

libdwarf

Generated by Doxygen 1.9.8

1 A Consumer Library Interface to DWARF	1
1.1 Suggestions for improvement are welcome.	2
1.2 Downloading Libdwarf	2
1.3 Introduction	2
1.4 Thread Safety	3
1.5 Error Handling in libdwarf	3
1.5.1 Error Handling at Initialization	3
1.5.2 Error Handling Everywhere	4
1.5.2.1 DW_DLV_OK	5
1.5.2.2 DW_DLV_NO_ENTRY	5
1.5.2.3 DW_DLV_ERROR	5
1.5.2.4 Slight Performance Enhancement	5
1.6 Extracting Data Per Compilation Unit	6
1.7 Line Table Registers	6
1.8 Reading Special Sections Independently	7
1.9 Special Frame Registers	7
1.10 .debug_pubnames etc DWARF2-DWARF4	8
1.11 Reading DWARF with no object file present	9
1.12 Section Groups: Split Dwarf, COMDAT groups	11
1.13 Details on separate DWARF object access	12
1.14 Linking against libdwarf.so (or dll or dylib)	13
1.15 Linking against libdwarf.a	14
1.16 Suppressing CRC calculation for debuglink	14
1.17 Object Reading By User Code	15
1.18 dwsec_mmap	15
1.19 Recent Changes	16
2 JIT and special case DWARF	23
2.1 Reading DWARF not in an object file	23
2.1.1 Describing the Interface	25
2.1.2 Describing A Section	25
2.1.3 Function Pointers	26
3 dwarf.h	29
4 libdwarf.h	31
5 checkexamples.c	33
6 Topic Index	35
6.1 Topics	35
7 Class Index	37
7.1 Class List	37

8 File Index	39
8.1 File List	39
9 Topic Documentation	41
9.1 Basic Library Datatypes Group	41
9.1.1 Detailed Description	41
9.1.2 Typedef Documentation	41
9.1.2.1 Dwarf_Addr	41
9.1.2.2 Dwarf_Bool	41
9.1.2.3 Dwarf_Half	42
9.1.2.4 Dwarf_Off	42
9.1.2.5 Dwarf_Ptr	42
9.1.2.6 Dwarf_Signed	42
9.1.2.7 Dwarf_Small	42
9.1.2.8 Dwarf_Unsigned	42
9.2 Enumerators with various purposes	42
9.2.1 Detailed Description	43
9.2.2 Enumeration Type Documentation	43
9.2.2.1 Dwarf_Form_Class	43
9.2.2.2 Dwarf_Ranges_Entry_Type	43
9.3 Defined and Opaque Structs	43
9.3.1 Detailed Description	45
9.3.2 Typedef Documentation	45
9.3.2.1 Dwarf_Abbrev	45
9.3.2.2 Dwarf_Arange	45
9.3.2.3 Dwarf_Attribute	45
9.3.2.4 Dwarf_Block	45
9.3.2.5 Dwarf_Cie	45
9.3.2.6 Dwarf_Debug	45
9.3.2.7 Dwarf_Debug_Addr_Table	46
9.3.2.8 Dwarf_Debug_Fission_Per_CU	46
9.3.2.9 Dwarf_Die	46
9.3.2.10 Dwarf_Dnames_Head	46
9.3.2.11 Dwarf_Dsc_Head	46
9.3.2.12 Dwarf_Error	46
9.3.2.13 Dwarf_Fde	46
9.3.2.14 Dwarf_Form_Data16	47
9.3.2.15 Dwarf_Frame_Instr_Head	47
9.3.2.16 Dwarf_Func	47
9.3.2.17 Dwarf_Gdbindex	47
9.3.2.18 Dwarf_Global	47
9.3.2.19 Dwarf_Gnu_Index_Head	47

9.3.2.20 Dwarf_Handler	47
9.3.2.21 Dwarf_Line	48
9.3.2.22 Dwarf_Line_Context	48
9.3.2.23 Dwarf_Loc_Head_c	48
9.3.2.24 Dwarf_Locdesc_c	48
9.3.2.25 Dwarf_Macro_Context	48
9.3.2.26 Dwarf_Macro_Details	48
9.3.2.27 Dwarf_Obj_Access_Interface_a	48
9.3.2.28 Dwarf_Obj_Access_Methods_a	48
9.3.2.29 Dwarf_Obj_Access_Section_a	49
9.3.2.30 dwarf_printf_callback_function_type	49
9.3.2.31 Dwarf_Ranges	49
9.3.2.32 Dwarf_Regtable3	49
9.3.2.33 Dwarf_Regtable_Entry3	50
9.3.2.34 Dwarf_Rnglists_Head	51
9.3.2.35 Dwarf_Section	51
9.3.2.36 Dwarf_Sig8	51
9.3.2.37 Dwarf_Str_Offsets_Table	51
9.3.2.38 Dwarf_Type	51
9.3.2.39 Dwarf_Var	52
9.3.2.40 Dwarf_Weak	52
9.3.2.41 Dwarf_Xu_Index_Header	52
9.3.3 Enumeration Type Documentation	52
9.3.3.1 Dwarf_Sec_Alloc_Pref	52
9.4 Default stack frame macros	52
9.4.1 Detailed Description	53
9.5 DW_DLA alloc/dealloc typename&number	53
9.5.1 Detailed Description	53
9.6 DW_DLE Dwarf_Error numbers	54
9.6.1 Detailed Description	63
9.6.2 Macro Definition Documentation	63
9.6.2.1 DW_DLE_LAST	63
9.7 Libdwarf Initialization Functions	64
9.7.1 Detailed Description	64
9.7.2 Initialization And Finish Operations	64
9.7.3 Function Documentation	64
9.7.3.1 dwarf_finish()	64
9.7.3.2 dwarf_get_tied_dbg()	65
9.7.3.3 dwarf_init_b()	65
9.7.3.4 dwarf_init_path()	66
9.7.3.5 dwarf_init_path_a()	67
9.7.3.6 dwarf_init_path_dl()	68

9.7.3.7 dwarf_init_path_dl_a()	69
9.7.3.8 dwarf_object_finish()	69
9.7.3.9 dwarf_object_init_b()	69
9.7.3.10 dwarf_set_tied_dbg()	70
9.8 Compilation Unit (CU) Access	71
9.8.1 Detailed Description	72
9.8.2 Function Documentation	72
9.8.2.1 dwarf_child()	72
9.8.2.2 dwarf_cu_header_basics()	72
9.8.2.3 dwarf_dealloc_die()	73
9.8.2.4 dwarf_die_from_hash_signature()	73
9.8.2.5 dwarf_find_die_given_sig8()	74
9.8.2.6 dwarf_get_die_infotypes_flag()	74
9.8.2.7 dwarf_next_cu_header_d()	75
9.8.2.8 dwarf_next_cu_header_e()	75
9.8.2.9 dwarf_offdie_b()	76
9.8.2.10 dwarf_siblingof_b()	77
9.8.2.11 dwarf_siblingof_c()	78
9.9 Debugging Information Entry (DIE) content	78
9.9.1 Detailed Description	80
9.9.2 Function Documentation	80
9.9.2.1 dwarf_addr_form_is_indexed()	80
9.9.2.2 dwarf_arrayorder()	81
9.9.2.3 dwarf_attr()	81
9.9.2.4 dwarf_bitoffset()	81
9.9.2.5 dwarf_bitsize()	82
9.9.2.6 dwarf_bytesize()	82
9.9.2.7 dwarf_CU_dieoffset_given_die()	83
9.9.2.8 dwarf_debug_addr_index_to_addr()	83
9.9.2.9 dwarf_die_abbrev_children_flag()	84
9.9.2.10 dwarf_die_abbrev_code()	84
9.9.2.11 dwarf_die_abbrev_global_offset()	84
9.9.2.12 dwarf_die_CU_offset()	85
9.9.2.13 dwarf_die_CU_offset_range()	85
9.9.2.14 dwarf_die_offsets()	86
9.9.2.15 dwarf_die_text()	86
9.9.2.16 dwarf_diename()	87
9.9.2.17 dwarf_dieoffset()	87
9.9.2.18 dwarf_dietype_offset()	88
9.9.2.19 dwarf_get_cu_die_offset_given_cu_header_offset_b()	88
9.9.2.20 dwarf_get_die_address_size()	89
9.9.2.21 dwarf_get_version_of_die()	89

9.9.2.22 dwarf_hasattr()	90
9.9.2.23 dwarf_highpc_b()	90
9.9.2.24 dwarf_language_version_data()	91
9.9.2.25 dwarf_language_version_string()	91
9.9.2.26 dwarf_lowpc()	92
9.9.2.27 dwarf_lvn_name()	92
9.9.2.28 dwarf_lvn_name_direct()	93
9.9.2.29 dwarf_lvn_table_entry()	93
9.9.2.30 dwarf_offset_list()	94
9.9.2.31 dwarf_srclang()	94
9.9.2.32 dwarf_srclangname()	95
9.9.2.33 dwarf_srclangname_version()	95
9.9.2.34 dwarf_tag()	96
9.9.2.35 dwarf_validate_die_sibling()	96
9.10 DIE Attribute and Attribute-Form Details	97
9.10.1 Detailed Description	99
9.10.2 Function Documentation	99
9.10.2.1 dwarf_attr_offset()	99
9.10.2.2 dwarf_attrlist()	99
9.10.2.3 dwarf_convert_to_global_offset()	100
9.10.2.4 dwarf_dealloc_attribute()	100
9.10.2.5 dwarf_dealloc_uncompressed_block()	101
9.10.2.6 dwarf_discr_entry_s()	101
9.10.2.7 dwarf_discr_entry_u()	101
9.10.2.8 dwarf_discr_list()	102
9.10.2.9 dwarf_formaddr()	102
9.10.2.10 dwarf_formblock()	103
9.10.2.11 dwarf_formdata16()	103
9.10.2.12 dwarf_formexprloc()	104
9.10.2.13 dwarf_formflag()	104
9.10.2.14 dwarf_formref()	105
9.10.2.15 dwarf_formsdata()	105
9.10.2.16 dwarf_formsig8()	106
9.10.2.17 dwarf_formsig8_const()	106
9.10.2.18 dwarf_formstring()	106
9.10.2.19 dwarf_formudata()	107
9.10.2.20 dwarf_get_debug_addr_index()	107
9.10.2.21 dwarf_get_debug_str_index()	108
9.10.2.22 dwarf_get_form_class()	108
9.10.2.23 dwarf_global_formref()	109
9.10.2.24 dwarf_global_formref_b()	109
9.10.2.25 dwarf_hasform()	110

9.10.2.26 dwarf_uncompress_integer_block_a()	110
9.10.2.27 dwarf_whatattr()	110
9.10.2.28 dwarf_whatform()	111
9.10.2.29 dwarf_whatform_direct()	111
9.11 Line Table For a CU	112
9.11.1 Detailed Description	114
9.11.2 Function Documentation	114
9.11.2.1 dwarf_check_lineheader_b()	114
9.11.2.2 dwarf_line_is_addr_set()	114
9.11.2.3 dwarf_line_srcfileno()	115
9.11.2.4 dwarf_lineaddr()	115
9.11.2.5 dwarf_linebeginstatement()	115
9.11.2.6 dwarf_lineblock()	116
9.11.2.7 dwarf_lineendsequence()	116
9.11.2.8 dwarf_lineno()	117
9.11.2.9 dwarf_lineoff_b()	117
9.11.2.10 dwarf_linesrc()	118
9.11.2.11 dwarf_print_lines()	118
9.11.2.12 dwarf_prologue_end_etc()	119
9.11.2.13 dwarf_register_printf_callback()	119
9.11.2.14 dwarf_srcfiles()	120
9.11.2.15 dwarf_srclines_b()	121
9.11.2.16 dwarf_srclines_comp_dir()	122
9.11.2.17 dwarf_srclines_dealloc_b()	122
9.11.2.18 dwarf_srclines_files_data_b()	122
9.11.2.19 dwarf_srclines_files_indexes()	123
9.11.2.20 dwarf_srclines_from_linecontext()	124
9.11.2.21 dwarf_srclines_include_dir_count()	124
9.11.2.22 dwarf_srclines_include_dir_data()	125
9.11.2.23 dwarf_srclines_subprog_count()	125
9.11.2.24 dwarf_srclines_subprog_data()	126
9.11.2.25 dwarf_srclines_table_offset()	126
9.11.2.26 dwarf_srclines_two_level_from_linecontext()	127
9.11.2.27 dwarf_srclines_version()	127
9.12 Ranges: code addresses in DWARF3-4	128
9.12.1 Detailed Description	128
9.12.2 Function Documentation	128
9.12.2.1 dwarf_dealloc_ranges()	128
9.12.2.2 dwarf_get_ranges_b()	129
9.12.2.3 dwarf_get_ranges_baseaddress()	129
9.13 Rnglists: code addresses in DWARF5	130
9.13.1 Detailed Description	131

9.13.2 Function Documentation	131
9.13.2.1 dwarf_dealloc_rnglists_head()	131
9.13.2.2 dwarf_get_rnglist_context_basics()	131
9.13.2.3 dwarf_get_rnglist_head_basics()	132
9.13.2.4 dwarf_get_rnglist_offset_index_value()	132
9.13.2.5 dwarf_get_rnglist_rle()	133
9.13.2.6 dwarf_get_rnglists_entry_fields_a()	133
9.13.2.7 dwarf_load_rnglists()	134
9.13.2.8 dwarf_rnglists_get_rle_head()	135
9.14 Locations of data: DWARF2-DWARF5	135
9.14.1 Detailed Description	137
9.14.2 Function Documentation	137
9.14.2.1 dwarf_dealloc_loc_head_c()	137
9.14.2.2 dwarf_get_location_op_value_c()	137
9.14.2.3 dwarf_get_locdesc_entry_d()	138
9.14.2.4 dwarf_get_locdesc_entry_e()	139
9.14.2.5 dwarf_get_loclist_c()	139
9.14.2.6 dwarf_get_loclist_context_basics()	140
9.14.2.7 dwarf_get_loclist_head_basics()	140
9.14.2.8 dwarf_get_loclist_head_kind()	141
9.14.2.9 dwarf_get_loclist_lle()	141
9.14.2.10 dwarf_get_loclist_offset_index_value()	142
9.14.2.11 dwarf_load_loclists()	142
9.14.2.12 dwarf_loclist_from_expr_c()	143
9.15 .debug_addr access: DWARF5	143
9.15.1 Detailed Description	144
9.15.2 Function Documentation	144
9.15.2.1 dwarf_dealloc_debug_addr_table()	144
9.15.2.2 dwarf_debug_addr_by_index()	144
9.15.2.3 dwarf_debug_addr_table()	145
9.16 Macro Access: DWARF5	146
9.16.1 Detailed Description	146
9.16.2 Function Documentation	147
9.16.2.1 dwarf_dealloc_macro_context()	147
9.16.2.2 dwarf_get_macro_context()	147
9.16.2.3 dwarf_get_macro_context_by_offset()	147
9.16.2.4 dwarf_get_macro_defundef()	148
9.16.2.5 dwarf_get_macro_import()	149
9.16.2.6 dwarf_get_macro_op()	149
9.16.2.7 dwarf_get_macro_startend_file()	151
9.16.2.8 dwarf_macro_context_head()	151
9.16.2.9 dwarf_macro_context_total_length()	152

9.16.2.10 dwarf_macro_operands_table()	152
9.17 Macro Access: DWARF2-4	153
9.17.1 Detailed Description	153
9.17.2 Function Documentation	153
9.17.2.1 dwarf_find_macro_value_start()	153
9.17.2.2 dwarf_get_macro_details()	154
9.18 Stack Frame Access	154
9.18.1 Detailed Description	157
9.18.2 Typedef Documentation	157
9.18.2.1 dwarf_iterate_fde_callback_function_type	157
9.18.3 Function Documentation	157
9.18.3.1 dwarf_cie_section_offset()	157
9.18.3.2 dwarf_dealloc_fde_cie_list()	158
9.18.3.3 dwarf_dealloc_frame_instr_head()	158
9.18.3.4 dwarf_expand_frame_instructions()	158
9.18.3.5 dwarf_fde_section_offset()	159
9.18.3.6 dwarf_get_cie_augmentation_data()	159
9.18.3.7 dwarf_get_cie_index()	160
9.18.3.8 dwarf_get_cie_info_b()	160
9.18.3.9 dwarf_get_cie_of_fde()	161
9.18.3.10 dwarf_get_fde_at_pc()	162
9.18.3.11 dwarf_get_fde_augmentation_data()	162
9.18.3.12 dwarf_get_fde_exception_info()	163
9.18.3.13 dwarf_get_fde_for_die()	163
9.18.3.14 dwarf_get_fde_info_for_all_regs3()	163
9.18.3.15 dwarf_get_fde_info_for_all_regs3_b()	164
9.18.3.16 dwarf_get_fde_info_for_cfa_reg3_b()	164
9.18.3.17 dwarf_get_fde_info_for_cfa_reg3_c()	165
9.18.3.18 dwarf_get_fde_info_for_reg3_b()	165
9.18.3.19 dwarf_get_fde_info_for_reg3_c()	166
9.18.3.20 dwarf_get_fde_instr_bytes()	167
9.18.3.21 dwarf_get_fde_list()	167
9.18.3.22 dwarf_get_fde_list_eh()	168
9.18.3.23 dwarf_get_fde_n()	168
9.18.3.24 dwarf_get_fde_range()	169
9.18.3.25 dwarf_get_frame_instruction()	169
9.18.3.26 dwarf_get_frame_instruction_a()	170
9.18.3.27 dwarf_iterate_fde_all_regs3()	171
9.18.3.28 dwarf_set_frame_cfa_value()	171
9.18.3.29 dwarf_set_frame_rule_initial_value()	172
9.18.3.30 dwarf_set_frame_rule_table_size()	172
9.18.3.31 dwarf_set_frame_same_value()	173

9.18.3.32 dwarf_set_frame_undefined_value()	173
9.19 Abbreviations Section Details	173
9.19.1 Detailed Description	174
9.19.2 Function Documentation	174
9.19.2.1 dwarf_get_abbrev()	174
9.19.2.2 dwarf_get_abbrev_children_flag()	175
9.19.2.3 dwarf_get_abbrev_code()	175
9.19.2.4 dwarf_get_abbrev_entry_b()	175
9.19.2.5 dwarf_get_abbrev_tag()	176
9.20 String Section .debug_str Details	177
9.20.1 Detailed Description	177
9.20.2 Function Documentation	177
9.20.2.1 dwarf_get_str()	177
9.21 Str_Offsets section details	178
9.21.1 Detailed Description	178
9.21.2 Function Documentation	178
9.21.2.1 dwarf_close_str_offsets_table_access()	178
9.21.2.2 dwarf_next_str_offsets_table()	179
9.21.2.3 dwarf_open_str_offsets_table_access()	180
9.21.2.4 dwarf_str_offsets_statistics()	180
9.21.2.5 dwarf_str_offsets_value_by_index()	180
9.22 Dwarf_Error Functions	181
9.22.1 Detailed Description	181
9.22.2 Function Documentation	181
9.22.2.1 dwarf_dealloc_error()	181
9.22.2.2 dwarf_errmsg()	182
9.22.2.3 dwarf_errmsg_by_number()	182
9.22.2.4 dwarf_errno()	182
9.22.2.5 dwarf_error_creation()	183
9.23 Generic dwarf_dealloc Function	183
9.23.1 Detailed Description	183
9.23.2 Function Documentation	184
9.23.2.1 dwarf_dealloc()	184
9.24 Access to Section .debug_sup	184
9.24.1 Detailed Description	184
9.24.2 Function Documentation	184
9.24.2.1 dwarf_get_debug_sup()	184
9.25 Fast Access to .debug_names DWARF5	185
9.25.1 Detailed Description	186
9.25.2 Function Documentation	186
9.25.2.1 dwarf_dealloc_dnames()	186
9.25.2.2 dwarf_dnames_abbrevtable()	186

9.25.2.3 dwarf_dnames_bucket()	187
9.25.2.4 dwarf_dnames_cu_table()	187
9.25.2.5 dwarf_dnames_entrypool()	188
9.25.2.6 dwarf_dnames_entrypool_values()	189
9.25.2.7 dwarf_dnames_header()	190
9.25.2.8 dwarf_dnames_name()	190
9.25.2.9 dwarf_dnames_offsets()	191
9.25.2.10 dwarf_dnames_sizes()	192
9.26 Fast Access to a CU given a code address	192
9.26.1 Detailed Description	193
9.26.2 Function Documentation	193
9.26.2.1 dwarf_get_arange()	193
9.26.2.2 dwarf_get_arange_cu_header_offset()	193
9.26.2.3 dwarf_get_arange_info_b()	194
9.26.2.4 dwarf_get_aranges()	194
9.26.2.5 dwarf_get_cu_die_offset()	195
9.27 Fast Access to .debug_pubnames and more.	195
9.27.1 Detailed Description	196
9.27.2 Function Documentation	196
9.27.2.1 dwarf_get_globals()	196
9.27.2.2 dwarf_get_globals_header()	197
9.27.2.3 dwarf_get_pubtypes()	197
9.27.2.4 dwarf_global_cu_offset()	198
9.27.2.5 dwarf_global_die_offset()	198
9.27.2.6 dwarf_global_name_offsets()	199
9.27.2.7 dwarf_global_tag_number()	199
9.27.2.8 dwarf_globals_by_type()	199
9.27.2.9 dwarf_globals_dealloc()	201
9.27.2.10 dwarf_globname()	201
9.27.2.11 dwarf_return_empty_pubnames()	202
9.28 Fast Access to GNU .debug_gnu_pubnames	202
9.28.1 Detailed Description	202
9.28.2 Function Documentation	203
9.28.2.1 dwarf_get_gnu_index_block()	203
9.28.2.2 dwarf_get_gnu_index_block_entry()	203
9.28.2.3 dwarf_get_gnu_index_head()	205
9.28.2.4 dwarf_gnu_index_dealloc()	205
9.29 Fast Access to Gdb Index	206
9.29.1 Detailed Description	207
9.29.2 Function Documentation	207
9.29.2.1 dwarf_dealloc_gdbindex()	207
9.29.2.2 dwarf_gdbindex_addressarea()	207

