)

  Find the best 1D dispersion polynomial in a given search space.
- void cpl_wlcalib_slitmodel_delete (cpl_wlcalib_slitmodel ∗self)

  Free memory associated with a cpl_wlcalib_slitmodel object.
- cpl_wlcalib_slitmodel ∗ cpl_wlcalib_slitmodel_new (void)

  Create a new line model to be initialized.
- cpl_error_code cpl_wlcalib_slitmodel_set_catalog (cpl_wlcalib_slitmodel ∗self, cpl_bivector ∗catalog)

  Set the catalog of lines to be used by the spectrum filler.
- cpl_error_code cpl_wlcalib_slitmodel_set_threshold (cpl_wlcalib_slitmodel ∗self, double value)

  The (positive) threshold for truncating the transfer function.
- cpl_error_code cpl_wlcalib_slitmodel_set_wfwhm (cpl_wlcalib_slitmodel ∗self, double value)

  Set the FWHM of the transfer function to be used by the spectrum filler.
- cpl_error_code cpl_wlcalib_slitmodel_set_wslit (cpl_wlcalib_slitmodel ∗self, double value)

  Set the slit width to be used by the spectrum filler.

4.50.1 Detailed Description

This module contains functions to perform 1D-wavelength calibration, typically of long-slit spectroscopy data.

Synopsis:

```
#include "cpl_wlcalib.h"
```

4.50.2 Function Documentation

4.50.2.1 cpl_wlcalib_fill_line_spectrum()

```
cpl_error_code cpl_wlcalib_fill_line_spectrum (
            cpl_vector * self,
            void * model,
            const cpl_polynomial * disp )
```

Generate a 1D spectrum from a model and a dispersion relation.

Parameters

self	Vector to fill with spectrum
model	Pointer to cpl_wlcalib_slitmodel object
disp	1D-Dispersion relation, at least of degree 1

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

Note

The model is passed as a *void* pointer so the function can be used with cpl_wlcalib_find_best_1d().

See also

cpl_wlcalib_find_best_1d()

The fill a vector with a spectrum, one must first initialize the parameters of the model (error checks omitted for brevity):

```
cpl_vector            * spectrum   = cpl_vector_new(nresolution);
cpl_wlcalib_slitmodel * model      = cpl_wlcalib_slitmodel_new();
cpl_bivector          * lines      = my_load_lines_catalog(filename);
cpl_polynomial        * dispersion = my_1d_dispersion();
cpl_wlcalib_slitmodel_set_wslit(model, 3.0);
cpl_wlcalib_slitmodel_set_wfwhm(model, 4.0);
cpl_wlcalib_slitmodel_set_threshold(model, 5.0);
cpl_wlcalib_slitmodel_set_catalog(model, lines);
```

With that the spectrum can be filled:

```
cpl_wlcalib_fill_line_spectrum(spectrum, model, dispersion);
```

Clean-up when no more spectra are needed (lines are deleted with the model):

```
cpl_wlcalib_slitmodel_delete(model);
cpl_polynomial_delete(dispersion);
cpl_vector_delete(spectrum);
```

Each line profile is given by the convolution of the Dirac delta function with a Gaussian with $\sigma = w_{FWHM}/(2\sqrt{(2\log(2))})$, and a top-hat with the slit width as width. This continuous line profile is then integrated over each pixel, wherever the intensity is above the threshold set by the given model. For a given line the value on a given pixel requires the evaluation of two calls to *erf()*.

Possible _cpl_error_code_ set by this function:

- CPL_ERROR_NULL_INPUT if an input pointer is *NULL*
- CPL_ERROR_INVALID_TYPE If the input polynomial is not 1D
- CPL_ERROR_ILLEGAL_INPUT If the input polynomial is non-increasing over the given input (pixel) range, or if a model parameter is non-physical (e.g. non-positive slit width).
- CPL_ERROR_DATA_NOT_FOUND If no catalog lines are available in the range of the dispersion relation
- CPL_ERROR_INCOMPATIBLE_INPUT If the wavelengths of two catalog lines are found to be in non-increasing order.

References cpl_ensure_code, and CPL_ERROR_NULL_INPUT.

4.50.2.2 cpl_wlcalib_fill_line_spectrum_fast()

```
cpl_error_code cpl_wlcalib_fill_line_spectrum_fast (
            cpl_vector * self,
            void * model,
            const cpl_polynomial * disp )
```

Generate a 1D spectrum from a model and a dispersion relation.

Parameters

self	Vector to fill with spectrum
model	Pointer to cpl_wlcalib_slitmodel object
disp	1D-Dispersion relation, at least of degree 1

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

Note

The generated spectrum will use an approximate line profile for speed

See also

cpl_wlcalib_fill_line_spectrum()

The approximation preserves the position of the maximum, the symmetry and the flux of the line profile.

The use of a given line in a spectrum requires the evaluation of four calls to *erf()*.

The fast spectrum generation can be useful when the model spectrum includes many catalog lines.

References cpl_ensure_code, and CPL_ERROR_NULL_INPUT.

4.50.2.3 cpl_wlcalib_fill_logline_spectrum()

```
cpl_error_code cpl_wlcalib_fill_logline_spectrum (
            cpl_vector * self,
            void * model,
            const cpl_polynomial * disp )
```

Generate a 1D spectrum from a model and a dispersion relation.

Parameters

self	Vector to fill with spectrum
model	Pointer to cpl_wlcalib_slitmodel object
disp	1D-Dispersion relation, at least of degree 1

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

Note

The spectrum is generated from 1 + the logarithm of the line intensities

See also

cpl_wlcalib_fill_line_spectrum()

References cpl_ensure_code, and CPL_ERROR_NULL_INPUT.

4.50.2.4 cpl_wlcalib_fill_logline_spectrum_fast()

```
cpl_error_code cpl_wlcalib_fill_logline_spectrum_fast (
            cpl_vector * self,
            void * model,
            const cpl_polynomial * disp )
```

Generate a 1D spectrum from a model and a dispersion relation.

Parameters

self	Vector to fill with spectrum
model	Pointer to cpl_wlcalib_slitmodel object
disp	1D-Dispersion relation, at least of degree 1

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

Note

The spectrum is generated from 1 + the logarithm of the line intensities and an approximate line profile for speed

See also

cpl_wlcalib_fill_line_spectrum_fast()

References cpl_ensure_code, and CPL_ERROR_NULL_INPUT.

4.50.2.5 cpl_wlcalib_find_best_1d()

```
cpl_error_code cpl_wlcalib_find_best_1d (
            cpl_polynomial * self,
            const cpl_polynomial * guess,
            const cpl_vector * spectrum,
            void * model,
            cpl_error_code(*)(cpl_vector *, void *, const cpl_polynomial *) filler,
            const cpl_vector * wl_search,
            cpl_size nsamples,
            cpl_size hsize,
            double * xcmax,
            cpl_vector * xcorrs )
```

Find the best 1D dispersion polynomial in a given search space.

Parameters

self	Pre-created 1D-polynomial for the result
guess	1D-polynomial with the guess, may equal self
spectrum	The vector with the observed 1D-spectrum
model	The spectrum model
filler	The function used to make the spectrum
wl_search	Search range around the anchor points, same unit as guess
nsamples	Number of samples around the anchor points
hsize	Maximum (pixel) displacement of the polynomial guess
xcmax	On success, the maximum cross-correlation
xcorrs	The vector to fill with the correlation values or NULL

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_wlcalib_fill_line_spectrum() for the model and filler.

Find the polynomial that maximizes the cross-correlation between an observed 1D-spectrum and a model spectrum based on the polynomial dispersion relation.

Each element in the vector of wavelength search ranges is in the same unit as the corresponding Y-value of the dispersion relation. Each value in the vector is the width of a search window centered on the corresponding value in the guess polynomial. The length D of the search vector thus determines the dimensionality of the search space for the dispersion polynomial. If for example the search vector consists of three elements, then the three lowest order coefficients of the dispersion relation may be modified by the search.

For each candidate polynomial P(x), the polynomial P(x+u), -hsize <= u <= hsize is also evaluated. The half-size hsize may be zero. When it is non-zero, an additional 2 * hsize cross-correlations are performed for each candidate polynomial, one for each possible shift. The maximizing polynomial among those shifted polynomials is kept. A well-chosen half-size can allow for the use of fewer number of samples around the anchor points, leading to a reduction of polynomials to be evaluated.

The complexity in terms of model spectra creation is O(N^D) and in terms of cross-correlations O(hsize * N^D), where N is nsamples and D is the length of wl_search.

xcorrs must be NULL or have a size of (at least) N^D*(1 + 2 * hsize).

Possible _cpl_error_code_ set by this function:

- CPL_ERROR_NULL_INPUT if an input pointer is *NULL*
- CPL_ERROR_INVALID_TYPE If an input polynomial is not 1D
- CPL_ERROR_ILLEGAL_INPUT If nfree is less than 2, or nsamples is less than 1, hsize negative or if wl_↩ search contains a zero search bound, or if xcorrs is non-NULL and too short.
- CPL_ERROR_DATA_NOT_FOUND If no model spectra can be created using the supplied model and filler

References cpl_ensure_code, CPL_ERROR_DATA_NOT_FOUND, CPL_ERROR_ILLEGAL_INPUT, CPL_ERR↩ OR_INVALID_TYPE, CPL_ERROR_NONE, CPL_ERROR_NULL_INPUT, cpl_errorstate_get(), cpl_errorstate_↩ is_equal(), cpl_polynomial_eval_1d(), cpl_polynomial_get_dimension(), cpl_vector_get_data(), cpl_vector_get_↩ data_const(), cpl_vector_get_size(), cpl_vector_new(), cpl_vector_set(), and cpl_vector_wrap().

4.50.2.6 cpl_wlcalib_slitmodel_delete()

```
void cpl_wlcalib_slitmodel_delete (
            cpl_wlcalib_slitmodel * self )
```

Free memory associated with a cpl_wlcalib_slitmodel object.

Parameters

self	The cpl_wlcalib_slitmodel object or *NULL*

Returns

Nothing

See also

cpl_wlcalib_slitmodel_new()

Note

If *self* is `NULL` nothing is done and no error is set.

References cpl_bivector_delete(), cpl_free(), and cpl_vector_delete().

4.50.2.7 cpl_wlcalib_slitmodel_new()

```
cpl_wlcalib_slitmodel* cpl_wlcalib_slitmodel_new (
            void  )
```

Create a new line model to be initialized.

Returns

1 newly allocated cpl_wlcalib_slitmodel

Note

All elements are initialized to either zero or NULL.

See also

cpl_wlcalib_slitmodel_delete() for object deallocation.

The model comprises these elements: Slit Width FWHM of transfer function Truncation threshold of the transfer function Catalog of lines (typically arc or sky)

The units of the X-values of the lines is a length, it is assumed to be the same as that of the Y-values of the dispersion relation (e.g. meter), the units of slit width and the FWHM are assumed to be the same as the X-values of the dispersion relation (e.g. pixel), while the units of the produced spectrum will be that of the Y-values of the lines.

References cpl_calloc().

4.50.2.8 cpl_wlcalib_slitmodel_set_catalog()

```
cpl_error_code cpl_wlcalib_slitmodel_set_catalog (
            cpl_wlcalib_slitmodel * self,
            cpl_bivector * catalog )
```

Set the catalog of lines to be used by the spectrum filler.

Parameters

self	The cpl_wlcalib_slitmodel object
catalog	The catalog of lines (e.g. arc lines)

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_wlcalib_slitmodel_new()

Note

The values in the X-vector must be increasing. Any previously set catalog is deallocated

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is *NULL*

References cpl_bivector_delete(), cpl_ensure_code, CPL_ERROR_NONE, and CPL_ERROR_NULL_INPUT.

4.50.2.9 cpl_wlcalib_slitmodel_set_threshold()

```
cpl_error_code cpl_wlcalib_slitmodel_set_threshold (
            cpl_wlcalib_slitmodel * self,
            double value )
```

The (positive) threshold for truncating the transfer function.

Parameters

self	The cpl_wlcalib_slitmodel object
value	The (non-negative) truncation threshold, 5 is a good value.

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_wlcalib_slitmodel_new()

Note

The threshold should be high enough to ensure a good line profile, but not too high to make the spectrum generation too costly.

The line profile is truncated at this distance [pixel] from its maximum: $x_{max} = w/2 + k * \sigma$, where w is the slit width and $\sigma = w_{FWHM}/(2\sqrt{(2\log(2))})$, where w_{FWHM} is the Full Width at Half Maximum (FWHM) of the transfer function and k is the user supplied value.

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is *NULL*
- CPL_ERROR_ILLEGAL_INPUT the value is negative

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, and CPL_ERROR_NULL↩ _INPUT.

4.50.2.10 cpl_wlcalib_slitmodel_set_wfwhm()

```
cpl_error_code cpl_wlcalib_slitmodel_set_wfwhm (
            cpl_wlcalib_slitmodel * self,
            double value )
```

Set the FWHM of the transfer function to be used by the spectrum filler.

Parameters

self	The cpl_wlcalib_slitmodel object
value	The (positive) FWHM

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_wlcalib_slitmodel_new()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is *NULL*
- CPL_ERROR_ILLEGAL_INPUT the value is non-positive

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, and CPL_ERROR_NULL↩ _INPUT.

4.50.2.11 cpl_wlcalib_slitmodel_set_wslit()

```
cpl_error_code cpl_wlcalib_slitmodel_set_wslit (
            cpl_wlcalib_slitmodel * self,
            double value )
```

Set the slit width to be used by the spectrum filler.

Parameters

self	The cpl_wlcalib_slitmodel object
value	The (positive) width of the slit

Returns

CPL_ERROR_NONE or the relevant _cpl_error_code_ on error

See also

cpl_wlcalib_slitmodel_new()

Possible _cpl_error_code_ set in this function:

- CPL_ERROR_NULL_INPUT if an input pointer is *NULL*
- CPL_ERROR_ILLEGAL_INPUT the value is non-positive

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, and CPL_ERROR_NULL↩ _INPUT.

4.51 World Coordinate System

Macros

- #define CPL_WCS_REGEXP

  A regular expression that matches the FITS keys used for WCS.

Functions

- cpl_error_code cpl_wcs_convert (const cpl_wcs *wcs, const cpl_matrix *from, cpl_matrix **to, cpl_array **status, cpl_wcs_trans_mode transform)

  Convert between physical and world coordinates.
- void cpl_wcs_delete (cpl_wcs *wcs)

  Destroy a WCS structure.
- const cpl_matrix * cpl_wcs_get_cd (const cpl_wcs *wcs)

  Accessor to get the CD matrix for a WCS.
- const cpl_array * cpl_wcs_get_crpix (const cpl_wcs *wcs)

  Accessor to get the CRPIX vector for a WCS.
- const cpl_array * cpl_wcs_get_crval (const cpl_wcs *wcs)

  Accessor to get the CRVAL vector for a WCS.
- const cpl_array * cpl_wcs_get_ctype (const cpl_wcs *wcs)

  Accessor to get the CTYPE vector for a WCS.
- const cpl_array * cpl_wcs_get_cunit (const cpl_wcs *wcs)

  Accessor to get the CUNIT vector for a WCS.
- const cpl_array * cpl_wcs_get_image_dims (const cpl_wcs *wcs)

  Accessor to get the axis lengths of the image associated with a WCS.
- int cpl_wcs_get_image_naxis (const cpl_wcs *wcs)

  Accessor to get the dimensionality of the image associated with a WCS.
- cpl_wcs * cpl_wcs_new_from_propertylist (const cpl_propertylist *plist)

  Create a wcs structure by parsing a propertylist.
- cpl_error_code cpl_wcs_platesol (const cpl_propertylist *ilist, const cpl_matrix *cel, const cpl_matrix *xy, int niter, float thresh, cpl_wcs_platesol_fitmode fitmode, cpl_wcs_platesol_outmode outmode, cpl_propertylist **olist)

  Do a 2d plate solution given physical and celestial coordinates.

4.51.1 Detailed Description

This module provides functions to manipulate FITS World Coordinate Systems

A *cpl_wcs* is an object containing a pointer to the WCSLIB structure and the physical dimensions of the image from which the WCS was read. The functionality provided includes general transformations between physical and world coordinates as well as a few convenience routines for x,y <=> RA,Dec transformations.

Synopsis:

```
#include "cpl_wcs.h"
```

4.51.2 Macro Definition Documentation

4.51.2.1 CPL_WCS_REGEXP

```
#define CPL_WCS_REGEXP
```

A regular expression that matches the FITS keys used for WCS.

4.51.3 Function Documentation

4.51.3.1 cpl_wcs_convert()

```
cpl_error_code cpl_wcs_convert (
            const cpl_wcs * wcs,
            const cpl_matrix * from,
            cpl_matrix ** to,
            cpl_array ** status,
            cpl_wcs_trans_mode transform )
```

Convert between physical and world coordinates.

Parameters

wcs	The input cpl_wcs structure
from	The input coordinate matrix
to	The output coordinate matrix
status	The output status array
transform	The transformation mode

Returns

An appropriate error code.

Errors

CPL_ERROR_NULL_INPUT	The parameter *wcs*, *from*, *to* or *status* is a `NULL` pointer or *wcs* is missing some of its information.
CPL_ERROR_UNSPECIFIED	No rows or columns in the input matrix, or an unspecified error has occurred in the WCSLIB routine
CPL_ERROR_UNSUPPORTED_MODE	The input conversion mode is not supported
CPL_ERROR_NO_WCS	The WCS sub library is not available.

This function converts between several types of coordinates. These include: – physical coordinates: The physical location on a detector (i.e. pixel coordinates) – world coordinates: The real astronomical coordinate system for the observations. This may be spectral, celestial, time, etc. – standard coordinates: These are an intermediate relative coordinate representation, defined as a distance from a reference point in the natural units of the world coordinate system. Any defined projection geometry will have already been included in the definition of standard coordinates.

The supported conversion modes are: – CPL_WCS_PHYS2WORLD: Converts input physical to world coordinates – CPL_WCS_WORLD2PHYS: Converts input world to physical coordinates – CPL_WCS_WORLD2STD: Converts input world to standard coordinates – CPL_WCS_PHYS2STD: Converts input physical to standard coordinates

The input cpl_matrix **from** has to be filled with coordinates. The number of rows equals the number of objects and the number of columns has to be equal to the value of the NAXIS keyword in the **wcs** structure. The same convention is used for the output cpl_matrix **to**. For example, if an image contains NAXIS = 2 and 100 stars with positions X,Y, the new matrix will be created:

```
from = cpl_matrix_new(100, 2);
```

Each element in column 0 will take a X coordinate and each element in column 1 will take a Y coordinate.

The output matrix and status arrays will be allocated here, and thus will need to be freed by the calling routine. The status array is used to flag input coordinates where there has been some sort of failure in the transformation.

References cpl_array_new(), cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NULL_INPUT, CPL_ERROR_UNSPECIFIED, cpl_matrix_get_ncol(), cpl_matrix_get_nrow(), cpl_matrix_new(), CPL_TYPE_DOUBLE, and CPL_TYPE_INT.

4.51.3.2 cpl_wcs_delete()

```
void cpl_wcs_delete (
            cpl_wcs * wcs )
```

Destroy a WCS structure.

Parameters

wcs	The WCS structure to destroy

Returns

Nothing.

Errors

CPL_ERROR_NO_WCS	The WCS sub library is not available.

The function destroys the WCS structure *wcs* and its whole contents. If *wcs* is `NULL`, nothing is done and no error is set.

References cpl_array_delete(), cpl_array_unwrap(), CPL_ERROR_NO_WCS, cpl_free(), and cpl_matrix_delete().

4.51.3.3 cpl_wcs_get_cd()

```
const cpl_matrix* cpl_wcs_get_cd (
            const cpl_wcs * wcs )
```

Accessor to get the CD matrix for a WCS.

Parameters

wcs	The WCS structure to examine

Returns

A handle to a matrix with the CDi_ja linear transformation matrix, or NULL on error.

Errors

CPL_ERROR_NULL_INPUT	The parameter *wcs* is a `NULL` pointer.
CPL_ERROR_NO_WCS	The WCS sub library is not available.

The function returns a handle to a matrix with the CD matrix defined for this WCS.

References cpl_ensure, CPL_ERROR_NO_WCS, and CPL_ERROR_NULL_INPUT.

4.51.3.4 cpl_wcs_get_crpix()

```
const cpl_array* cpl_wcs_get_crpix (
            const cpl_wcs * wcs )
```

Accessor to get the CRPIX vector for a WCS.

Parameters

wcs	The WCS structure to examine

Returns

A handle to an array with the CRPIXja keyvalues for each pixel axis, or NULL on error.

Errors

CPL_ERROR_NULL_INPUT	The parameter *wcs* is a `NULL` pointer.
CPL_ERROR_NO_WCS	The WCS sub library is not available.

The function returns a handle to an array with the CRPIX vector defined for this WCS.

References cpl_ensure, CPL_ERROR_NO_WCS, and CPL_ERROR_NULL_INPUT.

4.51.3.5 cpl_wcs_get_crval()

```
const cpl_array* cpl_wcs_get_crval (
            const cpl_wcs * wcs )
```

Accessor to get the CRVAL vector for a WCS.

Parameters

wcs	The WCS structure to examine

Returns

A handle to an array with the CRVALia keyvalues for each coord axis, or NULL on error.

Errors

CPL_ERROR_NULL_INPUT	The parameter *wcs* is a `NULL` pointer.
CPL_ERROR_NO_WCS	The WCS sub library is not available.

The function returns a handle to an array with the CRVAL vector defined for this WCS.

References cpl_ensure, CPL_ERROR_NO_WCS, and CPL_ERROR_NULL_INPUT.

4.51.3.6 cpl_wcs_get_ctype()

```
const cpl_array* cpl_wcs_get_ctype (
            const cpl_wcs * wcs )
```

Accessor to get the CTYPE vector for a WCS.

Parameters

wcs	The WCS structure to examine

Returns

A handle to an array with the CTYPEja keyvalues for each pixel axis, or NULL on error.

Errors

CPL_ERROR_NULL_INPUT	The parameter *wcs* is a `NULL` pointer.
CPL_ERROR_NO_WCS	The WCS sub library is not available.

The function returns a handle to an array with the CTYPE vector defined for this WCS.

References cpl_ensure, CPL_ERROR_NO_WCS, and CPL_ERROR_NULL_INPUT.

4.51.3.7 cpl_wcs_get_cunit()

```
const cpl_array* cpl_wcs_get_cunit (
            const cpl_wcs * wcs )
```

Accessor to get the CUNIT vector for a WCS.

Parameters

wcs	The WCS structure to examine

Returns

A handle to an array with the CUNITja keyvalues for each pixel axis, or NULL on error.

Errors

CPL_ERROR_NULL_INPUT	The parameter *wcs* is a `NULL` pointer.
CPL_ERROR_NO_WCS	The WCS sub library is not available.

The function returns a handle to an array with the CUNIT vector defined for this WCS.

References cpl_ensure, CPL_ERROR_NO_WCS, and CPL_ERROR_NULL_INPUT.

4.51.3.8 cpl_wcs_get_image_dims()

```
const cpl_array* cpl_wcs_get_image_dims (
            const cpl_wcs * wcs )
```

Accessor to get the axis lengths of the image associated with a WCS.

Parameters

wcs	The WCS structure to examine

Returns

An array with the image axis sizes, or NULL on error.

Errors

CPL_ERROR_NULL_INPUT	The parameter *wcs* is a `NULL` pointer.
CPL_ERROR_NO_WCS	The WCS sub library is not available.

The function returns a handle to an array with the axis lengths of the image associated with this WCS. If no image was used to define the WCS then a NULL value will be returned.

References cpl_ensure, CPL_ERROR_NO_WCS, and CPL_ERROR_NULL_INPUT.

4.51.3.9 cpl_wcs_get_image_naxis()

```
int cpl_wcs_get_image_naxis (
             const cpl_wcs * wcs )
```

Accessor to get the dimensionality of the image associated with a WCS.

Parameters

wcs	The WCS structure to examine

Returns

The image dimensionality, or zero on error

Errors

CPL_ERROR_NULL_INPUT	The parameter *wcs* is a `NULL` pointer.
CPL_ERROR_NO_WCS	The WCS sub library is not available.

The function returns the dimensionality of the image associated with a WCS. If no image was used to define the WCS then a value of zero is returned.

References cpl_ensure, CPL_ERROR_NO_WCS, and CPL_ERROR_NULL_INPUT.

4.51.3.10 cpl_wcs_new_from_propertylist()

```
cpl_wcs* cpl_wcs_new_from_propertylist (
            const cpl_propertylist * plist )
```

Create a wcs structure by parsing a propertylist.

Parameters

plist	The input propertylist

Returns

The newly created and filled cpl_wcs object or NULL if it could not be created. In the latter case an appropriate error code is set.

Errors

CPL_ERROR_NULL_INPUT	The parameter *plist* is a `NULL` pointer.
CPL_ERROR_TYPE_MISMATCH	NAXIS information in image propertylist is not an integer
CPL_ERROR_DATA_NOT_FOUND	Error in getting NAXIS information for image propertylists
CPL_ERROR_UNSPECIFIED	An unspecified error occurred in the WCSLIB routine.
CPL_ERROR_NO_WCS	The WCS sub library is not available.

The function allocates memory for a WCS structure. A pointer to the WCSLIB header information is created by parsing the FITS WCS keywords from the header of a file. A few ancillary items are also filled in.

It is allowed to pass a cpl_propertylist with a valid WCS structure and NAXIS = 0, such a propertylist can be created by cpl_wcs_platesol(). In this case a cpl_wcs object is returned for which the dimensional information (accessible via cpl_wcs_get_image_dims()) will be NULL.

The returned property must be destroyed using the wcs destructor **cpl_wcs_delete()**.

See also

cpl_wcs_delete()

References cpl_ensure, CPL_ERROR_NULL_INPUT, CPL_ERROR_UNSPECIFIED, cpl_malloc(), cpl_↩ propertylist_copy_property_regexp(), cpl_propertylist_delete(), cpl_propertylist_get_const(), cpl_propertylist_↩ get_size(), and cpl_propertylist_new().

Referenced by cpl_wcs_platesol().

4.51.3.11 cpl_wcs_platesol()

```
cpl_error_code cpl_wcs_platesol (
            const cpl_propertylist * ilist,
            const cpl_matrix * cel,
            const cpl_matrix * xy,
            int niter,
            float thresh,
            cpl_wcs_platesol_fitmode fitmode,
            cpl_wcs_platesol_outmode outmode,
            cpl_propertylist ** olist )
```

Do a 2d plate solution given physical and celestial coordinates.

Parameters

ilist	The input property list containing the first pass WCS
cel	The celestial coordinate matrix
xy	The physical coordinate matrix
niter	The number of fitting iterations
thresh	The threshold for the fitting rejection cycle
fitmode	The fitting mode (see below)
outmode	The output mode (see below)
olist	The output property list containing the new WCS

Returns

An appropriate error code.

Errors

CPL_ERROR_NULL↩ _INPUT	The parameter *cel* is a `NULL` pointer, the parameter *xy* is a `NULL` pointer or *ilist* is a `NULL` pointer.	CPL_ERROR_ILLEG↩ AL_INPUT	The parameter *niter* is non-positive.
CPL_ERROR_UNSP↩ ECIFIED	Unable to parse the input propertylist into a proper FITS WCS or there are too few points in the input matrices for a fit.		
CPL_ERROR_INCO↩ MPATIBLE_INPUT	The matrices *cel* and *xy* have different sizes.		
CPL_ERROR_UNSU↩ PPORTED_MODE	Either *fitmode* or *outmode* are specified incorrectly		
CPL_ERROR_DATA↩ _NOT_FOUND	The threshold is so low that no valid points are found. If the threshold is not positive, this error is certain to occur.		
CPL_ERROR_NO_↩ WCS	The WCS sub library is not available.		

This function allows for the following type of fits: – CPL_WCS_PLATESOL_4: Fit for zero point, 1 scale and 1 rotation. – CPL_WCS_PLATESOL_6: Fit for zero point, 2 scales, 1 rotation, 1 shear.

This function allows the zeropoint to be defined by shifting either the physical or the celestial coordinates of the reference point: – CPL_WCS_MV_CRVAL: Keeps the physical point fixed and shifts the celestial – CPL_WCS_↩ MV_CRPIX: Keeps the celestial point fixed and shifts the physical

The output property list contains WCS relevant information only.

The matrices *cel*, and *xy* have to be set up in the same way as it is required for **cpl_wcs_convert()**. See the documentation of **cpl_wcs_convert()** for details.

See also

cpl_wcs_convert()

References cpl_ensure_code, CPL_ERROR_ILLEGAL_INPUT, CPL_ERROR_NONE, CPL_ERROR_NULL_IN↩PUT, CPL_ERROR_UNSPECIFIED, CPL_ERROR_UNSUPPORTED_MODE, cpl_matrix_get_data_const(), cpl↩_matrix_get_nrow(), and cpl_wcs_new_from_propertylist().

Chapter 5

Class Documentation

5.1 _cpl_framedata_ Struct Reference

The public frame data object.

Public Attributes

- cpl_size max_count
- cpl_size min_count
- const char * tag

5.1.1 Detailed Description

The public frame data object.

The frame data object stores a frame identifier, the frame tag, and the minimum and maximum number of frames needed.

The data members of this structure are public to allow for a static initialization. Any other access of the public data members should still be done using the member functions.

5.1.2 Member Data Documentation

5.1.2.1 max_count

```
cpl_size _cpl_framedata_::max_count
```

The maximum number of frames of the kind given by the *tag*, the recipe requires in input. A value of `-1` means that the maximum number of frames is unspecified.

5.1.2.2 min_count

```
cpl_size _cpl_framedata_::min_count
```

The minimum number of frames of the kind given by the *tag*, the recipe requires in input. A value of `-1` means that the minimum number of frames is unspecified.

5.1.2.3 tag

```
const char* _cpl_framedata_::tag
```

The frame tag. A unique string identifier for a particular kind of frame.

5.2 _cpl_plugin_ Struct Reference

The type representation of the generic plugin interface.

Public Attributes

- unsigned int api
  The API version the Plugin complies to.
- const char * author
  Name of the plugin's author.
- const char * copyright
  Plugin's copyright.
- cpl_plugin_func deinitialize
  Deinitialization a plugin instance.
- const char * description
  Plugin's detailed description.
- const char * email
  Author's email address.
- cpl_plugin_func execute
  Executes a plugin instance.
- cpl_plugin_func initialize
  Initializes a plugin instance.
- const char * name
  Plugin's unique name.
- const char * synopsis
  Plugin's short help string.
- unsigned long type
  The Plugin type.
- unsigned long version
  The Plugin version.

5.2.1 Detailed Description

The type representation of the generic plugin interface.

5.2.2 Member Data Documentation

5.2.2.1 api

```
unsigned int _cpl_plugin_::api
```

The API version the Plugin complies to.

The API version number identifies the internal layout of the plugin interface structure. It may be used by an application calling a plugin to setup the correct interface to communicate with the plugin or, in the simplest case, to ignore any plugin which does not match the plugin API an application has been buid for.

Referenced by cpl_plugin_new().

5.2.2.2 author

```
const char* _cpl_plugin_::author
```

Name of the plugin's author.

Variable contains the null-terminated identifier string of the plugins author. If the plugin does not specify an author this pointer should be set to a `NULL` pointer.

5.2.2.3 copyright

```
const char* _cpl_plugin_::copyright
```

Plugin's copyright.

Variable contains the copyright and license string applying to the plugin. The returned string must be null-terminated. If no copyright applies this pointer should be set to a `NULL` pointer.

5.2.2.4 deinitialize

```
cpl_plugin_func _cpl_plugin_::deinitialize
```

Deinitialization a plugin instance.

Returns

The function must return 0 on success, and a non-zero value if the plugin deinitalization failed.

The function to deinitialize the plugin instance *plugin*. If this is `NULL` no deinitialization of the plugin instance is needed.

5.2.2.5 description

```
const char* _cpl_plugin_::description
```

Plugin's detailed description.

Variable contains the plugin's null-terminated detailed description string. The description is the detailed help for the plugin. For formatting the output the C special characters `'\n'`, `'\t'` maybe embedded in the returned string. If the plugin does not provide a detailed description the pointer should be set to a `NULL` pointer.

5.2.2.6 email

```
const char* _cpl_plugin_::email
```

Author's email address.

Variable contains the null-terminated string of the author's email address. If the plugin does not specify an email address this pointer should be set to a `NULL` pointer.

5.2.2.7 execute

```
cpl_plugin_func _cpl_plugin_::execute
```

Executes a plugin instance.

Parameters

plugin	The plugin to execute.

Returns

The function must return 0 on success, and a non-zero value if the plugin execution failed.

The function executes the plugin instance *plugin*.

5.2.2.8 initialize

```
cpl_plugin_func _cpl_plugin_::initialize
```

Initializes a plugin instance.

Parameters

plugin	The plugin to instantiate.

Returns

The function must return 0 on success, and a non-zero value if the plugin instantiation failed.

The function to initialize a plugin instance. This maybe NULL if the initialization of the plugin is not needed. Otherwise it has to be called before plugin type specific members are accessed.

5.2.2.9 name

```
const char* _cpl_plugin_::name
```

Plugin's unique name.

Variable contains the unique name of the Plugin. To ensure uniqueness across all possible Plugins one should follow the hierarchical naming convention mentioned in the CPL documentation.

5.2.2.10 synopsis

```
const char* _cpl_plugin_::synopsis
```

Plugin's short help string.

Variable contains the plugin's null-terminated short help string. The short help string should summarize the plugin's purpose in not more than a few lines. It may contain new line characters. If the plugin does not provide a short help the pointer should be set to a `NULL` pointer.

5.2.2.11 type

```
unsigned long _cpl_plugin_::type
```

The Plugin type.

The Plugin type identifies the type of plugin. The data type is not a cpl_plugin_type in order to keep this interface as generic as possible.

5.2.2.12 version

```
unsigned long _cpl_plugin_::version
```

The Plugin version.

The Plugin version number defines the version number for the plugin. The Plugin version number is an encoded version of the usual **MAJOR.MINOR.MICRO** form for version numbers.

5.3 _cpl_recipe_ Struct Reference

The type representation of the recipe plugin interface.

Public Attributes

- cpl_frameset * frames

  Pointer to a frame set, or NULL if no frame set is available.
- cpl_plugin interface

  Generic plugin interface.
- cpl_parameterlist * parameters

  Pointer to the recipes parameter list, or NULL if the recipe does not provide/accept any parameters.

5.3.1 Detailed Description

The type representation of the recipe plugin interface.

5.3.2 Member Data Documentation

5.3.2.1 frames

```
cpl_frameset* _cpl_recipe_::frames
```

Pointer to a frame set, or NULL if no frame set is available.

This member points to the frame set (see Frame Sets) the recipe should process. The frame set to process has to be provided by the application which is going to execute this recipe, i.e. this member has to be set by the application.

The recipe can rely on the availability of the frame set at the time the application executes the recipe by calling cpl_plugin::execute. The recipe is free to ignore a provided frame set if it does not need any input frames.

5.3.2.2 interface

```
cpl_plugin _cpl_recipe_::interface
```

Generic plugin interface.

See the Plugin Interface documentation for a detailed description.

5.3.2.3 parameters

```
cpl_parameterlist* _cpl_recipe_::parameters
```

Pointer to the recipes parameter list, or NULL if the recipe does not provide/accept any parameters.

This member points to a cpl_parameterlist, containing all parameters the recipe accepts, or NULL if the recipe does not need any parameters for execution.

An application which wants to execute the recipe may update this list with new parameter values, obtained from the command line for instance.

Index

_cpl_border_mode_
    Filters, 142
_cpl_error_code_
    Error handling, 126
_cpl_fft_mode_
    FFTW wrappers, 131
_cpl_filter_mode_
    Filters, 142
_cpl_fits_mode_
    FITS related basic routines, 136
_cpl_frame_group_
    Frames, 177
_cpl_frame_level_
    Frames, 177
_cpl_frame_type_
    Frames, 178
_cpl_framedata_, 999
    max_count, 999
    min_count, 999
    tag, 1000
_cpl_io_type_
    I/O, 252
_cpl_multiframe_id_mode_
    Multi Frames, 506
_cpl_parameter_class_
    Parameters, 534
_cpl_parameter_mode_
    Parameters, 534
_cpl_plugin_, 1000
    api, 1001
    author, 1001
    copyright, 1001
    deinitialize, 1001
    description, 1001
    email, 1002
    execute, 1002
    initialize, 1002
    name, 1003
    synopsis, 1003
    type, 1003
    version, 1003
_cpl_plugin_type_
    Plugin Interface, 582
_cpl_recipe_, 1003
    frames, 1004
    interface, 1004
    parameters, 1004
_cpl_regex_syntax_option_
    Regular Expression Filter, 758
_cpl_stats_mode_
    Statistics, 763
_cpl_type_
    Type codes, 916
_cpl_value_
    Images, 290

api
    _cpl_plugin_, 1001
Arrays, 9
    cpl_array_abs, 14
    cpl_array_add, 15
    cpl_array_add_scalar, 15
    cpl_array_add_scalar_complex, 16
    cpl_array_arg, 17
    cpl_array_cast, 17
    cpl_array_copy_data, 18
    cpl_array_copy_data_complex, 18
    cpl_array_copy_data_cplsize, 19
    cpl_array_copy_data_double, 19
    cpl_array_copy_data_double_complex, 20
    cpl_array_copy_data_float, 20
    cpl_array_copy_data_float_complex, 21
    cpl_array_copy_data_int, 21
    cpl_array_copy_data_long, 22
    cpl_array_copy_data_long_long, 22
    cpl_array_copy_data_string, 23
    cpl_array_count_invalid, 23
    cpl_array_delete, 24
    cpl_array_divide, 24
    cpl_array_divide_scalar, 25
    cpl_array_divide_scalar_complex, 26
    cpl_array_dump, 26
    cpl_array_dump_structure, 27
    cpl_array_duplicate, 28
    cpl_array_erase_window, 28
    cpl_array_exponential, 29
    cpl_array_extract, 29
    cpl_array_extract_imag, 30
    cpl_array_extract_real, 31
    cpl_array_fill_window, 31
    cpl_array_fill_window_complex, 32
    cpl_array_fill_window_cplsize, 33
    cpl_array_fill_window_double, 33
    cpl_array_fill_window_double_complex, 35
    cpl_array_fill_window_float, 36
    cpl_array_fill_window_float_complex, 36
    cpl_array_fill_window_int, 37
    cpl_array_fill_window_invalid, 38
    cpl_array_fill_window_long, 38

cpl_array_fill_window_long_long, 39
cpl_array_fill_window_string, 39
cpl_array_get, 40
cpl_array_get_complex, 41
cpl_array_get_cplsize, 41
cpl_array_get_data_cplsize, 42
cpl_array_get_data_cplsize_const, 42
cpl_array_get_data_double, 43
cpl_array_get_data_double_complex, 43
cpl_array_get_data_double_complex_const, 44
cpl_array_get_data_double_const, 44
cpl_array_get_data_float, 45
cpl_array_get_data_float_complex, 45
cpl_array_get_data_float_complex_const, 46
cpl_array_get_data_float_const, 46
cpl_array_get_data_int, 47
cpl_array_get_data_int_const, 47
cpl_array_get_data_long, 48
cpl_array_get_data_long_const, 48
cpl_array_get_data_long_long, 49
cpl_array_get_data_long_long_const, 49
cpl_array_get_data_string, 50
cpl_array_get_data_string_const, 50
cpl_array_get_double, 51
cpl_array_get_double_complex, 51
cpl_array_get_float, 52
cpl_array_get_float_complex, 52
cpl_array_get_int, 53
cpl_array_get_long, 53
cpl_array_get_long_long, 54
cpl_array_get_max, 55
cpl_array_get_maxpos, 55
cpl_array_get_mean, 56
cpl_array_get_mean_complex, 57
cpl_array_get_median, 57
cpl_array_get_min, 58
cpl_array_get_minpos, 58
cpl_array_get_size, 59
cpl_array_get_stdev, 60
cpl_array_get_string, 60
cpl_array_get_type, 61
cpl_array_has_invalid, 61
cpl_array_has_valid, 62
cpl_array_insert, 62
cpl_array_insert_window, 63
cpl_array_is_valid, 64
cpl_array_logarithm, 64
cpl_array_multiply, 65
cpl_array_multiply_scalar, 66
cpl_array_multiply_scalar_complex, 66
cpl_array_new, 67
cpl_array_power, 67
cpl_array_power_complex, 68
cpl_array_set, 69
cpl_array_set_complex, 69
cpl_array_set_cplsize, 70
cpl_array_set_double, 71
cpl_array_set_double_complex, 71
cpl_array_set_float, 72
cpl_array_set_float_complex, 72
cpl_array_set_int, 73
cpl_array_set_invalid, 73
cpl_array_set_long, 74
cpl_array_set_long_long, 75
cpl_array_set_size, 75
cpl_array_set_string, 76
cpl_array_subtract, 76
cpl_array_subtract_scalar, 77
cpl_array_subtract_scalar_complex, 78
cpl_array_unwrap, 78
cpl_array_wrap_cplsize, 79
cpl_array_wrap_double, 79
cpl_array_wrap_double_complex, 81
cpl_array_wrap_float, 81
cpl_array_wrap_float_complex, 82
cpl_array_wrap_int, 82
cpl_array_wrap_long, 83
cpl_array_wrap_long_long, 84
cpl_array_wrap_string, 84
author
_cpl_plugin_, 1001
Auxiliary Frame Data, 86
cpl_framedata, 87
cpl_framedata_clear, 87
cpl_framedata_create, 88
cpl_framedata_delete, 88
cpl_framedata_duplicate, 89
cpl_framedata_get_max_count, 89
cpl_framedata_get_min_count, 91
cpl_framedata_get_tag, 91
cpl_framedata_new, 92
cpl_framedata_set, 92
cpl_framedata_set_max_count, 93
cpl_framedata_set_min_count, 93
cpl_framedata_set_tag, 94