9.29.2.3 dwarf_gdbindex_addressarea_entry()	208
9.29.2.4 dwarf_gdbindex_culist_array()	208
9.29.2.5 dwarf_gdbindex_culist_entry()	209
9.29.2.6 dwarf_gdbindex_cuvector_inner_attributes()	209
9.29.2.7 dwarf_gdbindex_cuvector_instance_expand_value()	210
9.29.2.8 dwarf_gdbindex_cuvector_length()	210
9.29.2.9 dwarf_gdbindex_header()	211
9.29.2.10 dwarf_gdbindex_string_by_offset()	212
9.29.2.11 dwarf_gdbindex_symboltable_array()	212
9.29.2.12 dwarf_gdbindex_symboltable_entry()	212
9.29.2.13 dwarf_gdbindex_types_culist_array()	213
9.29.2.14 dwarf_gdbindex_types_culist_entry()	213
9.30 Fast Access to Split Dwarf (Debug Fission)	214
9.30.1 Detailed Description	215
9.30.2 Function Documentation	215
9.30.2.1 dwarf_dealloc_xu_header()	215
9.30.2.2 dwarf_get_debugfission_for_die()	215
9.30.2.3 dwarf_get_debugfission_for_key()	215
9.30.2.4 dwarf_get_xu_hash_entry()	217
9.30.2.5 dwarf_get_xu_index_header()	217
9.30.2.6 dwarf_get_xu_index_section_type()	218
9.30.2.7 dwarf_get_xu_section_names()	219
9.30.2.8 dwarf_get_xu_section_offset()	219
9.31 Access GNU .gnu_debuglink, build-id.	220
9.31.1 Detailed Description	220
9.31.2 Function Documentation	221
9.31.2.1 dwarf_add_debuglink_global_path()	221
9.31.2.2 dwarf_basic_crc32()	221
9.31.2.3 dwarf_crc32()	222
9.31.2.4 dwarf_gnu_debuglink()	222
9.31.2.5 dwarf_suppress_debuglink_crc()	223
9.32 Harmless Error recording	224
9.32.1 Detailed Description	225
9.32.2 Function Documentation	225
9.32.2.1 dwarf_get_harmless_error_list()	225
9.32.2.2 dwarf_insert_harmless_error()	226
9.32.2.3 dwarf_set_harmless_error_list_size()	226
9.32.2.4 dwarf_set_harmless_errors_enabled()	226
9.33 Names DW_TAG_member etc as strings	227
9.33.1 Detailed Description	228
9.33.2 Function Documentation	229
9.33.2.1 dwarf_get_EH_name()	229

9.33.2.2 dwarf_get_FORM_CLASS_name()	229
9.33.2.3 dwarf_get_FRAME_name()	229
9.33.2.4 dwarf_get_GNUKIND_name()	229
9.33.2.5 dwarf_get_GNUIVIS_name()	230
9.33.2.6 dwarf_get_LLEX_name()	230
9.33.2.7 dwarf_get_MACINFO_name()	230
9.33.2.8 dwarf_get_MACRO_name()	230
9.34 Object Sections Data	231
9.34.1 Detailed Description	232
9.34.2 Function Documentation	233
9.34.2.1 dwarf_get_address_size()	233
9.34.2.2 dwarf_get_die_section_name()	233
9.34.2.3 dwarf_get_die_section_name_b()	233
9.34.2.4 dwarf_get_frame_section_name()	234
9.34.2.5 dwarf_get_frame_section_name_eh_gnu()	234
9.34.2.6 dwarf_get_line_section_name_from_die()	234
9.34.2.7 dwarf_get_offset_size()	234
9.34.2.8 dwarf_get_real_section_name()	235
9.34.2.9 dwarf_get_section_count()	235
9.34.2.10 dwarf_get_section_info_by_index()	236
9.34.2.11 dwarf_get_section_info_by_index_a()	236
9.34.2.12 dwarf_get_section_info_by_name()	237
9.34.2.13 dwarf_get_section_info_by_name_a()	237
9.34.2.14 dwarf_get_section_max_offsets_d()	238
9.34.2.15 dwarf_machine_architecture()	239
9.34.2.16 dwarf_machine_architecture_a()	239
9.35 Section Groups Objectfile Data	240
9.35.1 Detailed Description	241
9.35.2 Function Documentation	241
9.35.2.1 dwarf_sec_group_map()	241
9.35.2.2 dwarf_sec_group_sizes()	241
9.36 LEB Encode and Decode	242
9.36.1 Detailed Description	242
9.37 Miscellaneous Functions	243
9.37.1 Detailed Description	243
9.37.2 Function Documentation	243
9.37.2.1 dwarf_get_universalbinary_count()	243
9.37.2.2 dwarf_library_allow_dup_attr()	244
9.37.2.3 dwarf_package_version()	244
9.37.2.4 dwarf_record_cmdline_options()	245
9.37.2.5 dwarf_set_de_alloc_flag()	245
9.37.2.6 dwarf_set_default_address_size()	245

9.37.2.7 dwarf_set_reloc_application()	246
9.37.2.8 dwarf_set_stringcheck()	246
9.37.3 Variable Documentation	247
9.37.3.1 dwarf_get_endian_copy_function	247
9.38 Determine Object Type of a File	247
9.38.1 Detailed Description	247
9.39 Section allocation: malloc or mmap	248
9.39.1 Detailed Description	248
9.39.2 Function Documentation	248
9.39.2.1 dwarf_get_mmap_count()	248
9.39.2.2 dwarf_set_load_preference()	249
9.40 Using dwarf_init_path()	249
9.41 Using dwarf_init_path_dl()	250
9.42 Using dwarf_attrlist()	251
9.43 Attaching a tied dbg	252
9.44 Detaching a tied dbg	253
9.45 Examining Section Group data	253
9.46 Using dwarf_siblingof_c()	254
9.47 Using dwarf_siblingof_b()	254
9.48 Using dwarf_child()	255
9.49 using dwarf_validate_die_sibling	255
9.50 Example walking CUs(e)	257
9.51 Example walking CUs(d)	258
9.52 Using dwarf_offdie_b()	260
9.53 Using dwarf_offset_given_die()	261
9.54 Using dwarf_attrlist()	261
9.55 Using dwarf_offset_list()	261
9.56 Documenting Form_Block	262
9.57 Using dwarf_discr_list()	263
9.58 Location/expression access	264
9.59 Reading a location expression	266
9.60 Using dwarf_srclines_b()	267
9.61 Using dwarf_srclines_b() and linecontext	269
9.62 Using dwarf_srcfiles()	269
9.63 Using dwarf_get_globals()	270
9.64 Using dwarf_globals_by_type()	271
9.65 Reading .debug_weaknames (nonstandard)	271
9.66 Reading .debug_funcnames (nonstandard)	271
9.67 Reading .debug_types (nonstandard)	272
9.68 Reading .debug_varnames data (nonstandard)	272
9.69 Reading .debug_names data	273
9.70 Reading .debug_macro data (DWARF5)	275

9.71 Reading .debug_machinfo (DWARF2-4)	278
9.72 Extracting fde, cie lists.	278
9.73 Reading the .eh_frame section	279
9.74 Using dwarf_expand_frame_instructions	279
9.75 Reading string offsets section data	280
9.76 Reading an aranges section	281
9.77 Example getting .debug_ranges data	282
9.78 Reading gdbindex data	284
9.79 Reading gdbindex addressarea	285
9.80 Reading the gdbindex symbol table	285
9.81 Reading cu and tu Debug Fission data	286
9.82 Reading Split Dwarf (Debug Fission) hash slots	287
9.83 Reading high pc from a DIE.	287
9.84 Reading Split Dwarf (Debug Fission) data	288
9.85 Retrieving tag,attribute,etc names	288
9.86 Using GNU debuglink data	289
9.87 Accessing accessing raw rnglist	290
9.88 Accessing rnglists section	291
9.89 Demonstrating reading DWARF without a file.	292
9.90 A simple report on section groups.	297
10 Class Documentation	301
10.1 Dwarf_Block_s Struct Reference	301
10.2 Dwarf_Cmdline_Options_s Struct Reference	301
10.2.1 Detailed Description	301
10.3 Dwarf_Debug_Fission_Per_CU_s Struct Reference	302
10.4 Dwarf_Form_Data16_s Struct Reference	302
10.5 Dwarf_Macro_Details_s Struct Reference	302
10.5.1 Detailed Description	303
10.6 Dwarf_Obj_Access_Interface_a_s Struct Reference	303
10.7 Dwarf_Obj_Access_Methods_a_s Struct Reference	303
10.7.1 Detailed Description	304
10.8 Dwarf_Obj_Access_Section_a_s Struct Reference	304
10.9 Dwarf_Printf_Callback_Info_s Struct Reference	305
10.9.1 Detailed Description	305
10.10 Dwarf_Ranges_s Struct Reference	305
10.11 Dwarf_Regtable3_s Struct Reference	306
10.12 Dwarf_Regtable_Entry3_s Struct Reference	306
10.13 Dwarf_Sig8_s Struct Reference	307
11 File Documentation	309
12 checkexamples.c	311

12.1 /home/davea/dwarf/code/src/bin/dwarfexample/jitreader.c File Reference	311
12.2 /home/davea/dwarf/code/src/bin/dwarfexample/showsectiongroups.c File Reference	311
13 dwarf.h	313
13.1 dwarf.h	313
14 libdwarf.h	333
14.1 libdwarf.h	333
Index	369

Chapter 1

A Consumer Library Interface to DWARF

1.1 Suggestions for improvement are welcome.	2
1.2 Downloading Libdwarf	2
1.3 Introduction	2
1.4 Thread Safety	3
1.5 Error Handling in libdwarf	3
1.5.1 Error Handling at Initialization	3
1.5.2 Error Handling Everywhere	4
1.6 Extracting Data Per Compilation Unit	6
1.7 Line Table Registers	6
1.8 Reading Special Sections Independently	7
1.9 Special Frame Registers	7
1.10 .debug_pubnames etc DWARF2-DWARF4	8
1.11 Reading DWARF with no object file present	9
1.12 Section Groups: Split Dwarf, COMDAT groups	11
1.13 Details on separate DWARF object access	12
1.14 Linking against libdwarf.so (or dll or dylib)	13
1.15 Linking against libdwarf.a	14
1.16 Suppressing CRC calculation for debuglink	14
1.17 Object Reading By User Code	15
1.18 dwsec_mmap	15
1.19 Recent Changes	16

Author

David Anderson

Copyright

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Date

2026-02-01 v2.3.0

1.1 Suggestions for improvement are welcome.

Your thoughts on the document?

A) Are the section and subsection titles on Main Page meaningful to you?

B) Are the titles on the Modules page meaningful to you?

Anything else you find misleading or confusing? Send suggestions to (libdwarf (at) linuxmail with final characters .org). Sorry about the simple obfuscation to keep bots away.

Thanks in advance for any suggestions.

1.2 Downloading Libdwarf

Project page is at <https://github.com/davea42/libdwarf-code>

There is a Releases area on the project page, click Latest and you will be presented with options to download the source in three different forms.

For details on licensing, see COPYING in the files list.

README.md may be of interest (automatically shown on the project page on github..

Examples of using libdwarf are in [doc/checkexamples.c](#) and `src/bin/dwarfexamples`.

To download source, one can also do:

git clone <https://github.com/davea42/libdwarf-code> **code**

Some tests simply assume the project source base name is **code** which is why the above is as shown. This is a grave historical misfeature that needs to be fixed in the test scripts.

1.3 Introduction

This document describes an interface to *libdwarf*, a library of functions to provide access to DWARF debugging information records, DWARF line number information, DWARF address range and global names information, weak names information, DWARF frame description information, DWARF static function names, DWARF static variables, and DWARF type information. In addition the library provides access to several object sections (created by compiler writers and for debuggers) related to debugging but not mentioned in any DWARF standard.

The DWARF Standard has long mentioned the "Unix International Programming Languages Special Interest Group" (PLSIG), under whose auspices the DWARF committee was formed around 1991. "Unix International" was disbanded in the 1990s and no longer exists.

The DWARF committee published DWARF2 July 27, 1993, DWARF3 in 2005, DWARF4 in 2010, and DWARF5 in 2017.

In the mid 1990s this document and the library it describes (which the committee never endorsed, having decided not to endorse or approve any particular library interface) was made available on the internet by Silicon Graphics, Inc.

In 2005 the DWARF committee began an affiliation with FreeStandards.org. In 2007 FreeStandards.org merged with The Linux Foundation. The DWARF committee dropped its affiliation with FreeStandards.org in 2007 and established the dwarfstd.org website.

See also

<https://www.dwarfstd.org> for current information on standardization activities and a copy of the standard.

1.4 Thread Safety

Libdwarf can safely open multiple Dwarf_Debug pointers simultaneously but all such Dwarf_Debug pointers must be opened within the same thread. And all *libdwarf* calls must be made from within that single (same) thread.

1.5 Error Handling in libdwarf

Essentially every *libdwarf* call could involve dealing with an error (possibly data corruption in the object file). Here we explain the two main approaches the library provides (though we think only one of them is truly appropriate except in toy programs). In all cases where the library returns an error code (almost every library function does) the caller should check whether the returned integer is DW_DLV_OK, DW_DLV_ERROR, or DW_DLV_NO_ENTRY and then act accordingly.

A) The recommended approach is to define a Dwarf_Error and initialize it to 0.

```
Dwarf_Error error = 0;
```

Then, in every call where there is a Dwarf_Error argument pass its address. For example:

```
int res = dwarf_tag(die,DW_TAG_compile_unit,&error);
```

The possible return values to res are, in general:

```
DW_DLV_OK
DW_DLV_NO_ENTRY
DW_DLV_ERROR
```

If **DW_DLV_ERROR** is returned then error is set (by the library) to a pointer to important details about the error and the library will not pass back any data through other pointer arguments. If **DW_DLV_NO_ENTRY** is returned the error argument is ignored by the library and the library will not pass back any data through pointer arguments. If **DW_DLV_OK** is returned argument pointers that are defined as ways to return data to your code are used and values are set in your data by the library.

Some functions cannot possibly return some of these three values. As defined later for each function.

B) An alternative (not recommended) approach is to pass NULL to the error argument.

```
int res = dwarf_tag(die,DW_TAG_compile_unit,NULL);
```

If your initialization provided an 'errhand' function pointer argument (see below) the library will call errhand if an error is encountered. (Your errhand function could exit if you so choose.)

The the library will then return DW_DLV_ERROR, though you will have no way to identify what the error was. Could be a malloc fail or data corruption or an invalid argument to the call, or something else.

That is the whole picture. The library never calls exit() under any circumstances.

1.5.1 Error Handling at Initialization

Each initialization call (for example)

```
Dwarf_Debug dbg = 0;
const char *path = "myobjectfile";
char *true_path = 0;
unsigned int true_pathlen = 0;
Dwarf_Handler errhand = 0;
Dwarf_Ptr errarg = 0;
Dwarf_Error error = 0;
int res = 0;

res = dwarf_init_path(path,true_path,true_pathlen,
    DW_GROUPNUMBER_ANY,errhand,errarg,&dbg,&error);
```

has two arguments that appear nowhere else in the library.

```
Dwarf_Handler errhand
Dwarf_Ptr      errarg
```

For the **recommended A)** approach:

Just pass NULL to both those arguments. If the initialization call returns DW_DLV_ERROR you should then call `dwarf_dealloc_error` (dbg, error);

to free the Dwarf_Error data because `dwarf_finish()` does not clean up a dwarf-init error. This works even though `dbg` will be NULL.

For the **not recommended B)** approach:

Because `dw_errarg` is a general pointer one could create a struct with data of interest and use a pointer to the struct as the `dw_errarg`. Or one could use an integer or NULL, it just depends what you want to do in the Dwarf_Handler function you write.

If you wish to provide a `dw_errhand`, define a function (this first example is not a good choice as it terminates the application!).

```
void bad_dw_errhandler(Dwarf_Error error, Dwarf_Ptr ptr)
{
    printf("ERROR Exit on %lx due to error 0x%lx %s\n",
        (unsigned long)ptr,
        (unsigned long)dwarf_errno(error),
        dwarf_errmsg(error));
    exit(1)
}
```

and pass `bad_dw_errhandler` (as a function pointer, no parentheses).

The Dwarf_Ptr argument your error handler function receives is the value you passed in as `dw_errarg`, and can be anything, it allows you to associate the callback with a particular `dwarf_init*` call if you wish to make such an association.

By doing an `exit()` you guarantee that your application abruptly stops. This is only acceptable in toy or practice programs.

A better `dw_errhand` function is

```
void my_dw_errhandler(Dwarf_Error error, Dwarf_Ptr ptr)
{
    /* Clearly one could write to a log file or do
       whatever the application finds useful. */
    printf("ERROR on %lx due to error 0x%lx %s\n",
        (unsigned long)ptr,
        (unsigned long)dwarf_errno(error),
        dwarf_errmsg(error));
}
```

because it returns rather than exiting. It is not ideal. The DW_DLV_ERROR code is returned from *libdwarf* and your code can do what it likes with the error situation. The library will continue from the error and will return an error code on returning to your `@libdwarf` call ... but the calling function will not know what the error was.

```
Dwarf_Ptr x = address of some struct I want in the errhandler;
res = dwarf_init_path(..., my_dw_errhandler, x, ... );
if (res == ...)
```

If you do not wish to provide a `dw_errhand`, just pass both arguments as NULL.

1.5.2 Error Handling Everywhere

So let us examine a simple case where anything could happen. We are taking the **recommended A)** method of using a non-null Dwarf_Error*:

```
int func(Dwarf_Dbg dbg, Dwarf_Die die, Dwarf_Error* error) {
    Dwarf_Die newdie = 0;
    int res = 0;

    res = dwarf_siblingof_c(die, &newdie, error);
    if (res != DW_DLV_OK) {
        /* Whether DW_DLV_ERROR or DW_DLV_NO_ENTRY
           (the latter is actually impossible
           for this function) returning res is the
           appropriate default thing to do. */
        return res;
    }
    /* Do something with newdie. */
    dwarf_dealloc_die(newdie);
    newdie = 0; /* A good habit... */
    return DW_DLV_OK;
}
```

1.5.2.1 DW_DLV_OK

When `res == DW_DLV_OK` `newdie` is a valid pointer and when appropriate we should do `dwarf_dealloc_die(newdie)`. For other *libdwarf* calls the meaning depends on the function called, so read the description of the function you called for more information.

1.5.2.2 DW_DLV_NO_ENTRY

When `res == DW_DLV_NO_ENTRY` then `newdie` is not set and there is no error. It means `die` was the last of a siblinglist. For other *libdwarf* calls the meaning depends on the function called, so read the description of the function you called for more information.

1.5.2.3 DW_DLV_ERROR

When `res == DW_DLV_ERROR` Something bad happened. The only way to know what happened is to examine the `*error` as in

```
int ev = dwarf_errno(*error);
or
char * msg = dwarf_errmsg(*error);
```

or both and report that somehow.

The above three values are the only returns possible from the great majority of *libdwarf* functions, and for these functions the return type is always `int`.

If it is a decently large or long-running program then you want to free any local memory you allocated and return `res`. If it is a small or experimental program print something and `exit` (possibly leaking memory).

If you want to discard the error report from the `dwarf_siblingof_c()` call then possibly do

```
dwarf_dealloc_error(dbg, *error);
*error = 0;
return DW_DLV_OK;
```

Except in a special case involving function `dwarf_set_de_alloc_flag()` (which you will not usually call), any `dwarf_dealloc()` that is needed will happen automatically when you call `dwarf_finish()`.

1.5.2.4 Slight Performance Enhancement

Very long running library access programs using relevant appropriate `dwarf_dealloc` calls should consider calling `dwarf_set_de_alloc_flag(0)`. Using this one could get a performance enhancement of perhaps five percent in *libdwarf* CPU time and a reduction in memory use.

Be sure to test using `valgrind` (at runtime) or the `gcc` option `-fsanitize` at compile time and run tests appropriately to ensure your code really does the extra `dwarf_dealloc` calls needed since when using `dwarf_set_de_alloc_flag(0)` `dwarf_finish()` does only limited cleanup.

1.6 Extracting Data Per Compilation Unit

The library is designed to run a single pass through the set of Compilation Units (CUs), via a sequence of calls to [dwarf_next_cu_header_e\(\)](#). ([dwarf_next_cu_header_d\(\)](#) is supported but its use requires that it be immediately followed by a call to [dwarf_siblingof_b\(\)](#). see [dwarf_next_cu_header_d\(\)](#).)

Within a CU opened with [dwarf_next_cu_header_e\(\)](#) do something (if desired) on the CU_DIE returned, and call [dwarf_child\(\)](#) on the CU_DIE to begin recursing through all DIEs. If you save the CU_DIE you can repeat passes beginning with [dwarf_child\(\)](#) on the CU_DIE, though it almost certainly faster to remember, in your data structures, what you need from the first pass.

The general plan:

create your local data structure(s)

- A. Check your local data structures to see **if** you have what you need
- B. If sufficient data present act on it, ensuring your data structures are kept **for** further use.
- C. Otherwise Read a CU, recording relevant data in your structures and loop back to A.

For an example (best approach)

See also

[Example walking CUs\(e\)](#) or (second-best approach)

[Example walking CUs\(d\)](#) Write your code to record relevant (to you) information from each CU as you go so your code has no need for a second pass through the CUs. This is much much faster than allowing multiple passes would be.

1.7 Line Table Registers

Line Table Registers

Please refer to the DWARF5 Standard for details. The line table registers are named in Section 6.2.2 State Machine Registers and are not much changed from DWARF2.

Certain functions on Dwarf_Line data return values for these 'registers' as these are the data available for debuggers and other tools to relate a code address to a source file name and possibly also to a line number and column-number within the source file.

```
address
op_index
file
line
column
is_stmt
basic_block
end_sequence
prologue_end
epilogue_begin
isa
discriminator
```


1.8 Reading Special Sections Independently

DWARF defines (in each version of DWARF) sections which have a somewhat special character. These are referenced from compilation units and other places and the Standard does not forbid blocks of random bytes at the start or end or between the areas referenced from elsewhere.

Sometimes compilers (or linkers) leave trash behind as a result of optimizations. If there is a lot of space wasted that way it is a quality of implementation issue. But usually the wasted space, if any, is small.

Compiler writers or others may be interested in looking at these sections independently so *libdwarf* provides functions that allow reading the sections without reference to what references them.

[Abbreviations can be read independently](#)

[Strings can be read independently](#)

[String Offsets can be read independently](#)

[The addr table can be read independently](#)

Those functions allow starting at byte 0 of the section and provide a length so you can calculate the next section offset to call or refer to.