Bi-vector object, 96
cpl_bivector_copy, 97
cpl_bivector_delete, 97
cpl_bivector_dump, 98
cpl_bivector_duplicate, 98
cpl_bivector_get_size, 99
cpl_bivector_get_x, 99
cpl_bivector_get_x_const, 100
cpl_bivector_get_x_data, 100
cpl_bivector_get_x_data_const, 101
cpl_bivector_get_y, 102
cpl_bivector_get_y_const, 102
cpl_bivector_get_y_data, 103
cpl_bivector_get_y_data_const, 103
cpl_bivector_interpolate_linear, 104
cpl_bivector_new, 104
cpl_bivector_read, 105
cpl_bivector_sort, 106
cpl_bivector_unwrap_vectors, 107
cpl_bivector_wrap_vectors, 107

copyright
_cpl_plugin_, 1001
cpl_apertures_delete
High level functions to handle apertures, 211
cpl_apertures_dump
High level functions to handle apertures, 212
cpl_apertures_extract
High level functions to handle apertures, 212
cpl_apertures_extract_mask
High level functions to handle apertures, 213
cpl_apertures_extract_sigma
High level functions to handle apertures, 214
cpl_apertures_extract_window
High level functions to handle apertures, 215
cpl_apertures_get_bottom
High level functions to handle apertures, 215
cpl_apertures_get_bottom_x
High level functions to handle apertures, 217
cpl_apertures_get_centroid_x
High level functions to handle apertures, 217
cpl_apertures_get_centroid_y
High level functions to handle apertures, 218
cpl_apertures_get_flux
High level functions to handle apertures, 219
cpl_apertures_get_fwhm
High level functions to handle apertures, 219
cpl_apertures_get_left
High level functions to handle apertures, 220
cpl_apertures_get_left_y
High level functions to handle apertures, 220
cpl_apertures_get_max
High level functions to handle apertures, 221
cpl_apertures_get_max_x
High level functions to handle apertures, 222
cpl_apertures_get_max_y
High level functions to handle apertures, 222
cpl_apertures_get_maxpos_x
High level functions to handle apertures, 223
cpl_apertures_get_maxpos_y
High level functions to handle apertures, 223
cpl_apertures_get_mean
High level functions to handle apertures, 224
cpl_apertures_get_median
High level functions to handle apertures, 224
cpl_apertures_get_min
High level functions to handle apertures, 225
cpl_apertures_get_minpos_x
High level functions to handle apertures, 225
cpl_apertures_get_minpos_y
High level functions to handle apertures, 226
cpl_apertures_get_npix
High level functions to handle apertures, 227
cpl_apertures_get_pos_x
High level functions to handle apertures, 227
cpl_apertures_get_pos_y
High level functions to handle apertures, 228
cpl_apertures_get_right
High level functions to handle apertures, 228
cpl_apertures_get_right_y
High level functions to handle apertures, 229
cpl_apertures_get_size
High level functions to handle apertures, 230
cpl_apertures_get_stdev
High level functions to handle apertures, 230
cpl_apertures_get_top
High level functions to handle apertures, 231
cpl_apertures_get_top_x
High level functions to handle apertures, 231
cpl_apertures_new_from_image
High level functions to handle apertures, 232
cpl_apertures_sort_by_flux
High level functions to handle apertures, 233
cpl_apertures_sort_by_max
High level functions to handle apertures, 233
cpl_apertures_sort_by_npix
High level functions to handle apertures, 234
cpl_array_abs
Arrays, 14
cpl_array_add
Arrays, 15
cpl_array_add_scalar
Arrays, 15
cpl_array_add_scalar_complex
Arrays, 16
cpl_array_arg
Arrays, 17
cpl_array_cast
Arrays, 17
cpl_array_copy_data
Arrays, 18
cpl_array_copy_data_complex
Arrays, 18
cpl_array_copy_data_cplsize
Arrays, 19
cpl_array_copy_data_double
Arrays, 19
cpl_array_copy_data_double_complex
Arrays, 20
cpl_array_copy_data_float
Arrays, 20
cpl_array_copy_data_float_complex
Arrays, 21
cpl_array_copy_data_int
Arrays, 21
cpl_array_copy_data_long
Arrays, 22
cpl_array_copy_data_long_long
Arrays, 22
cpl_array_copy_data_string
Arrays, 23
cpl_array_count_invalid
Arrays, 23
cpl_array_delete
Arrays, 24
cpl_array_divide
Arrays, 24

cpl_array_divide_scalar
Arrays, 25
cpl_array_divide_scalar_complex
Arrays, 26
cpl_array_dump
Arrays, 26
cpl_array_dump_structure
Arrays, 27
cpl_array_duplicate
Arrays, 28
cpl_array_erase_window
Arrays, 28
cpl_array_exponential
Arrays, 29
cpl_array_extract
Arrays, 29
cpl_array_extract_imag
Arrays, 30
cpl_array_extract_real
Arrays, 31
cpl_array_fill_window
Arrays, 31
cpl_array_fill_window_complex
Arrays, 32
cpl_array_fill_window_cplsize
Arrays, 33
cpl_array_fill_window_double
Arrays, 33
cpl_array_fill_window_double_complex
Arrays, 35
cpl_array_fill_window_float
Arrays, 36
cpl_array_fill_window_float_complex
Arrays, 36
cpl_array_fill_window_int
Arrays, 37
cpl_array_fill_window_invalid
Arrays, 38
cpl_array_fill_window_long
Arrays, 38
cpl_array_fill_window_long_long
Arrays, 39
cpl_array_fill_window_string
Arrays, 39
cpl_array_get
Arrays, 40
cpl_array_get_complex
Arrays, 41
cpl_array_get_cplsize
Arrays, 41
cpl_array_get_data_cplsize
Arrays, 42
cpl_array_get_data_cplsize_const
Arrays, 42
cpl_array_get_data_double
Arrays, 43
cpl_array_get_data_double_complex
Arrays, 43
cpl_array_get_data_double_complex_const
Arrays, 44
cpl_array_get_data_double_const
Arrays, 44
cpl_array_get_data_float
Arrays, 45
cpl_array_get_data_float_complex
Arrays, 45
cpl_array_get_data_float_complex_const
Arrays, 46
cpl_array_get_data_float_const
Arrays, 46
cpl_array_get_data_int
Arrays, 47
cpl_array_get_data_int_const
Arrays, 47
cpl_array_get_data_long
Arrays, 48
cpl_array_get_data_long_const
Arrays, 48
cpl_array_get_data_long_long
Arrays, 49
cpl_array_get_data_long_long_const
Arrays, 49
cpl_array_get_data_string
Arrays, 50
cpl_array_get_data_string_const
Arrays, 50
cpl_array_get_double
Arrays, 51
cpl_array_get_double_complex
Arrays, 51
cpl_array_get_float
Arrays, 52
cpl_array_get_float_complex
Arrays, 52
cpl_array_get_int
Arrays, 53
cpl_array_get_long
Arrays, 53
cpl_array_get_long_long
Arrays, 54
cpl_array_get_max
Arrays, 55
cpl_array_get_maxpos
Arrays, 55
cpl_array_get_mean
Arrays, 56
cpl_array_get_mean_complex
Arrays, 57
cpl_array_get_median
Arrays, 57
cpl_array_get_min
Arrays, 58
cpl_array_get_minpos
Arrays, 58
cpl_array_get_size
Arrays, 59

cpl_array_get_stdev
Arrays, 60
cpl_array_get_string
Arrays, 60
cpl_array_get_type
Arrays, 61
cpl_array_has_invalid
Arrays, 61
cpl_array_has_valid
Arrays, 62
cpl_array_insert
Arrays, 62
cpl_array_insert_window
Arrays, 63
cpl_array_is_valid
Arrays, 64
cpl_array_logarithm
Arrays, 64
cpl_array_multiply
Arrays, 65
cpl_array_multiply_scalar
Arrays, 66
cpl_array_multiply_scalar_complex
Arrays, 66
cpl_array_new
Arrays, 67
cpl_array_power
Arrays, 67
cpl_array_power_complex
Arrays, 68
cpl_array_set
Arrays, 69
cpl_array_set_complex
Arrays, 69
cpl_array_set_cplsize
Arrays, 70
cpl_array_set_double
Arrays, 71
cpl_array_set_double_complex
Arrays, 71
cpl_array_set_float
Arrays, 72
cpl_array_set_float_complex
Arrays, 72
cpl_array_set_int
Arrays, 73
cpl_array_set_invalid
Arrays, 73
cpl_array_set_long
Arrays, 74
cpl_array_set_long_long
Arrays, 75
cpl_array_set_size
Arrays, 75
cpl_array_set_string
Arrays, 76
cpl_array_subtract
Arrays, 76
cpl_array_subtract_scalar
Arrays, 77
cpl_array_subtract_scalar_complex
Arrays, 78
cpl_array_unwrap
Arrays, 78
cpl_array_wrap_cplsize
Arrays, 79
cpl_array_wrap_double
Arrays, 79
cpl_array_wrap_double_complex
Arrays, 81
cpl_array_wrap_float
Arrays, 81
cpl_array_wrap_float_complex
Arrays, 82
cpl_array_wrap_int
Arrays, 82
cpl_array_wrap_long
Arrays, 83
cpl_array_wrap_long_long
Arrays, 84
cpl_array_wrap_string
Arrays, 84
cpl_bitmask
Type codes, 915
cpl_bivector_copy
Bi-vector object, 97
cpl_bivector_delete
Bi-vector object, 97
cpl_bivector_dump
Bi-vector object, 98
cpl_bivector_duplicate
Bi-vector object, 98
cpl_bivector_get_size
Bi-vector object, 99
cpl_bivector_get_x
Bi-vector object, 99
cpl_bivector_get_x_const
Bi-vector object, 100
cpl_bivector_get_x_data
Bi-vector object, 100
cpl_bivector_get_x_data_const
Bi-vector object, 101
cpl_bivector_get_y
Bi-vector object, 102
cpl_bivector_get_y_const
Bi-vector object, 102
cpl_bivector_get_y_data
Bi-vector object, 103
cpl_bivector_get_y_data_const
Bi-vector object, 103
cpl_bivector_interpolate_linear
Bi-vector object, 104
cpl_bivector_new
Bi-vector object, 104
cpl_bivector_read
Bi-vector object, 105

cpl_bivector_sort
Bi-vector object, 106
cpl_bivector_unwrap_vectors
Bi-vector object, 107
cpl_bivector_wrap_vectors
Bi-vector object, 107
CPL_BORDER_COPY
Filters, 142
CPL_BORDER_CROP
Filters, 142
CPL_BORDER_FILTER
Filters, 142
cpl_border_mode
Filters, 141
CPL_BORDER_NOP
Filters, 142
CPL_BORDER_ZERO
Filters, 142
CPL_BPP_16_SIGNED
I/O, 250
CPL_BPP_16_UNSIGNED
I/O, 250
CPL_BPP_32_SIGNED
I/O, 251
CPL_BPP_8_UNSIGNED
I/O, 251
CPL_BPP_IEEE_DOUBLE
I/O, 251
CPL_BPP_IEEE_FLOAT
I/O, 251
cpl_calloc
Memory Management Utilities, 481
cpl_detector_interpolate_rejected
High-level functions to compute detector features, 246
CPL_DFS_PRO_CATG
DFS related functions, 110
CPL_DFS_PRO_SCIENCE
DFS related functions, 110
CPL_DFS_PRO_TECH
DFS related functions, 110
CPL_DFS_PRO_TYPE
DFS related functions, 110
cpl_dfs_save_imagelist
DFS related functions, 111
cpl_dfs_save_paf
DFS related functions, 112
cpl_dfs_save_propertylist
DFS related functions, 113
cpl_dfs_save_table
DFS related functions, 114
cpl_dfs_setup_product_header
DFS related functions, 115
cpl_dfs_sign_products
DFS related functions, 116
CPL_DFS_SIGNATURE_CHECKSUM
DFS related functions, 111
Tables, 785
CPL_DFS_SIGNATURE_DATAMD5
DFS related functions, 111
Tables, 785
CPL_DFS_SIGNATURE_NONE
DFS related functions, 111
Tables, 785
cpl_dfs_update_product_header
DFS related functions, 117
cpl_end
Library Initialization, 396
cpl_ensure
Error handling, 121
cpl_ensure_code
Error handling, 122
CPL_ERROR_ACCESS_OUT_OF_RANGE
Error handling, 127
CPL_ERROR_ASSIGNING_STREAM
Error handling, 127
CPL_ERROR_BAD_FILE_FORMAT
Error handling, 127
cpl_error_code
Error handling, 126
CPL_ERROR_CONTINUE
Error handling, 127
CPL_ERROR_DATA_NOT_FOUND
Error handling, 127
CPL_ERROR_DIVISION_BY_ZERO
Error handling, 127
CPL_ERROR_DUPLICATING_STREAM
Error handling, 127
cpl_error_ensure
Error handling, 122
CPL_ERROR_EOL
Error handling, 127
CPL_ERROR_FILE_ALREADY_OPEN
Error handling, 127
CPL_ERROR_FILE_IO
Error handling, 127
CPL_ERROR_FILE_NOT_CREATED
Error handling, 127
CPL_ERROR_FILE_NOT_FOUND
Error handling, 127
cpl_error_get_code
Error handling, 127
cpl_error_get_file
Error handling, 128
cpl_error_get_function
Error handling, 128
cpl_error_get_line
Error handling, 128
cpl_error_get_message
Error handling, 129
cpl_error_get_message_default
Error handling, 129
cpl_error_get_where
Error handling, 130
CPL_ERROR_HISTORY_LOST
Error handling, 127

CPL_ERROR_ILLEGAL_INPUT
Error handling, 127
CPL_ERROR_ILLEGAL_OUTPUT
Error handling, 127
CPL_ERROR_INCOMPATIBLE_INPUT
Error handling, 127
CPL_ERROR_INVALID_TYPE
Error handling, 127
CPL_ERROR_MAX_MESSAGE_LENGTH
Error handling, 123
CPL_ERROR_NO_WCS
Error handling, 127
CPL_ERROR_NONE
Error handling, 127
CPL_ERROR_NULL_INPUT
Error handling, 127
cpl_error_set
Error handling, 124
cpl_error_set_message
Error handling, 124
cpl_error_set_where
Error handling, 125
CPL_ERROR_SINGULAR_MATRIX
Error handling, 127
CPL_ERROR_TYPE_MISMATCH
Error handling, 127
CPL_ERROR_UNSPECIFIED
Error handling, 127
CPL_ERROR_UNSUPPORTED_MODE
Error handling, 127
cpl_errorstate_dump
Handling of multiple CPL errors, 198
cpl_errorstate_dump_one
Handling of multiple CPL errors, 199
cpl_errorstate_dump_one_debug
Handling of multiple CPL errors, 200
cpl_errorstate_dump_one_info
Handling of multiple CPL errors, 200
cpl_errorstate_dump_one_warning
Handling of multiple CPL errors, 201
cpl_errorstate_get
Handling of multiple CPL errors, 201
cpl_errorstate_is_equal
Handling of multiple CPL errors, 202
cpl_errorstate_set
Handling of multiple CPL errors, 203
CPL_FFT_BACKWARD
FFTW wrappers, 132
CPL_FFT_FIND_EXHAUSTIVE
FFTW wrappers, 132
CPL_FFT_FIND_MEASURE
FFTW wrappers, 132
CPL_FFT_FIND_PATIENT
FFTW wrappers, 132
CPL_FFT_FORWARD
FFTW wrappers, 132
cpl_fft_image
FFTW wrappers, 132
cpl_fft_imagelist
FFTW wrappers, 133
cpl_fft_mode
FFTW wrappers, 131
CPL_FFT_NOSCALE
FFTW wrappers, 132
CPL_FILTER_AVERAGE
Filters, 144
CPL_FILTER_AVERAGE_FAST
Filters, 144
CPL_FILTER_CLOSING
Filters, 143
CPL_FILTER_DILATION
Filters, 143
CPL_FILTER_EROSION
Filters, 143
CPL_FILTER_LINEAR
Filters, 143
CPL_FILTER_LINEAR_SCALE
Filters, 144
CPL_FILTER_MEDIAN
Filters, 144
cpl_filter_mode
Filters, 142
CPL_FILTER_MORPHO
Filters, 145
CPL_FILTER_MORPHO_SCALE
Filters, 146
CPL_FILTER_OPENING
Filters, 143
CPL_FILTER_STDEV
Filters, 145
CPL_FILTER_STDEV_FAST
Filters, 145
cpl_fit_image_gaussian
High-level functions for non-linear fitting, 235
cpl_fit_imagelist_polynomial
High-level functions for non-linear fitting, 237
cpl_fit_imagelist_polynomial_window
High-level functions for non-linear fitting, 238
cpl_fit_lvmq
High-level functions for non-linear fitting, 240
cpl_fits_count_extensions
FITS related basic routines, 136
cpl_fits_find_extension
FITS related basic routines, 136
cpl_fits_get_extension_nb
FITS related basic routines, 137
cpl_fits_get_mode
FITS related basic routines, 138
cpl_fits_get_nb_extensions
FITS related basic routines, 138
cpl_fits_mode
FITS related basic routines, 135
CPL_FITS_ONE
FITS related basic routines, 136
CPL_FITS_RESTART_CACHING
FITS related basic routines, 136

cpl_fits_set_mode
FITS related basic routines, 138
CPL_FITS_START_CACHING
FITS related basic routines, 136
CPL_FITS_STOP_CACHING
FITS related basic routines, 136
cpl_flux_get_bias_window
High-level functions to compute detector features, 247
cpl_flux_get_noise_ring
High-level functions to compute detector features, 247
cpl_flux_get_noise_window
High-level functions to compute detector features, 248
cpl_frame
Frames, 176
cpl_frame_compare_func
Frames, 176
cpl_frame_delete
Frames, 178
cpl_frame_dump
Frames, 179
cpl_frame_duplicate
Frames, 179
cpl_frame_get_filename
Frames, 180
cpl_frame_get_group
Frames, 181
cpl_frame_get_level
Frames, 181
cpl_frame_get_nextensions
Frames, 182
cpl_frame_get_tag
Frames, 182
cpl_frame_get_type
Frames, 183
cpl_frame_group
Frames, 176
CPL_FRAME_GROUP_CALIB
Frames, 177
CPL_FRAME_GROUP_CALIB_ID
Frames, 175
CPL_FRAME_GROUP_NONE
Frames, 177
CPL_FRAME_GROUP_PRODUCT
Frames, 177
CPL_FRAME_GROUP_PRODUCT_ID
Frames, 176
CPL_FRAME_GROUP_RAW
Frames, 177
CPL_FRAME_GROUP_RAW_ID
Frames, 176
cpl_frame_level
Frames, 177
CPL_FRAME_LEVEL_FINAL
Frames, 178
CPL_FRAME_LEVEL_INTERMEDIATE
Frames, 178
CPL_FRAME_LEVEL_NONE
Frames, 178
CPL_FRAME_LEVEL_TEMPORARY
Frames, 178
cpl_frame_new
Frames, 184
cpl_frame_set_filename
Frames, 184
cpl_frame_set_group
Frames, 184
cpl_frame_set_level
Frames, 185
cpl_frame_set_tag
Frames, 186
cpl_frame_set_type
Frames, 186
cpl_frame_type
Frames, 177
CPL_FRAME_TYPE_ANY
Frames, 178
CPL_FRAME_TYPE_IMAGE
Frames, 178
CPL_FRAME_TYPE_MATRIX
Frames, 178
CPL_FRAME_TYPE_NONE
Frames, 178
CPL_FRAME_TYPE_PAF
Frames, 178
CPL_FRAME_TYPE_TABLE
Frames, 178
cpl_framedata
Auxiliary Frame Data, 87
cpl_framedata_clear
Auxiliary Frame Data, 87
cpl_framedata_create
Auxiliary Frame Data, 88
cpl_framedata_delete
Auxiliary Frame Data, 88
cpl_framedata_duplicate
Auxiliary Frame Data, 89
cpl_framedata_get_max_count
Auxiliary Frame Data, 89
cpl_framedata_get_min_count
Auxiliary Frame Data, 91
cpl_framedata_get_tag
Auxiliary Frame Data, 91
cpl_framedata_new
Auxiliary Frame Data, 92
cpl_framedata_set
Auxiliary Frame Data, 92
cpl_framedata_set_max_count
Auxiliary Frame Data, 93
cpl_framedata_set_min_count
Auxiliary Frame Data, 93
cpl_framedata_set_tag
Auxiliary Frame Data, 94
cpl_frameset

Frame Sets, 155
cpl_frameset_count_tags
Frame Sets, 155
cpl_frameset_delete
Frame Sets, 156
cpl_frameset_dump
Frame Sets, 156
cpl_frameset_duplicate
Frame Sets, 157
cpl_frameset_erase
Frame Sets, 157
cpl_frameset_erase_frame
Frame Sets, 158
cpl_frameset_extract
Frame Sets, 159
cpl_frameset_find
Frame Sets, 159
cpl_frameset_find_const
Frame Sets, 160
cpl_frameset_get_first
Frame Sets, 161
cpl_frameset_get_first_const
Frame Sets, 162
cpl_frameset_get_frame
Frame Sets, 162
cpl_frameset_get_frame_const
Frame Sets, 163
cpl_frameset_get_next
Frame Sets, 164
cpl_frameset_get_next_const
Frame Sets, 165
cpl_frameset_get_position
Frame Sets, 166
cpl_frameset_get_position_const
Frame Sets, 166
cpl_frameset_get_size
Frame Sets, 167
cpl_frameset_insert
Frame Sets, 168
cpl_frameset_is_empty
Frame Sets, 168
cpl_frameset_iterator
Frame Set Iterators, 147
cpl_frameset_iterator_advance
Frame Set Iterators, 148
cpl_frameset_iterator_assign
Frame Set Iterators, 148
cpl_frameset_iterator_delete
Frame Set Iterators, 149
cpl_frameset_iterator_distance
Frame Set Iterators, 149
cpl_frameset_iterator_duplicate
Frame Set Iterators, 150
cpl_frameset_iterator_get
Frame Set Iterators, 151
cpl_frameset_iterator_get_const
Frame Set Iterators, 151
cpl_frameset_iterator_new
Frame Set Iterators, 152
cpl_frameset_iterator_reset
Frame Set Iterators, 153
cpl_frameset_join
Frame Sets, 169
cpl_frameset_labelise
Frame Sets, 170
cpl_frameset_new
Frame Sets, 170
cpl_frameset_sort
Frame Sets, 171
cpl_free
Memory Management Utilities, 482
cpl_gaussian_eval_2d
High-level functions for non-linear fitting, 242
cpl_geom_combine
High level functions for geometric transformations, 204
CPL_GEOM_FIRST
High level functions for geometric transformations, 204
cpl_geom_img_offset_combine
High level functions for geometric transformations, 205
cpl_geom_img_offset_fine
High level functions for geometric transformations, 206
cpl_geom_img_offset_saa
High level functions for geometric transformations, 207
CPL_GEOM_INTERSECT
High level functions for geometric transformations, 204
CPL_GEOM_UNION
High level functions for geometric transformations, 204
cpl_get_description
Library Initialization, 396
cpl_get_license
Recipe Definition, 753
CPL_HAVE_VA_ARGS
Error handling, 126
cpl_image_abs
Images, 291
cpl_image_abs_create
Images, 291
cpl_image_accept
Images, 292
cpl_image_accept_all
Images, 292
cpl_image_add
Images, 293
cpl_image_add_create
Images, 293
cpl_image_add_scalar
Images, 294
cpl_image_add_scalar_create
Images, 294