Usually that works fine. If there is some random data somewhere outside of referenced areas or the data format is a gcc extension of an early DWARF version the reader function may fail, returning DW_DLV_ERROR. Such an error is neither a compiler bug nor a *libdwarf* bug.

1.9 Special Frame Registers

In dealing with `.debug_frame` or `.eh_frame` there are five values that must be set unless one has relatively few registers in the target ABI (anything under 188 registers, see [dwarf.h](#) DW_FRAME_LAST_REG_NUM for this default).

The requirements stem from the design of the section. See the DWARF5 Standard for details. The `.debug_frame` section is basically the same from DWARF2 on. The `.eh_frame` section is similar to `.debug_frame` but is intended to support exception handling and has fields and data not present in `.debug_frame`.

Keep in mind that register values correspond to columns in the theoretical fully complete line table of a row per pc and a column per register.

There is no time or space penalty in setting **Undefined_Value**, **Same_Value**, and **CFA_Column** much larger than the **Table_Size**.

Here are the five values.

Table_Size: This sets the number of columns in the theoretical table. It starts at DW_FRAME_LAST_REG_NUM which defaults to 188. This is the only value you might need to change, given the defaults of the others are set reasonably large by default. Setting this higher than necessary wastes cpu time and memory space.

Undefined_Value: A register number that means the register value is undefined. For example due to a call clobbering the register. DW_FRAME_UNDEFINED_VAL defaults to 12288. There no such column in the table.

Same_Value: A register number that means the register value is the same as the value at the call. Nothing can have clobbered it. DW_FRAME_SAME_VAL defaults to 12289. There no such column in the table.

Initial_Value: The value must be either DW_FRAME_UNDEFINED_VAL or DW_FRAME_SAME_VAL to represent how most registers are to be thought of at a function call. This is a property of the ABI and instruction set. Specific frame instructions in the CIE or FDE will override this for registers not matching this value.

CFA_Column: A number for the CFA. Defined so we can use a register number to refer to it. DW_FRAME_CFA_↵_COL defaults to 12290. There no such column in the table. See [libdwarf.h](#) struct [Dwarf_Regtable3_s](#) member [rt3_cfa_rule](#) or function [dwarf_get_fde_info_for_cfa_reg3_b\(\)](#) or function [dwarf_get_fde_info_for_cfa_reg3_c\(\)](#) .

A set of functions allow these to be changed at runtime. The set should be called (if needed) immediately after initializing a Dwarf_Debug and before any other calls on that Dwarf_Debug. If just one value (for example, Table_↵_Size) needs altering, then just call that single function.

For the library accessing frame data to work properly there are certain invariants that must be true once the set of functions have been called.

REQUIRED:

```
Table_Size      > the number of registers in the ABI.
Undefined_Value != Same_Value
CFA_Column      != Undefined_value
CFA_Column      != Same_value
Initial_Value   == Same_Value ||
                (Initial_Value == Undefined_value)
Undefined_Value > Table_Size
Same_Value      > Table_Size
CFA_Column      > Table_Size
```

If these conditions are not honored the library will return a DW_DLV_ERROR code rather than use an improper set of values.

1.10 .debug_pubnames etc DWARF2-DWARF4

Each section consists of a header for a specific compilation unit (CU) followed by an a set of tuples, each tuple consisting of an offset of a compilation unit followed by a null-terminated namestring. The tuple set is ended by a 0,0 pair. Then followed with the data for the next CU and so on.

The function set provided for each such section allows one to print all the section data as it literally appears in the section (with headers and tuples) or to treat it as a single array with CU data columns.

Each has a set of 6 functions.

Section	typename	Standard
.debug_pubnames	Dwarf_Global	DWARF2-DWARF4
.debug_pubtypes	Dwarf_Global	DWARF3,DWARF4

These sections are accessed calling [dwarf_globals_by_type\(\)](#) using type of DW_GL_GLOBALS or DW_GL_↵PUBTYPES. Or call [dwarf_get_pubtypes\(\)](#).

The following four were defined in SGI/IRIX compilers in the 1990s but were never part of the DWARF standard. These sections are accessed calling [dwarf_globals_by_type\(\)](#) using type of DW_GL_FUNCS,DW_GL_↵TYPES,DW_GL_VARS, or DW_GL_WEAKS.

It not likely you will encounter these four sections.

```
.debug_funcs
.debug_typenames
.debug_vars
.debug_weaks
```

1.11 Reading DWARF with no object file present

This most commonly happens with just-in-time compilation, and someone working on the code wants to debug this on-the-fly code in a situation where nothing can be written to disc, but DWARF can be constructed in memory.

For a simple example of this

See also

[Demonstrating reading DWARF without a file.](#)

But the *libdwarf* feature can be used in a wide variety of ways.

For example, the DWARF data could be kept in simple files of bytes on the internet. Or on the local net. Or if files can be written locally each section could be kept in a simple stream of bytes in the local file system.

Another example is a non-standard file system, or file format, with the intent of obfuscating the file or the DWARF.

For this to work the code generator must generate standard DWARF.

Overall the idea is a simple one: You write a small handful of functions and supply function pointers and code implementing the functions. These are part of your application or library, not part of *libdwarf*.

You set up a little bit of data with that code (all described below) and then you have essentially written the dwarf↵_init_path equivalent and you can access compilation units, line tables etc and the standard *libdwarf* function calls work.

Take great care in your object reader code because any failure to reject object files with corrupt data in areas such as object headers, section headers, or relocation records nullifies the intended guarantee that the library will not crash.

Data you need to create involves these types. What follows describes how to fill them in and how to make them work for you.

```
typedef struct Dwarf_Obj_Access_Interface_a_s
Dwarf_Obj_Access_Interface_a;
struct Dwarf_Obj_Access_Interface_a_s {
    void*          ai_object;
    const Dwarf_Obj_Access_Methods_a *ai_methods;
};

typedef struct Dwarf_Obj_Access_Methods_a_s
Dwarf_Obj_Access_Methods_a
struct Dwarf_Obj_Access_Methods_a_s {
    int            (*om_get_section_info)(void* obj,
        Dwarf_Unsigned section_index,
        Dwarf_Obj_Access_Section_a* return_section,
        int* error);
    Dwarf_Small    (*om_get_byte_order)(void* obj);
    Dwarf_Small    (*om_get_length_size)(void* obj);
    Dwarf_Small    (*om_get_pointer_size)(void* obj);
    Dwarf_Unsigned (*om_get_filesize)(void* obj);

    Dwarf_Unsigned (*om_get_section_count)(void* obj);
    int            (*om_load_section)(void* obj,
        Dwarf_Unsigned section_index,
        Dwarf_Small** return_data, int* error);
    int            (*om_relocate_a_section)(void* obj,
        Dwarf_Unsigned section_index,
        Dwarf_Debug dbg,
        int* error);
};

typedef struct Dwarf_Obj_Access_Section_a_s
Dwarf_Obj_Access_Section_a
struct Dwarf_Obj_Access_Section_a_s {
    const char*    as_name;
    Dwarf_Unsigned as_type;
    Dwarf_Unsigned as_flags;
    Dwarf_Addr     as_addr;
```

```

Dwarf_Unsigned as_offset;
Dwarf_Unsigned as_size;
Dwarf_Unsigned as_link;
Dwarf_Unsigned as_info;
Dwarf_Unsigned as_addralign;
Dwarf_Unsigned as_entsize;
};

```

Dwarf_Obj_Access_Section_a: Your implementation of a **om_get_section_info** must fill in a few fields for *libdw*. The fields here are standard Elf, but for most you can just use the value zero. We assume you will not be doing relocations at runtime and do not describe how to do relocations.

as_name: Here you set a section name via the pointer. The section names must be names as defined in the DWARF standard, so if such do not appear in your data you have to create the strings yourself.

as_type: Fill in zero.

as_flags: Fill in zero.

as_addr: Fill in the address, in local memory, where the bytes of the section are.

as_offset: Fill in zero.

as_size: Fill in the size, in bytes, of the section you are telling *libdw* about.

as_link: Fill in zero.

as_info: Fill in zero.

as_addralign: Fill in zero.

as_entsize: Fill in one(1).

Dwarf_Obj_Access_Methods_a_s: The functions we need to access object data from *libdw* are declared here.

In these function pointer declarations 'void *obj' is intended to be a pointer (the object field in Dwarf_Obj_Access_Interface_s) that hides the library-specific and object-specific data that makes it possible to handle multiple object formats and multiple libraries. It is not required that one handles multiple such in a single *libdw* archive/shared-library (but not ruled out either). See dwarf_elf_object_access_internals_t and dwarf_elf_access.c for an example.

Usually the struct **Dwarf_Obj_Access_Methods_a_s** is statically defined and the function pointers are set at compile time.

The om_get_filesize member is new September 4, 2021. Its position is NOT at the end of the list. The member names all now have om_ prefix.

1.12 Section Groups: Split Dwarf, COMDAT groups

A typical executable or shared object is unlikely to have any section groups as the linker would have removed those when building an executable or shared-object, and in that case what follows is irrelevant and unimportant.

COMDAT groups are defined by the Elf ABI and enable compilers and linkers to work together to eliminate blocks of duplicate DWARF and duplicate CODE.

Split Dwarf (sometimes referred to as Debug Fission) allows compilers and linkers to separate large amounts of DWARF from the executable, shrinking disk space needed in the executable while allowing full debugging (also applies to shared objects).

See the DWARF5 Standard, Section E.1 Using Compilation Units page 364.

To name COMDAT groups (defined later here) we add the following defines to [libdwf.h](#) (the DWARF standard does not specify how to do any of this).

```
/* These support opening DWARF5 split dwarf objects and
   Elf SHT_GROUP blocks of DWARF sections. */
#define DW_GROUPNUMBER_ANY 0
#define DW_GROUPNUMBER_BASE 1
#define DW_GROUPNUMBER_DWO 2
```

The DW_GROUPNUMBER_ are used in *libdwf* functions [dwarf_init_path\(\)](#), [dwarf_init_path_dl\(\)](#) and [dwarf_init_b\(\)](#). In all those cases unless you know there is any complexity in your object file, pass in DW_↵GROUPNUMBER_ANY.

To see section groups usage, see the example source:

See also

[A simple report on section groups.](#)
[Examining Section Group data](#)

The function interface declarations:

See also

[dwarf_sec_group_sizes](#)
[dwarf_sec_group_map](#)

If an object file has multiple groups *libdwf* will not reveal contents of more than the single requested group with a given [dwarf_init_path\(\)](#) call. One must pass in another groupnumber to another [dwarf_init_path\(\)](#), meaning initialize a new Dwarf_Debug, to get *libdwf* to access that group.

When opening a Dwarf_Debug the following applies:

If DW_GROUPNUMBER_ANY is passed in *libdwf* will choose either of DW_GROUPNUMBER_BASE(1) or DW_↵GROUPNUMBER_DWO (2) depending on the object content. If both groups one and two are in the object *libdwf* will chose DW_GROUPNUMBER_BASE.

If DW_GROUPNUMBER_BASE is passed in *libdwf* will choose it if non-split DWARF is in the object, else the init call will return DW_DLV_NO_ENTRY.

If DW_GROUPNUMBER_DWO is passed in *libdwf* will choose it if .dwo sections are in the object, else the init will call return DW_DLV_NO_ENTRY.

If a groupnumber greater than two is passed in *libdwf* accepts it, whether any sections corresponding to that groupnumber exist or not. If the groupnumber is not an actual group the init will call return DW_DLV_NO_ENTRY.

For information on groups "dwarfdump -i" on an object file will show all section group information **unless** the object file is a simple standard object with no .dwo sections and no COMDAT groups (in which case the output will be silent on groups). Look for **Section Groups data** in the dwarfdump output. The groups information will be appearing very early in the dwarfdump output.

Sections that are part of an Elf COMDAT GROUP are assigned a group number > 2 . There can be many such COMDAT groups in an object file (but none in an executable or shared object). Each such COMDAT group will have a small set of sections in it and each section in such a group will be assigned the same group number by *libdwarf*.

Sections that are in a .dwp .dwo object file are assigned to DW_GROUPNUMBER_DWO,

Sections not part of a .dwp package file or a.dwo section, or a COMDAT group are assigned DW_GROUPNUMBER_BASE.

At least one compiler relies on relocations to identify COMDAT groups, but the compiler authors do not publicly document how this works so we ignore such (these COMDAT groups will result in *libdwarf* returning DW_DLV_ERROR).

Popular compilers and tools are using such sections. There is no detailed documentation that we can find (so far) on how the COMDAT section groups are used, so *libdwarf* is based on observations of what compilers generate.

1.13 Details on separate DWARF object access

There are, at present, three distinct approaches in use to put DWARF information into separate objects to significantly shrink the size of the executable. All of them involve identifying a separate file.

Split Dwarf is one method. It defines the attribute **DW_AT_dwo_name** (if present) as having a file-system appropriate name of the split object with most of the DWARF.

The second is MacOS dSYM. It is a convention of placing the DWARF-containing object (separate from the object containing code) in a specific subdirectory tree.

The third involves GNU debuglink and GNU debug_id. These are two distinct ways (outside of DWARF) to provide names of alternative DWARF-containing objects elsewhere in a file system.

If one initializes a Dwarf_Debug object with `dwarf_init_path()` or `dwarf_init_path_dl()` appropriately *libdwarf* will automatically open the alternate dSYM or debuglink/debug_id object on the object with most of the DWARF.

See also

<https://sourceware.org/gdb/onlinedocs/gdb/Separate-Debug-Files.html>

libdwarf provides means to automatically read the alternate object (in place of the one named in the init call) or to suppress that and read the named object file.

```
int dwarf_init_path(const char * dw_path,
char * dw_true_path_out_buffer,
unsigned int dw_true_path_bufferlen,
unsigned int dw_groupnumber,
Dwarf_Handler dw_errhand,
Dwarf_Ptr dw_errarg,
Dwarf_Debug* dw_dbg,
Dwarf_Error* dw_error);

int dwarf_init_path_dl(const char *dw_path,
char * true_path_out_buffer,
unsigned true_path_bufferlen,
unsigned groupnumber,
Dwarf_Handler errhand,
Dwarf_Ptr errarg,
Dwarf_Debug * ret_dbg,
char ** dl_path_array,
```

```
unsigned int    dl_path_count,
unsigned char   * path_source,
Dwarf_Error    * error);
```

Case 1:

If *dw_true_path_out_buffer* or *dw_true_path_bufferlen* is passed in as zero then the library will not look for an alternative object.

Case 2:

If *dw_true_path_out_buffer* passes a pointer to space you provide and *dw_true_path_bufferlen* passes in the length, in bytes, of the buffer, *libdwarf* will look for alternate DWARF-containing objects. We advise that the caller zero all the bytes in *dw_true_path_out_buffer* before calling.

If the alternate object name (with its null-terminator) is too long to fit in the buffer the call will return DW_DLV_ERROR with *dw_error* providing error code DW_DLE_PATH_SIZE_TOO_SMALL.

If the alternate object name fits in the buffer *libdwarf* will open and use that alternate file in the returned Dwarf_Dbg.

It is up to callers to notice that *dw_true_path_out_buffer* now contains a string and callers will probably wish to do something with the string.

If the initial byte of *dw_true_path_out_buffer* is a non-null when the call returns then an alternative object was found and opened.

The second function, *dwarf_init_path_dl()*, is the same as *dwarf_init_path()* except the *_dl* version has three additional arguments, as follows:

Pass in NULL or *dw_dl_path_array*, an array of pointers to strings with alternate GNU debuglink paths you want searched. For most people, passing in NULL suffices.

Pass in *dw_dl_path_array_size*, the number of elements in *dw_dl_path_array*.

Pass in *dw_dl_path_source* as NULL or a pointer to char. If non-null *libdwarf* will set it to one of three values:

- DW_PATHSOURCE_basic which means the original input *dw_path* is the one opened in *dw_dbg*.
- DW_PATHSOURCE_dsym which means a MacOS dSYM object was found and is the one opened in *dw_dbg*. *dw_true_path_out_buffer* contains the dSYM object path.
- DW_PATHSOURCE_debuglink which means a GNU debuglink or GNU debug-id path was found and names the one opened in *dw_dbg*. *dw_true_path_out_buffer* contains the object path.

1.14 Linking against libdwarf.so (or dll or dylib)

If you wish to do the basic *libdwarf* tests and are linking against a shared library *libdwarf* you must do an install for the tests to succeed (in some environments it is not strictly necessary).

For example, if building with configure, do

```
make
make install
make check
```

You can install anywhere, there is no need to install in a system directory! Creating a temporary directory and installing there suffices. If installed in appropriate system directories that works too.

When compiling to link against a shared library *libdwarf* you **must not define** *LIBDWARF_STATIC*.

For examples of this for all three build systems read the project shell script

```
scripts/allsimplebuilds.sh
```

1.15 Linking against libdwarf.a

- If you are building an application
- And are linking your application against a static library libdwarf.a
- Then you must ensure that each source file compilation with an include of [libdwarf.h](#) has the macro **LIBDWARF_STATIC** defined to your source compilation.
- If *libdwarf* was built with zlib and zstd decompression library enabled you must add `-lz -lzstd` to the link line of the build of your application.

To pass **LIBDWARF_STATIC** to the preprocessor with Visual Studio:

- Right click on a project name
- In the contextual menu, click on **Properties** at the very bottom.
- In the new window, double click on **C/C++**
- On the right, click on **Preprocessor definitions**
- There is a small down arrow on the right, click on it then click on **Modify**
- Add **LIBDWARF_STATIC** to the values
- Click on **OK** to close the windows

1.16 Suppressing CRC calculation for debuglink

GNU Debuglink-specific issue:

If GNU debuglink is present and considered by [dwarf_init_path\(\)](#) or [dwarf_init_path_dl\(\)](#) the library may be required to compute a 32bit crc (Cyclic Redundancy Check) on the file found via GNU debuglink.

See also

https://en.wikipedia.org/wiki/Cyclic_redundancy_check

For people doing repeated builds of objects using such the crc check is a waste of time as they know the crc comparison will pass.

For such situations a special interface function lets the [dwarf_init_path\(\)](#) or [dwarf_init_path_dl\(\)](#) caller suppress the crc check without having any effect on anything else in *libdwarf*.

It might be used as follows (the same pattern applies to [dwarf_init_path_dl\(\)](#)) for any program that might do multiple [dwarf_init_path\(\)](#) or [dwarf_init_path_dl\(\)](#) calls in a single program execution.

```
int res = 0;
int crc_check = 0;

crc_check = dwarf_suppress_debuglink_crc(1);
res = dwarf_init_path(..usual arguments);
/* Reset the crc flag to previous value. */
dwarf_suppress_debuglink_crc(crc_check);
/* Now check res in the usual way. */
```

This pattern ensures the crc check is suppressed for this single [dwarf_init_path\(\)](#) or [dwarf_init_path_dl\(\)](#) call while leaving the setting unchanged for further [dwarf_init_path\(\)](#) or [dwarf_init_path_dl\(\)](#) calls in the running program.

1.17 Object Reading By User Code

With the usual libdwf initialization functions the library does extensive checks of object formats and attempts to find all object errors and returns error codes reading objects with errors affecting the library. Including, for example, object header errors and relocation section errors. The intent is to return correct output for correct DWARF and avoid program crashes in the library when the object or DWARF data is corrupt (returning error codes as appropriate).

`dwarf_object_init_b()` is an alternative to the usual libdwf initialization functions – making it possible to read object formats libdwf knows nothing about. This interface delegates the object reading to user code.

`dwarf_object_init_b()` requires the user code to provide correct and complete section information for the sections the library requires. The library cannot guarantee correct, successful, crash-free operation if the user code reading an object is insufficiently careful, thorough, and consistent in returning `DW_DLV_OK`, `DW_DLV_NO_ENTRY`, and `DW_DLV_ERROR` as applicable.

1.18 dwsec_mmap

As of version 0.12.0 libdwf allows callers to select mmap (instead of malloc/read) to access object section DWARF data. Even if mmap is selected it is possible libdwf will chose to use malloc in specific cases.

If at library build time the required functions/header are not available the following will have no effect.

One way to select mmap is to call

```
dwarf_set_load_preference(Dwarf_Alloc_Mmap);
```

Another way to select mmap is with an environment variable

```
export DWARF_WHICH_ALLOC=mmap
```

so libdwf will see the variable at runtime.

The environment variable overrides the function call.

Calling `dwarf_set_load_preference(0)` will return the current overall preference will return the current overall preference, an instance of

```
enum Dwarf_Sec_Alloc_Pref
```

The new function

```
dwarf_get_mmap_count(Dwarf_Debug dw_dbg)
```

returns the application count and size of allocations for DWARF sections from the open `Dwarf_Debug` pointer.

Each supported build environment has a new build option to prevent libdwf from assuming that things in the build are always present.

1.19 Recent Changes

We list these with newest first.

Changes 2.2.0 to 2.3.0

Released 1 February 2026

Added function [dwarf_iterate_fde_all_regs3\(\)](#) which lets callers get all rows of the `.debug_frame` and `.eh_frame` sections via callbacks the function makes to a function you write. Simple to use and, for FDEs with many rows, sixty times faster. A prerelease version failed to allow retrieving the CFA pseudo-register value, but that is fixed. See `dwarfexample/frame2.c` for an example of use.

Added function [dwarf_set_harmless_errors_enabled\(Dwarf_Debug dw_dbg,int dw_v\)](#). Passing in `dw_v` of 0 (zero) turns off checking for what are really harmless errors, which makes a meaningful improvement in library performance for some calls. By default harmless errors are checked-for.

[dwarf_set_frame_rule_table_size\(\)](#) now allows setting register rules table size as low as 50 entries. The default remains arbitrary: `DW_FRAME_LAST_REG_NUM` (189) as that suffices for most architectures without being excessive.

Changes 2.1.0 to 2.2.0

Released 10 October 2025

Added functions [dwarf_lvn_name_direct\(\)](#) [dwarf_lvn_name\(\)](#) [dwarf_lvn_table_entry\(\)](#) enabling access to all the fields relevant in DWARF6 `DW_AT_language_version` attributes.

In builds using (for example) `cc -std=c99 gcc` will turn off visibility of `strdup()` in `string.h` leading to a build failure. So now we define `_GNU_SOURCE` in builds.

Corrected a bug in reading line table data that used `DW_FORM_strx` (and other `strx` forms).

Fixed various failures to handle corrupted (fuzzed) Apple Mach-o object files.

Changes 2.0.0 to 2.1.0

Released 20 July 2025

Corrected (and tested) use of DWARF6 attributes `DW_AT_language_name` and `DW_AT_language_version`. As of July 2025 we are not aware of a released compiler providing these attributes.

Added function [dwarf_srclangName\(\)](#) so that `DW_AT_language_name` attribute values can be accessed. Added [dwarf_language_version_data\(\)](#) because [dwarf_language_version_string\(\)](#) is not an appropriate function name here. The old name still exists and works.

Added [dwarf_srclangName_version\(\)](#) so that the data provided in DWARF6 `DW_AT_language_version` can be returned.

Fixed minor warnings from a compiler (`dwarfgn`) and from `meson`. No change to output.

Removed heuristic checks for decompress reasonableness as such proved to be ... unreasonable in certain real object files..

Corrected the `cmake` build of shared-library `libdwarf/CMakeLists.txt`

Given an unusual object using debuglink but with no sections with names starting with `.debug_` or `_eh_frame`, libdwarf would complain about not having any DWARF sections and ignore the debuglink data. See github issue 297 for details of the fix.

Changes 0.12.0 to 2.0.0

Released 20 May 2025.

Skipping all versions 1.x.x because before libdwarf used Semantic Versioning gcc built libdwarf.so.1.0.0 .

Fixed a longstanding bug in `configure.ac` which began to cause builds to fail with recent `autoconf`.

Fixed a problem in `test/CmakeLists.txt` that caused current builds to fail on Msys2 Mingw64. Had been working for many months.

Updated the error report (for `zlib`, `zstd`) when decompression exceeds a heuristic. Now reports the compressed-len and the uncompressed-len. Increased the heuristic multiple allowed from 16 to 32.

Changes 0.11.1 to 0.12.0

Released 02 April 2025

To optionally support `mmap/munmap` of object files sections we read we have added a function prototype for struct [Dwarf_Obj_Access_Methods_a_s](#) function `om_load_section()`. This will help when reading multi gigabyte object files. And we added a function prototype for destructing the object specific data while removing library internal public functions.

If an application does not call any of the functions which are new in v0.12.0 then it will work without recompilation.

Any application calling the new functions (for example, v0.12.0 `dwarfdump`) will only work with a v0.12.0 libdwarf.

If one is calling [dwarf_object_init_b\(\)](#) (almost no one ever calls this function) one is therefore instantiating struct [Dwarf_Obj_Access_Methods_a_s](#) oneself, you will surely find that your application will not work with libdwarf 0.12.0. Moreover, recompilation will fail unless you update your source to add the two new pointers to your instantiation (typically just add two zeros or NULLs in that struct instance).

Added new API function [dwarf_machine_architecture_a\(\)](#) which has an additional argument added to let `dwarfdump` create an better `.text` (etc) address-range for the object file being read for improved checking (fewer incorrect error reports) in `dwarfdump -k` output.

Up through December 2024 libdwarf could be made to be very very slow (Denial of Service) with calls with thousands of duplicate attributes in an abbreviation list of a specially constructed Compilation Unit.

Beginning 2025 by default that cannot happen as the library quickly notices and returns `DW_DLV_ERROR` with error details noted. Callers should check the return value and act appropriately, as always, when calling the library.

In case one has (and cannot fix) object files with duplicated attributes one can call a new API function↵: [dwarf_library_allow_dup_attr\(\)](#). The library defaults to false (0) meaning the checks are done in libdwarf by default. Pass non-zero value to allow duplicate attributes in a Debugging Information Entry through to callers.

Added the ability to select, at runtime, whether libdwarf will use `malloc` to load section content from an object file being read (previously the only option) or will use `mmap` instead.

If the build determines `mmap` is unavailable then `malloc` will be used.

Added API function [dwarf_set_load_preference\(\)](#) giving callers the option to choose the default section load functions. libdwarf now recognizes the environment variable `DWARF_WHICH_ALLOC` to select whether the library uses `mmap` or `malloc`/read to load object section data, and the environment variable values `'DWARF_WHICH_↵ALLOC=mmap'` or `'DWARF_WHICH_ALLOC=malloc'` are the only values recognized. A recognized environment

variable overrides [dwarf_set_load_preference\(\)](#) values. If the libdwarf build determines mmap is unavailable then only malloc will be used.

Added API function [dwarf_get_mmap_count\(\)](#) giving callers the ability to determine what section loads were used and the total amount of section data loaded.

Added API function [dwarf_get_LANGUAGE_name\(\)](#) to be able to easily get a string for DW_LNAME_Ada etc.

Added API function [dwarf_language_version_string\(\)](#). This returns information defined by DWARF 6 and useful in interpreting DWARF6 language-version strings based on a name accessed from DW_AT_language_name attribute.

Changes 0.11.0 to 0.11.1

Corrected handling of DWARF5 .debug_rnglists and .debug_loclists. No API change, no incompatibilities.

Changes 0.10.1 to 0.11.0

Added function [dwarf_get_ranges_baseaddress\(\)](#) to the api to allow dwarfdump and other library callers to easily derive the (cooked) address from the raw data in the DWARF2, DWARF3, DWARF4 .debug_ranges section. An example of use is in [doc/checkexamples.c](#) (see examplev).

Changes 0.9.2 to 0.10.1

Released 01 July 2024 (Release 0.10.0 was missing a CMakeLists.txt file and is withdrawn).

Added API function [dwarf_get_locdesc_entry_e\(\)](#) to allow dwarfdump to report some data from .debug_loclists more completely – it reports a byte length of each loclist item. This is of little interest to anyone, surely. [dwarf_get_locdesc_entry_d\(\)](#) is still what you should be using.

[dwarf_debug_addr_table\(\)](#) now supports reading the DWARF4 GNU extension .debug_addr table.

A heuristic sanity check for PE object files was too conservative in limiting VirtualSize to 200MB. A library user has an exe with .debug_info size of over 200MB. Increased the limit to be 2000MB and changed the names of the errors for the three heuristic checks to include *HEURISTIC* so it is easier to know the kind of error/failure it is.

When doing a shared-library build with cmake we were not emitting the correct .so version names nor setting SONAME with the correct version name. This long-standing mistake is now fixed.

Changes 0.9.1 to 0.9.2

Version 0.9.2 released 2 April 2024

Vulnerabilities DW202402-001, DW202402-002, DW202402-003, and DW202403-001 could crash *libdwarf* given a carefully corrupted (fuzzed) DWARF object file. Now the library returns an error for these corruptions. DW_CFA↵_high_user (in [dwarf.h](#)) was a misspelling. Added the correct spelling DW_CFA_hi_user and a comment on the incorrect spelling.

Changes 0.9.0 to 0.9.1

Version 0.9.1 released 27 January 2024

The abbreviation code type returned by [dwarf_die_abbrev_code\(\)](#) changed from **int** to **Dwarf_Unsigned** as abbrev codes are not constrained by the DWARF Standard.

The section count returned by [dwarf_get_section_count\(\)](#) is now of type **Dwarf_Unsigned**. The previous type of **int** never made sense in *libdwarf*. Callers will, in practice, see the same value as before.

All type-warnings issued by MSVC have been fixed.

Problems reading Macho (Apple) relocatable object files have been fixed.

Each of the build systems available now has an option which eliminates *libdwarf* references to the object section decompression libraries. See the respective READMEs.

Changes 0.8.0 to 0.9.0

Version 0.9.0 released 8 December 2023

Adding functions (rarely needed) for callers with special requirements. Added [dwarf_get_section_info_by_name_a\(\)](#) and [dwarf_get_section_info_by_index_a\(\)](#) which add `dw_section_flags` pointer argument to return the object section file flags (whose meaning depends entirely on the object file format), and `dw_section_offset` pointer argument to return the object-relevant offset of the section (here too the meaning depends on the object format). Also added [dwarf_machine_architecture\(\)](#) which returns a few top level data items about the object *libdwarf* has opened, including the 'machine' and 'flags' from object headers (all supported object types).

This adds new library functions [dwarf_next_cu_header_e\(\)](#) and [dwarf_siblingof_c\(\)](#). Used exactly as documented [dwarf_next_cu_header_d\(\)](#) and [dwarf_siblingof_b\(\)](#) work fine and continue to be supported for the foreseeable future. However it would be easy to misuse as the requirement that [dwarf_siblingof_b\(\)](#) be called immediately after a successful call to [dwarf_next_cu_header_d\(\)](#) was never stated and that dependency was impossible to enforce. The dependency was an API mistake made in 1992.

So [dwarf_next_cu_header_e\(\)](#) now returns the compilation-unit DIE as well as header data and [dwarf_siblingof_c\(\)](#) is not needed except to traverse sibling DIEs. (the compilation-unit DIE by definition has no siblings).

Changes were required to support Mach-O (Apple) universal binaries, which were not readable by earlier versions of the library.

We have new library functions [dwarf_init_path_a\(\)](#), [dwarf_init_path_dl_a\(\)](#), and [dwarf_get_universalbinary_count\(\)](#).

The first two allow a caller to specify which (numbering from zero) object file to report on by adding a new argument `dw_universalnumber`. Passing zero as the `dw_universalnumber` argument is always safe.

The third lets callers retrieve the number being used.

These new calls do not replace anything so existing code will work fine.

Applying the previously existing calls [dwarf_init_path\(\)](#) [dwarf_init_path_dl\(\)](#) to a Mach-O universal binary works, but the library will return data on the first (index zero) as a default since there is no `dw_universalnumber` argument possible.

For improved performance in reading Fde data when iterating though all usable pc values we add [dwarf_get_fde_info_for_all_regs3_b\(\)](#), which returns the next pc value with actual frame data. We retain [dwarf_get_fde_info_for_all_regs3\(\)](#) so existing code need not change.

Changes 0.7.0 to 0.8.0

v0.8.0 released 2023-09-20

New functions [dwarf_get_fde_info_for_reg3_c\(\)](#), [dwarf_get_fde_info_for_cfa_reg3_c\(\)](#) are defined. The advantage of the new versions is they correctly type the `dw_offset` argument return value as `Dwarf_Signed` instead of the earlier and incorrect type `Dwarf_Unsigned`.

The original functions [dwarf_get_fde_info_for_reg3_b\(\)](#) and [dwarf_get_fde_info_for_cfa_reg3_b\(\)](#) continue to exist and work for compatibility with the previous release.

For all `open()` calls for which the `O_CLOEXEC` flag exists we now add that flag to the `open()` call.

Vulnerabilities involving reading corrupt object files (created by fuzzing) have been fixed: DW202308-001 (ossfuzz 59576), DW202307-001 (ossfuzz 60506), DW202306-011 (ossfuzz 59950), DW202306-009 (ossfuzz 59755), DW202306-006 (ossfuzz 59727), DW202306-005 (ossfuzz 59717), DW202306-004 (ossfuzz 59695), DW202306-002 (ossfuzz 59519), DW202306-001 (ossfuzz 59597), DW202305-010 (ossfuzz 59478), DW202305-009 (ossfuzz 56451), DW202305-008 (ossfuzz 56451), DW202305-007 (ossfuzz 56474), DW202305-006 (ossfuzz 56472), DW202305-005 (ossfuzz 56462), DW202305-004 (ossfuzz 56446).

Changes 0.6.0 to 0.7.0

v0.7.0 released 2023-05-20

Elf section counts can exceed 16 bits (on linux see **man 5 elf**) so some function prototype members of struct [Dwarf_Obj_Access_Methods_a_s](#) changed. Specifically, `om_get_section_info()`, `om_load_section()`, and `om_relocate_a_section()` now pass section indexes as `Dwarf_Unsigned` instead of `Dwarf_Half`. Without this change executables/objects with more than 64K sections cannot be read by *libdwarf*. This is unlikely to affect your code since for most users *libdwarf* takes care of this and `dwarfdump` is aware of this change.

Two functions have been removed from [libdwarf.h](#) and the library: `dwarf_dnames_abbrev_by_code()` and `dwarf_dnames_abbrev_form_by_index()`.

`dwarf_dnames_abbrev_by_code()` is slow and pointless. Use either [dwarf_dnames_name\(\)](#) or [dwarf_dnames_abbrevtable\(\)](#) instead, depending on what you want to accomplish.

`dwarf_dnames_abbrev_form_by_index()` is not needed, was difficult to call due to argument list requirements, and never worked.

Changes 0.5.0 to 0.6.0

v0.6.0 released 2023-02-20 The dealloc required by [dwarf_offset_list\(\)](#) was wrong. The call could crash *libdwarf* on systems with 32bit pointers. The new and proper dealloc (for all pointer sizes) is `dwarf_dealloc(dbg, offsetlistptr, DW_DLA_UARRAY)`;

A memory leak from [dwarf_load_loclists\(\)](#) and [dwarf_load_rnglists\(\)](#) is fixed and the *libdwarf*-regressiontests error that hid the leak has also been fixed.

A **compatibility** change affects callers of [dwarf_dietype_offset\(\)](#), which on success returns the offset of the target of the `DW_AT_type` attribute (if such exists in the `Dwarf_Die`). Added a pointer argument so the function can (when appropriate) return a `FALSE` argument indicating the offset refers to DWARF4 `.debug_types` section, rather than `TRUE` value when `.debug_info` is the section the offset refers to. If anyone was using this function it would fail badly (while pretending success) with a DWARF4 `DW_FORM_ref_sig8` on a `DW_AT_type` attribute from the `Dwarf_Die` argument. One will likely encounter DWARF4 content so a single correct function seemed necessary. New regression tests will ensure this will continue to work.

A **compatibility** change affects callers of [dwarf_get_pubtypes\(\)](#). If an application reads `.debug_pubtypes` there is a **compatibility break**. Such applications must be recompiled with latest *libdwarf*, change `Dwarf_Type` declarations to use `Dwarf_Global`, and can only use the latest *libdwarf*. We are correcting a 1993 library design mistake that created extra work and documentation for library users and inflated the *libdwarf* API and documentation for no good reason.

The changes are: the data type `Dwarf_Type` disappears as do `dwarf_pubtypename()`, `dwarf_pubtype_die_offset()`, `dwarf_pubtype_cu_offset()`, `dwarf_pubtype_name_offsets()` and `dwarf_pubtypes_dealloc()`. Instead the type is `Dwarf_Global`, the type and functions used for [dwarf_get_globals\(\)](#). The existing read/dealloc functions for `Dwarf_Global` apply to `pubtypes` data too.

No one should be referring to the 1990s SGI/IRIX sections `.debug_weaknames`, `.debug_funcnames`, `.debug_varnames`, or `.debug_typenames` as they are not emitted by any compiler except from SGI/IRIX/MIPS in that period. There is (revised) support in *libdwarf* to read these sections, but we will not mention details here.

Any use of DW_FORM_strx3 or DW_FORM_addrx3 in DWARF would, in 0.5.0 and earlier, result in *libdwarf* reporting erroneous data. A copy-paste error in *libdwarf/dwarf_util.c* was noticed and fixed 24 January 2023 for 0.6.0. Bug **DW202301-001**.

Changes 0.4.2 to 0.5.0

v0.5.0 released 2022-11-22 The handling of the `.debug_abbrev` data in *libdwarf* is now more cpu-efficient (measurably faster) so access to DIEs and attribute lists is faster. The changes are library-internal so are not visible in the API.

Corrects CU and TU indexes in the `.debug_names` (fast access) section to be zero-based. The code for that section was previously unusable as it did not follow the DWARF5 documentation.

`dwarf_get_globals()` now returns a list of Dwarf_Global names and DIE offsets whether such are defined in the `.debug_names` or `.debug_pubnames` section or both. Previously it only read `.debug_pubnames`.

A new function, `dwarf_global_tag_number()`, returns the DW_TAG of any Dwarf_Global that was derived from the `.debug_names` section.

Three new functions enable printing of the `.debug_addr` table. `dwarf_debug_addr_table()`, `dwarf_debug_addr_by_index()`, and `dwarf_dealloc_debug_addr_table()`. Actual use of the table(s) in `.debug_addr` is handled for you when an attribute invoking such is encountered (see DW_FORM_addrx, DW_FORM_addrx1 etc).

Added doc/libdwarf.dox to the distribution (left out by accident earlier).

Changes 0.4.1 to 0.4.2

0.4.2 released 2022-09-13. No API changes. No API additions. Corrected a bug in `dwarf_tsearchhash.c` where a delete request was accidentally assumed in all hash tree searches. It was invisible to *libdwarf* uses. Vulnerabilities DW202207-001 and DW202208-001 were fixed so error conditions when reading fuzzed object files can no longer crash *libdwarf* (the crash was possible but not certain before the fixes). In this release we believe neither *libdwarf* nor *dwarfdump* leak memory even when there are malloc failures. Any GNU debuglink or build-id section contents were not being properly freed (if malloced, meaning a compressed section) until 9 September 2022.

It is now possible to run the build sanity tests in all three build mechanisms (configure,cmake,meson) on linux, MacOS, FreeBSD, and Mingw msys2 (windows). *libdwarf* README.md (or README) and README.cmake document how to do builds for each supported platform and build mechanism.

Changes 0.4.0 to 0.4.1

Reading a carefully corrupted DIE with form DW_FORM_ref_sig8 could result in reading memory outside any section, possibly leading to a segmentation violation or other crash. Fixed.

See also

<https://www.prevanders.net/dwarfbug.xml> DW202206-001

Reading a carefully corrupted `.debug_pubnames/.debug_pubtypes` could lead to reading memory outside the section being read, possibly leading to a segmentation violation or other crash. Fixed.

See also

<https://www.prevanders.net/dwarfbug.xml> DW202205-001

libdwf accepts DW_AT_entry_pc in a compilation unit DIE as a base address for location lists (though it will prefer DW_AT_low_pc if present, per DWARF3). A particular compiler emits DW_AT_entry_pc in a DWARF2 object, requiring this change.

libdwf adds `dwarf_suppress_debuglink_crc()` so that library callers can suppress crc calculations. (useful to save the time of crc when building and testing the same thing(s) over and over; it just loses a little checking.) Additionally, *libdwf* now properly handles objects with only GNU debug-id or only GNU debuglink.

`dwarfdump` adds `--show-args`, an option to print its arguments and version. Without that new option the version and arguments are not shown. The output of `-v` (`--version`) is a little more complete.

`dwarfdump` adds `--suppress-debuglink-crc`, an option to avoid crc calculations when rebuilding and rerunning tests depending on GNU `.note.gnu.buildid` or `.gnu_debuglink` sections. The help text and the `dwarfdump.1` man page are more specific documenting `--suppress-debuglink-crc` and `--no-follow-debuglink`

Changes 0.3.4 to 0.4.0

Removed the unused `Dwarf_Error` argument from `dwarf_return_empty_pubnames()` as the function can only return DW_DLV_OK. `dwarf_xu_header_free()` renamed to `dwarf_dealloc_xu_header()`. `dwarf_gdbindex_free()` renamed to `dwarf_dealloc_gdbindex()`. `dwarf_loc_head_c_dealloc` renamed to `dwarf_dealloc_loc_head_c()`.

`dwarf_get_location_op_value_d()` renamed to `dwarf_get_location_op_value_c()`, and 3 pointless arguments removed. The `dwarf_get_location_op_value_d` version and the three arguments were added for DWARF5 in `libdwf-20210528` but the change was a mistake. Now reverted to the previous version.

The `.debug_names` section interfaces have changed. Added `dwarf_dnames_offsets()` to provide details of facts useful in problems reading the section. `dwarf_dnames_name()` now does work and the interface was changed to make it easier to use.

Changes 0.3.3 to 0.3.4

Replaced the groff -mm based `libdwf.pdf` with a `libdwf.pdf` generated by doxygen and latex.

Added support for the meson build system.

Updated an include in `libdwf` source files. Improved doxygen documentation of *libdwf*. Now 'make check -j8' and the like works correctly. Fixed a bug where reading a PE (Windows) object could fail for certain section virtual size values. Added initializers to two uninitialized local variables in `dwarfdump` source so a compiler warning cannot not kill a `--enable-wall` build.

Added `src/bin/dwarfexample/showsectiongroups.c` so it is easy to see what groups are present in an object without all the other `dwarfdump` output.

Changes 20210528 to 0.3.3 (28 January 2022)

There were major revisions in going from date versioning to Semantic Versioning. Many functions were deleted and various functions changed their list of arguments. Many many filenames changed. Include lists were simplified. Far too much changed to list here.

Chapter 2

JIT and special case DWARF

html 2

2.1 Reading DWARF not in an object file

If the DWARF you work with is in standard object files (Elf, PE, MacOS) then you can ignore this section entirely. All that this section describes is used, but it's already done for you in functions in the library:

See also

[dwarf_init_path](#) [dwarf_init_path_dl](#)
[dwarf_init_b](#) and
[dwarf_finish](#) .

This section describes how to use calls

See also

[dwarf_object_init_b](#)
[dwarf_object_finish](#) .

These functions are useful if someone is doing just-in-time compilation, and someone working on the code wants to debug this on-the-fly code in a situation where nothing can be written to disc, but DWARF can be constructed in memory.

For a simple example of this with DWARF in local arrays

See also

[Demonstrating reading DWARF without a file.](#)

But the *libdwarf* feature can be useful in a variety of circumstances.

For example, the DWARF data were kept in simple files of bytes on the internet. Or on the local net. Or if files can be written locally each section could be kept in a simple stream of bytes in the local file system.

Another example is a non-standard file system, or file format, with the intent of obfuscating the file or the DWARF.

For this to work the code generator must generate standard DWARF.

Overall the idea is a simple one: You write a small handful of functions and supply function pointers and code implementing the functions. These are part of your application or library, not part of *libdwarf*. Your code accesses the data in whatever way applies and you write code that provides the interfaces so standard *libdwarf* can access your DWARF content.

You set up a little bit of data with that code (described below) and then you have essentially written the dwarf_↵ init_path equivalent and you can access compilation units, line tables etc and the standard *libdwarf* function calls simply work.