cpl_image_and
Images, 295
cpl_image_and_scalar
Images, 296
cpl_image_average_create
Images, 296
cpl_image_cast
Images, 297
cpl_image_collapse_create
Images, 298
cpl_image_collapse_median_create
Images, 299
cpl_image_collapse_window_create
Images, 300
cpl_image_conjugate
Images, 300
cpl_image_copy
Images, 301
cpl_image_count_rejected
Images, 302
cpl_image_delete
Images, 302
cpl_image_divide
Images, 303
cpl_image_divide_create
Images, 304
cpl_image_divide_scalar
Images, 305
cpl_image_divide_scalar_create
Images, 305
cpl_image_dump_structure
Images, 306
cpl_image_dump_window
Images, 306
cpl_image_duplicate
Images, 307
cpl_image_exponential
Images, 308
cpl_image_exponential_create
Images, 309
cpl_image_extract
Images, 309
cpl_image_extract_subsample
Images, 310
cpl_image_fft
Images, 311
cpl_image_fill_abs_arg
Images, 312
cpl_image_fill_gaussian
Images, 313
cpl_image_fill_jacobian
Images, 314
cpl_image_fill_jacobian_polynomial
Images, 315
cpl_image_fill_noise_uniform
Images, 316
cpl_image_fill_polynomial
Images, 317
cpl_image_fill_re_im
Images, 318
cpl_image_fill_rejected
Images, 319
cpl_image_fill_test_create
Images, 319
cpl_image_fill_window
Images, 320
cpl_image_filter
Images, 321
cpl_image_filter_linear
Images, 322
cpl_image_filter_mask
Images, 323
cpl_image_filter_median
Images, 324
cpl_image_filter_morpho
Images, 326
cpl_image_filter_stdev
Images, 326
cpl_image_fit_gaussian
Images, 327
cpl_image_flip
Images, 328
cpl_image_get
Images, 329
cpl_image_get_absflux
Images, 330
cpl_image_get_absflux_window
Images, 330
cpl_image_get_bpm
Images, 331
cpl_image_get_bpm_const
Images, 332
cpl_image_get_centroid_x
Images, 332
cpl_image_get_centroid_x_window
Images, 333
cpl_image_get_centroid_y
Images, 333
cpl_image_get_centroid_y_window
Images, 334
cpl_image_get_complex
Images, 334
cpl_image_get_data
Images, 335
cpl_image_get_data_const
Images, 336
cpl_image_get_data_double
Images, 336
cpl_image_get_data_double_complex
Images, 337
cpl_image_get_data_double_complex_const
Images, 337
cpl_image_get_data_double_const
Images, 338
cpl_image_get_data_float
Images, 338

cpl_image_get_data_float_complex
Images, 339
cpl_image_get_data_float_complex_const
Images, 339
cpl_image_get_data_float_const
Images, 340
cpl_image_get_data_int
Images, 340
cpl_image_get_data_int_const
Images, 341
cpl_image_get_flux
Images, 341
cpl_image_get_flux_window
Images, 341
cpl_image_get_fwhm
Images, 342
cpl_image_get_interpolated
Images, 343
cpl_image_get_mad
Images, 344
cpl_image_get_mad_window
Images, 345
cpl_image_get_max
Images, 346
cpl_image_get_max_window
Images, 346
cpl_image_get_maxpos
Images, 347
cpl_image_get_maxpos_window
Images, 347
cpl_image_get_mean
Images, 348
cpl_image_get_mean_window
Images, 349
cpl_image_get_median
Images, 349
cpl_image_get_median_dev
Images, 350
cpl_image_get_median_dev_window
Images, 350
cpl_image_get_median_window
Images, 351
cpl_image_get_min
Images, 352
cpl_image_get_min_window
Images, 353
cpl_image_get_minpos
Images, 353
cpl_image_get_minpos_window
Images, 354
cpl_image_get_size_x
Images, 355
cpl_image_get_size_y
Images, 355
cpl_image_get_sqflux
Images, 356
cpl_image_get_sqflux_window
Images, 356
cpl_image_get_stdev
Images, 357
cpl_image_get_stdev_window
Images, 357
cpl_image_get_type
Images, 358
cpl_image_hypot
Images, 359
cpl_image_iqe
Images, 359
cpl_image_is_rejected
Images, 361
cpl_image_labelise_mask_create
Images, 361
cpl_image_load
Images, 362
cpl_image_load_window
Images, 363
cpl_image_logarithm
Images, 364
cpl_image_logarithm_create
Images, 365
cpl_image_move
Images, 365
cpl_image_multiply
Images, 366
cpl_image_multiply_create
Images, 367
cpl_image_multiply_scalar
Images, 367
cpl_image_multiply_scalar_create
Images, 368
cpl_image_new
Images, 368
cpl_image_new_from_accepted
Imagelists, 255
cpl_image_new_from_mask
Images, 369
cpl_image_normalise
Images, 370
cpl_image_normalise_create
Images, 370
cpl_image_not
Images, 371
cpl_image_or
Images, 372
cpl_image_or_scalar
Images, 373
cpl_image_power
Images, 373
cpl_image_power_create
Images, 374
cpl_image_rebin
Images, 374
cpl_image_reject
Images, 375
cpl_image_reject_from_mask
Images, 376

cpl_image_reject_value
Images, 376
cpl_image_save
Images, 377
cpl_image_set
Images, 378
cpl_image_set_bpm
Images, 379
cpl_image_set_complex
Images, 380
cpl_image_shift
Images, 380
cpl_image_subtract
Images, 381
cpl_image_subtract_create
Images, 382
cpl_image_subtract_scalar
Images, 382
cpl_image_subtract_scalar_create
Images, 383
cpl_image_threshold
Images, 383
cpl_image_turn
Images, 384
cpl_image_unset_bpm
Images, 385
cpl_image_unwrap
Images, 386
cpl_image_warp
Images, 386
cpl_image_warp_polynomial
Images, 388
cpl_image_wrap_double
Images, 389
cpl_image_wrap_double_complex
Images, 390
cpl_image_wrap_float
Images, 390
cpl_image_wrap_float_complex
Images, 391
cpl_image_wrap_int
Images, 392
cpl_image_xor
Images, 392
cpl_image_xor_scalar
Images, 393
cpl_imagelist_add
Imagelists, 255
cpl_imagelist_add_image
Imagelists, 256
cpl_imagelist_add_scalar
Imagelists, 256
cpl_imagelist_cast
Imagelists, 257
cpl_imagelist_collapse_create
Imagelists, 258
cpl_imagelist_collapse_median_create
Imagelists, 259
cpl_imagelist_collapse_minmax_create
Imagelists, 259
cpl_imagelist_collapse_sigclip_create
Imagelists, 260
cpl_imagelist_delete
Imagelists, 262
cpl_imagelist_divide
Imagelists, 262
cpl_imagelist_divide_image
Imagelists, 263
cpl_imagelist_divide_scalar
Imagelists, 263
cpl_imagelist_dump_structure
Imagelists, 264
cpl_imagelist_dump_window
Imagelists, 264
cpl_imagelist_duplicate
Imagelists, 265
cpl_imagelist_empty
Imagelists, 266
cpl_imagelist_erase
Imagelists, 266
cpl_imagelist_exponential
Imagelists, 267
cpl_imagelist_get
Imagelists, 268
cpl_imagelist_get_const
Imagelists, 268
cpl_imagelist_get_size
Imagelists, 269
cpl_imagelist_is_uniform
Imagelists, 269
cpl_imagelist_load
Imagelists, 270
cpl_imagelist_load_frameset
Frame Sets IO functions, 173
cpl_imagelist_load_window
Imagelists, 271
cpl_imagelist_logarithm
Imagelists, 272
cpl_imagelist_multiply
Imagelists, 272
cpl_imagelist_multiply_image
Imagelists, 273
cpl_imagelist_multiply_scalar
Imagelists, 273
cpl_imagelist_new
Imagelists, 274
cpl_imagelist_normalise
Imagelists, 274
cpl_imagelist_power
Imagelists, 275
cpl_imagelist_save
Imagelists, 275
cpl_imagelist_set
Imagelists, 277
cpl_imagelist_subtract
Imagelists, 278

cpl_imagelist_subtract_image
Imagelists, 278
cpl_imagelist_subtract_scalar
Imagelists, 279
cpl_imagelist_swap_axis_create
Imagelists, 279
cpl_imagelist_threshold
Imagelists, 280
cpl_imagelist_unset
Imagelists, 281
cpl_imagelist_unwrap
Imagelists, 281
cpl_init
Library Initialization, 397
CPL_IO_APPEND
I/O, 252
CPL_IO_COMPRESS_GZIP
I/O, 252
CPL_IO_COMPRESS_HCOMPRESS
I/O, 252
CPL_IO_COMPRESS_PLIO
I/O, 252
CPL_IO_COMPRESS_RICE
I/O, 252
CPL_IO_CREATE
I/O, 252
CPL_IO_DEFAULT
I/O, 252
CPL_IO_EXTEND
I/O, 252
CPL_IO_MAX
I/O, 252
cpl_io_type
I/O, 252
cpl_malloc
Memory Management Utilities, 482
cpl_mask_and
Masks of pixels, 403
cpl_mask_closing
Masks of pixels, 403
cpl_mask_collapse_create
Masks of pixels, 404
cpl_mask_copy
Masks of pixels, 404
cpl_mask_count
Masks of pixels, 405
cpl_mask_count_window
Masks of pixels, 406
cpl_mask_delete
Masks of pixels, 406
cpl_mask_dilation
Masks of pixels, 407
cpl_mask_dump_window
Masks of pixels, 407
cpl_mask_duplicate
Masks of pixels, 409
cpl_mask_erosion
Masks of pixels, 409
cpl_mask_extract
Masks of pixels, 410
cpl_mask_extract_subsample
Masks of pixels, 411
cpl_mask_filter
Masks of pixels, 411
cpl_mask_flip
Masks of pixels, 413
cpl_mask_get
Masks of pixels, 414
cpl_mask_get_data
Masks of pixels, 414
cpl_mask_get_data_const
Masks of pixels, 415
cpl_mask_get_size_x
Masks of pixels, 415
cpl_mask_get_size_y
Masks of pixels, 416
cpl_mask_is_empty
Masks of pixels, 416
cpl_mask_is_empty_window
Masks of pixels, 417
cpl_mask_load
Masks of pixels, 418
cpl_mask_load_window
Masks of pixels, 418
cpl_mask_move
Masks of pixels, 419
cpl_mask_new
Masks of pixels, 420
cpl_mask_not
Masks of pixels, 421
cpl_mask_opening
Masks of pixels, 421
cpl_mask_or
Masks of pixels, 422
cpl_mask_save
Masks of pixels, 422
cpl_mask_set
Masks of pixels, 423
cpl_mask_shift
Masks of pixels, 424
cpl_mask_threshold_image
Masks of pixels, 425
cpl_mask_threshold_image_create
Masks of pixels, 426
cpl_mask_turn
Masks of pixels, 426
cpl_mask_unwrap
Masks of pixels, 427
cpl_mask_wrap
Masks of pixels, 428
cpl_mask_xor
Masks of pixels, 428
CPL_MATH_1_PI
Fundamental math functionality, 189
CPL_MATH_2_PI
Fundamental math functionality, 189

CPL_MATH_2_SQRTPI
Fundamental math functionality, 190
CPL_MATH_2PI
Fundamental math functionality, 190
CPL_MATH_4_PI
Fundamental math functionality, 190
CPL_MATH_DEG_RAD
Fundamental math functionality, 191
CPL_MATH_E
Fundamental math functionality, 191
CPL_MATH_FWHM_SIG
Fundamental math functionality, 191
CPL_MATH_LN10
Fundamental math functionality, 192
CPL_MATH_LN2
Fundamental math functionality, 192
CPL_MATH_LOG10E
Fundamental math functionality, 192
CPL_MATH_LOG2E
Fundamental math functionality, 192
CPL_MATH_PI
Fundamental math functionality, 193
CPL_MATH_PI_2
Fundamental math functionality, 193
CPL_MATH_PI_4
Fundamental math functionality, 193
CPL_MATH_RAD_DEG
Fundamental math functionality, 194
CPL_MATH_SIG_FWHM
Fundamental math functionality, 194
CPL_MATH_SQRT1_2
Fundamental math functionality, 194
CPL_MATH_SQRT2
Fundamental math functionality, 195
CPL_MATH_SQRT2PI
Fundamental math functionality, 195
CPL_MATH_SQRT3
Fundamental math functionality, 195
CPL_MATH_STD_MAD
Fundamental math functionality, 196
cpl_matrix_add
Matrices, 433
cpl_matrix_add_scalar
Matrices, 433
cpl_matrix_append
Matrices, 434
cpl_matrix_copy
Matrices, 435
cpl_matrix_decomp_chol
Matrices, 435
cpl_matrix_delete
Matrices, 437
cpl_matrix_divide
Matrices, 437
cpl_matrix_divide_scalar
Matrices, 438
cpl_matrix_dump
Matrices, 439
cpl_matrix_duplicate
Matrices, 439
cpl_matrix_erase_columns
Matrices, 440
cpl_matrix_erase_rows
Matrices, 441
cpl_matrix_exponential
Matrices, 441
cpl_matrix_extract
Matrices, 442
cpl_matrix_extract_column
Matrices, 443
cpl_matrix_extract_diagonal
Matrices, 443
cpl_matrix_extract_row
Matrices, 444
cpl_matrix_fill
Matrices, 445
cpl_matrix_fill_column
Matrices, 445
cpl_matrix_fill_diagonal
Matrices, 446
cpl_matrix_fill_row
Matrices, 447
cpl_matrix_fill_window
Matrices, 447
cpl_matrix_flip_columns
Matrices, 448
cpl_matrix_flip_rows
Matrices, 449
cpl_matrix_get
Matrices, 449
cpl_matrix_get_data
Matrices, 450
cpl_matrix_get_data_const
Matrices, 451
cpl_matrix_get_determinant
Matrices, 451
cpl_matrix_get_max
Matrices, 452
cpl_matrix_get_maxpos
Matrices, 452
cpl_matrix_get_mean
Matrices, 453
cpl_matrix_get_median
Matrices, 454
cpl_matrix_get_min
Matrices, 454
cpl_matrix_get_minpos
Matrices, 455
cpl_matrix_get_ncol
Matrices, 455
cpl_matrix_get_ncol_
Matrices, 456
cpl_matrix_get_nrow
Matrices, 457
cpl_matrix_get_stdev
Matrices, 457

cpl_matrix_invert_create
Matrices, 458
cpl_matrix_is_diagonal
Matrices, 459
cpl_matrix_is_identity
Matrices, 459
cpl_matrix_is_zero
Matrices, 460
cpl_matrix_logarithm
Matrices, 461
cpl_matrix_multiply
Matrices, 461
cpl_matrix_multiply_scalar
Matrices, 462
cpl_matrix_new
Matrices, 462
cpl_matrix_power
Matrices, 463
cpl_matrix_product_create
Matrices, 464
cpl_matrix_resize
Matrices, 464
cpl_matrix_set_
Matrices, 465
cpl_matrix_set_size
Matrices, 467
cpl_matrix_shift
Matrices, 468
cpl_matrix_solve
Matrices, 468
cpl_matrix_solve_chol
Matrices, 469
cpl_matrix_solve_normal
Matrices, 470
cpl_matrix_solve_svd
Matrices, 471
cpl_matrix_solve_svd_threshold
Matrices, 472
cpl_matrix_sort_columns
Matrices, 473
cpl_matrix_sort_rows
Matrices, 473
cpl_matrix_subtract
Matrices, 474
cpl_matrix_subtract_scalar
Matrices, 475
cpl_matrix_swap_columns
Matrices, 475
cpl_matrix_swap_rowcolumn
Matrices, 476
cpl_matrix_swap_rows
Matrices, 477
cpl_matrix_threshold_small
Matrices, 477
cpl_matrix_transpose_create
Matrices, 478
cpl_matrix_unwrap
Matrices, 479
cpl_matrix_wrap
Matrices, 479
CPL_MAX
Fundamental math functionality, 196
cpl_memory_dump
Memory Management Utilities, 483
cpl_memory_is_empty
Memory Management Utilities, 483
CPL_MIN
Fundamental math functionality, 196
cpl_msg_debug
Messages, 489
cpl_msg_error
Messages, 489
cpl_msg_get_domain
Messages, 490
cpl_msg_get_level
Messages, 490
cpl_msg_get_log_level
Messages, 490
cpl_msg_get_log_name
Messages, 490
cpl_msg_indent
Messages, 491
cpl_msg_indent_less
Messages, 491
cpl_msg_indent_more
Messages, 491
cpl_msg_info
Messages, 492
cpl_msg_info_overwritable
Messages, 492
cpl_msg_init
Messages, 493
cpl_msg_progress
Messages, 494
cpl_msg_set_component_off
Messages, 494
cpl_msg_set_component_on
Messages, 495
cpl_msg_set_domain
Messages, 495
cpl_msg_set_domain_off
Messages, 496
cpl_msg_set_domain_on
Messages, 496
cpl_msg_set_indentation
Messages, 497
cpl_msg_set_level
Messages, 497
cpl_msg_set_level_from_env
Messages, 498
cpl_msg_set_log_level
Messages, 498
cpl_msg_set_log_name
Messages, 499
cpl_msg_set_threadid_off
Messages, 500

cpl_msg_set_threadid_on
Messages, 500
cpl_msg_set_time_off
Messages, 501
cpl_msg_set_time_on
Messages, 501
cpl_msg_set_width
Messages, 502
cpl_msg_stop
Messages, 502
cpl_msg_stop_log
Messages, 503
cpl_msg_warning
Messages, 503
cpl_multiframe
Multi Frames, 506
cpl_multiframe_add_empty
Multi Frames, 507
cpl_multiframe_append_datagroup
Multi Frames, 507
cpl_multiframe_append_datagroup_from_position
Multi Frames, 508
cpl_multiframe_append_dataset
Multi Frames, 509
cpl_multiframe_append_dataset_from_position
Multi Frames, 510
cpl_multiframe_delete
Multi Frames, 511
cpl_multiframe_get_size
Multi Frames, 511
CPL_MULTIFRAME_ID_JOIN
Multi Frames, 506
cpl_multiframe_id_mode
Multi Frames, 506
CPL_MULTIFRAME_ID_PREFIX
Multi Frames, 506
CPL_MULTIFRAME_ID_SET
Multi Frames, 506
cpl_multiframe_new
Multi Frames, 512
cpl_multiframe_write
Multi Frames, 512
cpl_parameter
Parameters, 533
cpl_parameter_class
Parameters, 533
CPL_PARAMETER_CLASS_ENUM
Parameters, 534
CPL_PARAMETER_CLASS_INVALID
Parameters, 534
CPL_PARAMETER_CLASS_RANGE
Parameters, 534
CPL_PARAMETER_CLASS_VALUE
Parameters, 534
cpl_parameter_delete
Parameters, 535
cpl_parameter_disable
Parameters, 535
cpl_parameter_dump
Parameters, 536
cpl_parameter_duplicate
Parameters, 536
cpl_parameter_enable
Parameters, 537
cpl_parameter_get_alias
Parameters, 537
cpl_parameter_get_bool
Parameters, 538
cpl_parameter_get_class
Parameters, 539
cpl_parameter_get_context
Parameters, 539
cpl_parameter_get_default_bool
Parameters, 540
cpl_parameter_get_default_double
Parameters, 541
cpl_parameter_get_default_flag
Parameters, 541
cpl_parameter_get_default_int
Parameters, 542
cpl_parameter_get_default_string
Parameters, 543
cpl_parameter_get_double
Parameters, 543
cpl_parameter_get_enum_double
Parameters, 544
cpl_parameter_get_enum_int
Parameters, 545
cpl_parameter_get_enum_size
Parameters, 545
cpl_parameter_get_enum_string
Parameters, 546
cpl_parameter_get_help
Parameters, 547
cpl_parameter_get_id
Parameters, 548
cpl_parameter_get_int
Parameters, 548
cpl_parameter_get_name
Parameters, 549
cpl_parameter_get_range_max_double
Parameters, 550
cpl_parameter_get_range_max_int
Parameters, 550
cpl_parameter_get_range_min_double
Parameters, 551
cpl_parameter_get_range_min_int
Parameters, 551
cpl_parameter_get_string
Parameters, 552
cpl_parameter_get_tag
Parameters, 553
cpl_parameter_get_type
Parameters, 553
cpl_parameter_is_enabled
Parameters, 554

cpl_parameter_mode
Parameters, 534
CPL_PARAMETER_MODE_CFG
Parameters, 534
CPL_PARAMETER_MODE_CLI
Parameters, 534
CPL_PARAMETER_MODE_ENV
Parameters, 534
cpl_parameter_new_enum
Parameters, 555
cpl_parameter_new_enum_from_array
Parameters, 556
cpl_parameter_new_range
Parameters, 557
cpl_parameter_new_value
Parameters, 558
cpl_parameter_set_alias
Parameters, 559
cpl_parameter_set_bool
Parameters, 560
cpl_parameter_set_default_bool
Parameters, 561
cpl_parameter_set_default_double
Parameters, 562
cpl_parameter_set_default_flag
Parameters, 562
cpl_parameter_set_default_int
Parameters, 563
cpl_parameter_set_default_string
Parameters, 564
cpl_parameter_set_double
Parameters, 564
cpl_parameter_set_id
Parameters, 565
cpl_parameter_set_int
Parameters, 565
cpl_parameter_set_string
Parameters, 566
cpl_parameter_set_tag
Parameters, 567
cpl_parameterlist
Parameter Lists, 515
cpl_parameterlist_append
Parameter Lists, 515
cpl_parameterlist_delete
Parameter Lists, 516
cpl_parameterlist_dump
Parameter Lists, 516
cpl_parameterlist_find
Parameter Lists, 517
cpl_parameterlist_find_const
Parameter Lists, 517
cpl_parameterlist_find_context
Parameter Lists, 518
cpl_parameterlist_find_context_const
Parameter Lists, 519
cpl_parameterlist_find_tag
Parameter Lists, 519
cpl_parameterlist_find_tag_const
Parameter Lists, 520
cpl_parameterlist_find_type
Parameter Lists, 522
cpl_parameterlist_find_type_const
Parameter Lists, 522
cpl_parameterlist_get_first
Parameter Lists, 524
cpl_parameterlist_get_first_const
Parameter Lists, 525
cpl_parameterlist_get_last
Parameter Lists, 525
cpl_parameterlist_get_last_const
Parameter Lists, 526
cpl_parameterlist_get_next
Parameter Lists, 526
cpl_parameterlist_get_next_const
Parameter Lists, 527
cpl_parameterlist_get_size
Parameter Lists, 528
cpl_parameterlist_new
Parameter Lists, 528
cpl_photom_fill_blackbody
High-level functions that are photometry related, 244
cpl_plot_bivector
Plotting of CPL objects, 569
cpl_plot_bivectors
Plotting of CPL objects, 569
cpl_plot_column
Plotting of CPL objects, 570
cpl_plot_columns
Plotting of CPL objects, 571
cpl_plot_image
Plotting of CPL objects, 572
cpl_plot_image_col
Plotting of CPL objects, 573
cpl_plot_image_row
Plotting of CPL objects, 574
cpl_plot_mask
Plotting of CPL objects, 575
cpl_plot_vector
Plotting of CPL objects, 576
cpl_plot_vectors
Plotting of CPL objects, 576
cpl_plugin
Plugin Interface, 581
CPL_PLUGIN_API
Plugin Interface, 581
cpl_plugin_copy
Plugin Interface, 582
cpl_plugin_delete
Plugin Interface, 583
cpl_plugin_dump
Plugin Interface, 584
cpl_plugin_get_api
Plugin Interface, 584
cpl_plugin_get_author

Plugin Interface, 585
cpl_plugin_get_copyright
Plugin Interface, 585
cpl_plugin_get_deinit
Plugin Interface, 586
cpl_plugin_get_description
Plugin Interface, 587
cpl_plugin_get_email
Plugin Interface, 587
cpl_plugin_get_exec
Plugin Interface, 588
cpl_plugin_get_info
Plugin Interface, 589
cpl_plugin_get_init
Plugin Interface, 589
cpl_plugin_get_name
Plugin Interface, 590
cpl_plugin_get_synopsis
Plugin Interface, 590
cpl_plugin_get_type
Plugin Interface, 591
cpl_plugin_get_type_string
Plugin Interface, 592
cpl_plugin_get_version
Plugin Interface, 592
cpl_plugin_get_version_string
Plugin Interface, 593
cpl_plugin_init
Plugin Interface, 593
cpl_plugin_new
Plugin Interface, 595
cpl_plugin_set_api
Plugin Interface, 595
cpl_plugin_set_author
Plugin Interface, 596
cpl_plugin_set_copyright
Plugin Interface, 596
cpl_plugin_set_deinit
Plugin Interface, 597
cpl_plugin_set_description
Plugin Interface, 598
cpl_plugin_set_email
Plugin Interface, 598
cpl_plugin_set_exec
Plugin Interface, 599
cpl_plugin_set_init
Plugin Interface, 600
cpl_plugin_set_name
Plugin Interface, 600
cpl_plugin_set_synopsis
Plugin Interface, 601
cpl_plugin_set_type
Plugin Interface, 602
cpl_plugin_set_version
Plugin Interface, 602
cpl_plugin_type
Plugin Interface, 581
CPL_PLUGIN_TYPE_NONE
Plugin Interface, 582
CPL_PLUGIN_TYPE_RECIPE
Plugin Interface, 582
CPL_PLUGIN_TYPE_RECIPE_V2
Plugin Interface, 582
cpl_pluginlist
Plugin List, 604
cpl_pluginlist_append
Plugin List, 605
cpl_pluginlist_delete
Plugin List, 605
cpl_pluginlist_dump
Plugin List, 606
cpl_pluginlist_find
Plugin List, 606
cpl_pluginlist_get_first
Plugin List, 607
cpl_pluginlist_get_last
Plugin List, 607
cpl_pluginlist_get_next
Plugin List, 608
cpl_pluginlist_get_size
Plugin List, 609
cpl_pluginlist_new
Plugin List, 609
cpl_pluginlist_prepend
Plugin List, 609
cpl_polynomial_add
Polynomials, 617
cpl_polynomial_compare
Polynomials, 618
cpl_polynomial_copy
Polynomials, 619
cpl_polynomial_delete
Polynomials, 619
cpl_polynomial_derivative
Polynomials, 620
cpl_polynomial_dump
Polynomials, 621
cpl_polynomial_duplicate
Polynomials, 621
cpl_polynomial_eval
Polynomials, 622
cpl_polynomial_eval_1d
Polynomials, 623
cpl_polynomial_eval_1d_diff
Polynomials, 624
cpl_polynomial_eval_2d
Polynomials, 624
cpl_polynomial_eval_3d
Polynomials, 625
cpl_polynomial_extract
Polynomials, 626
cpl_polynomial_fit
Polynomials, 627
cpl_polynomial_fit_1d_create
Polynomials, 629
cpl_polynomial_fit_2d_create

Polynomials, 630
cpl_polynomial_get_coeff
Polynomials, 630
cpl_polynomial_get_degree
Polynomials, 631
cpl_polynomial_get_dimension
Polynomials, 632
cpl_polynomial_multiply
Polynomials, 632
cpl_polynomial_multiply_scalar
Polynomials, 633
cpl_polynomial_set_coeff
Polynomials, 634
cpl_polynomial_shift_1d
Polynomials, 635
cpl_polynomial_solve_1d
Polynomials, 635
cpl_polynomial_subtract
Polynomials, 636
cpl_ppm_match_points
Point pattern matching module, 611
cpl_ppm_match_positions
Point pattern matching module, 614
cpl_property
Properties, 641
cpl_property_delete
Properties, 641
cpl_property_dump
Properties, 642
cpl_property_duplicate
Properties, 642
cpl_property_get_bool
Properties, 643
cpl_property_get_char
Properties, 644
cpl_property_get_comment
Properties, 644
cpl_property_get_double
Properties, 645
cpl_property_get_double_complex
Properties, 645
cpl_property_get_float
Properties, 646
cpl_property_get_float_complex
Properties, 647
cpl_property_get_int
Properties, 647
cpl_property_get_long
Properties, 648
cpl_property_get_long_long
Properties, 649
cpl_property_get_name
Properties, 649
cpl_property_get_size
Properties, 650
cpl_property_get_string
Properties, 651
cpl_property_get_type
Properties, 651
cpl_property_new
Properties, 652
cpl_property_new_array
Properties, 653
cpl_property_set_bool
Properties, 654
cpl_property_set_char
Properties, 654
cpl_property_set_comment
Properties, 655
cpl_property_set_double
Properties, 656
cpl_property_set_double_complex
Properties, 656
cpl_property_set_float
Properties, 657
cpl_property_set_float_complex
Properties, 658
cpl_property_set_int
Properties, 659
cpl_property_set_long
Properties, 659
cpl_property_set_long_long
Properties, 660
cpl_property_set_name
Properties, 661
cpl_property_set_string
Properties, 661
cpl_propertylist
Property Lists, 668
cpl_propertylist_append
Property Lists, 668
cpl_propertylist_append_bool
Property Lists, 669
cpl_propertylist_append_char
Property Lists, 669
cpl_propertylist_append_double
Property Lists, 670
cpl_propertylist_append_double_complex
Property Lists, 671
cpl_propertylist_append_float
Property Lists, 671
cpl_propertylist_append_float_complex
Property Lists, 672
cpl_propertylist_append_int
Property Lists, 673
cpl_propertylist_append_long
Property Lists, 673
cpl_propertylist_append_long_long
Property Lists, 674
cpl_propertylist_append_property
Property Lists, 675
cpl_propertylist_append_string
Property Lists, 675
cpl_propertylist_compare_func
Property Lists, 668
cpl_propertylist_copy_property

Property Lists, 676
cpl_propertylist_copy_property_regexp
Property Lists, 677
cpl_propertylist_delete
Property Lists, 678
cpl_propertylist_dump
Property Lists, 678
cpl_propertylist_duplicate
Property Lists, 679
cpl_propertylist_empty
Property Lists, 679
cpl_propertylist_erase
Property Lists, 680
cpl_propertylist_erase_regexp
Property Lists, 680
cpl_propertylist_get
Property Lists, 681
cpl_propertylist_get_bool
Property Lists, 682
cpl_propertylist_get_char
Property Lists, 682
cpl_propertylist_get_comment
Property Lists, 683
cpl_propertylist_get_const
Property Lists, 684
cpl_propertylist_get_double
Property Lists, 684
cpl_propertylist_get_double_complex
Property Lists, 685
cpl_propertylist_get_float
Property Lists, 686
cpl_propertylist_get_float_complex
Property Lists, 687
cpl_propertylist_get_int
Property Lists, 687
cpl_propertylist_get_long
Property Lists, 688
cpl_propertylist_get_long_long
Property Lists, 689
cpl_propertylist_get_property
Property Lists, 689
cpl_propertylist_get_property_const
Property Lists, 690
cpl_propertylist_get_size
Property Lists, 691
cpl_propertylist_get_string
Property Lists, 691
cpl_propertylist_get_type
Property Lists, 692
cpl_propertylist_has
Property Lists, 693
cpl_propertylist_insert_after_bool
Property Lists, 693
cpl_propertylist_insert_after_char
Property Lists, 694
cpl_propertylist_insert_after_double
Property Lists, 695
cpl_propertylist_insert_after_double_complex
Property Lists, 695
cpl_propertylist_insert_after_float
Property Lists, 697
cpl_propertylist_insert_after_float_complex
Property Lists, 698
cpl_propertylist_insert_after_int
Property Lists, 698
cpl_propertylist_insert_after_long
Property Lists, 700
cpl_propertylist_insert_after_long_long
Property Lists, 701
cpl_propertylist_insert_after_property
Property Lists, 701
cpl_propertylist_insert_after_string
Property Lists, 702
cpl_propertylist_insert_bool
Property Lists, 703
cpl_propertylist_insert_char
Property Lists, 703
cpl_propertylist_insert_double
Property Lists, 705
cpl_propertylist_insert_double_complex
Property Lists, 706
cpl_propertylist_insert_float
Property Lists, 706
cpl_propertylist_insert_float_complex
Property Lists, 708
cpl_propertylist_insert_int
Property Lists, 709
cpl_propertylist_insert_long
Property Lists, 709
cpl_propertylist_insert_long_long
Property Lists, 711
cpl_propertylist_insert_property
Property Lists, 712
cpl_propertylist_insert_string
Property Lists, 712
cpl_propertylist_is_empty
Property Lists, 713
cpl_propertylist_load
Property Lists, 714
cpl_propertylist_load_regexp
Property Lists, 714
cpl_propertylist_new
Property Lists, 715
cpl_propertylist_prepend_bool
Property Lists, 716
cpl_propertylist_prepend_char
Property Lists, 717
cpl_propertylist_prepend_double
Property Lists, 717
cpl_propertylist_prepend_double_complex
Property Lists, 718
cpl_propertylist_prepend_float
Property Lists, 718
cpl_propertylist_prepend_float_complex
Property Lists, 719
cpl_propertylist_prepend_int

Property Lists, 720
cpl_propertylist_prepend_long
Property Lists, 720
cpl_propertylist_prepend_long_long
Property Lists, 721
cpl_propertylist_prepend_property
Property Lists, 722
cpl_propertylist_prepend_string
Property Lists, 722
cpl_propertylist_save
Property Lists, 723
cpl_propertylist_set_bool
Property Lists, 724
cpl_propertylist_set_char
Property Lists, 725
cpl_propertylist_set_comment
Property Lists, 725
cpl_propertylist_set_double
Property Lists, 726
cpl_propertylist_set_double_complex
Property Lists, 727
cpl_propertylist_set_float
Property Lists, 727
cpl_propertylist_set_float_complex
Property Lists, 728
cpl_propertylist_set_int
Property Lists, 729
cpl_propertylist_set_long
Property Lists, 729
cpl_propertylist_set_long_long
Property Lists, 730
cpl_propertylist_set_string
Property Lists, 731
cpl_propertylist_sort
Property Lists, 731
cpl_propertylist_update_bool
Property Lists, 732
cpl_propertylist_update_char
Property Lists, 733
cpl_propertylist_update_double
Property Lists, 733
cpl_propertylist_update_double_complex
Property Lists, 734
cpl_propertylist_update_float
Property Lists, 735
cpl_propertylist_update_float_complex
Property Lists, 736
cpl_propertylist_update_int
Property Lists, 736
cpl_propertylist_update_long
Property Lists, 737
cpl_propertylist_update_long_long
Property Lists, 738
cpl_propertylist_update_string
Property Lists, 738
cpl_realloc
Memory Management Utilities, 483
cpl_recipe
Recipes, 756
CPL_RECIPE_DEFINE
Recipe Definition, 754
cpl_recipe_define
Recipe Definition, 753
cpl_recipeconfig_clear
Recipe Configurations, 741
cpl_recipeconfig_delete
Recipe Configurations, 741
cpl_recipeconfig_get_inputs
Recipe Configurations, 741
cpl_recipeconfig_get_max_count
Recipe Configurations, 742
cpl_recipeconfig_get_min_count
Recipe Configurations, 743
cpl_recipeconfig_get_outputs
Recipe Configurations, 744
cpl_recipeconfig_get_tags
Recipe Configurations, 744
cpl_recipeconfig_is_required
Recipe Configurations, 746
cpl_recipeconfig_new
Recipe Configurations, 747
cpl_recipeconfig_set_input
Recipe Configurations, 747
cpl_recipeconfig_set_inputs
Recipe Configurations, 748
cpl_recipeconfig_set_output
Recipe Configurations, 749
cpl_recipeconfig_set_outputs
Recipe Configurations, 750
cpl_recipeconfig_set_tag
Recipe Configurations, 751
cpl_recipeconfig_set_tags
Recipe Configurations, 752
cpl_regex
Regular Expression Filter, 757
cpl_regex_apply
Regular Expression Filter, 758
CPL_REGEX_BASIC
Regular Expression Filter, 758
cpl_regex_delete
Regular Expression Filter, 759
CPL_REGEX_EXTENDED
Regular Expression Filter, 758
CPL_REGEX_ICASE
Regular Expression Filter, 758
cpl_regex_is_negated
Regular Expression Filter, 759
cpl_regex_negate
Regular Expression Filter, 759
cpl_regex_new
Regular Expression Filter, 760
CPL_REGEX_NOSUBS
Regular Expression Filter, 758
cpl_regex_syntax_option
Regular Expression Filter, 757
cpl_size

Type codes, 916
CPL_SIZE_FORMAT
Type codes, 915
CPL_SIZE_MAX
Type codes, 915
CPL_SIZE_MIN
Type codes, 915
cpl_sprintf
Memory Management Utilities, 484
cpl_stats
Statistics, 762
CPL_STATS_ABSFLUX
Statistics, 763
CPL_STATS_ALL
Statistics, 763
CPL_STATS_CENTROID
Statistics, 763
cpl_stats_delete
Statistics, 763
cpl_stats_dump
Statistics, 764
CPL_STATS_FLUX
Statistics, 763
cpl_stats_get_absflux
Statistics, 764
cpl_stats_get_centroid_x
Statistics, 765
cpl_stats_get_centroid_y
Statistics, 765
cpl_stats_get_flux
Statistics, 766
cpl_stats_get_mad
Statistics, 766
cpl_stats_get_max
Statistics, 767
cpl_stats_get_max_x
Statistics, 767
cpl_stats_get_max_y
Statistics, 768
cpl_stats_get_mean
Statistics, 768
cpl_stats_get_median
Statistics, 769
cpl_stats_get_median_dev
Statistics, 769
cpl_stats_get_min
Statistics, 770
cpl_stats_get_min_x
Statistics, 770
cpl_stats_get_min_y
Statistics, 771
cpl_stats_get_npix
Statistics, 771
cpl_stats_get_sqflux
Statistics, 772
cpl_stats_get_stdev
Statistics, 772
CPL_STATS_MAD
Statistics, 763
CPL_STATS_MAX
Statistics, 763
CPL_STATS_MAXPOS
Statistics, 763
CPL_STATS_MEAN
Statistics, 763
CPL_STATS_MEDIAN
Statistics, 763
CPL_STATS_MEDIAN_DEV
Statistics, 763
CPL_STATS_MIN
Statistics, 763
CPL_STATS_MINPOS
Statistics, 763
cpl_stats_mode
Statistics, 762
cpl_stats_new_from_image
Statistics, 773
cpl_stats_new_from_image_window
Statistics, 773
CPL_STATS_SQFLUX
Statistics, 763
CPL_STATS_STDEV
Statistics, 763
cpl_strdup
Memory Management Utilities, 485
cpl_table_abs_column
Tables, 785
cpl_table_add_columns
Tables, 785
cpl_table_add_scalar
Tables, 786
cpl_table_add_scalar_complex
Tables, 787
cpl_table_and_selected
Tables, 787
cpl_table_and_selected_double
Tables, 788
cpl_table_and_selected_double_complex
Tables, 789
cpl_table_and_selected_float
Tables, 789
cpl_table_and_selected_float_complex
Tables, 790
cpl_table_and_selected_int
Tables, 791
cpl_table_and_selected_invalid
Tables, 791
cpl_table_and_selected_long
Tables, 792
cpl_table_and_selected_long_long
Tables, 793
cpl_table_and_selected_string
Tables, 793
cpl_table_and_selected_window
Tables, 794
cpl_table_arg_column

Tables, 795
cpl_table_cast_column
Tables, 796
cpl_table_compare_structure
Tables, 797
cpl_table_conjugate_column
Tables, 798
cpl_table_copy_data_double
Tables, 798
cpl_table_copy_data_double_complex
Tables, 799
cpl_table_copy_data_float
Tables, 799
cpl_table_copy_data_float_complex
Tables, 800
cpl_table_copy_data_int
Tables, 801
cpl_table_copy_data_long
Tables, 801
cpl_table_copy_data_long_long
Tables, 802
cpl_table_copy_data_string
Tables, 802
cpl_table_copy_structure
Tables, 804
cpl_table_count_invalid
Tables, 805
cpl_table_count_selected
Tables, 805
cpl_table_delete
Tables, 806
cpl_table_divide_columns
Tables, 806
cpl_table_divide_scalar
Tables, 807
cpl_table_divide_scalar_complex
Tables, 808
cpl_table_dump
Tables, 808
cpl_table_dump_structure
Tables, 809
cpl_table_duplicate
Tables, 810
cpl_table_duplicate_column
Tables, 810
cpl_table_erase_column
Tables, 811
cpl_table_erase_invalid
Tables, 811
cpl_table_erase_invalid_rows
Tables, 812
cpl_table_erase_selected
Tables, 813
cpl_table_erase_window
Tables, 814
cpl_table_exponential_column
Tables, 814
cpl_table_extract
Tables, 815
cpl_table_extract_selected
Tables, 816
cpl_table_fill_column_window
Tables, 816
cpl_table_fill_column_window_array
Tables, 817
cpl_table_fill_column_window_complex
Tables, 818
cpl_table_fill_column_window_double
Tables, 819
cpl_table_fill_column_window_double_complex
Tables, 820
cpl_table_fill_column_window_float
Tables, 820
cpl_table_fill_column_window_float_complex
Tables, 821
cpl_table_fill_column_window_int
Tables, 822
cpl_table_fill_column_window_long
Tables, 823
cpl_table_fill_column_window_long_long
Tables, 824
cpl_table_fill_column_window_string
Tables, 824
cpl_table_fill_invalid_double
Tables, 825
cpl_table_fill_invalid_double_complex
Tables, 826
cpl_table_fill_invalid_float
Tables, 827
cpl_table_fill_invalid_float_complex
Tables, 828
cpl_table_fill_invalid_int
Tables, 828
cpl_table_fill_invalid_long
Tables, 829
cpl_table_fill_invalid_long_long
Tables, 830
cpl_table_get
Tables, 831
cpl_table_get_array
Tables, 832
cpl_table_get_column_depth
Tables, 832
cpl_table_get_column_dimension
Tables, 833
cpl_table_get_column_dimensions
Tables, 834
cpl_table_get_column_format
Tables, 834
cpl_table_get_column_max
Tables, 835
cpl_table_get_column_maxpos
Tables, 835
cpl_table_get_column_mean
Tables, 836
cpl_table_get_column_mean_complex

Tables, 836
cpl_table_get_column_median
Tables, 837
cpl_table_get_column_min
Tables, 837
cpl_table_get_column_minpos
Tables, 838
cpl_table_get_column_name
Tables, 838
cpl_table_get_column_names
Tables, 839
cpl_table_get_column_stdev
Tables, 839
cpl_table_get_column_type
Tables, 840
cpl_table_get_column_unit
Tables, 841
cpl_table_get_complex
Tables, 841
cpl_table_get_data_array
Tables, 842
cpl_table_get_data_array_const
Tables, 843
cpl_table_get_data_double
Tables, 843
cpl_table_get_data_double_complex
Tables, 844
cpl_table_get_data_double_complex_const
Tables, 845
cpl_table_get_data_double_const
Tables, 846
cpl_table_get_data_float
Tables, 846
cpl_table_get_data_float_complex
Tables, 847
cpl_table_get_data_float_complex_const
Tables, 848
cpl_table_get_data_float_const
Tables, 849
cpl_table_get_data_int
Tables, 849
cpl_table_get_data_int_const
Tables, 850
cpl_table_get_data_long
Tables, 851
cpl_table_get_data_long_const
Tables, 851
cpl_table_get_data_long_long
Tables, 852
cpl_table_get_data_long_long_const
Tables, 853
cpl_table_get_data_string
Tables, 853
cpl_table_get_data_string_const
Tables, 854
cpl_table_get_double
Tables, 855
cpl_table_get_double_complex
Tables, 855
cpl_table_get_float
Tables, 856
cpl_table_get_float_complex
Tables, 857
cpl_table_get_int
Tables, 857
cpl_table_get_long
Tables, 858
cpl_table_get_long_long
Tables, 859
cpl_table_get_ncol
Tables, 860
cpl_table_get_nrow
Tables, 860
cpl_table_get_string
Tables, 861
cpl_table_has_column
Tables, 862
cpl_table_has_invalid
Tables, 862
cpl_table_has_valid
Tables, 863
cpl_table_imag_column
Tables, 864
cpl_table_insert
Tables, 864
cpl_table_insert_window
Tables, 865
cpl_table_is_selected
Tables, 866
cpl_table_is_valid
Tables, 866
cpl_table_load
Tables, 867
cpl_table_load_window
Tables, 868
cpl_table_logarithm_column
Tables, 869
cpl_table_move_column
Tables, 870
cpl_table_multiply_columns
Tables, 870
cpl_table_multiply_scalar
Tables, 871
cpl_table_multiply_scalar_complex
Tables, 872
cpl_table_name_column
Tables, 872
cpl_table_new
Tables, 873
cpl_table_new_column
Tables, 873
cpl_table_new_column_array
Tables, 874
cpl_table_not_selected
Tables, 875
cpl_table_or_selected