Data you need to create involves the following types. What follows describes how to fill them in and how to make them work for you.

```
typedef struct Dwarf_Obj_Access_Interface_a_s
    Dwarf_Obj_Access_Interface_a;
struct Dwarf_Obj_Access_Interface_a_s {
    void                *ai_object;
    const Dwarf_Obj_Access_Methods_a *ai_methods;
};

typedef struct Dwarf_Obj_Access_Methods_a_s
    Dwarf_Obj_Access_Methods_a
struct Dwarf_Obj_Access_Methods_a_s {
    int                (*om_get_section_info)(void* obj,
        Dwarf_Half      section_index,
        Dwarf_Obj_Access_Sections_a* return_section,
        int              * error);
    Dwarf_Small        (*om_get_byte_order)(void* obj);
    Dwarf_Small        (*om_get_length_size)(void* obj);
    Dwarf_Small        (*om_get_pointer_size)(void* obj);
    Dwarf_Unsigned     (*om_get_filesize)(void* obj);
    Dwarf_Unsigned     (*om_get_section_count)(void* obj);
    int                (*om_load_section)(void* obj,
        Dwarf_Half      section_index,
        Dwarf_Small**   return_data,
        int              * error);
    int                (*om_relocate_a_section)(void* obj,
        Dwarf_Half      section_index,
        Dwarf_Debug      dbg,
        int              *error);
};

typedef struct Dwarf_Obj_Access_Sections_a_s
    Dwarf_Obj_Access_Sections_a
struct Dwarf_Obj_Access_Sections_a_s {
    const char*        as_name;
    Dwarf_Unsigned     as_type;
    Dwarf_Unsigned     as_flags;
    Dwarf_Addr         as_addr;
    Dwarf_Unsigned     as_offset;
    Dwarf_Unsigned     as_size;
    Dwarf_Unsigned     as_link;
    Dwarf_Unsigned     as_info;
    Dwarf_Unsigned     as_addralign;
    Dwarf_Unsigned     as_entrysize;
};
```

2.1.1 Describing the Interface

struct struct Dwarf_Obj_Access_Interface_a_s

Your code must create and fill in this struct's two pointer members. Libdwarf needs these to access your DWARF data. You pass a pointer to this filled-in struct to **dwarf_object_init_b**. When it is time to conclude all access to the created Dwarf_Debug call **dwarf_object_finish**. Any allocations you made in setting these things up you must then free after calling **dwarf_object_finish**.

ai_object

Allocate a local struct (*libdwarf* will not touch this struct and will not know anything of its contents). You will need one of these for each Dwarf_Debug you open. Put a pointer to this into ai_object. Then fill in all the data you need to access information you will pass back via the ai_methods functions. In the description of the methods functions described later here, this pointer is named **obj**.

ai_methods

Usually you allocate a static structure and fill it in with function pointers (to functions you write). Then put a pointer to the static structure into this field.

2.1.2 Describing A Section

Dwarf_Obj_Access_Section_a:

The set of fields here is a set that is sufficient to describe a single object section to *libdwarf*. Your implementation of a **om_get_section_info** must simply fill in a few fields (leaving most zero) for *libdwarf* for the section indexed. The fields here are standard Elf, and for most you can just fill in the value zero. For section index zero as_name should be set to an empty string (see below about section index numbers).

as_name: Here you set a section name via the pointer. The section names must be names as defined in the DWARF standard, so if such do not appear in your data you have to create the strings yourself.

as_type: Just fill in zero.

as_flags: Just fill in zero.

as_addr: Fill in the address, in local memory, where the bytes of the section are.

as_offset: Just fill in zero.

as_size: Fill in the size, in bytes, of the section you are telling *libdwarf* about.

as_link: Just fill in zero.

as_info: Just fill in zero.

as_addralign: Just fill in zero.

as_entrysize: Just fill in one.

2.1.3 Function Pointers

struct Dwarf_Obj_Access_Methods_a_s:

The functions *libdwarf* needs to access object data are declared here. Usually the struct is statically defined and the function pointers are set at compile time. You must implement these functions based on your knowledge of how the actual data is represented and where to get it.

Each has a first-parameter of **obj** which is a struct you define to hold data you need to implement this set of functions. You refer to it When *libdwarf* calls your set of functions (these described now) it passes the *ai_object* pointer you provided to these functions as **obj** parameter .

This is the final part of your work for *libdwarf*. In the source file with your code you will be allocating data, making a provision for an array (real or conceptual) for per-section data, and returning values *libdwarf* needs. Note that the section array should include an index zero with all zero field values. That means interesting fields start with index one. This special case of index zero Elf is required and matches the standard Elf object format.

Notice that the **error** argument, where applicable, is an *int** . Error codes passed back are DW_DLE codes and **dwarf_errmsg_by_number** may be used (by your code) to get the standard error string for that error.

om_get_section_info

Get address, size, and name info about a section.

```
Parameters
obj          - Your data
section_index - Zero-based index.
return_section - Pointer to a structure in which
                section info will be placed. Caller must
                provide a valid pointer to a structure area.
                The structure's contents will be overwritten
                by the call to get_section_info.
error        - A pointer to an integer in which an error
                code may be stored.

Return
DW_DLV_OK      - Everything ok.
DW_DLV_ERROR   - Error occurred. Use 'error' to determine the
                @e libdwarf defined error.
DW_DLV_NO_ENTRY - No such section.
```

om_get_byte_order

This retrieves data you put into your **ai_object** struct that you filled out.

Get from your @b *ai_object* whether the object file represented by this *interface* is big-endian (DW_END_big) or little endian (DW_END_little).

```
Parameters
obj - Your data

Return
Endianness of object, DW_END_big or DW_END_little.
```

om_get_length_size

This retrieves data you put into your **ai_object** struct that you filled out.

Get the size of a length field in the underlying object file.
@e *libdwarf* currently supports * 4 and 8 byte sizes, but may support larger in the future.
Perhaps the *return* type should be an enumeration?

```
Parameters
obj - Your data

Return
Size of length. Cannot fail.
```

om_get_pointer_size

This retrieves data you put into your **ai_object** struct that you filled out.

Get the size of a pointer field in the underlying object file.

@e libdwarf currently supports 4 and 8 byte sizes.
Perhaps the `return` type should be an enumeration?

Return
Size of pointer. Cannot fail. */

om_get_filesize

This retrieves data you put into your `ai_object` struct that you filled out.

Parameters
obj - Your data

Return
Must `return` a value at least as large as any section @e libdwarf might read. Returns a value that is a sanity check on offsets @e libdwarf reads `for this` DWARF set. It need not be a tight bound.

om_get_section_count

This retrieves data you put into your `ai_object` struct that you filled out.

Get the number of sections in the `object` file, including the index zero section with no content.

Parameters
obj - Your data

Return
Number of sections.

om_load_section

This retrieves data you put into your `ai_object` struct that you filled out.

Get a pointer to an array of bytes that are the section content.

Get a pointer to an array of bytes that represent the section.

Parameters
obj - Your data
section_index - Zero-based section index.
return_data - Place the address of `this` section content into `*return_data`.
error - Pointer to an integer `for` returning libdwarf-defined error numbers.

Return
DW_DLV_OK - No error.
DW_DLV_ERROR - Error. Use 'error' to indicate a libdwarf-defined error number.
DW_DLV_NO_ENTRY - No such section. */

om_relocate_a_section

Leave `this` pointer NULL.
If relocations are required it is probably simpler `for` you `do` to them yourself in your implementation of @b `om_load_section`.
Any relocations `this` function pointer is to use must be in standard Elf relocation (32 or 64 bit) form and must be in an appropriately named Elf relocation section.

Parameters
obj - Your data
section_index - Zero-based index of the section to be relocated.
error - Pointer to an integer `for` returning libdwarf-defined error numbers.

Return
DW_DLV_OK - No error.
DW_DLV_ERROR - Error. Use 'error' to indicate a libdwarf-defined error number.
DW_DLV_NO_ENTRY - No such section.

Chapter 3

dwarf.h

[dwarf.h](#) contains all the identifiers such as `DW_TAG_compile_unit` etc from the various versions of the DWARF Standard beginning with DWARF2 and containing all later Dwarf Standard identifiers.

In addition, it contains all user-defined identifiers that we have been able to find.

All identifiers here are C defines with the prefix `"DW_"`.

Chapter 4

libdwarf.h

[libdwarf.h](#) contains all the type declarations and function declarations needed to use the library. It is essential that coders include [dwarf.h](#) before including [libdwarf.h](#).

All identifiers here in the public namespace begin with DW_ or Dwarf_ or dwarf_ . All function argument names declared here begin with dw_ .

Chapter 5

checkexamples.c

[checkexamples.c](#) contains what user code should be. Hence the code typed in [checkexamples.c](#) is PUBLIC DO-MAIN and may be copied, used, and altered without any restrictions.

[checkexamples.c](#) need not be compiled routinely nor should it ever be executed.

To verify syntatic correctness compile in the libdwarf-code/doc directory with:

```
cc -c -Wall -O0 -Wpointer-arith \
-Wdeclaration-after-statement \
-Wextra -Wcomment -Wformat -Wpedantic -Wuninitialized \
-Wno-long-long -Wshadow -Wbad-function-cast \
-Wmissing-parameter-type -Wnested-externs \
-I../src/lib/libdwarf checkexamples.c
```


Chapter 6

Topic Index

6.1 Topics

Here is a list of all topics with brief descriptions:

Basic Library Datatypes Group	41
Enumerators with various purposes	42
Defined and Opaque Structs	43
Default stack frame macros	52
DW_DLA alloc/dealloc typename&number	53
DW_DLE Dwarf_Error numbers	54
Libdwarf Initialization Functions	64
Compilation Unit (CU) Access	71
Debugging Information Entry (DIE) content	78
DIE Attribute and Attribute-Form Details	97
Line Table For a CU	112
Ranges: code addresses in DWARF3-4	128
Rnglists: code addresses in DWARF5	130
Locations of data: DWARF2-DWARF5	135
.debug_addr access: DWARF5	143
Macro Access: DWARF5	146
Macro Access: DWARF2-4	153
Stack Frame Access	154
Abbreviations Section Details	173
String Section .debug_str Details	177
Str_Offsets section details	178
Dwarf_Error Functions	181
Generic dwarf_dealloc Function	183
Access to Section .debug_sup	184
Fast Access to .debug_names DWARF5	185
Fast Access to a CU given a code address	192
Fast Access to .debug_pubnames and more.	195
Fast Access to GNU .debug_gnu_pubnames	202
Fast Access to Gdb Index	206
Fast Access to Split Dwarf (Debug Fission)	214
Access GNU .gnu_debuglink, build-id.	220
Harmless Error recording	224
Names DW_TAG_member etc as strings	227
Object Sections Data	231
Section Groups Objectfile Data	240

LEB Encode and Decode	242
Miscellaneous Functions	243
Determine Object Type of a File	247
Section allocation: malloc or mmap	248
Using dwarf_init_path()	249
Using dwarf_init_path_dl()	250
Using dwarf_attrlist()	251
Attaching a tied dbg	252
Detaching a tied dbg	253
Examining Section Group data	253
Using dwarf_siblingof_c()	254
Using dwarf_siblingof_b()	254
Using dwarf_child()	255
using dwarf_validate_die_sibling	255
Example walking CUs(e)	257
Example walking CUs(d)	258
Using dwarf_offdie_b()	260
Using dwarf_offset_given_die()	261
Using dwarf_attrlist()	261
Using dwarf_offset_list()	261
Documenting Form_Block	262
Using dwarf_discr_list()	263
Location/expression access	264
Reading a location expression	266
Using dwarf_srclines_b()	267
Using dwarf_srclines_b() and linecontext	269
Using dwarf_srcfiles()	269
Using dwarf_get_globals()	270
Using dwarf_globals_by_type()	271
Reading .debug_weaknames (nonstandard)	271
Reading .debug_funcnames (nonstandard)	271
Reading .debug_types (nonstandard)	272
Reading .debug_varnames data (nonstandard)	272
Reading .debug_names data	273
Reading .debug_macro data (DWARF5)	275
Reading .debug_macinfo (DWARF2-4)	278
Extracting fde, cie lists.	278
Reading the .eh_frame section	279
Using dwarf_expand_frame_instructions	279
Reading string offsets section data	280
Reading an aranges section	281
Example getting .debug_ranges data	282
Reading gdbindex data	284
Reading gdbindex addressarea	285
Reading the gdbindex symbol table	285
Reading cu and tu Debug Fission data	286
Reading Split Dwarf (Debug Fission) hash slots	287
Reading high pc from a DIE.	287
Reading Split Dwarf (Debug Fission) data	288
Retrieving tag,attribute,etc names	288
Using GNU debuglink data	289
Accessing accessing raw rnglist	290
Accessing rnglists section	291
Demonstrating reading DWARF without a file.	292
A simple report on section groups.	297

Chapter 7

Class Index

7.1 Class List

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

Dwarf_Block_s	301
Dwarf_Cmdline_Options_s	301
Dwarf_Debug_Fission_Per_CU_s	302
Dwarf_Form_Data16_s	302
Dwarf_Macro_Details_s	302
Dwarf_Obj_Access_Interface_a_s	303
Dwarf_Obj_Access_Methods_a_s	303
Dwarf_Obj_Access_Section_a_s	304
Dwarf_Printf_Callback_Info_s	305
Dwarf_Ranges_s	305
Dwarf_Regtable3_s	306
Dwarf_Regtable_Entry3_s	306
Dwarf_Sig8_s	307

Chapter 8

File Index

8.1 File List

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

checkexamples.c	311
/home/davea/dwarf/code/src/bin/dwarfexample/ jitreader.c	311
/home/davea/dwarf/code/src/bin/dwarfexample/ showsectiongroups.c	311
/home/davea/dwarf/code/src/lib/libdwarf/ dwarf.h	313
/home/davea/dwarf/code/src/lib/libdwarf/ libdwarf.h	333

Chapter 9

Topic Documentation

9.1 Basic Library Datatypes Group

Typedefs

- typedef unsigned long long [Dwarf_Unsigned](#)
- typedef signed long long [Dwarf_Signed](#)
- typedef unsigned long long [Dwarf_Off](#)
- typedef unsigned long long [Dwarf_Addr](#)
- typedef int [Dwarf_Bool](#)
- typedef unsigned short [Dwarf_Half](#)
- typedef unsigned char [Dwarf_Small](#)
- typedef void * [Dwarf_Ptr](#)

9.1.1 Detailed Description

9.1.2 Typedef Documentation

9.1.2.1 Dwarf_Addr

[Dwarf_Addr](#)

Used when a data item is a an address represented in DWARF. 64 bits. Must be as large as the largest object address size.

9.1.2.2 Dwarf_Bool

[Dwarf_Bool](#)

A TRUE(non-zero)/FALSE(zero) data item.

9.1.2.3 Dwarf_Half

`Dwarf_Half`

Many libdwarf values (attribute codes, for example) are defined by the standard to be 16 bits, and this datatype reflects that (the type must be at least 16 bits wide).

9.1.2.4 Dwarf_Off

`Dwarf_Off`

Used for offsets. It should be same size as Dwarf_Unsigned.

9.1.2.5 Dwarf_Ptr

`Dwarf_Ptr`

A generic pointer type. It uses void * so it cannot be added-to or subtracted-from.

9.1.2.6 Dwarf_Signed

`Dwarf_Signed`

The basic signed data type. Intended to be a signed 64bit value.

9.1.2.7 Dwarf_Small

`Dwarf_Small`

Used for small unsigned integers and used as Dwarf_Small* for pointers and it supports pointer addition and subtraction conveniently.

9.1.2.8 Dwarf_Unsigned

`Dwarf_Unsigned`

The basic unsigned data type. Intended to be an unsigned 64bit value.

9.2 Enumerators with various purposes

Enumerations

- enum `Dwarf_Ranges_Entry_Type` { `DW_RANGES_ENTRY` , `DW_RANGES_ADDRESS_SELECTION` , `DW_RANGES_END` }
- enum `Dwarf_Form_Class` {
`DW_FORM_CLASS_UNKNOWN` = 0 , `DW_FORM_CLASS_ADDRESS` = 1 , `DW_FORM_CLASS_BLOCK` = 2 , `DW_FORM_CLASS_CONSTANT` =3 ,
`DW_FORM_CLASS_EXPRLOC` = 4 , `DW_FORM_CLASS_FLAG` = 5 , `DW_FORM_CLASS_LINEPTR` = 6 ,
`DW_FORM_CLASS_LOCLISTPTR` =7 ,
`DW_FORM_CLASS_MACPTR` = 8 , `DW_FORM_CLASS_RANGELISTPTR` =9 , `DW_FORM_CLASS_↵`
`REFERENCE` =10 , `DW_FORM_CLASS_STRING` = 11 ,
`DW_FORM_CLASS_FRAMEPTR` = 12 , `DW_FORM_CLASS_MACROPTR` = 13 , `DW_FORM_CLASS_↵`
`ADDRPTR` = 14 , `DW_FORM_CLASS_LOCLIST` = 15 ,
`DW_FORM_CLASS_LOCLISTSPTR` =16 , `DW_FORM_CLASS_RNGLIST` =17 , `DW_FORM_CLASS_↵`
`RNGLISTSPTR` =18 , `DW_FORM_CLASS_STROFFSETSPTR` =19 }

9.2.1 Detailed Description

9.2.2 Enumeration Type Documentation

9.2.2.1 Dwarf_Form_Class

```
enum Dwarf_Form_Class
```

The dwarf specification separates FORMs into different classes. To do the separation properly requires 4 pieces of data as of DWARF4 (thus the function arguments listed here). The DWARF4 specification class definition suffices to describe all DWARF versions. See section 7.5.4, Attribute Encodings. A return of DW_FORM_CLASS_UNKNOWN means the library could not properly figure out what form-class it is.

DW_FORM_CLASS_FRAMEPTR is MIPS/IRIX only, and refers to the DW_AT_MIPS_fde attribute (a reference to the .debug_frame section).

DWARF5: DW_FORM_CLASS_LOCLISTSPTR is like DW_FORM_CLASS_LOCLIST except that LOCLISTSPTR is always a section offset, never an index, and LOCLISTSPTR is only referenced by DW_AT_loclists_base. Note DW_FORM_CLASS_LOCLISTSPTR spelling to distinguish from DW_FORM_CLASS_LOCLISTPTR.

DWARF5: DW_FORM_CLASS_RNGLISTSPTR is like DW_FORM_CLASS_RNGLIST except that RNGLISTSPTR is always a section offset, never an index. DW_FORM_CLASS_RNGLISTSPTR is only referenced by DW_AT_↔rnglists_base.

9.2.2.2 Dwarf_Ranges_Entry_Type

```
enum Dwarf_Ranges_Entry_Type
```

The dwr_addr1/addr2 data is either pair of offsets of a base pc address (DW_RANGES_ENTRY) or a base pc address (dwr_addr2 in DW_RANGES_ADDRESS_SELECTION) or both are zero(end of list, DW_RANGES_END) or both non-zero but identical (means an empty range, DW_RANGES_ENTRY). These are for use with DWARF 2,3,4.

DW_RANGES_ADDRESS_SELECTION should have been spelled DW_RANGES_BASE_ADDRESS. but it is not worth changing as it is widely used.

The DW_RANGES_ENTRY values are raw pc offset data recorded in the section, not addresses.

See also

[Example getting .debug_ranges data](#)

Dwarf_Ranges* apply to DWARF2,3, and 4. Not to DWARF5 (the data is different and in a new DWARF5 section).

9.3 Defined and Opaque Structs

Classes

- struct [Dwarf_Form_Data16_s](#)
- struct [Dwarf_Sig8_s](#)
- struct [Dwarf_Block_s](#)
- struct [Dwarf_Printf_Callback_Info_s](#)
- struct [Dwarf_Cmdline_Options_s](#)
- struct [Dwarf_Ranges_s](#)
- struct [Dwarf_Regtable_Entry3_s](#)
- struct [Dwarf_Regtable3_s](#)
- struct [Dwarf_Macro_Details_s](#)
- struct [Dwarf_Obj_Access_Section_a_s](#)
- struct [Dwarf_Obj_Access_Methods_a_s](#)
- struct [Dwarf_Obj_Access_Interface_a_s](#)
- struct [Dwarf_Debug_Fission_Per_CU_s](#)

Typedefs

- typedef struct [Dwarf_Form_Data16_s](#) Dwarf_Form_Data16
- typedef struct [Dwarf_Sig8_s](#) Dwarf_Sig8
- typedef struct [Dwarf_Block_s](#) Dwarf_Block
- typedef struct Dwarf_Locdesc_c_s * [Dwarf_Locdesc_c](#)
- typedef struct Dwarf_Loc_Head_c_s * [Dwarf_Loc_Head_c](#)
- typedef struct Dwarf_Gnu_Index_Head_s * [Dwarf_Gnu_Index_Head](#)
- typedef struct Dwarf_Dsc_Head_s * [Dwarf_Dsc_Head](#)
- typedef struct Dwarf_Frame_Instr_Head_s * [Dwarf_Frame_Instr_Head](#)
- typedef void(* [dwarf_printf_callback_function_type](#)) (void *dw_user_pointer, const char *dw_linecontent)
- typedef struct [Dwarf_Cmdline_Options_s](#) Dwarf_Cmdline_Options
- typedef struct Dwarf_Str_Offsets_Table_s * [Dwarf_Str_Offsets_Table](#)
- typedef struct [Dwarf_Ranges_s](#) Dwarf_Ranges
- typedef struct [Dwarf_Regtable_Entry3_s](#) Dwarf_Regtable_Entry3
- typedef struct [Dwarf_Regtable3_s](#) Dwarf_Regtable3
- typedef struct Dwarf_Error_s * [Dwarf_Error](#)
- typedef struct Dwarf_Debug_s * [Dwarf_Debug](#)
- typedef struct Dwarf_Section_s * [Dwarf_Section](#)
- typedef struct Dwarf_Die_s * [Dwarf_Die](#)
- typedef struct Dwarf_Debug_Addr_Table_s * [Dwarf_Debug_Addr_Table](#)
- typedef struct Dwarf_Line_s * [Dwarf_Line](#)
- typedef struct Dwarf_Global_s * [Dwarf_Global](#)
- typedef struct Dwarf_Type_s * [Dwarf_Type](#)
- typedef struct Dwarf_Func_s * [Dwarf_Func](#)
- typedef struct Dwarf_Var_s * [Dwarf_Var](#)
- typedef struct Dwarf_Weak_s * [Dwarf_Weak](#)
- typedef struct Dwarf_Attribute_s * [Dwarf_Attribute](#)
- typedef struct Dwarf_Abbrev_s * [Dwarf_Abbrev](#)
- typedef struct Dwarf_Fde_s * [Dwarf_Fde](#)
- typedef struct Dwarf_Cie_s * [Dwarf_Cie](#)
- typedef struct Dwarf_Arange_s * [Dwarf_Arange](#)
- typedef struct Dwarf_Gdbindex_s * [Dwarf_Gdbindex](#)
- typedef struct Dwarf_Xu_Index_Header_s * [Dwarf_Xu_Index_Header](#)
- typedef struct Dwarf_Line_Context_s * [Dwarf_Line_Context](#)
- typedef struct Dwarf_Macro_Context_s * [Dwarf_Macro_Context](#)
- typedef struct Dwarf_Dnames_Head_s * [Dwarf_Dnames_Head](#)
- typedef void(* [Dwarf_Handler](#)) (Dwarf_Error dw_error, Dwarf_Ptr dw_errarg)
- typedef struct [Dwarf_Macro_Details_s](#) Dwarf_Macro_Details
- typedef struct [Dwarf_Debug_Fission_Per_CU_s](#) Dwarf_Debug_Fission_Per_CU
- typedef struct [Dwarf_Obj_Access_Interface_a_s](#) Dwarf_Obj_Access_Interface_a
- typedef struct [Dwarf_Obj_Access_Methods_a_s](#) Dwarf_Obj_Access_Methods_a
- typedef struct [Dwarf_Obj_Access_Section_a_s](#) Dwarf_Obj_Access_Section_a
- typedef struct Dwarf_Rnglists_Head_s * [Dwarf_Rnglists_Head](#)

Enumerations

- enum [Dwarf_Sec_Alloc_Pref](#) { Dwarf_Alloc_None =0 , Dwarf_Alloc_Malloc =1 , Dwarf_Alloc_Mmap =2 }

9.3.1 Detailed Description

9.3.2 Typedef Documentation

9.3.2.1 Dwarf_Abbrev

[Dwarf_Abbrev](#)

Used to reference a Dwarf_Abbrev. Usually Dwarf_Abbrev are fully handled inside the library so one rarely needs to declare the type.