Tables, 876
cpl_table_or_selected_double
Tables, 876
cpl_table_or_selected_double_complex
Tables, 877
cpl_table_or_selected_float
Tables, 878
cpl_table_or_selected_float_complex
Tables, 878
cpl_table_or_selected_int
Tables, 879
cpl_table_or_selected_invalid
Tables, 880
cpl_table_or_selected_long
Tables, 880
cpl_table_or_selected_long_long
Tables, 881
cpl_table_or_selected_string
Tables, 882
cpl_table_or_selected_window
Tables, 883
cpl_table_power_column
Tables, 883
cpl_table_real_column
Tables, 884
cpl_table_save
Tables, 885
cpl_table_select_all
Tables, 886
cpl_table_select_row
Tables, 887
cpl_table_set
Tables, 888
cpl_table_set_array
Tables, 888
cpl_table_set_column_depth
Tables, 889
cpl_table_set_column_dimensions
Tables, 890
cpl_table_set_column_format
Tables, 890
cpl_table_set_column_invalid
Tables, 891
cpl_table_set_column_unit
Tables, 892
cpl_table_set_complex
Tables, 892
cpl_table_set_double
Tables, 893
cpl_table_set_double_complex
Tables, 894
cpl_table_set_float
Tables, 895
cpl_table_set_float_complex
Tables, 896
cpl_table_set_int
Tables, 896
cpl_table_set_invalid
Tables, 897
cpl_table_set_long
Tables, 898
cpl_table_set_long_long
Tables, 899
cpl_table_set_size
Tables, 899
cpl_table_set_string
Tables, 900
cpl_table_shift_column
Tables, 901
cpl_table_sort
Tables, 902
cpl_table_subtract_columns
Tables, 902
cpl_table_subtract_scalar
Tables, 903
cpl_table_subtract_scalar_complex
Tables, 905
cpl_table_unselect_all
Tables, 905
cpl_table_unselect_row
Tables, 906
cpl_table_unwrap
Tables, 906
cpl_table_where_selected
Tables, 907
cpl_table_wrap_double
Tables, 908
cpl_table_wrap_double_complex
Tables, 908
cpl_table_wrap_float
Tables, 909
cpl_table_wrap_float_complex
Tables, 910
cpl_table_wrap_int
Tables, 910
cpl_table_wrap_long
Tables, 911
cpl_table_wrap_long_long
Tables, 912
cpl_table_wrap_string
Tables, 912
cpl_test
Unit testing functions, 920
cpl_test_abs
Unit testing functions, 921
cpl_test_abs_complex
Unit testing functions, 921
cpl_test_array_abs
Unit testing functions, 922
cpl_test_assert
Unit testing functions, 923
cpl_test_end
Unit testing functions, 938
cpl_test_eq
Unit testing functions, 923
cpl_test_eq_error

Unit testing functions, 924
cpl_test_eq_mask
Unit testing functions, 924
cpl_test_eq_ptr
Unit testing functions, 925
cpl_test_eq_string
Unit testing functions, 925
cpl_test_error
Unit testing functions, 927
cpl_test_errorstate
Unit testing functions, 927
cpl_test_fits
Unit testing functions, 928
cpl_test_get_bytes_image
Unit testing functions, 939
cpl_test_get_bytes_imagelist
Unit testing functions, 939
cpl_test_get_bytes_matrix
Unit testing functions, 940
cpl_test_get_bytes_vector
Unit testing functions, 940
cpl_test_get_failed
Unit testing functions, 941
cpl_test_get_tested
Unit testing functions, 941
cpl_test_get_walltime
Unit testing functions, 942
cpl_test_image_abs
Unit testing functions, 928
cpl_test_image_rel
Unit testing functions, 929
cpl_test_imagelist_abs
Unit testing functions, 930
cpl_test_init
Unit testing functions, 930
cpl_test_leq
Unit testing functions, 931
cpl_test_lt
Unit testing functions, 931
cpl_test_matrix_abs
Unit testing functions, 932
cpl_test_memory_is_empty
Unit testing functions, 933
cpl_test_noneq
Unit testing functions, 933
cpl_test_noneq_ptr
Unit testing functions, 933
cpl_test_noneq_string
Unit testing functions, 934
cpl_test_nonnull
Unit testing functions, 934
cpl_test_null
Unit testing functions, 935
cpl_test_polynomial_abs
Unit testing functions, 935
cpl_test_rel
Unit testing functions, 936
cpl_test_vector_abs
Unit testing functions, 937
cpl_test_zero
Unit testing functions, 937
cpl_type
Type codes, 916
CPL_TYPE_BITMASK
Type codes, 917
CPL_TYPE_BOOL
Type codes, 916
cpl_type_bpp
I/O, 251
CPL_TYPE_CHAR
Type codes, 916
CPL_TYPE_COMPLEX
Type codes, 917
CPL_TYPE_DOUBLE
Type codes, 917
CPL_TYPE_DOUBLE_COMPLEX
Type codes, 917
CPL_TYPE_FLAG_ARRAY
Type codes, 916
CPL_TYPE_FLOAT
Type codes, 917
CPL_TYPE_FLOAT_COMPLEX
Type codes, 917
cpl_type_get_name
Type codes, 917
cpl_type_get_sizeof
Type codes, 918
CPL_TYPE_INT
Type codes, 916
CPL_TYPE_INVALID
Type codes, 916
CPL_TYPE_LONG
Type codes, 917
CPL_TYPE_LONG_LONG
Type codes, 917
CPL_TYPE_POINTER
Type codes, 917
CPL_TYPE_SHORT
Type codes, 916
CPL_TYPE_SIZE
Type codes, 917
CPL_TYPE_STRING
Type codes, 917
CPL_TYPE_UCHAR
Type codes, 916
CPL_TYPE_UINT
Type codes, 917
CPL_TYPE_ULONG
Type codes, 917
CPL_TYPE_UNSPECIFIED
Type codes, 917
CPL_TYPE_USHORT
Type codes, 916
cpl_value
Images, 290
CPL_VALUE_INF

Images, 290
CPL_VALUE_MINUSINF
Images, 290
CPL_VALUE_NAN
Images, 290
CPL_VALUE_NOTFINITE
Images, 290
CPL_VALUE_PLUSINF
Images, 290
CPL_VALUE_ZERO
Images, 290
cpl_vector_add
Vector, 945
cpl_vector_add_scalar
Vector, 945
cpl_vector_convolve_symmetric
Vector, 946
cpl_vector_copy
Vector, 946
cpl_vector_correlate
Vector, 947
cpl_vector_cycle
Vector, 948
cpl_vector_delete
Vector, 949
cpl_vector_divide
Vector, 950
cpl_vector_divide_scalar
Vector, 950
cpl_vector_dump
Vector, 951
cpl_vector_duplicate
Vector, 951
cpl_vector_exponential
Vector, 952
cpl_vector_extract
Vector, 953
cpl_vector_fill
Vector, 953
cpl_vector_fill_kernel_profile
Vector, 954
cpl_vector_fill_polynomial
Polynomials, 637
cpl_vector_fill_polynomial_fit_residual
Polynomials, 638
cpl_vector_filter_lowpass_create
Vector, 955
cpl_vector_filter_median_create
Vector, 956
cpl_vector_find
Vector, 956
cpl_vector_fit_gaussian
Vector, 957
cpl_vector_get
Vector, 959
cpl_vector_get_data
Vector, 960
cpl_vector_get_data_const
Vector, 960
cpl_vector_get_max
Vector, 961
cpl_vector_get_maxpos
Vector, 961
cpl_vector_get_mean
Vector, 962
cpl_vector_get_median
Vector, 962
cpl_vector_get_median_const
Vector, 963
cpl_vector_get_min
Vector, 963
cpl_vector_get_minpos
Vector, 964
cpl_vector_get_size
Vector, 964
cpl_vector_get_stdev
Vector, 965
cpl_vector_get_sum
Vector, 966
cpl_vector_load
Vector, 966
cpl_vector_logarithm
Vector, 967
cpl_vector_multiply
Vector, 967
cpl_vector_multiply_scalar
Vector, 968
cpl_vector_new
Vector, 968
cpl_vector_new_from_image_column
Images, 394
cpl_vector_new_from_image_row
Images, 394
cpl_vector_new_lss_kernel
Vector, 969
cpl_vector_power
Vector, 970
cpl_vector_product
Vector, 970
cpl_vector_read
Vector, 971
cpl_vector_save
Vector, 971
cpl_vector_set
Vector, 972
cpl_vector_set_size
Vector, 973
cpl_vector_sort
Vector, 974
cpl_vector_sqrt
Vector, 974
cpl_vector_subtract
Vector, 975
cpl_vector_subtract_scalar
Vector, 975
cpl_vector_unwrap

Vector, 976
cpl_vector_wrap
Vector, 976
cpl_version_get_binary_age
Library Version Information, 398
cpl_version_get_binary_version
Library Version Information, 398
cpl_version_get_interface_age
Library Version Information, 399
cpl_version_get_major
Library Version Information, 399
cpl_version_get_micro
Library Version Information, 399
cpl_version_get_minor
Library Version Information, 399
cpl_version_get_version
Library Version Information, 400
cpl_vsprintf
Memory Management Utilities, 486
cpl_wcs_convert
World Coordinate System, 988
cpl_wcs_delete
World Coordinate System, 989
cpl_wcs_get_cd
World Coordinate System, 989
cpl_wcs_get_crpix
World Coordinate System, 991
cpl_wcs_get_crval
World Coordinate System, 991
cpl_wcs_get_ctype
World Coordinate System, 992
cpl_wcs_get_cunit
World Coordinate System, 993
cpl_wcs_get_image_dims
World Coordinate System, 993
cpl_wcs_get_image_naxis
World Coordinate System, 994
cpl_wcs_new_from_propertylist
World Coordinate System, 994
cpl_wcs_platesol
World Coordinate System, 996
CPL_WCS_REGEXP
World Coordinate System, 987
cpl_wlcalib_fill_line_spectrum
Wavelength calibration, 978
cpl_wlcalib_fill_line_spectrum_fast
Wavelength calibration, 979
cpl_wlcalib_fill_logline_spectrum
Wavelength calibration, 980
cpl_wlcalib_fill_logline_spectrum_fast
Wavelength calibration, 981
cpl_wlcalib_find_best_1d
Wavelength calibration, 981
cpl_wlcalib_slitmodel_delete
Wavelength calibration, 983
cpl_wlcalib_slitmodel_new
Wavelength calibration, 983
cpl_wlcalib_slitmodel_set_catalog
Wavelength calibration, 984
cpl_wlcalib_slitmodel_set_threshold
Wavelength calibration, 984
cpl_wlcalib_slitmodel_set_wfwhm
Wavelength calibration, 985
cpl_wlcalib_slitmodel_set_wslit
Wavelength calibration, 986

deinitialize
_cpl_plugin_, 1001
description
_cpl_plugin_, 1001
DFS related functions, 109
CPL_DFS_PRO_CATG, 110
CPL_DFS_PRO_SCIENCE, 110
CPL_DFS_PRO_TECH, 110
CPL_DFS_PRO_TYPE, 110
cpl_dfs_save_imagelist, 111
cpl_dfs_save_paf, 112
cpl_dfs_save_propertylist, 113
cpl_dfs_save_table, 114
cpl_dfs_setup_product_header, 115
cpl_dfs_sign_products, 116
CPL_DFS_SIGNATURE_CHECKSUM, 111
CPL_DFS_SIGNATURE_DATAMD5, 111
CPL_DFS_SIGNATURE_NONE, 111
cpl_dfs_update_product_header, 117
DICB specific property functionality, 119

email
_cpl_plugin_, 1002
Error handling, 120
_cpl_error_code_, 126
cpl_ensure, 121
cpl_ensure_code, 122
CPL_ERROR_ACCESS_OUT_OF_RANGE, 127
CPL_ERROR_ASSIGNING_STREAM, 127
CPL_ERROR_BAD_FILE_FORMAT, 127
cpl_error_code, 126
CPL_ERROR_CONTINUE, 127
CPL_ERROR_DATA_NOT_FOUND, 127
CPL_ERROR_DIVISION_BY_ZERO, 127
CPL_ERROR_DUPLICATING_STREAM, 127
cpl_error_ensure, 122
CPL_ERROR_EOL, 127
CPL_ERROR_FILE_ALREADY_OPEN, 127
CPL_ERROR_FILE_IO, 127
CPL_ERROR_FILE_NOT_CREATED, 127
CPL_ERROR_FILE_NOT_FOUND, 127
cpl_error_get_code, 127
cpl_error_get_file, 128
cpl_error_get_function, 128
cpl_error_get_line, 128
cpl_error_get_message, 129
cpl_error_get_message_default, 129
cpl_error_get_where, 130
CPL_ERROR_HISTORY_LOST, 127
CPL_ERROR_ILLEGAL_INPUT, 127
CPL_ERROR_ILLEGAL_OUTPUT, 127

CPL_ERROR_INCOMPATIBLE_INPUT, 127
CPL_ERROR_INVALID_TYPE, 127
CPL_ERROR_MAX_MESSAGE_LENGTH, 123
CPL_ERROR_NO_WCS, 127
CPL_ERROR_NONE, 127
CPL_ERROR_NULL_INPUT, 127
cpl_error_set, 124
cpl_error_set_message, 124
cpl_error_set_where, 125
CPL_ERROR_SINGULAR_MATRIX, 127
CPL_ERROR_TYPE_MISMATCH, 127
CPL_ERROR_UNSPECIFIED, 127
CPL_ERROR_UNSUPPORTED_MODE, 127
CPL_HAVE_VA_ARGS, 126
execute
_cpl_plugin_, 1002

FFTW wrappers, 131
_cpl_fft_mode_, 131
CPL_FFT_BACKWARD, 132
CPL_FFT_FIND_EXHAUSTIVE, 132
CPL_FFT_FIND_MEASURE, 132
CPL_FFT_FIND_PATIENT, 132
CPL_FFT_FORWARD, 132
cpl_fft_image, 132
cpl_fft_imagelist, 133
cpl_fft_mode, 131
CPL_FFT_NOSCALE, 132
Filters, 141
_cpl_border_mode_, 142
_cpl_filter_mode_, 142
CPL_BORDER_COPY, 142
CPL_BORDER_CROP, 142
CPL_BORDER_FILTER, 142
cpl_border_mode, 141
CPL_BORDER_NOP, 142
CPL_BORDER_ZERO, 142
CPL_FILTER_AVERAGE, 144
CPL_FILTER_AVERAGE_FAST, 144
CPL_FILTER_CLOSING, 143
CPL_FILTER_DILATION, 143
CPL_FILTER_EROSION, 143
CPL_FILTER_LINEAR, 143
CPL_FILTER_LINEAR_SCALE, 144
CPL_FILTER_MEDIAN, 144
cpl_filter_mode, 142
CPL_FILTER_MORPHO, 145
CPL_FILTER_MORPHO_SCALE, 146
CPL_FILTER_OPENING, 143
CPL_FILTER_STDEV, 145
CPL_FILTER_STDEV_FAST, 145
FITS card related basic routines, 134
FITS related basic routines, 135
_cpl_fits_mode_, 136
cpl_fits_count_extensions, 136
cpl_fits_find_extension, 136
cpl_fits_get_extension_nb, 137
cpl_fits_get_mode, 138
cpl_fits_get_nb_extensions, 138
cpl_fits_mode, 135
CPL_FITS_ONE, 136
CPL_FITS_RESTART_CACHING, 136
cpl_fits_set_mode, 138
CPL_FITS_START_CACHING, 136
CPL_FITS_STOP_CACHING, 136
Frame Set Iterators, 147
cpl_frameset_iterator, 147
cpl_frameset_iterator_advance, 148
cpl_frameset_iterator_assign, 148
cpl_frameset_iterator_delete, 149
cpl_frameset_iterator_distance, 149
cpl_frameset_iterator_duplicate, 150
cpl_frameset_iterator_get, 151
cpl_frameset_iterator_get_const, 151
cpl_frameset_iterator_new, 152
cpl_frameset_iterator_reset, 153
Frame Sets, 154
cpl_frameset, 155
cpl_frameset_count_tags, 155
cpl_frameset_delete, 156
cpl_frameset_dump, 156
cpl_frameset_duplicate, 157
cpl_frameset_erase, 157
cpl_frameset_erase_frame, 158
cpl_frameset_extract, 159
cpl_frameset_find, 159
cpl_frameset_find_const, 160
cpl_frameset_get_first, 161
cpl_frameset_get_first_const, 162
cpl_frameset_get_frame, 162
cpl_frameset_get_frame_const, 163
cpl_frameset_get_next, 164
cpl_frameset_get_next_const, 165
cpl_frameset_get_position, 166
cpl_frameset_get_position_const, 166
cpl_frameset_get_size, 167
cpl_frameset_insert, 168
cpl_frameset_is_empty, 168
cpl_frameset_join, 169
cpl_frameset_labelise, 170
cpl_frameset_new, 170
cpl_frameset_sort, 171
Frame Sets IO functions, 173
cpl_imagelist_load_frameset, 173
Frames, 174
_cpl_frame_group_, 177
_cpl_frame_level_, 177
_cpl_frame_type_, 178
cpl_frame, 176
cpl_frame_compare_func, 176
cpl_frame_delete, 178
cpl_frame_dump, 179
cpl_frame_duplicate, 179
cpl_frame_get_filename, 180
cpl_frame_get_group, 181
cpl_frame_get_level, 181
cpl_frame_get_nextensions, 182

cpl_frame_get_tag, 182
cpl_frame_get_type, 183
cpl_frame_group, 176
CPL_FRAME_GROUP_CALIB, 177
CPL_FRAME_GROUP_CALIB_ID, 175
CPL_FRAME_GROUP_NONE, 177
CPL_FRAME_GROUP_PRODUCT, 177
CPL_FRAME_GROUP_PRODUCT_ID, 176
CPL_FRAME_GROUP_RAW, 177
CPL_FRAME_GROUP_RAW_ID, 176
cpl_frame_level, 177
CPL_FRAME_LEVEL_FINAL, 178
CPL_FRAME_LEVEL_INTERMEDIATE, 178
CPL_FRAME_LEVEL_NONE, 178
CPL_FRAME_LEVEL_TEMPORARY, 178
cpl_frame_new, 184
cpl_frame_set_filename, 184
cpl_frame_set_group, 184
cpl_frame_set_level, 185
cpl_frame_set_tag, 186
cpl_frame_set_type, 186
cpl_frame_type, 177
CPL_FRAME_TYPE_ANY, 178
CPL_FRAME_TYPE_IMAGE, 178
CPL_FRAME_TYPE_MATRIX, 178
CPL_FRAME_TYPE_NONE, 178
CPL_FRAME_TYPE_PAF, 178
CPL_FRAME_TYPE_TABLE, 178
frames
_cpl_recipe_, 1004
Fundamental math functionality, 188
CPL_MATH_1_PI, 189
CPL_MATH_2_PI, 189
CPL_MATH_2_SQRTPI, 190
CPL_MATH_2PI, 190
CPL_MATH_4_PI, 190
CPL_MATH_DEG_RAD, 191
CPL_MATH_E, 191
CPL_MATH_FWHM_SIG, 191
CPL_MATH_LN10, 192
CPL_MATH_LN2, 192
CPL_MATH_LOG10E, 192
CPL_MATH_LOG2E, 192
CPL_MATH_PI, 193
CPL_MATH_PI_2, 193
CPL_MATH_PI_4, 193
CPL_MATH_RAD_DEG, 194
CPL_MATH_SIG_FWHM, 194
CPL_MATH_SQRT1_2, 194
CPL_MATH_SQRT2, 195
CPL_MATH_SQRT2PI, 195
CPL_MATH_SQRT3, 195
CPL_MATH_STD_MAD, 196
CPL_MAX, 196
CPL_MIN, 196

Handling of multiple CPL errors, 198
cpl_errorstate_dump, 198
cpl_errorstate_dump_one, 199
cpl_errorstate_dump_one_debug, 200
cpl_errorstate_dump_one_info, 200
cpl_errorstate_dump_one_warning, 201
cpl_errorstate_get, 201
cpl_errorstate_is_equal, 202
cpl_errorstate_set, 203
High level functions for geometric transformations, 204
cpl_geom_combine, 204
CPL_GEOM_FIRST, 204
cpl_geom_img_offset_combine, 205
cpl_geom_img_offset_fine, 206
cpl_geom_img_offset_saa, 207
CPL_GEOM_INTERSECT, 204
CPL_GEOM_UNION, 204
High level functions to handle apertures, 210
cpl_apertures_delete, 211
cpl_apertures_dump, 212
cpl_apertures_extract, 212
cpl_apertures_extract_mask, 213
cpl_apertures_extract_sigma, 214
cpl_apertures_extract_window, 215
cpl_apertures_get_bottom, 215
cpl_apertures_get_bottom_x, 217
cpl_apertures_get_centroid_x, 217
cpl_apertures_get_centroid_y, 218
cpl_apertures_get_flux, 219
cpl_apertures_get_fwhm, 219
cpl_apertures_get_left, 220
cpl_apertures_get_left_y, 220
cpl_apertures_get_max, 221
cpl_apertures_get_max_x, 222
cpl_apertures_get_max_y, 222
cpl_apertures_get_maxpos_x, 223
cpl_apertures_get_maxpos_y, 223
cpl_apertures_get_mean, 224
cpl_apertures_get_median, 224
cpl_apertures_get_min, 225
cpl_apertures_get_minpos_x, 225
cpl_apertures_get_minpos_y, 226
cpl_apertures_get_npix, 227
cpl_apertures_get_pos_x, 227
cpl_apertures_get_pos_y, 228
cpl_apertures_get_right, 228
cpl_apertures_get_right_y, 229
cpl_apertures_get_size, 230
cpl_apertures_get_stdev, 230
cpl_apertures_get_top, 231
cpl_apertures_get_top_x, 231
cpl_apertures_new_from_image, 232
cpl_apertures_sort_by_flux, 233
cpl_apertures_sort_by_max, 233
cpl_apertures_sort_by_npix, 234
High-level functions for non-linear fitting, 235
cpl_fit_image_gaussian, 235
cpl_fit_imagelist_polynomial, 237
cpl_fit_imagelist_polynomial_window, 238
cpl_fit_lvmq, 240
cpl_gaussian_eval_2d, 242