9.3.2.2 Dwarf_Arange

[Dwarf_Arange](#)

Used to reference a code address range in a section such as .debug_info.

9.3.2.3 Dwarf_Attribute

[Dwarf_Attribute](#)

Used to reference a Dwarf_Die attribute

9.3.2.4 Dwarf_Block

[Dwarf_Block](#)

Used to hold uninterpreted blocks of data. bl_data refers to on an uninterpreted block of data Used with certain location information functions, a frame expression function, expanded frame instructions, and DW_FORM_block functions.

See also

[dwarf_formblock](#)

[Documenting Form_Block](#)

9.3.2.5 Dwarf_Cie

[Dwarf_Cie](#)

Used to reference .debug_frame or .eh_frame CIE.

9.3.2.6 Dwarf_Debug

[Dwarf_Debug](#)

An open Dwarf_Debug points to data that libdwarf maintains to support libdwarf calls.

9.3.2.7 Dwarf_Debug_Addr_Table

[Dwarf_Debug_Addr_Table](#)

Used to reference a table in section .debug_addr

9.3.2.8 Dwarf_Debug_Fission_Per_CU

[Dwarf_Debug_Fission_Per_CU](#)

A handy short name for a [Dwarf_Debug_Fission_Per_CU_s](#) struct.

9.3.2.9 Dwarf_Die

[Dwarf_Die](#)

Used to reference a DWARF Debugging Information Entry.

9.3.2.10 Dwarf_Dnames_Head

[Dwarf_Dnames_Head](#)

Used as the general reference to the DWARF5 .debug_names section.

9.3.2.11 Dwarf_Dsc_Head

[Dwarf_Dsc_Head](#)

Access to DW_AT_discr_list array of discriminant values.

9.3.2.12 Dwarf_Error

```
Dwarf_Error  
Dwarf_Error error = 0;  
dres = dwarf_siblingof_c(in_die, &return_sib, &error);
```

&error is used in calls to return error details when the call returns DW_DLV_ERROR.

9.3.2.13 Dwarf_Fde

[Dwarf_Fde](#)

Used to reference .debug_frame or .eh_frame FDE.

9.3.2.14 Dwarf_Form_Data16

`Dwarf_Form_Data16`

a container for a DW_FORM_data16 data item. We have no integer types suitable so this special struct is used instead. It is up to consumers/producers to deal with the contents.

9.3.2.15 Dwarf_Frame_Instr_Head

`Dwarf_Frame_Instr_Head`

The basis for access to DWARF frame instructions (FDE or CIE) in full detail.

9.3.2.16 Dwarf_Func

`Dwarf_Func`

An SGI extension type which is no longer used at all. As of release 0.6.0 use Dwarf_Global instead.

9.3.2.17 Dwarf_Gdbindex

`Dwarf_Gdbindex`

Used to reference .gdb_index section data which is a fast-access section by and for gdb.

9.3.2.18 Dwarf_Global

`Dwarf_Global`

Used to reference a reference to an entry in the .debug_pubnames section.

9.3.2.19 Dwarf_Gnu_Index_Head

`Dwarf_Gnu_Index_Head`

A pointer to a struct Dwarf_Gnu_Index_Head_s for sections .debug_gnu_pubtypes or .debug_gnu_pubnames. These are not standard DWARF, and can appear with gcc -gdwarf-5

9.3.2.20 Dwarf_Handler

`Dwarf_Handler`

Used in rare cases (mainly tiny programs) with `dwarf_init_path()` etc initialization calls to provide a pointer to a generic-error-handler function you write.

9.3.2.21 Dwarf_Line

[Dwarf_Line](#)

Used to reference a line reference from the .debug_line section.

9.3.2.22 Dwarf_Line_Context

[Dwarf_Line_Context](#)

Used as the general reference line data (.debug_line).

9.3.2.23 Dwarf_Loc_Head_c

[Dwarf_Loc_Head_c](#)

provides access to any sort of location description for DWARF2,3,4, or 5.

9.3.2.24 Dwarf_Locdesc_c

[Dwarf_Locdesc_c](#)

Provides access to Dwarf_Locdesc_c, a single location description

9.3.2.25 Dwarf_Macro_Context

[Dwarf_Macro_Context](#)

Used as the general reference to DWARF5 .debug_macro data.

9.3.2.26 Dwarf_Macro_Details

[Dwarf_Macro_Details](#)

A handy short name for a Dwarf_Macro_Details_S struct.

9.3.2.27 Dwarf_Obj_Access_Interface_a

[Dwarf_Obj_Access_Interface_a](#)

Used for access to and setting up special data allowing access to DWARF even with no object files present

9.3.2.28 Dwarf_Obj_Access_Methods_a

[Dwarf_Obj_Access_Methods_a](#)

Used for access to and setting up special data allowing access to DWARF even with no object files present

9.3.2.29 Dwarf_Obj_Access_Section_a

[Dwarf_Obj_Access_Section_a](#)

Used for access to and setting up special data allowing access to DWARF even with no object files present. The fields match up with Elf section headers, but for non-Elf many of the fields can be set to zero.

9.3.2.30 dwarf_printf_callback_function_type

[dwarf_printf_callback_function_type](#)

Used as a function pointer to a user-written callback function. This provides a detailed content of line table data.

The default contents of the callback data are all zero bytes. So no callbacks involving this data will be done.

See [dwarf_register_printf_callback\(\)](#)

Parameters

<i>dw_user_pointer</i>	Passes your callback a pointer to space you allocated as an identifier of some kind in calling dwarf_register_printf_callback..
<i>dw_linecontent</i>	Passes your callback null-terminated string with one line of detailed line table content.

9.3.2.31 Dwarf_Ranges

[Dwarf_Ranges](#)

Details of of non-contiguous address ranges of DIEs for DWARF2, DWARF3, and DWARF4. Sufficient for older dwarf.

dwr_addr1 and dwr_addr2 in the struct are offsets from a base address in the CU involved. To calculate actual range pc addresses see the example:

See also

[Example getting .debug_ranges data](#)

9.3.2.32 Dwarf_Regtable3

[Dwarf_Regtable3](#)

This structs provides a way for applications to select the number of frame registers and to select names for them.

rt3_rules and rt3_reg_table_size must be filled in before calling libdwf. Filled in with a pointer to an array (pointer and array set up by the calling application) of rt3_reg_table_size [Dwarf_Regtable_Entry3_s](#) structs. libdwf does not allocate or deallocate space for the rules, you must do so. libdwf will initialize the contents rules array, you do not need to do so (though if you choose to initialize the array somehow that is ok: libdwf will overwrite your initializations with its own).

Note that this definition can only deal correctly with register table size that fits in a 16 bit unsigned value.

9.3.2.33 Dwarf_Regtable_Entry3

Dwarf_Regtable_Entry3

For each index *i* (naming a hardware register with dwarf number *i*) the following is true and defines the value of that register:

```
If dw_regnum is Register DW_FRAME_UNDEFINED_VAL
    it is not DWARF register number but
    a place holder indicating the register
    has no defined value.
If dw_regnum is Register DW_FRAME_SAME_VAL
    it is not DWARF register number but
    a place holder indicating the register has the same
    value in the previous frame.

DW_FRAME_UNDEFINED_VAL, DW_FRAME_SAME_VAL and
DW_FRAME_CFA_COL are only present at libdwarf runtime.
Never on disk.
DW_FRAME_* Values present on disk are in dwarf.h
Because DW_FRAME_SAME_VAL and DW_FRAME_UNDEFINED_VAL
and DW_FRAME_CFA_COL are definable at runtime
consider the names symbolic in this comment,
not absolute.

Otherwise: the register number is a DWARF register number
(see ABI documents for how this translates to hardware/
software register numbers in the machine hardware)
and the following applies:

In a cfa-defining entry (rt3_cfa_rule) the regnum is the
CFA 'register number'. Which is some 'normal' register,
not DW_FRAME_CFA_COL, nor DW_FRAME_SAME_VAL, nor
DW_FRAME_UNDEFINED_VAL.

If dw_value_type == DW_EXPR_OFFSET (the only
possible case for dwarf2):
    If dw_offset_relevant is non-zero, then
        the value is stored at the address
        CFA+N where N (dw_offset) is a signed offset,
        (not unsigned) and must be cast to Dwarf_Signed
        before use.
        dw_regnum is the cfa register rule which means
        one ignores dw_regnum and uses the CFA appropriately.
        Rule: Offset(N)
    If dw_offset_relevant is zero, then the
        value of the register
        is the value of (DWARF) register number dw_regnum.
        Rule: register(R)
If dw_value_type == DW_EXPR_VAL_OFFSET
    the value of this register is CFA +N where
    N (dw_offset) is a signed offset (not unsigned)
    and must be cast to Dwarf_Signed before use.
    dw_regnum is the cfa register rule which means
    one ignores dw_regnum and uses the CFA appropriately.
    Rule: val_offset(N)
If dw_value_type == DW_EXPR_EXPRESSION
    The value of the register is the value at the address
    computed by evaluating the DWARF expression E.
    Rule: expression(E)
    The expression E byte stream is pointed to by
    block.bl_data.
    The expression length in bytes is given by
    block.bl_len.
If dw_value_type == DW_EXPR_VAL_EXPRESSION
    The value of the register is the value
    computed by evaluating the DWARF expression E.
    Rule: val_expression(E)
    The expression E byte stream is pointed to
    by block.bl_data.
```

The expression length in bytes is given by
block.bl_len.
Other values of dw_value_type are an error.

DWARF is showing what a debugger would act on to calculate actual register values. Libdwarf does not know any register values and cannot calculate any. If a caller wishes to actually do the proper calculations the caller must provide its own register data space and calculate new values and new register status in the caller's register data.

Note that this definition can only deal correctly with register numbers that fit in a 16 bit unsigned value. Removing this restriction would force an incompatible change to several functions in the libdwarf API.

9.3.2.34 Dwarf_Rnglists_Head

[Dwarf_Rnglists_Head](#)

Used for access to a set of DWARF5 debug_rnglists entries.

9.3.2.35 Dwarf_Section

[Dwarf_Section](#)

An open Dwarf_Section points to data that libdwarf maintains to record object section data.

9.3.2.36 Dwarf_Sig8

[Dwarf_Sig8](#)

Used for signatures where ever they appear. It is not a string, it is 8 bytes of a signature one would use to find a type unit.

See also

[dwarf_formsig8](#)

9.3.2.37 Dwarf_Str_Offsets_Table

[Dwarf_Str_Offsets_Table](#)

Provides an access to the .debug_str_offsets section independently of other DWARF sections. Mainly of use in examining the .debug_str_offsets section content for problems.

9.3.2.38 Dwarf_Type

[Dwarf_Type](#)

Before release 0.6.0 used to reference a reference to an entry in the .debug_pubtypes section (as well as the SGI-only extension .debug_types). However, we use Dwarf_Global instead now.

9.3.2.39 Dwarf_Var

`Dwarf_Var`

An SGI extension type which is no longer used at all. As of release 0.6.0 use Dwarf_Global instead.

9.3.2.40 Dwarf_Weak

`Dwarf_Weak`

An SGI extension type which is no longer used at all. As of release 0.6.0 use Dwarf_Global instead.

9.3.2.41 Dwarf_Xu_Index_Header

`Dwarf_Xu_Index_Header`

Used to reference .debug_cu_index or .debug_tu_index sections in a split-dwarf package file.

9.3.3 Enumeration Type Documentation

9.3.3.1 Dwarf_Sec_Alloc_Pref

enum `Dwarf_Sec_Alloc_Pref`

Since

{0.12.0}

This is part of the allowance of mmap for loading sections of an object file.

The option of using mmap() only applies to Elf object files in this release.

See also

`dwarf_set_load_preference()`

9.4 Default stack frame macros

Macros

- `#define DW_DLX_NO_EH_OFFSET (-1LL)`
- `#define DW_DLX_EH_OFFSET_UNAVAILABLE (-2LL)`
- `#define DW_CIE_AUGMENTER_STRING_V0 "z"`
- `#define DW_REG_TABLE_SIZE DW_FRAME_LAST_REG_NUM`
- `#define DW_FRAME_REG_INITIAL_VALUE DW_FRAME_SAME_VAL`
- `#define DW_EXPR_OFFSET 0 /* offset is from CFA reg */`
- `#define DW_EXPR_VAL_OFFSET 1`
- `#define DW_EXPR_EXPRESSION 2`
- `#define DW_EXPR_VAL_EXPRESSION 3`

9.4.1 Detailed Description

9.5 DW_DLA alloc/dealloc typename&number

Macros

- `#define DW_DLA_STRING 0x01 /* char* */`
- `#define DW_DLA_LOC 0x02 /* Dwarf_Loc */`
- `#define DW_DLA_LOCDISC 0x03 /* Dwarf_Locdesc */`
- `#define DW_DLA_ELLIST 0x04 /* Dwarf_Ellist (not used) */`
- `#define DW_DLA_BOUNDS 0x05 /* Dwarf_Bounds (not used) */`
- `#define DW_DLA_BLOCK 0x06 /* Dwarf_Block */`
- `#define DW_DLA_DEBUG 0x07 /* Dwarf_Debug */`
- `#define DW_DLA_DIE 0x08 /* Dwarf_Die */`
- `#define DW_DLA_LINE 0x09 /* Dwarf_Line */`
- `#define DW_DLA_ATTR 0x0a /* Dwarf_Attribute */`
- `#define DW_DLA_TYPE 0x0b /* Dwarf_Type (not used) */`
- `#define DW_DLA_SUBSCR 0x0c /* Dwarf_Subscr (not used) */`
- `#define DW_DLA_GLOBAL 0x0d /* Dwarf_Global */`
- `#define DW_DLA_ERROR 0x0e /* Dwarf_Error */`
- `#define DW_DLA_LIST 0x0f /* a list */`
- `#define DW_DLA_LINEBUF 0x10 /* Dwarf_Line* (not used) */`
- `#define DW_DLA_ARANGE 0x11 /* Dwarf_Arange */`
- `#define DW_DLA_ABBREV 0x12 /* Dwarf_Abbrev */`
- `#define DW_DLA_FRAME_INSTR_HEAD 0x13 /* Dwarf_Frame_Instr_Head */`
- `#define DW_DLA_CIE 0x14 /* Dwarf_Cie */`
- `#define DW_DLA_FDE 0x15 /* Dwarf_Fde */`
- `#define DW_DLA_LOC_BLOCK 0x16 /* Dwarf_Loc */`
- `#define DW_DLA_FRAME_OP 0x17 /* Dwarf_Frame_Op (not used) */`
- `#define DW_DLA_FUNC 0x18 /* Dwarf_Func */`
- `#define DW_DLA_UARRAY 0x19 /* Array of Dwarf_Off:Jan2023 */`
- `#define DW_DLA_VAR 0x1a /* Dwarf_Var */`
- `#define DW_DLA_WEAK 0x1b /* Dwarf_Weak */`
- `#define DW_DLA_ADDR 0x1c /* Dwarf_Addr sized entries */`
- `#define DW_DLA_RANGES 0x1d /* Dwarf_Ranges */`
- `#define DW_DLA_GNU_INDEX_HEAD 0x35`
- `#define DW_DLA_RNGLISTS_HEAD 0x36 /* .debug_rnglists DW5 */`
- `#define DW_DLA_GDBINDEX 0x37 /* Dwarf_Gdbindex */`
- `#define DW_DLA_XU_INDEX 0x38 /* Dwarf_Xu_Index_Header */`
- `#define DW_DLA_LOC_BLOCK_C 0x39 /* Dwarf_Loc_c */`
- `#define DW_DLA_LOCDISC_C 0x3a /* Dwarf_Locdesc_c */`
- `#define DW_DLA_LOC_HEAD_C 0x3b /* Dwarf_Loc_Head_c */`
- `#define DW_DLA_MACRO_CONTEXT 0x3c /* Dwarf_Macro_Context */`
- `#define DW_DLA_DSC_HEAD 0x3e /* Dwarf_Dsc_Head */`
- `#define DW_DLA_DNAMES_HEAD 0x3f /* Dwarf_Dnames_Head */`
- `#define DW_DLA_STR_OFFSETS 0x40`
- `#define DW_DLA_DEBUG_ADDR 0x41`

9.5.1 Detailed Description

These identify the various allocate/dealloc types. The allocation happens within libdwf, and the deallocation is usually done by user code.

9.6 DW_DLE Dwarf_Error numbers

Macros

- `#define DW_DLE_NE 0 /* no error */`
- `#define DW_DLE_VMM 1 /* dwarf format/library version mismatch */`
- `#define DW_DLE_MAP 2 /* memory map failure */`
- `#define DW_DLE_LEE 3 /* libelf error */`
- `#define DW_DLE_NDS 4 /* no debug section */`
- `#define DW_DLE_NLS 5 /* no line section */`
- `#define DW_DLE_ID 6 /* invalid descriptor for query */`
- `#define DW_DLE_IOF 7 /* I/O failure */`
- `#define DW_DLE_MAF 8 /* memory allocation failure */`
- `#define DW_DLE_IA 9 /* invalid argument */`
- `#define DW_DLE_MDE 10 /* mangled debugging entry */`
- `#define DW_DLE_MLE 11 /* mangled line number entry */`
- `#define DW_DLE_FNO 12 /* file not open */`
- `#define DW_DLE_FNR 13 /* file not a regular file */`
- `#define DW_DLE_FWA 14 /* file open with wrong access */`
- `#define DW_DLE_NOB 15 /* not an object file */`
- `#define DW_DLE_MOF 16 /* mangled object file header */`
- `#define DW_DLE_EOLL 17 /* end of location list entries */`
- `#define DW_DLE_NOLL 18 /* no location list section */`
- `#define DW_DLE_BADOFF 19 /* Invalid offset */`
- `#define DW_DLE_EOS 20 /* end of section */`
- `#define DW_DLE_ATRUNC 21 /* abbreviations section appears truncated*/`
- `#define DW_DLE_BADBITC 22 /* Address size passed to dwarf bad,*/`
- `#define DW_DLE_DBG_ALLOC 23`
- `#define DW_DLE_FSTAT_ERROR 24`
- `#define DW_DLE_FSTAT_MODE_ERROR 25`
- `#define DW_DLE_INIT_ACCESS_WRONG 26`
- `#define DW_DLE_ELF_BEGIN_ERROR 27`
- `#define DW_DLE_ELF_GETEHDR_ERROR 28`
- `#define DW_DLE_ELF_GETSHDR_ERROR 29`
- `#define DW_DLE_ELF_STRPTR_ERROR 30`
- `#define DW_DLE_DEBUG_INFO_DUPLICATE 31`
- `#define DW_DLE_DEBUG_INFO_NULL 32`
- `#define DW_DLE_DEBUG_ABBREV_DUPLICATE 33`
- `#define DW_DLE_DEBUG_ABBREV_NULL 34`
- `#define DW_DLE_DEBUG_ARANGES_DUPLICATE 35`
- `#define DW_DLE_DEBUG_ARANGES_NULL 36`
- `#define DW_DLE_DEBUG_LINE_DUPLICATE 37`
- `#define DW_DLE_DEBUG_LINE_NULL 38`
- `#define DW_DLE_DEBUG_LOC_DUPLICATE 39`
- `#define DW_DLE_DEBUG_LOC_NULL 40`
- `#define DW_DLE_DEBUG_MACINFO_DUPLICATE 41`
- `#define DW_DLE_DEBUG_MACINFO_NULL 42`
- `#define DW_DLE_DEBUG_PUBNAMES_DUPLICATE 43`
- `#define DW_DLE_DEBUG_PUBNAMES_NULL 44`
- `#define DW_DLE_DEBUG_STR_DUPLICATE 45`
- `#define DW_DLE_DEBUG_STR_NULL 46`
- `#define DW_DLE_CU_LENGTH_ERROR 47`
- `#define DW_DLE_VERSION_STAMP_ERROR 48`
- `#define DW_DLE_ABBREV_OFFSET_ERROR 49`

- `#define DW_DLE_ADDRESS_SIZE_ERROR 50`
- `#define DW_DLE_DEBUG_INFO_PTR_NULL 51`
- `#define DW_DLE_DIE_NULL 52`
- `#define DW_DLE_STRING_OFFSET_BAD 53`
- `#define DW_DLE_DEBUG_LINE_LENGTH_BAD 54`
- `#define DW_DLE_LINE_PROLOG_LENGTH_BAD 55`
- `#define DW_DLE_LINE_NUM_OPERANDS_BAD 56`
- `#define DW_DLE_LINE_SET_ADDR_ERROR 57`
- `#define DW_DLE_LINE_EXT_OPCODE_BAD 58`
- `#define DW_DLE_DWARF_LINE_NULL 59`
- `#define DW_DLE_INCL_DIR_NUM_BAD 60`
- `#define DW_DLE_LINE_FILE_NUM_BAD 61`
- `#define DW_DLE_ALLOC_FAIL 62`
- `#define DW_DLE_NO_CALLBACK_FUNC 63`
- `#define DW_DLE_SECT_ALLOC 64`
- `#define DW_DLE_FILE_ENTRY_ALLOC 65`
- `#define DW_DLE_LINE_ALLOC 66`
- `#define DW_DLE_FPGM_ALLOC 67`
- `#define DW_DLE_INCDIR_ALLOC 68`
- `#define DW_DLE_STRING_ALLOC 69`
- `#define DW_DLE_CHUNK_ALLOC 70`
- `#define DW_DLE_BYTEOFF_ERR 71`
- `#define DW_DLE_CIE_ALLOC 72`
- `#define DW_DLE_FDE_ALLOC 73`
- `#define DW_DLE_REGNO_OVFL 74`
- `#define DW_DLE_CIE_OFFS_ALLOC 75`
- `#define DW_DLE_WRONG_ADDRESS 76`
- `#define DW_DLE_EXTRA_NEIGHBORS 77`
- `#define DW_DLE_WRONG_TAG 78`
- `#define DW_DLE_DIE_ALLOC 79`
- `#define DW_DLE_PARENT_EXISTS 80`
- `#define DW_DLE_DBG_NULL 81`
- `#define DW_DLE_DEBUGLINE_ERROR 82`
- `#define DW_DLE_DEBUGFRAME_ERROR 83`
- `#define DW_DLE_DEBUGINFO_ERROR 84`
- `#define DW_DLE_ATTR_ALLOC 85`
- `#define DW_DLE_ABBREV_ALLOC 86`
- `#define DW_DLE_OFFSET_UFLW 87`
- `#define DW_DLE_ELF_SECT_ERR 88`
- `#define DW_DLE_DEBUG_FRAME_LENGTH_BAD 89`
- `#define DW_DLE_FRAME_VERSION_BAD 90`
- `#define DW_DLE_CIE_RET_ADDR_REG_ERROR 91`
- `#define DW_DLE_FDE_NULL 92`
- `#define DW_DLE_FDE_DBG_NULL 93`
- `#define DW_DLE_CIE_NULL 94`
- `#define DW_DLE_CIE_DBG_NULL 95`
- `#define DW_DLE_FRAME_TABLE_COL_BAD 96`
- `#define DW_DLE_PC_NOT_IN_FDE_RANGE 97`
- `#define DW_DLE_CIE_INSTR_EXEC_ERROR 98`
- `#define DW_DLE_FRAME_INSTR_EXEC_ERROR 99`
- `#define DW_DLE_FDE_PTR_NULL 100`
- `#define DW_DLE_RET_OP_LIST_NULL 101`
- `#define DW_DLE_LINE_CONTEXT_NULL 102`
- `#define DW_DLE_DBG_NO_CU_CONTEXT 103`
- `#define DW_DLE_DIE_NO_CU_CONTEXT 104`

- `#define DW_DLE_FIRST_DIE_NOT_CU` 105
- `#define DW_DLE_NEXT_DIE_PTR_NULL` 106
- `#define DW_DLE_DEBUG_FRAME_DUPLICATE` 107
- `#define DW_DLE_DEBUG_FRAME_NULL` 108
- `#define DW_DLE_ABBREV_DECODE_ERROR` 109
- `#define DW_DLE_DWARF_ABBREV_NULL` 110
- `#define DW_DLE_ATTR_NULL` 111
- `#define DW_DLE_DIE_BAD` 112
- `#define DW_DLE_DIE_ABBREV_BAD` 113
- `#define DW_DLE_ATTR_FORM_BAD` 114
- `#define DW_DLE_ATTR_NO_CU_CONTEXT` 115
- `#define DW_DLE_ATTR_FORM_SIZE_BAD` 116
- `#define DW_DLE_ATTR_DBG_NULL` 117
- `#define DW_DLE_BAD_REF_FORM` 118
- `#define DW_DLE_ATTR_FORM_OFFSET_BAD` 119
- `#define DW_DLE_LINE_OFFSET_BAD` 120
- `#define DW_DLE_DEBUG_STR_OFFSET_BAD` 121
- `#define DW_DLE_STRING_PTR_NULL` 122
- `#define DW_DLE_PUBNAMES_VERSION_ERROR` 123
- `#define DW_DLE_PUBNAMES_LENGTH_BAD` 124
- `#define DW_DLE_GLOBAL_NULL` 125
- `#define DW_DLE_GLOBAL_CONTEXT_NULL` 126
- `#define DW_DLE_DIR_INDEX_BAD` 127
- `#define DW_DLE_LOC_EXPR_BAD` 128
- `#define DW_DLE_DIE_LOC_EXPR_BAD` 129
- `#define DW_DLE_ADDR_ALLOC` 130
- `#define DW_DLE_OFFSET_BAD` 131
- `#define DW_DLE_MAKE_CU_CONTEXT_FAIL` 132
- `#define DW_DLE_REL_ALLOC` 133
- `#define DW_DLE_ARANGE_OFFSET_BAD` 134
- `#define DW_DLE_SEGMENT_SIZE_BAD` 135
- `#define DW_DLE_ARANGE_LENGTH_BAD` 136
- `#define DW_DLE_ARANGE_DECODE_ERROR` 137
- `#define DW_DLE_ARANGES_NULL` 138
- `#define DW_DLE_ARANGE_NULL` 139
- `#define DW_DLE_NO_FILE_NAME` 140
- `#define DW_DLE_NO_COMP_DIR` 141
- `#define DW_DLE_CU_ADDRESS_SIZE_BAD` 142
- `#define DW_DLE_INPUT_ATTR_BAD` 143
- `#define DW_DLE_EXPR_NULL` 144
- `#define DW_DLE_BAD_EXPR_OPCODE` 145
- `#define DW_DLE_EXPR_LENGTH_BAD` 146
- `#define DW_DLE_MULTIPLE_RELOC_IN_EXPR` 147
- `#define DW_DLE_ELF_GETIDENT_ERROR` 148
- `#define DW_DLE_NO_AT_MIPS_FDE` 149
- `#define DW_DLE_NO_CIE_FOR_FDE` 150
- `#define DW_DLE_DIE_ABBREV_LIST_NULL` 151
- `#define DW_DLE_DEBUG_FUNCNAMES_DUPLICATE` 152
- `#define DW_DLE_DEBUG_FUNCNAMES_NULL` 153
- `#define DW_DLE_DEBUG_FUNCNAMES_VERSION_ERROR` 154
- `#define DW_DLE_DEBUG_FUNCNAMES_LENGTH_BAD` 155
- `#define DW_DLE_FUNC_NULL` 156
- `#define DW_DLE_FUNC_CONTEXT_NULL` 157
- `#define DW_DLE_DEBUG_TYPENAMES_DUPLICATE` 158
- `#define DW_DLE_DEBUG_TYPENAMES_NULL` 159

- `#define DW_DLE_DEBUG_TYPENAMES_VERSION_ERROR` 160
- `#define DW_DLE_DEBUG_TYPENAMES_LENGTH_BAD` 161
- `#define DW_DLE_TYPE_NULL` 162
- `#define DW_DLE_TYPE_CONTEXT_NULL` 163
- `#define DW_DLE_DEBUG_VARNAME_DUPLICATE` 164
- `#define DW_DLE_DEBUG_VARNAME_NULL` 165
- `#define DW_DLE_DEBUG_VARNAME_VERSION_ERROR` 166
- `#define DW_DLE_DEBUG_VARNAME_LENGTH_BAD` 167
- `#define DW_DLE_VAR_NULL` 168
- `#define DW_DLE_VAR_CONTEXT_NULL` 169
- `#define DW_DLE_DEBUG_WEAKNAME_DUPLICATE` 170
- `#define DW_DLE_DEBUG_WEAKNAME_NULL` 171
- `#define DW_DLE_DEBUG_WEAKNAME_VERSION_ERROR` 172
- `#define DW_DLE_DEBUG_WEAKNAME_LENGTH_BAD` 173
- `#define DW_DLE_WEAK_NULL` 174
- `#define DW_DLE_WEAK_CONTEXT_NULL` 175
- `#define DW_DLE_LOCDISC_COUNT_WRONG` 176
- `#define DW_DLE_MACINFO_STRING_NULL` 177
- `#define DW_DLE_MACINFO_STRING_EMPTY` 178
- `#define DW_DLE_MACINFO_INTERNAL_ERROR_SPACE` 179
- `#define DW_DLE_MACINFO_MALLOC_FAIL` 180
- `#define DW_DLE_DEBUGMACINFO_ERROR` 181
- `#define DW_DLE_DEBUG_MACRO_LENGTH_BAD` 182
- `#define DW_DLE_DEBUG_MACRO_MAX_BAD` 183
- `#define DW_DLE_DEBUG_MACRO_INTERNAL_ERR` 184
- `#define DW_DLE_DEBUG_MACRO_MALLOC_SPACE` 185
- `#define DW_DLE_DEBUG_MACRO_INCONSISTENT` 186
- `#define DW_DLE_DF_NO_CIE_AUGMENTATION` 187
- `#define DW_DLE_DF_REG_NUM_TOO_HIGH` 188
- `#define DW_DLE_DF_MAKE_INSTR_NO_INIT` 189
- `#define DW_DLE_DF_NEW_LOC_LESS_OLD_LOC` 190
- `#define DW_DLE_DF_POP_EMPTY_STACK` 191
- `#define DW_DLE_DF_ALLOC_FAIL` 192
- `#define DW_DLE_DF_FRAME_DECODING_ERROR` 193
- `#define DW_DLE_DEBUG_LOC_SECTION_SHORT` 194
- `#define DW_DLE_FRAME_AUGMENTATION_UNKNOWN` 195
- `#define DW_DLE_PUBTYPE_CONTEXT` 196 /* Unused. */
- `#define DW_DLE_DEBUG_PUBTYPES_LENGTH_BAD` 197
- `#define DW_DLE_DEBUG_PUBTYPES_VERSION_ERROR` 198
- `#define DW_DLE_DEBUG_PUBTYPES_DUPLICATE` 199
- `#define DW_DLE_FRAME_CIE_DECODE_ERROR` 200
- `#define DW_DLE_FRAME_REGISTER_UNREPRESENTABLE` 201
- `#define DW_DLE_FRAME_REGISTER_COUNT_MISMATCH` 202
- `#define DW_DLE_LINK_LOOP` 203
- `#define DW_DLE_STRP_OFFSET_BAD` 204
- `#define DW_DLE_DEBUG_RANGES_DUPLICATE` 205
- `#define DW_DLE_DEBUG_RANGES_OFFSET_BAD` 206
- `#define DW_DLE_DEBUG_RANGES_MISSING_END` 207
- `#define DW_DLE_DEBUG_RANGES_OUT_OF_MEM` 208
- `#define DW_DLE_DEBUG_SYMTAB_ERR` 209
- `#define DW_DLE_DEBUG_STRTAB_ERR` 210
- `#define DW_DLE_RELOC_MISMATCH_INDEX` 211
- `#define DW_DLE_RELOC_MISMATCH_RELOC_INDEX` 212
- `#define DW_DLE_RELOC_MISMATCH_STRTAB_INDEX` 213
- `#define DW_DLE_RELOC_SECTION_MISMATCH` 214

- `#define DW_DLE_RELOC_SECTION_MISSING_INDEX` 215
- `#define DW_DLE_RELOC_SECTION_LENGTH_ODD` 216
- `#define DW_DLE_RELOC_SECTION_PTR_NULL` 217
- `#define DW_DLE_RELOC_SECTION_MALLOC_FAIL` 218
- `#define DW_DLE_NO_ELF64_SUPPORT` 219
- `#define DW_DLE_MISSING_ELF64_SUPPORT` 220
- `#define DW_DLE_ORPHAN_FDE` 221
- `#define DW_DLE_DUPLICATE_INST_BLOCK` 222
- `#define DW_DLE_BAD_REF_SIG8_FORM` 223
- `#define DW_DLE_ATTR_EXPRLOC_FORM_BAD` 224
- `#define DW_DLE_FORM_SEC_OFFSET_LENGTH_BAD` 225
- `#define DW_DLE_NOT_REF_FORM` 226
- `#define DW_DLE_DEBUG_FRAME_LENGTH_NOT_MULTIPLE` 227
- `#define DW_DLE_REF_SIG8_NOT_HANDLED` 228
- `#define DW_DLE_DEBUG_FRAME_POSSIBLE_ADDRESS_BOTCH` 229
- `#define DW_DLE_LOC_BAD_TERMINATION` 230
- `#define DW_DLE_SYMTAB_SECTION_LENGTH_ODD` 231
- `#define DW_DLE_RELOC_SECTION_SYMBOL_INDEX_BAD` 232
- `#define DW_DLE_RELOC_SECTION_RELOC_TARGET_SIZE_UNKNOWN` 233
- `#define DW_DLE_SYMTAB_SECTION_ENTRYSIZE_ZERO` 234
- `#define DW_DLE_LINE_NUMBER_HEADER_ERROR` 235
- `#define DW_DLE_DEBUG_TYPES_NULL` 236
- `#define DW_DLE_DEBUG_TYPES_DUPLICATE` 237
- `#define DW_DLE_DEBUG_TYPES_ONLY_DWARF4` 238
- `#define DW_DLE_DEBUG_TYPEOFFSET_BAD` 239
- `#define DW_DLE_GNU_OPCODE_ERROR` 240
- `#define DW_DLE_DEBUGPUBTYPES_ERROR` 241
- `#define DW_DLE_AT_FIXUP_NULL` 242
- `#define DW_DLE_AT_FIXUP_DUP` 243
- `#define DW_DLE_BAD_ABINAME` 244
- `#define DW_DLE_TOO_MANY_DEBUG` 245
- `#define DW_DLE_DEBUG_STR_OFFSETS_DUPLICATE` 246
- `#define DW_DLE_SECTION_DUPLICATION` 247
- `#define DW_DLE_SECTION_ERROR` 248
- `#define DW_DLE_DEBUG_ADDR_DUPLICATE` 249
- `#define DW_DLE_DEBUG_CU_UNAVAILABLE_FOR_FORM` 250
- `#define DW_DLE_DEBUG_FORM_HANDLING_INCOMPLETE` 251
- `#define DW_DLE_NEXT_DIE_PAST_END` 252
- `#define DW_DLE_NEXT_DIE_WRONG_FORM` 253
- `#define DW_DLE_NEXT_DIE_NO_ABBREV_LIST` 254
- `#define DW_DLE_NESTED_FORM_INDIRECT_ERROR` 255
- `#define DW_DLE_CU_DIE_NO_ABBREV_LIST` 256
- `#define DW_DLE_MISSING_NEEDED_DEBUG_ADDR_SECTION` 257
- `#define DW_DLE_ATTR_FORM_NOT_ADDR_INDEX` 258
- `#define DW_DLE_ATTR_FORM_NOT_STR_INDEX` 259
- `#define DW_DLE_DUPLICATE_GDB_INDEX` 260
- `#define DW_DLE_ERRONEOUS_GDB_INDEX_SECTION` 261
- `#define DW_DLE_GDB_INDEX_COUNT_ERROR` 262
- `#define DW_DLE_GDB_INDEX_COUNT_ADDR_ERROR` 263
- `#define DW_DLE_GDB_INDEX_INDEX_ERROR` 264
- `#define DW_DLE_GDB_INDEX_CUVEC_ERROR` 265
- `#define DW_DLE_DUPLICATE_CU_INDEX` 266
- `#define DW_DLE_DUPLICATE_TU_INDEX` 267
- `#define DW_DLE_XU_TYPE_ARG_ERROR` 268
- `#define DW_DLE_XU_IMPOSSIBLE_ERROR` 269

- `#define DW_DLE_XU_NAME_COL_ERROR` 270
- `#define DW_DLE_XU_HASH_ROW_ERROR` 271
- `#define DW_DLE_XU_HASH_INDEX_ERROR` 272
- `#define DW_DLE_FAILSAFE_ERRVAL` 273
- `#define DW_DLE_ARANGE_ERROR` 274
- `#define DW_DLE_PUBNAMES_ERROR` 275
- `#define DW_DLE_FUNCNAMES_ERROR` 276
- `#define DW_DLE_TYPENAMES_ERROR` 277
- `#define DW_DLE_VARNAME_ERROR` 278
- `#define DW_DLE_WEAKNAMES_ERROR` 279
- `#define DW_DLE_RELOCS_ERROR` 280
- `#define DW_DLE_ATTR_OUTSIDE_SECTION` 281
- `#define DW_DLE_FFISSION_INDEX_WRONG` 282
- `#define DW_DLE_FFISSION_VERSION_ERROR` 283
- `#define DW_DLE_NEXT_DIE_LOW_ERROR` 284
- `#define DW_DLE_CU_UT_TYPE_ERROR` 285
- `#define DW_DLE_NO_SUCH_SIGNATURE_FOUND` 286
- `#define DW_DLE_SIGNATURE_SECTION_NUMBER_WRONG` 287
- `#define DW_DLE_ATTR_FORM_NOT_DATA8` 288
- `#define DW_DLE_SIG_TYPE_WRONG_STRING` 289
- `#define DW_DLE_MISSING_REQUIRED_TU_OFFSET_HASH` 290
- `#define DW_DLE_MISSING_REQUIRED_CU_OFFSET_HASH` 291
- `#define DW_DLE_DWP_MISSING_DWO_ID` 292
- `#define DW_DLE_DWP_SIBLING_ERROR` 293
- `#define DW_DLE_DEBUG_FFISSION_INCOMPLETE` 294
- `#define DW_DLE_FFISSION_SECNUM_ERR` 295
- `#define DW_DLE_DEBUG_MACRO_DUPLICATE` 296
- `#define DW_DLE_DEBUG_NAMES_DUPLICATE` 297
- `#define DW_DLE_DEBUG_LINE_STR_DUPLICATE` 298
- `#define DW_DLE_DEBUG_SUP_DUPLICATE` 299
- `#define DW_DLE_NO_SIGNATURE_TO_LOOKUP` 300
- `#define DW_DLE_NO_TIED_ADDR_AVAILABLE` 301
- `#define DW_DLE_NO_TIED_SIG_AVAILABLE` 302
- `#define DW_DLE_STRING_NOT_TERMINATED` 303
- `#define DW_DLE_BAD_LINE_TABLE_OPERATION` 304
- `#define DW_DLE_LINE_CONTEXT_BOTCH` 305
- `#define DW_DLE_LINE_CONTEXT_INDEX_WRONG` 306
- `#define DW_DLE_NO_TIED_STRING_AVAILABLE` 307
- `#define DW_DLE_NO_TIED_FILE_AVAILABLE` 308
- `#define DW_DLE_CU_TYPE_MISSING` 309
- `#define DW_DLE_LLE_CODE_UNKNOWN` 310
- `#define DW_DLE_LOCLIST_INTERFACE_ERROR` 311
- `#define DW_DLE_LOCLIST_INDEX_ERROR` 312
- `#define DW_DLE_INTERFACE_NOT_SUPPORTED` 313
- `#define DW_DLE_ZDEBUG_REQUIRES_ZLIB` 314
- `#define DW_DLE_ZDEBUG_INPUT_FORMAT_ODD` 315
- `#define DW_DLE_ZLIB_BUF_ERROR` 316
- `#define DW_DLE_ZLIB_DATA_ERROR` 317
- `#define DW_DLE_MACRO_OFFSET_BAD` 318
- `#define DW_DLE_MACRO_OPCODE_BAD` 319
- `#define DW_DLE_MACRO_OPCODE_FORM_BAD` 320
- `#define DW_DLE_UNKNOWN_FORM` 321
- `#define DW_DLE_BAD_MACRO_HEADER_POINTER` 322
- `#define DW_DLE_BAD_MACRO_INDEX` 323
- `#define DW_DLE_MACRO_OP_UNHANDLED` 324

- `#define DW_DLE_MACRO_PAST_END` 325
- `#define DW_DLE_LINE_STRP_OFFSET_BAD` 326
- `#define DW_DLE_STRING_FORM_IMPROPER` 327
- `#define DW_DLE_ELF_FLAGS_NOT_AVAILABLE` 328
- `#define DW_DLE_LEB_IMPROPER` 329
- `#define DW_DLE_DEBUG_LINE_RANGE_ZERO` 330
- `#define DW_DLE_READ_LITTLEENDIAN_ERROR` 331
- `#define DW_DLE_READ_BIGENDIAN_ERROR` 332
- `#define DW_DLE_RELOC_INVALID` 333
- `#define DW_DLE_INFO_HEADER_ERROR` 334
- `#define DW_DLE_ARANGES_HEADER_ERROR` 335
- `#define DW_DLE_LINE_OFFSET_WRONG_FORM` 336
- `#define DW_DLE_FORM_BLOCK_LENGTH_ERROR` 337
- `#define DW_DLE_ZLIB_SECTION_SHORT` 338
- `#define DW_DLE_CIE_INSTR_PTR_ERROR` 339
- `#define DW_DLE_FDE_INSTR_PTR_ERROR` 340
- `#define DW_DLE_FFISSION_ADDITION_ERROR` 341
- `#define DW_DLE_HEADER_LEN_BIGGER_THAN_SECSIZE` 342
- `#define DW_DLE_LOEXPR_OFF_SECTION_END` 343
- `#define DW_DLE_POINTER_SECTION_UNKNOWN` 344
- `#define DW_DLE_ERRONEOUS_XU_INDEX_SECTION` 345
- `#define DW_DLE_DIRECTORY_FORMAT_COUNT_VS_DIRECTORIES_MISMATCH` 346
- `#define DW_DLE_COMPRESSED_EMPTY_SECTION` 347
- `#define DW_DLE_SIZE_WRAPAROUND` 348
- `#define DW_DLE_ILLOGICAL_TSEARCH` 349
- `#define DW_DLE_BAD_STRING_FORM` 350
- `#define DW_DLE_DEBUGSTR_ERROR` 351
- `#define DW_DLE_DEBUGSTR_UNEXPECTED_REL` 352
- `#define DW_DLE_DISCR_ARRAY_ERROR` 353
- `#define DW_DLE_LEB_OUT_ERROR` 354
- `#define DW_DLE_SIBLING_LIST_IMPROPER` 355
- `#define DW_DLE_LOCLIST_OFFSET_BAD` 356
- `#define DW_DLE_LINE_TABLE_BAD` 357
- `#define DW_DLE_DEBUG_LOCLISTS_DUPLICATE` 358
- `#define DW_DLE_DEBUG_RNGLISTS_DUPLICATE` 359
- `#define DW_DLE_ABBREV_OFF_END` 360
- `#define DW_DLE_FORM_STRING_BAD_STRING` 361
- `#define DW_DLE_AUGMENTATION_STRING_OFF_END` 362
- `#define DW_DLE_STRING_OFF_END_PUBNAMES_LIKE` 363
- `#define DW_DLE_LINE_STRING_BAD` 364
- `#define DW_DLE_DEFINE_FILE_STRING_BAD` 365
- `#define DW_DLE_MACRO_STRING_BAD` 366
- `#define DW_DLE_MACINFO_STRING_BAD` 367
- `#define DW_DLE_ZLIB_UNCOMPRESS_ERROR` 368
- `#define DW_DLE_IMPROPER_DWO_ID` 369
- `#define DW_DLE_GROUPNUMBER_ERROR` 370
- `#define DW_DLE_ADDRESS_SIZE_ZERO` 371
- `#define DW_DLE_DEBUG_NAMES_HEADER_ERROR` 372
- `#define DW_DLE_DEBUG_NAMES_AUG_STRING_ERROR` 373
- `#define DW_DLE_DEBUG_NAMES_PAD_NON_ZERO` 374
- `#define DW_DLE_DEBUG_NAMES_OFF_END` 375
- `#define DW_DLE_DEBUG_NAMES_ABBREV_OVERFLOW` 376
- `#define DW_DLE_DEBUG_NAMES_ABBREV_CORRUPTION` 377
- `#define DW_DLE_DEBUG_NAMES_NULL_POINTER` 378
- `#define DW_DLE_DEBUG_NAMES_BAD_INDEX_ARG` 379