High-level functions that are photometry related, 244
cpl_photom_fill_blackbody, 244
High-level functions to compute detector features, 246
cpl_detector_interpolate_rejected, 246
cpl_flux_get_bias_window, 247
cpl_flux_get_noise_ring, 247
cpl_flux_get_noise_window, 248

I/O, 250
_cpl_io_type_, 252
CPL_BPP_16_SIGNED, 250
CPL_BPP_16_UNSIGNED, 250
CPL_BPP_32_SIGNED, 251
CPL_BPP_8_UNSIGNED, 251
CPL_BPP_IEEE_DOUBLE, 251
CPL_BPP_IEEE_FLOAT, 251
CPL_IO_APPEND, 252
CPL_IO_COMPRESS_GZIP, 252
CPL_IO_COMPRESS_HCOMPRESS, 252
CPL_IO_COMPRESS_PLIO, 252
CPL_IO_COMPRESS_RICE, 252
CPL_IO_CREATE, 252
CPL_IO_DEFAULT, 252
CPL_IO_EXTEND, 252
CPL_IO_MAX, 252
cpl_io_type, 252
cpl_type_bpp, 251
Imagelists, 253
cpl_image_new_from_accepted, 255
cpl_imagelist_add, 255
cpl_imagelist_add_image, 256
cpl_imagelist_add_scalar, 256
cpl_imagelist_cast, 257
cpl_imagelist_collapse_create, 258
cpl_imagelist_collapse_median_create, 259
cpl_imagelist_collapse_minmax_create, 259
cpl_imagelist_collapse_sigclip_create, 260
cpl_imagelist_delete, 262
cpl_imagelist_divide, 262
cpl_imagelist_divide_image, 263
cpl_imagelist_divide_scalar, 263
cpl_imagelist_dump_structure, 264
cpl_imagelist_dump_window, 264
cpl_imagelist_duplicate, 265
cpl_imagelist_empty, 266
cpl_imagelist_erase, 266
cpl_imagelist_exponential, 267
cpl_imagelist_get, 268
cpl_imagelist_get_const, 268
cpl_imagelist_get_size, 269
cpl_imagelist_is_uniform, 269
cpl_imagelist_load, 270
cpl_imagelist_load_window, 271
cpl_imagelist_logarithm, 272
cpl_imagelist_multiply, 272
cpl_imagelist_multiply_image, 273
cpl_imagelist_multiply_scalar, 273
cpl_imagelist_new, 274
cpl_imagelist_normalise, 274
cpl_imagelist_power, 275
cpl_imagelist_save, 275
cpl_imagelist_set, 277
cpl_imagelist_subtract, 278
cpl_imagelist_subtract_image, 278
cpl_imagelist_subtract_scalar, 279
cpl_imagelist_swap_axis_create, 279
cpl_imagelist_threshold, 280
cpl_imagelist_unset, 281
cpl_imagelist_unwrap, 281
Images, 283
_cpl_value_, 290
cpl_image_abs, 291
cpl_image_abs_create, 291
cpl_image_accept, 292
cpl_image_accept_all, 292
cpl_image_add, 293
cpl_image_add_create, 293
cpl_image_add_scalar, 294
cpl_image_add_scalar_create, 294
cpl_image_and, 295
cpl_image_and_scalar, 296
cpl_image_average_create, 296
cpl_image_cast, 297
cpl_image_collapse_create, 298
cpl_image_collapse_median_create, 299
cpl_image_collapse_window_create, 300
cpl_image_conjugate, 300
cpl_image_copy, 301
cpl_image_count_rejected, 302
cpl_image_delete, 302
cpl_image_divide, 303
cpl_image_divide_create, 304
cpl_image_divide_scalar, 305
cpl_image_divide_scalar_create, 305
cpl_image_dump_structure, 306
cpl_image_dump_window, 306
cpl_image_duplicate, 307
cpl_image_exponential, 308
cpl_image_exponential_create, 309
cpl_image_extract, 309
cpl_image_extract_subsample, 310
cpl_image_fft, 311
cpl_image_fill_abs_arg, 312
cpl_image_fill_gaussian, 313
cpl_image_fill_jacobian, 314
cpl_image_fill_jacobian_polynomial, 315
cpl_image_fill_noise_uniform, 316
cpl_image_fill_polynomial, 317
cpl_image_fill_re_im, 318
cpl_image_fill_rejected, 319
cpl_image_fill_test_create, 319
cpl_image_fill_window, 320
cpl_image_filter, 321
cpl_image_filter_linear, 322
cpl_image_filter_mask, 323
cpl_image_filter_median, 324
cpl_image_filter_morpho, 326

cpl_image_filter_stdev, 326
cpl_image_fit_gaussian, 327
cpl_image_flip, 328
cpl_image_get, 329
cpl_image_get_absflux, 330
cpl_image_get_absflux_window, 330
cpl_image_get_bpm, 331
cpl_image_get_bpm_const, 332
cpl_image_get_centroid_x, 332
cpl_image_get_centroid_x_window, 333
cpl_image_get_centroid_y, 333
cpl_image_get_centroid_y_window, 334
cpl_image_get_complex, 334
cpl_image_get_data, 335
cpl_image_get_data_const, 336
cpl_image_get_data_double, 336
cpl_image_get_data_double_complex, 337
cpl_image_get_data_double_complex_const, 337
cpl_image_get_data_double_const, 338
cpl_image_get_data_float, 338
cpl_image_get_data_float_complex, 339
cpl_image_get_data_float_complex_const, 339
cpl_image_get_data_float_const, 340
cpl_image_get_data_int, 340
cpl_image_get_data_int_const, 341
cpl_image_get_flux, 341
cpl_image_get_flux_window, 341
cpl_image_get_fwhm, 342
cpl_image_get_interpolated, 343
cpl_image_get_mad, 344
cpl_image_get_mad_window, 345
cpl_image_get_max, 346
cpl_image_get_max_window, 346
cpl_image_get_maxpos, 347
cpl_image_get_maxpos_window, 347
cpl_image_get_mean, 348
cpl_image_get_mean_window, 349
cpl_image_get_median, 349
cpl_image_get_median_dev, 350
cpl_image_get_median_dev_window, 350
cpl_image_get_median_window, 351
cpl_image_get_min, 352
cpl_image_get_min_window, 353
cpl_image_get_minpos, 353
cpl_image_get_minpos_window, 354
cpl_image_get_size_x, 355
cpl_image_get_size_y, 355
cpl_image_get_sqflux, 356
cpl_image_get_sqflux_window, 356
cpl_image_get_stdev, 357
cpl_image_get_stdev_window, 357
cpl_image_get_type, 358
cpl_image_hypot, 359
cpl_image_iqe, 359
cpl_image_is_rejected, 361
cpl_image_labelise_mask_create, 361
cpl_image_load, 362
cpl_image_load_window, 363
cpl_image_logarithm, 364
cpl_image_logarithm_create, 365
cpl_image_move, 365
cpl_image_multiply, 366
cpl_image_multiply_create, 367
cpl_image_multiply_scalar, 367
cpl_image_multiply_scalar_create, 368
cpl_image_new, 368
cpl_image_new_from_mask, 369
cpl_image_normalise, 370
cpl_image_normalise_create, 370
cpl_image_not, 371
cpl_image_or, 372
cpl_image_or_scalar, 373
cpl_image_power, 373
cpl_image_power_create, 374
cpl_image_rebin, 374
cpl_image_reject, 375
cpl_image_reject_from_mask, 376
cpl_image_reject_value, 376
cpl_image_save, 377
cpl_image_set, 378
cpl_image_set_bpm, 379
cpl_image_set_complex, 380
cpl_image_shift, 380
cpl_image_subtract, 381
cpl_image_subtract_create, 382
cpl_image_subtract_scalar, 382
cpl_image_subtract_scalar_create, 383
cpl_image_threshold, 383
cpl_image_turn, 384
cpl_image_unset_bpm, 385
cpl_image_unwrap, 386
cpl_image_warp, 386
cpl_image_warp_polynomial, 388
cpl_image_wrap_double, 389
cpl_image_wrap_double_complex, 390
cpl_image_wrap_float, 390
cpl_image_wrap_float_complex, 391
cpl_image_wrap_int, 392
cpl_image_xor, 392
cpl_image_xor_scalar, 393
cpl_value, 290
CPL_VALUE_INF, 290
CPL_VALUE_MINUSINF, 290
CPL_VALUE_NAN, 290
CPL_VALUE_NOTFINITE, 290
CPL_VALUE_PLUSINF, 290
CPL_VALUE_ZERO, 290
cpl_vector_new_from_image_column, 394
cpl_vector_new_from_image_row, 394
initialize
_cpl_plugin_, 1002
interface
_cpl_recipe_, 1004

Library Initialization, 396
cpl_end, 396
cpl_get_description, 396

cpl_init, 397
Library Version Information, 398
cpl_version_get_binary_age, 398
cpl_version_get_binary_version, 398
cpl_version_get_interface_age, 399
cpl_version_get_major, 399
cpl_version_get_micro, 399
cpl_version_get_minor, 399
cpl_version_get_version, 400

Masks of pixels, 401
cpl_mask_and, 403
cpl_mask_closing, 403
cpl_mask_collapse_create, 404
cpl_mask_copy, 404
cpl_mask_count, 405
cpl_mask_count_window, 406
cpl_mask_delete, 406
cpl_mask_dilation, 407
cpl_mask_dump_window, 407
cpl_mask_duplicate, 409
cpl_mask_erosion, 409
cpl_mask_extract, 410
cpl_mask_extract_subsample, 411
cpl_mask_filter, 411
cpl_mask_flip, 413
cpl_mask_get, 414
cpl_mask_get_data, 414
cpl_mask_get_data_const, 415
cpl_mask_get_size_x, 415
cpl_mask_get_size_y, 416
cpl_mask_is_empty, 416
cpl_mask_is_empty_window, 417
cpl_mask_load, 418
cpl_mask_load_window, 418
cpl_mask_move, 419
cpl_mask_new, 420
cpl_mask_not, 421
cpl_mask_opening, 421
cpl_mask_or, 422
cpl_mask_save, 422
cpl_mask_set, 423
cpl_mask_shift, 424
cpl_mask_threshold_image, 425
cpl_mask_threshold_image_create, 426
cpl_mask_turn, 426
cpl_mask_unwrap, 427
cpl_mask_wrap, 428
cpl_mask_xor, 428
Matrices, 430
cpl_matrix_add, 433
cpl_matrix_add_scalar, 433
cpl_matrix_append, 434
cpl_matrix_copy, 435
cpl_matrix_decomp_chol, 435
cpl_matrix_delete, 437
cpl_matrix_divide, 437
cpl_matrix_divide_scalar, 438
cpl_matrix_dump, 439
cpl_matrix_duplicate, 439
cpl_matrix_erase_columns, 440
cpl_matrix_erase_rows, 441
cpl_matrix_exponential, 441
cpl_matrix_extract, 442
cpl_matrix_extract_column, 443
cpl_matrix_extract_diagonal, 443
cpl_matrix_extract_row, 444
cpl_matrix_fill, 445
cpl_matrix_fill_column, 445
cpl_matrix_fill_diagonal, 446
cpl_matrix_fill_row, 447
cpl_matrix_fill_window, 447
cpl_matrix_flip_columns, 448
cpl_matrix_flip_rows, 449
cpl_matrix_get, 449
cpl_matrix_get_data, 450
cpl_matrix_get_data_const, 451
cpl_matrix_get_determinant, 451
cpl_matrix_get_max, 452
cpl_matrix_get_maxpos, 452
cpl_matrix_get_mean, 453
cpl_matrix_get_median, 454
cpl_matrix_get_min, 454
cpl_matrix_get_minpos, 455
cpl_matrix_get_ncol, 455
cpl_matrix_get_ncol_, 456
cpl_matrix_get_nrow, 457
cpl_matrix_get_stdev, 457
cpl_matrix_invert_create, 458
cpl_matrix_is_diagonal, 459
cpl_matrix_is_identity, 459
cpl_matrix_is_zero, 460
cpl_matrix_logarithm, 461
cpl_matrix_multiply, 461
cpl_matrix_multiply_scalar, 462
cpl_matrix_new, 462
cpl_matrix_power, 463
cpl_matrix_product_create, 464
cpl_matrix_resize, 464
cpl_matrix_set_, 465
cpl_matrix_set_size, 467
cpl_matrix_shift, 468
cpl_matrix_solve, 468
cpl_matrix_solve_chol, 469
cpl_matrix_solve_normal, 470
cpl_matrix_solve_svd, 471
cpl_matrix_solve_svd_threshold, 472
cpl_matrix_sort_columns, 473
cpl_matrix_sort_rows, 473
cpl_matrix_subtract, 474
cpl_matrix_subtract_scalar, 475
cpl_matrix_swap_columns, 475
cpl_matrix_swap_rowcolumn, 476
cpl_matrix_swap_rows, 477
cpl_matrix_threshold_small, 477
cpl_matrix_transpose_create, 478
cpl_matrix_unwrap, 479

cpl_matrix_wrap, 479
max_count
_cpl_framedata_, 999
Memory Management Utilities, 481
cpl_calloc, 481
cpl_free, 482
cpl_malloc, 482
cpl_memory_dump, 483
cpl_memory_is_empty, 483
cpl_realloc, 483
cpl_sprintf, 484
cpl_strdup, 485
cpl_vsprintf, 486
Messages, 487
cpl_msg_debug, 489
cpl_msg_error, 489
cpl_msg_get_domain, 490
cpl_msg_get_level, 490
cpl_msg_get_log_level, 490
cpl_msg_get_log_name, 490
cpl_msg_indent, 491
cpl_msg_indent_less, 491
cpl_msg_indent_more, 491
cpl_msg_info, 492
cpl_msg_info_overwritable, 492
cpl_msg_init, 493
cpl_msg_progress, 494
cpl_msg_set_component_off, 494
cpl_msg_set_component_on, 495
cpl_msg_set_domain, 495
cpl_msg_set_domain_off, 496
cpl_msg_set_domain_on, 496
cpl_msg_set_indentation, 497
cpl_msg_set_level, 497
cpl_msg_set_level_from_env, 498
cpl_msg_set_log_level, 498
cpl_msg_set_log_name, 499
cpl_msg_set_threadid_off, 500
cpl_msg_set_threadid_on, 500
cpl_msg_set_time_off, 501
cpl_msg_set_time_on, 501
cpl_msg_set_width, 502
cpl_msg_stop, 502
cpl_msg_stop_log, 503
cpl_msg_warning, 503
min_count
_cpl_framedata_, 999
Multi Frames, 505
_cpl_multiframe_id_mode_, 506
cpl_multiframe, 506
cpl_multiframe_add_empty, 507
cpl_multiframe_append_datagroup, 507
cpl_multiframe_append_datagroup_from_position, 508
cpl_multiframe_append_dataset, 509
cpl_multiframe_append_dataset_from_position, 510
cpl_multiframe_delete, 511
cpl_multiframe_get_size, 511
CPL_MULTIFRAME_ID_JOIN, 506
cpl_multiframe_id_mode, 506
CPL_MULTIFRAME_ID_PREFIX, 506
CPL_MULTIFRAME_ID_SET, 506
cpl_multiframe_new, 512
cpl_multiframe_write, 512

name
_cpl_plugin_, 1003

Parameter Lists, 514
cpl_parameterlist, 515
cpl_parameterlist_append, 515
cpl_parameterlist_delete, 516
cpl_parameterlist_dump, 516
cpl_parameterlist_find, 517
cpl_parameterlist_find_const, 517
cpl_parameterlist_find_context, 518
cpl_parameterlist_find_context_const, 519
cpl_parameterlist_find_tag, 519
cpl_parameterlist_find_tag_const, 520
cpl_parameterlist_find_type, 522
cpl_parameterlist_find_type_const, 522
cpl_parameterlist_get_first, 524
cpl_parameterlist_get_first_const, 525
cpl_parameterlist_get_last, 525
cpl_parameterlist_get_last_const, 526
cpl_parameterlist_get_next, 526
cpl_parameterlist_get_next_const, 527
cpl_parameterlist_get_size, 528
cpl_parameterlist_new, 528
Parameters, 530
_cpl_parameter_class_, 534
_cpl_parameter_mode_, 534
cpl_parameter, 533
cpl_parameter_class, 533
CPL_PARAMETER_CLASS_ENUM, 534
CPL_PARAMETER_CLASS_INVALID, 534
CPL_PARAMETER_CLASS_RANGE, 534
CPL_PARAMETER_CLASS_VALUE, 534
cpl_parameter_delete, 535
cpl_parameter_disable, 535
cpl_parameter_dump, 536
cpl_parameter_duplicate, 536
cpl_parameter_enable, 537
cpl_parameter_get_alias, 537
cpl_parameter_get_bool, 538
cpl_parameter_get_class, 539
cpl_parameter_get_context, 539
cpl_parameter_get_default_bool, 540
cpl_parameter_get_default_double, 541
cpl_parameter_get_default_flag, 541
cpl_parameter_get_default_int, 542
cpl_parameter_get_default_string, 543
cpl_parameter_get_double, 543
cpl_parameter_get_enum_double, 544
cpl_parameter_get_enum_int, 545
cpl_parameter_get_enum_size, 545

cpl_parameter_get_enum_string, 546
cpl_parameter_get_help, 547
cpl_parameter_get_id, 548
cpl_parameter_get_int, 548
cpl_parameter_get_name, 549
cpl_parameter_get_range_max_double, 550
cpl_parameter_get_range_max_int, 550
cpl_parameter_get_range_min_double, 551
cpl_parameter_get_range_min_int, 551
cpl_parameter_get_string, 552
cpl_parameter_get_tag, 553
cpl_parameter_get_type, 553
cpl_parameter_is_enabled, 554
cpl_parameter_mode, 534
CPL_PARAMETER_MODE_CFG, 534
CPL_PARAMETER_MODE_CLI, 534
CPL_PARAMETER_MODE_ENV, 534
cpl_parameter_new_enum, 555
cpl_parameter_new_enum_from_array, 556
cpl_parameter_new_range, 557
cpl_parameter_new_value, 558
cpl_parameter_set_alias, 559
cpl_parameter_set_bool, 560
cpl_parameter_set_default_bool, 561
cpl_parameter_set_default_double, 562
cpl_parameter_set_default_flag, 562
cpl_parameter_set_default_int, 563
cpl_parameter_set_default_string, 564
cpl_parameter_set_double, 564
cpl_parameter_set_id, 565
cpl_parameter_set_int, 565
cpl_parameter_set_string, 566
cpl_parameter_set_tag, 567
parameters
_cpl_recipe_, 1004
Plotting of CPL objects, 568
cpl_plot_bivector, 569
cpl_plot_bivectors, 569
cpl_plot_column, 570
cpl_plot_columns, 571
cpl_plot_image, 572
cpl_plot_image_col, 573
cpl_plot_image_row, 574
cpl_plot_mask, 575
cpl_plot_vector, 576
cpl_plot_vectors, 576
Plugin Interface, 579
_cpl_plugin_type_, 582
cpl_plugin, 581
CPL_PLUGIN_API, 581
cpl_plugin_copy, 582
cpl_plugin_delete, 583
cpl_plugin_dump, 584
cpl_plugin_get_api, 584
cpl_plugin_get_author, 585
cpl_plugin_get_copyright, 585
cpl_plugin_get_deinit, 586
cpl_plugin_get_description, 587
cpl_plugin_get_email, 587
cpl_plugin_get_exec, 588
cpl_plugin_get_info, 589
cpl_plugin_get_init, 589
cpl_plugin_get_name, 590
cpl_plugin_get_synopsis, 590
cpl_plugin_get_type, 591
cpl_plugin_get_type_string, 592
cpl_plugin_get_version, 592
cpl_plugin_get_version_string, 593
cpl_plugin_init, 593
cpl_plugin_new, 595
cpl_plugin_set_api, 595
cpl_plugin_set_author, 596
cpl_plugin_set_copyright, 596
cpl_plugin_set_deinit, 597
cpl_plugin_set_description, 598
cpl_plugin_set_email, 598
cpl_plugin_set_exec, 599
cpl_plugin_set_init, 600
cpl_plugin_set_name, 600
cpl_plugin_set_synopsis, 601
cpl_plugin_set_type, 602
cpl_plugin_set_version, 602
cpl_plugin_type, 581
CPL_PLUGIN_TYPE_NONE, 582
CPL_PLUGIN_TYPE_RECIPE, 582
CPL_PLUGIN_TYPE_RECIPE_V2, 582
Plugin List, 604
cpl_pluginlist, 604
cpl_pluginlist_append, 605
cpl_pluginlist_delete, 605
cpl_pluginlist_dump, 606
cpl_pluginlist_find, 606
cpl_pluginlist_get_first, 607
cpl_pluginlist_get_last, 607
cpl_pluginlist_get_next, 608
cpl_pluginlist_get_size, 609
cpl_pluginlist_new, 609
cpl_pluginlist_prepend, 609
Point pattern matching module, 611
cpl_ppm_match_points, 611
cpl_ppm_match_positions, 614
Polynomials, 616
cpl_polynomial_add, 617
cpl_polynomial_compare, 618
cpl_polynomial_copy, 619
cpl_polynomial_delete, 619
cpl_polynomial_derivative, 620
cpl_polynomial_dump, 621
cpl_polynomial_duplicate, 621
cpl_polynomial_eval, 622
cpl_polynomial_eval_1d, 623
cpl_polynomial_eval_1d_diff, 624
cpl_polynomial_eval_2d, 624
cpl_polynomial_eval_3d, 625
cpl_polynomial_extract, 626
cpl_polynomial_fit, 627

cpl_polynomial_fit_1d_create, 629
cpl_polynomial_fit_2d_create, 630
cpl_polynomial_get_coeff, 630
cpl_polynomial_get_degree, 631
cpl_polynomial_get_dimension, 632
cpl_polynomial_multiply, 632
cpl_polynomial_multiply_scalar, 633
cpl_polynomial_set_coeff, 634
cpl_polynomial_shift_1d, 635
cpl_polynomial_solve_1d, 635
cpl_polynomial_subtract, 636
cpl_vector_fill_polynomial, 637
cpl_vector_fill_polynomial_fit_residual, 638
Properties, 640
cpl_property, 641
cpl_property_delete, 641
cpl_property_dump, 642
cpl_property_duplicate, 642
cpl_property_get_bool, 643
cpl_property_get_char, 644
cpl_property_get_comment, 644
cpl_property_get_double, 645
cpl_property_get_double_complex, 645
cpl_property_get_float, 646
cpl_property_get_float_complex, 647
cpl_property_get_int, 647
cpl_property_get_long, 648
cpl_property_get_long_long, 649
cpl_property_get_name, 649
cpl_property_get_size, 650
cpl_property_get_string, 651
cpl_property_get_type, 651
cpl_property_new, 652
cpl_property_new_array, 653
cpl_property_set_bool, 654
cpl_property_set_char, 654
cpl_property_set_comment, 655
cpl_property_set_double, 656
cpl_property_set_double_complex, 656
cpl_property_set_float, 657
cpl_property_set_float_complex, 658
cpl_property_set_int, 659
cpl_property_set_long, 659
cpl_property_set_long_long, 660
cpl_property_set_name, 661
cpl_property_set_string, 661
Property Lists, 663
cpl_propertylist, 668
cpl_propertylist_append, 668
cpl_propertylist_append_bool, 669
cpl_propertylist_append_char, 669
cpl_propertylist_append_double, 670
cpl_propertylist_append_double_complex, 671
cpl_propertylist_append_float, 671
cpl_propertylist_append_float_complex, 672
cpl_propertylist_append_int, 673
cpl_propertylist_append_long, 673
cpl_propertylist_append_long_long, 674
cpl_propertylist_append_property, 675
cpl_propertylist_append_string, 675
cpl_propertylist_compare_func, 668
cpl_propertylist_copy_property, 676
cpl_propertylist_copy_property_regexp, 677
cpl_propertylist_delete, 678
cpl_propertylist_dump, 678
cpl_propertylist_duplicate, 679
cpl_propertylist_empty, 679
cpl_propertylist_erase, 680
cpl_propertylist_erase_regexp, 680
cpl_propertylist_get, 681
cpl_propertylist_get_bool, 682
cpl_propertylist_get_char, 682
cpl_propertylist_get_comment, 683
cpl_propertylist_get_const, 684
cpl_propertylist_get_double, 684
cpl_propertylist_get_double_complex, 685
cpl_propertylist_get_float, 686
cpl_propertylist_get_float_complex, 687
cpl_propertylist_get_int, 687
cpl_propertylist_get_long, 688
cpl_propertylist_get_long_long, 689
cpl_propertylist_get_property, 689
cpl_propertylist_get_property_const, 690
cpl_propertylist_get_size, 691
cpl_propertylist_get_string, 691
cpl_propertylist_get_type, 692
cpl_propertylist_has, 693
cpl_propertylist_insert_after_bool, 693
cpl_propertylist_insert_after_char, 694
cpl_propertylist_insert_after_double, 695
cpl_propertylist_insert_after_double_complex, 695
cpl_propertylist_insert_after_float, 697
cpl_propertylist_insert_after_float_complex, 698
cpl_propertylist_insert_after_int, 698
cpl_propertylist_insert_after_long, 700
cpl_propertylist_insert_after_long_long, 701
cpl_propertylist_insert_after_property, 701
cpl_propertylist_insert_after_string, 702
cpl_propertylist_insert_bool, 703
cpl_propertylist_insert_char, 703
cpl_propertylist_insert_double, 705
cpl_propertylist_insert_double_complex, 706
cpl_propertylist_insert_float, 706
cpl_propertylist_insert_float_complex, 708
cpl_propertylist_insert_int, 709
cpl_propertylist_insert_long, 709
cpl_propertylist_insert_long_long, 711
cpl_propertylist_insert_property, 712
cpl_propertylist_insert_string, 712
cpl_propertylist_is_empty, 713
cpl_propertylist_load, 714
cpl_propertylist_load_regexp, 714
cpl_propertylist_new, 715
cpl_propertylist_prepend_bool, 716
cpl_propertylist_prepend_char, 717
cpl_propertylist_prepend_double, 717

cpl_propertylist_prepend_double_complex, 718
cpl_propertylist_prepend_float, 718
cpl_propertylist_prepend_float_complex, 719
cpl_propertylist_prepend_int, 720
cpl_propertylist_prepend_long, 720
cpl_propertylist_prepend_long_long, 721
cpl_propertylist_prepend_property, 722
cpl_propertylist_prepend_string, 722
cpl_propertylist_save, 723
cpl_propertylist_set_bool, 724
cpl_propertylist_set_char, 725
cpl_propertylist_set_comment, 725
cpl_propertylist_set_double, 726
cpl_propertylist_set_double_complex, 727
cpl_propertylist_set_float, 727
cpl_propertylist_set_float_complex, 728
cpl_propertylist_set_int, 729
cpl_propertylist_set_long, 729
cpl_propertylist_set_long_long, 730
cpl_propertylist_set_string, 731
cpl_propertylist_sort, 731
cpl_propertylist_update_bool, 732
cpl_propertylist_update_char, 733
cpl_propertylist_update_double, 733
cpl_propertylist_update_double_complex, 734
cpl_propertylist_update_float, 735
cpl_propertylist_update_float_complex, 736
cpl_propertylist_update_int, 736
cpl_propertylist_update_long, 737
cpl_propertylist_update_long_long, 738
cpl_propertylist_update_string, 738

Recipe Configurations, 740
cpl_recipeconfig_clear, 741
cpl_recipeconfig_delete, 741
cpl_recipeconfig_get_inputs, 741
cpl_recipeconfig_get_max_count, 742
cpl_recipeconfig_get_min_count, 743
cpl_recipeconfig_get_outputs, 744
cpl_recipeconfig_get_tags, 744
cpl_recipeconfig_is_required, 746
cpl_recipeconfig_new, 747
cpl_recipeconfig_set_input, 747
cpl_recipeconfig_set_inputs, 748
cpl_recipeconfig_set_output, 749
cpl_recipeconfig_set_outputs, 750
cpl_recipeconfig_set_tag, 751
cpl_recipeconfig_set_tags, 752
Recipe Definition, 753
cpl_get_license, 753
CPL_RECIPE_DEFINE, 754
cpl_recipe_define, 753
Recipes, 756
cpl_recipe, 756
Regular Expression Filter, 757
_cpl_regex_syntax_option_, 758
cpl_regex, 757
cpl_regex_apply, 758
CPL_REGEX_BASIC, 758
cpl_regex_delete, 759
CPL_REGEX_EXTENDED, 758
CPL_REGEX_ICASE, 758
cpl_regex_is_negated, 759
cpl_regex_negate, 759
cpl_regex_new, 760
CPL_REGEX_NOSUBS, 758
cpl_regex_syntax_option, 757

Statistics, 761
_cpl_stats_mode_, 763
cpl_stats, 762
CPL_STATS_ABSFLUX, 763
CPL_STATS_ALL, 763
CPL_STATS_CENTROID, 763
cpl_stats_delete, 763
cpl_stats_dump, 764
CPL_STATS_FLUX, 763
cpl_stats_get_absflux, 764
cpl_stats_get_centroid_x, 765
cpl_stats_get_centroid_y, 765
cpl_stats_get_flux, 766
cpl_stats_get_mad, 766
cpl_stats_get_max, 767
cpl_stats_get_max_x, 767
cpl_stats_get_max_y, 768
cpl_stats_get_mean, 768
cpl_stats_get_median, 769
cpl_stats_get_median_dev, 769
cpl_stats_get_min, 770
cpl_stats_get_min_x, 770
cpl_stats_get_min_y, 771
cpl_stats_get_npix, 771
cpl_stats_get_sqflux, 772
cpl_stats_get_stdev, 772
CPL_STATS_MAD, 763
CPL_STATS_MAX, 763
CPL_STATS_MAXPOS, 763
CPL_STATS_MEAN, 763
CPL_STATS_MEDIAN, 763
CPL_STATS_MEDIAN_DEV, 763
CPL_STATS_MIN, 763
CPL_STATS_MINPOS, 763
cpl_stats_mode, 762
cpl_stats_new_from_image, 773
cpl_stats_new_from_image_window, 773
CPL_STATS_SQFLUX, 763
CPL_STATS_STDEV, 763
synopsis
_cpl_plugin_, 1003

Tables, 776
CPL_DFS_SIGNATURE_CHECKSUM, 785
CPL_DFS_SIGNATURE_DATAMD5, 785
CPL_DFS_SIGNATURE_NONE, 785
cpl_table_abs_column, 785
cpl_table_add_columns, 785
cpl_table_add_scalar, 786
cpl_table_add_scalar_complex, 787

cpl_table_and_selected, 787
cpl_table_and_selected_double, 788
cpl_table_and_selected_double_complex, 789
cpl_table_and_selected_float, 789
cpl_table_and_selected_float_complex, 790
cpl_table_and_selected_int, 791
cpl_table_and_selected_invalid, 791
cpl_table_and_selected_long, 792
cpl_table_and_selected_long_long, 793
cpl_table_and_selected_string, 793
cpl_table_and_selected_window, 794
cpl_table_arg_column, 795
cpl_table_cast_column, 796
cpl_table_compare_structure, 797
cpl_table_conjugate_column, 798
cpl_table_copy_data_double, 798
cpl_table_copy_data_double_complex, 799
cpl_table_copy_data_float, 799
cpl_table_copy_data_float_complex, 800
cpl_table_copy_data_int, 801
cpl_table_copy_data_long, 801
cpl_table_copy_data_long_long, 802
cpl_table_copy_data_string, 802
cpl_table_copy_structure, 804
cpl_table_count_invalid, 805
cpl_table_count_selected, 805
cpl_table_delete, 806
cpl_table_divide_columns, 806
cpl_table_divide_scalar, 807
cpl_table_divide_scalar_complex, 808
cpl_table_dump, 808
cpl_table_dump_structure, 809
cpl_table_duplicate, 810
cpl_table_duplicate_column, 810
cpl_table_erase_column, 811
cpl_table_erase_invalid, 811
cpl_table_erase_invalid_rows, 812
cpl_table_erase_selected, 813
cpl_table_erase_window, 814
cpl_table_exponential_column, 814
cpl_table_extract, 815
cpl_table_extract_selected, 816
cpl_table_fill_column_window, 816
cpl_table_fill_column_window_array, 817
cpl_table_fill_column_window_complex, 818
cpl_table_fill_column_window_double, 819
cpl_table_fill_column_window_double_complex, 820
cpl_table_fill_column_window_float, 820
cpl_table_fill_column_window_float_complex, 821
cpl_table_fill_column_window_int, 822
cpl_table_fill_column_window_long, 823
cpl_table_fill_column_window_long_long, 824
cpl_table_fill_column_window_string, 824
cpl_table_fill_invalid_double, 825
cpl_table_fill_invalid_double_complex, 826
cpl_table_fill_invalid_float, 827
cpl_table_fill_invalid_float_complex, 828
cpl_table_fill_invalid_int, 828
cpl_table_fill_invalid_long, 829
cpl_table_fill_invalid_long_long, 830
cpl_table_get, 831
cpl_table_get_array, 832
cpl_table_get_column_depth, 832
cpl_table_get_column_dimension, 833
cpl_table_get_column_dimensions, 834
cpl_table_get_column_format, 834
cpl_table_get_column_max, 835
cpl_table_get_column_maxpos, 835
cpl_table_get_column_mean, 836
cpl_table_get_column_mean_complex, 836
cpl_table_get_column_median, 837
cpl_table_get_column_min, 837
cpl_table_get_column_minpos, 838
cpl_table_get_column_name, 838
cpl_table_get_column_names, 839
cpl_table_get_column_stdev, 839
cpl_table_get_column_type, 840
cpl_table_get_column_unit, 841
cpl_table_get_complex, 841
cpl_table_get_data_array, 842
cpl_table_get_data_array_const, 843
cpl_table_get_data_double, 843
cpl_table_get_data_double_complex, 844
cpl_table_get_data_double_complex_const, 845
cpl_table_get_data_double_const, 846
cpl_table_get_data_float, 846
cpl_table_get_data_float_complex, 847
cpl_table_get_data_float_complex_const, 848
cpl_table_get_data_float_const, 849
cpl_table_get_data_int, 849
cpl_table_get_data_int_const, 850
cpl_table_get_data_long, 851
cpl_table_get_data_long_const, 851
cpl_table_get_data_long_long, 852
cpl_table_get_data_long_long_const, 853
cpl_table_get_data_string, 853
cpl_table_get_data_string_const, 854
cpl_table_get_double, 855
cpl_table_get_double_complex, 855
cpl_table_get_float, 856
cpl_table_get_float_complex, 857
cpl_table_get_int, 857
cpl_table_get_long, 858
cpl_table_get_long_long, 859
cpl_table_get_ncol, 860
cpl_table_get_nrow, 860
cpl_table_get_string, 861
cpl_table_has_column, 862
cpl_table_has_invalid, 862
cpl_table_has_valid, 863
cpl_table_imag_column, 864
cpl_table_insert, 864
cpl_table_insert_window, 865
cpl_table_is_selected, 866
cpl_table_is_valid, 866

cpl_table_load, 867
cpl_table_load_window, 868
cpl_table_logarithm_column, 869
cpl_table_move_column, 870
cpl_table_multiply_columns, 870
cpl_table_multiply_scalar, 871
cpl_table_multiply_scalar_complex, 872
cpl_table_name_column, 872
cpl_table_new, 873
cpl_table_new_column, 873
cpl_table_new_column_array, 874
cpl_table_not_selected, 875
cpl_table_or_selected, 876
cpl_table_or_selected_double, 876
cpl_table_or_selected_double_complex, 877
cpl_table_or_selected_float, 878
cpl_table_or_selected_float_complex, 878
cpl_table_or_selected_int, 879
cpl_table_or_selected_invalid, 880
cpl_table_or_selected_long, 880
cpl_table_or_selected_long_long, 881
cpl_table_or_selected_string, 882
cpl_table_or_selected_window, 883
cpl_table_power_column, 883
cpl_table_real_column, 884
cpl_table_save, 885
cpl_table_select_all, 886
cpl_table_select_row, 887
cpl_table_set, 888
cpl_table_set_array, 888
cpl_table_set_column_depth, 889
cpl_table_set_column_dimensions, 890
cpl_table_set_column_format, 890
cpl_table_set_column_invalid, 891
cpl_table_set_column_unit, 892
cpl_table_set_complex, 892
cpl_table_set_double, 893
cpl_table_set_double_complex, 894
cpl_table_set_float, 895
cpl_table_set_float_complex, 896
cpl_table_set_int, 896
cpl_table_set_invalid, 897
cpl_table_set_long, 898
cpl_table_set_long_long, 899
cpl_table_set_size, 899
cpl_table_set_string, 900
cpl_table_shift_column, 901
cpl_table_sort, 902
cpl_table_subtract_columns, 902
cpl_table_subtract_scalar, 903
cpl_table_subtract_scalar_complex, 905
cpl_table_unselect_all, 905
cpl_table_unselect_row, 906
cpl_table_unwrap, 906
cpl_table_where_selected, 907
cpl_table_wrap_double, 908
cpl_table_wrap_double_complex, 908
cpl_table_wrap_float, 909
cpl_table_wrap_float_complex, 910
cpl_table_wrap_int, 910
cpl_table_wrap_long, 911
cpl_table_wrap_long_long, 912
cpl_table_wrap_string, 912
tag
_cpl_framedata_, 1000
type
_cpl_plugin_, 1003
Type codes, 914
_cpl_type_, 916
cpl_bitmask, 915
cpl_size, 916
CPL_SIZE_FORMAT, 915
CPL_SIZE_MAX, 915
CPL_SIZE_MIN, 915
cpl_type, 916
CPL_TYPE_BITMASK, 917
CPL_TYPE_BOOL, 916
CPL_TYPE_CHAR, 916
CPL_TYPE_COMPLEX, 917
CPL_TYPE_DOUBLE, 917
CPL_TYPE_DOUBLE_COMPLEX, 917
CPL_TYPE_FLAG_ARRAY, 916
CPL_TYPE_FLOAT, 917
CPL_TYPE_FLOAT_COMPLEX, 917
cpl_type_get_name, 917
cpl_type_get_sizeof, 918
CPL_TYPE_INT, 916
CPL_TYPE_INVALID, 916
CPL_TYPE_LONG, 917
CPL_TYPE_LONG_LONG, 917
CPL_TYPE_POINTER, 917
CPL_TYPE_SHORT, 916
CPL_TYPE_SIZE, 917
CPL_TYPE_STRING, 917
CPL_TYPE_UCHAR, 916
CPL_TYPE_UINT, 917
CPL_TYPE_ULONG, 917
CPL_TYPE_UNSPECIFIED, 917
CPL_TYPE_USHORT, 916

Unit testing functions, 919
cpl_test, 920
cpl_test_abs, 921
cpl_test_abs_complex, 921
cpl_test_array_abs, 922
cpl_test_assert, 923
cpl_test_end, 938
cpl_test_eq, 923
cpl_test_eq_error, 924
cpl_test_eq_mask, 924
cpl_test_eq_ptr, 925
cpl_test_eq_string, 925
cpl_test_error, 927
cpl_test_errorstate, 927
cpl_test_fits, 928
cpl_test_get_bytes_image, 939
cpl_test_get_bytes_imagelist, 939

cpl_test_get_bytes_matrix, 940
cpl_test_get_bytes_vector, 940
cpl_test_get_failed, 941
cpl_test_get_tested, 941
cpl_test_get_walltime, 942
cpl_test_image_abs, 928
cpl_test_image_rel, 929
cpl_test_imagelist_abs, 930
cpl_test_init, 930
cpl_test_leq, 931
cpl_test_lt, 931
cpl_test_matrix_abs, 932
cpl_test_memory_is_empty, 933
cpl_test_noneq, 933
cpl_test_noneq_ptr, 933
cpl_test_noneq_string, 934
cpl_test_nonnull, 934
cpl_test_null, 935
cpl_test_polynomial_abs, 935
cpl_test_rel, 936
cpl_test_vector_abs, 937
cpl_test_zero, 937

Vector, 943
cpl_vector_add, 945
cpl_vector_add_scalar, 945
cpl_vector_convolve_symmetric, 946
cpl_vector_copy, 946
cpl_vector_correlate, 947
cpl_vector_cycle, 948
cpl_vector_delete, 949
cpl_vector_divide, 950
cpl_vector_divide_scalar, 950
cpl_vector_dump, 951
cpl_vector_duplicate, 951
cpl_vector_exponential, 952
cpl_vector_extract, 953
cpl_vector_fill, 953
cpl_vector_fill_kernel_profile, 954
cpl_vector_filter_lowpass_create, 955
cpl_vector_filter_median_create, 956
cpl_vector_find, 956
cpl_vector_fit_gaussian, 957
cpl_vector_get, 959
cpl_vector_get_data, 960
cpl_vector_get_data_const, 960
cpl_vector_get_max, 961
cpl_vector_get_maxpos, 961
cpl_vector_get_mean, 962
cpl_vector_get_median, 962
cpl_vector_get_median_const, 963
cpl_vector_get_min, 963
cpl_vector_get_minpos, 964
cpl_vector_get_size, 964
cpl_vector_get_stdev, 965
cpl_vector_get_sum, 966
cpl_vector_load, 966
cpl_vector_logarithm, 967
cpl_vector_multiply, 967
cpl_vector_multiply_scalar, 968
cpl_vector_new, 968
cpl_vector_new_lss_kernel, 969
cpl_vector_power, 970
cpl_vector_product, 970
cpl_vector_read, 971
cpl_vector_save, 971
cpl_vector_set, 972
cpl_vector_set_size, 973
cpl_vector_sort, 974
cpl_vector_sqrt, 974
cpl_vector_subtract, 975
cpl_vector_subtract_scalar, 975
cpl_vector_unwrap, 976
cpl_vector_wrap, 976
version
_cpl_plugin_, 1003

Wavelength calibration, 978
cpl_wlcalib_fill_line_spectrum, 978
cpl_wlcalib_fill_line_spectrum_fast, 979
cpl_wlcalib_fill_logline_spectrum, 980
cpl_wlcalib_fill_logline_spectrum_fast, 981
cpl_wlcalib_find_best_1d, 981
cpl_wlcalib_slitmodel_delete, 983
cpl_wlcalib_slitmodel_new, 983
cpl_wlcalib_slitmodel_set_catalog, 984
cpl_wlcalib_slitmodel_set_threshold, 984
cpl_wlcalib_slitmodel_set_wfwhm, 985
cpl_wlcalib_slitmodel_set_wslit, 986
World Coordinate System, 987
cpl_wcs_convert, 988
cpl_wcs_delete, 989
cpl_wcs_get_cd, 989
cpl_wcs_get_crpix, 991
cpl_wcs_get_crval, 991
cpl_wcs_get_ctype, 992
cpl_wcs_get_cunit, 993
cpl_wcs_get_image_dims, 993
cpl_wcs_get_image_naxis, 994
cpl_wcs_new_from_propertylist, 994
cpl_wcs_platesol, 996
CPL_WCS_REGEXP, 987