- `#define DW_DLE_DEBUG_NAMES_ENTRYPOOL_OFFSET` 380
- `#define DW_DLE_DEBUG_NAMES_UNHANDLED_FORM` 381
- `#define DW_DLE_LNCT_CODE_UNKNOWN` 382
- `#define DW_DLE_LNCT_FORM_CODE_NOT_HANDLED` 383
- `#define DW_DLE_LINE_HEADER_LENGTH_BOTCH` 384
- `#define DW_DLE_STRING_HASHTAB_IDENTITY_ERROR` 385
- `#define DW_DLE_UNIT_TYPE_NOT_HANDLED` 386
- `#define DW_DLE_GROUP_MAP_ALLOC` 387
- `#define DW_DLE_GROUP_MAP_DUPLICATE` 388
- `#define DW_DLE_GROUP_COUNT_ERROR` 389
- `#define DW_DLE_GROUP_INTERNAL_ERROR` 390
- `#define DW_DLE_GROUP_LOAD_ERROR` 391
- `#define DW_DLE_GROUP_LOAD_READ_ERROR` 392
- `#define DW_DLE_AUG_DATA_LENGTH_BAD` 393
- `#define DW_DLE_ABBREV_MISSING` 394
- `#define DW_DLE_NO_TAG_FOR_DIE` 395
- `#define DW_DLE_LOWPC_WRONG_CLASS` 396
- `#define DW_DLE_HIGHPC_WRONG_FORM` 397
- `#define DW_DLE_STR_OFFSETS_BASE_WRONG_FORM` 398
- `#define DW_DLE_DATA16_OUTSIDE_SECTION` 399
- `#define DW_DLE_LNCT_MD5_WRONG_FORM` 400
- `#define DW_DLE_LINE_HEADER_CORRUPT` 401
- `#define DW_DLE_STR_OFFSETS_NULLARGUMENT` 402
- `#define DW_DLE_STR_OFFSETS_NULL_DBG` 403
- `#define DW_DLE_STR_OFFSETS_NO_MAGIC` 404
- `#define DW_DLE_STR_OFFSETS_ARRAY_SIZE` 405
- `#define DW_DLE_STR_OFFSETS_VERSION_WRONG` 406
- `#define DW_DLE_STR_OFFSETS_ARRAY_INDEX_WRONG` 407
- `#define DW_DLE_STR_OFFSETS_EXTRA_BYTES` 408
- `#define DW_DLE_DUP_ATTR_ON_DIE` 409
- `#define DW_DLE_SECTION_NAME_BIG` 410
- `#define DW_DLE_FILE_UNAVAILABLE` 411
- `#define DW_DLE_FILE_WRONG_TYPE` 412
- `#define DW_DLE_SIBLING_OFFSET_WRONG` 413
- `#define DW_DLE_OPEN_FAIL` 414
- `#define DW_DLE_OFFSET_SIZE` 415
- `#define DW_DLE_MACH_O_SEGOFFSET_BAD` 416
- `#define DW_DLE_FILE_OFFSET_BAD` 417
- `#define DW_DLE_SEEK_ERROR` 418
- `#define DW_DLE_READ_ERROR` 419
- `#define DW_DLE_ELF_CLASS_BAD` 420
- `#define DW_DLE_ELF_ENDIAN_BAD` 421
- `#define DW_DLE_ELF_VERSION_BAD` 422
- `#define DW_DLE_FILE_TOO_SMALL` 423
- `#define DW_DLE_PATH_SIZE_TOO_SMALL` 424
- `#define DW_DLE_BAD_TYPE_SIZE` 425
- `#define DW_DLE_PE_SIZE_SMALL` 426
- `#define DW_DLE_PE_OFFSET_BAD` 427
- `#define DW_DLE_PE_STRING_TOO_LONG` 428
- `#define DW_DLE_IMAGE_FILE_UNKNOWN_TYPE` 429
- `#define DW_DLE_LINE_TABLE_LINENO_ERROR` 430
- `#define DW_DLE_PRODUCER_CODE_NOT_AVAILABLE` 431
- `#define DW_DLE_NO_ELF_SUPPORT` 432
- `#define DW_DLE_NO_STREAM_RELOC_SUPPORT` 433
- `#define DW_DLE_RETURN_EMPTY_PUBNAMES_ERROR` 434

- `#define DW_DLE_SECTION_SIZE_ERROR` 435
- `#define DW_DLE_INTERNAL_NULL_POINTER` 436
- `#define DW_DLE_SECTION_STRING_OFFSET_BAD` 437
- `#define DW_DLE_SECTION_INDEX_BAD` 438
- `#define DW_DLE_INTEGER_TOO_SMALL` 439
- `#define DW_DLE_ELF_SECTION_LINK_ERROR` 440
- `#define DW_DLE_ELF_SECTION_GROUP_ERROR` 441
- `#define DW_DLE_ELF_SECTION_COUNT_MISMATCH` 442
- `#define DW_DLE_ELF_STRING_SECTION_MISSING` 443
- `#define DW_DLE_SEEK_OFF_END` 444
- `#define DW_DLE_READ_OFF_END` 445
- `#define DW_DLE_ELF_SECTION_ERROR` 446
- `#define DW_DLE_ELF_STRING_SECTION_ERROR` 447
- `#define DW_DLE_MIXING_SPLIT_DWARF_VERSIONS` 448
- `#define DW_DLE_TAG_CORRUPT` 449
- `#define DW_DLE_FORM_CORRUPT` 450
- `#define DW_DLE_ATTR_CORRUPT` 451
- `#define DW_DLE_ABBREV_ATTR_DUPLICATION` 452
- `#define DW_DLE_DWP_SIGNATURE_MISMATCH` 453
- `#define DW_DLE_CU_UT_TYPE_VALUE` 454
- `#define DW_DLE_DUPLICATE_GNU_DEBUGLINK` 455
- `#define DW_DLE_CORRUPT_GNU_DEBUGLINK` 456
- `#define DW_DLE_CORRUPT_NOTE_GNU_DEBUGID` 457
- `#define DW_DLE_CORRUPT_GNU_DEBUGID_SIZE` 458
- `#define DW_DLE_CORRUPT_GNU_DEBUGID_STRING` 459
- `#define DW_DLE_HEX_STRING_ERROR` 460
- `#define DW_DLE_DECIMAL_STRING_ERROR` 461
- `#define DW_DLE_PRO_INIT_EXTRAS_UNKNOWN` 462
- `#define DW_DLE_PRO_INIT_EXTRAS_ERR` 463
- `#define DW_DLE_NULL_ARGS_DWARF_ADD_PATH` 464
- `#define DW_DLE_DWARF_INIT_DBG_NULL` 465
- `#define DW_DLE_ELF_RELOC_SECTION_ERROR` 466
- `#define DW_DLE_USER_DECLARED_ERROR` 467
- `#define DW_DLE_RNGLISTS_ERROR` 468
- `#define DW_DLE_LOCLISTS_ERROR` 469
- `#define DW_DLE_SECTION_SIZE_OR_OFFSET_LARGE` 470
- `#define DW_DLE_GDBINDEX_STRING_ERROR` 471
- `#define DW_DLE_GNU_PUBNAMES_ERROR` 472
- `#define DW_DLE_GNU_PUBTYPES_ERROR` 473
- `#define DW_DLE_DUPLICATE_GNU_DEBUG_PUBNAMES` 474
- `#define DW_DLE_DUPLICATE_GNU_DEBUG_PUBTYPES` 475
- `#define DW_DLE_DEBUG_SUP_STRING_ERROR` 476
- `#define DW_DLE_DEBUG_SUP_ERROR` 477
- `#define DW_DLE_LOCATION_ERROR` 478
- `#define DW_DLE_DEBUGLINK_PATH_SHORT` 479
- `#define DW_DLE_SIGNATURE_MISMATCH` 480
- `#define DW_DLE_MACRO_VERSION_ERROR` 481
- `#define DW_DLE_NEGATIVE_SIZE` 482
- `#define DW_DLE_UDATA_VALUE_NEGATIVE` 483
- `#define DW_DLE_DEBUG_NAMES_ERROR` 484
- `#define DW_DLE_CFA_INSTRUCTION_ERROR` 485
- `#define DW_DLE_MACHO_CORRUPT_HEADER` 486
- `#define DW_DLE_MACHO_CORRUPT_COMMAND` 487
- `#define DW_DLE_MACHO_CORRUPT_SECTIONDETAILS` 488
- `#define DW_DLE_RELOCATION_SECTION_SIZE_ERROR` 489

- `#define DW_DLE_SYMBOL_SECTION_SIZE_ERROR` 490
- `#define DW_DLE_PE_SECTION_SIZE_ERROR` 491
- `#define DW_DLE_DEBUG_ADDR_ERROR` 492
- `#define DW_DLE_NO_SECT_STRINGS` 493
- `#define DW_DLE_TOO_FEW_SECTIONS` 494
- `#define DW_DLE_BUILD_ID_DESCRIPTION_SIZE` 495
- `#define DW_DLE_BAD_SECTION_FLAGS` 496
- `#define DW_DLE_IMPROPER_SECTION_ZERO` 497
- `#define DW_DLE_INVALID_NULL_ARGUMENT` 498
- `#define DW_DLE_LINE_INDEX_WRONG` 499
- `#define DW_DLE_LINE_COUNT_WRONG` 500
- `#define DW_DLE_ARITHMETIC_OVERFLOW` 501
- `#define DW_DLE_UNIVERSAL_BINARY_ERROR` 502
- `#define DW_DLE_UNIV_BIN_OFFSET_SIZE_ERROR` 503
- `#define DW_DLE_PE_SECTION_SIZE_HEURISTIC_FAIL` 504
- `#define DW_DLE_LLE_ERROR` 505
- `#define DW_DLE_RLE_ERROR` 506
- `#define DW_DLE_MACHO_SEGMENT_COUNT_HEURISTIC_FAIL` 507
- `#define DW_DLE_DUPLICATE_NOTE_GNU_BUILD_ID` 508
- `#define DW_DLE_SYSCONF_VALUE_UNUSABLE` 509
- `#define DW_DLE_FRAME_ITERATOR_ERR` 510
- `#define DW_DLE_FRAME_FDE_TABLE_ERR` 511
- `#define DW_DLE_LAST` 511
- `#define DW_DLE_LO_USER` 0x10000

9.6.1 Detailed Description

These identify the various error codes that have been used. Not all of them are still use. We do not recycle obsolete codes into new uses. The codes 1 through 22 are historic and it is unlikely they are used anywhere in the library.

9.6.2 Macro Definition Documentation

9.6.2.1 DW_DLE_LAST

```
#define DW_DLE_LAST 511
```

Note

DW_DLE_LAST MUST EQUAL LAST ERROR NUMBER

9.7 Libdwarf Initialization Functions

Functions

- DW_API int [dwarf_init_path](#) (const char *dw_path, char *dw_true_path_out_buffer, unsigned int dw_true_path_bufferlen, unsigned int dw_groupnumber, [Dwarf_Handler](#) dw_errhand, [Dwarf_Ptr](#) dw_errarg, [Dwarf_Debug](#) *dw_dbg, [Dwarf_Error](#) *dw_error)
Initialization based on path, the most common initialization.
- DW_API int [dwarf_init_path_a](#) (const char *dw_path, char *dw_true_path_out_buffer, unsigned int dw_true_path_bufferlen, unsigned int dw_groupnumber, unsigned int dw_universalnumber, [Dwarf_Handler](#) dw_errhand, [Dwarf_Ptr](#) dw_errarg, [Dwarf_Debug](#) *dw_dbg, [Dwarf_Error](#) *dw_error)
Initialization based on path.
- DW_API int [dwarf_init_path_dl](#) (const char *dw_path, char *dw_true_path_out_buffer, unsigned int dw_true_path_bufferlen, unsigned int dw_groupnumber, [Dwarf_Handler](#) dw_errhand, [Dwarf_Ptr](#) dw_errarg, [Dwarf_Debug](#) *dw_dbg, char **dw_dl_path_array, unsigned int dw_dl_path_array_size, unsigned char *dw_dl_path_source, [Dwarf_Error](#) *dw_error)
Initialization following GNU debuglink section data.
- DW_API int [dwarf_init_path_dl_a](#) (const char *dw_path, char *dw_true_path_out_buffer, unsigned int dw_true_path_bufferlen, unsigned int dw_groupnumber, unsigned int dw_universalnumber, [Dwarf_Handler](#) dw_errhand, [Dwarf_Ptr](#) dw_errarg, [Dwarf_Debug](#) *dw_dbg, char **dw_dl_path_array, unsigned int dw_dl_path_array_size, unsigned char *dw_dl_path_source, [Dwarf_Error](#) *dw_error)
Initialization based on path with debuglink.
- DW_API int [dwarf_init_b](#) (int dw_fd, unsigned int dw_groupnumber, [Dwarf_Handler](#) dw_errhand, [Dwarf_Ptr](#) dw_errarg, [Dwarf_Debug](#) *dw_dbg, [Dwarf_Error](#) *dw_error)
Initialization based on Unix/Linux (etc) fd.
- DW_API int [dwarf_finish](#) ([Dwarf_Debug](#) dw_dbg)
Close the initialized dw_dbg and free all data libdwarf has for this dw_dbg.
- DW_API int [dwarf_object_init_b](#) ([Dwarf_Obj_Access_Interface_a](#) *dw_obj, [Dwarf_Handler](#) dw_errhand, [Dwarf_Ptr](#) dw_errarg, unsigned int dw_groupnumber, [Dwarf_Debug](#) *dw_dbg, [Dwarf_Error](#) *dw_error)
Used to access DWARF information in memory or in an object format unknown to libdwarf.
- DW_API int [dwarf_object_finish](#) ([Dwarf_Debug](#) dw_dbg)
Used to close the object_init dw_dbg.
- DW_API int [dwarf_set_tied_dbg](#) ([Dwarf_Debug](#) dw_split_dbg, [Dwarf_Debug](#) dw_tied_dbg, [Dwarf_Error](#) *dw_error)
Use with split dwarf.
- DW_API int [dwarf_get_tied_dbg](#) ([Dwarf_Debug](#) dw_dbg, [Dwarf_Debug](#) *dw_tieddbg_out, [Dwarf_Error](#) *dw_error)
Use with split dwarf.

9.7.1 Detailed Description

9.7.2 Initialization And Finish Operations

Opening and closing libdwarf on object files.

9.7.3 Function Documentation

9.7.3.1 dwarf_finish()

```
DW_API int dwarf_finish (
    Dwarf\_Debug dw_dbg )
```

Close the initialized dw_dbg and free all data libdwarf has for this dw_dbg.

Parameters

<i>dw_dbg</i>	Close the dbg.
---------------	----------------

Returns

May return DW_DLV_ERROR if something is very wrong: no further information is available.. May return DW_DLV_NO_ENTRY but no further information is available. Normally returns DW_DLV_OK.

There is nothing the caller can do with the return value except report it somehow. Most callers ignore the return value.

9.7.3.2 dwarf_get_tied_dbg()

```
DW_API int dwarf_get_tied_dbg (
    Dwarf_Debug dw_dbg,
    Dwarf_Debug * dw_tieddbg_out,
    Dwarf_Error * dw_error )
```

Use with split dwarf.

Given a main Dwarf_Debug this returns the tied Dwarf_Debug if there is one or else returns null(0).

Before v0.11.0 it was not defined what this returned if the tied-Dwarf_Debug was passed in, but it would have returned null(0) in that case. Unlikely anyone uses this call as callers had the tied and base dbg when calling [dwarf_set_tied_dbg\(\)](#).

Parameters

<i>dw_dbg</i>	Pass in a non-null Dwarf_Debug which is either a main-Dwarf_Debug or a tied-Dwarf_Debug.
<i>dw_tieddbg_out</i>	On success returns the applicable tied-Dwarf_Debug through the pointer. If dw_dbg is a tied-Dwarf_Debug the function returns null(0) through the pointer. If there is no tied-Dwarf_Debug (meaning there is just a main-Dwarf_Debug) the function returns null (0) through the pointer.
<i>dw_error</i>	If the dw_dbg is invalid or damaged then the function returns DW_DLV_ERROR and dw_error is set to point to the error details.

Returns

DW_DLV_OK or DW_DLV_ERROR. Never returns DW_DLV_NO_ENTRY.

9.7.3.3 dwarf_init_b()

```
DW_API int dwarf_init_b (
    int dw_fd,
    unsigned int dw_groupnumber,
    Dwarf_Handler dw_errhand,
    Dwarf_Ptr dw_errarg,
```

```
Dwarf_Debug * dw_dbg,
Dwarf_Error * dw_error )
```

Initialization based on Unix/Linux (etc) fd.

In case DW_DLV_ERROR returned be sure to call dwarf_dealloc_error even though the returned Dwarf_Debug is NULL.

Parameters

<i>dw_fd</i>	An open Unix/Linux/etc fd on the object file.
<i>dw_groupnumber</i>	The value passed in should be DW_GROUPNUMBER_ANY unless one wishes to other than a standard group.
<i>dw_errhand</i>	Pass in NULL unless one wishes libdwarf to call this error handling function (which you must write) instead of passing meaningful values to the dw_error argument.
<i>dw_errarg</i>	If dw_errhand is non-null, then this value (a pointer or integer that means something to you) is passed to the dw_errhand function in case that is helpful to you.
<i>dw_dbg</i>	On success, *dw_dbg is set to a pointer to a new Dwarf_Debug structure to be used in calls to libdwarf functions.
<i>dw_error</i>	In case return is DW_DLV_ERROR dw_error is set to point to the error details.

Returns

DW_DLV_OK etc.

9.7.3.4 dwarf_init_path()

```
DW_API int dwarf_init_path (
    const char * dw_path,
    char * dw_true_path_out_buffer,
    unsigned int dw_true_path_bufferlen,
    unsigned int dw_groupnumber,
    Dwarf_Handler dw_errhand,
    Dwarf_Ptr dw_errarg,
    Dwarf_Debug * dw_dbg,
    Dwarf_Error * dw_error )
```

Initialization based on path, the most common initialization.

On a Mach-O universal binary this function can only return information about the first (zero index) object in the universal binary.

Parameters

<i>dw_path</i>	Pass in the path to the object file to open.
<i>dw_true_path_out_buffer</i>	Pass in NULL or the name of a string buffer (The buffer should be initialized with an initial NUL byte) The returned string will be null-terminated. The path actually used is copied to true_path_out. If true_path_buffer len is zero or true_path_out_buffer is zero then the Special Macos processing will not occur, nor will the GNU_debuglink processing occur. In case GNU debuglink data was followed or Macos dSYM applies the true_path_out will not match path and the initial byte will be non-null. The value put in true_path_out is the actual file name.
<i>dw_true_path_bufferlen</i>	Pass in the length in bytes of the buffer.

Parameters

<i>dw_groupnumber</i>	The value passed in should be DW_GROUPNUMBER_ANY unless one wishes to other than a standard group.
<i>dw_errhand</i>	Pass in NULL unless one wishes libdwarf to call this error handling function (which you must write) instead of passing meaningful values to the <i>dw_error</i> argument.
<i>dw_errarg</i>	If <i>dw_errhand</i> is non-null, then this value (a pointer or integer that means something to you) is passed to the <i>dw_errhand</i> function in case that is helpful to you.
<i>dw_dbg</i>	On success, <i>*dw_dbg</i> is set to a pointer to a new Dwarf_Debug structure to be used in calls to libdwarf functions.
<i>dw_error</i>	In case return is DW_DLV_ERROR <i>dw_error</i> is set to point to the error details.

Returns

DW_DLV_OK etc.

[Details on separate DWARF object access](#)

See also

[dwarf_init_path_dl](#) [dwarf_init_b](#)

[Using dwarf_init_path\(\)](#)

9.7.3.5 dwarf_init_path_a()

```
DW_API int dwarf_init_path_a (
    const char * dw_path,
    char * dw_true_path_out_buffer,
    unsigned int dw_true_path_bufferlen,
    unsigned int dw_groupnumber,
    unsigned int dw_universalnumber,
    Dwarf_Handler dw_errhand,
    Dwarf_Ptr dw_errarg,
    Dwarf_Debug * dw_dbg,
    Dwarf_Error * dw_error )
```

Initialization based on path.

This identical to [dwarf_init_path\(\)](#) except that it adds a new argument, *dw_universalnumber*, with which you can specify which object in a Mach-O universal binary you wish to open.

It is always safe and appropriate to pass zero as the *dw_universalnumber*. Elf and PE and (non-universal) Mach-O object files ignore the value of *dw_universalnumber*.

9.7.3.6 dwarf_init_path_dl()

```
DW_API int dwarf_init_path_dl (
    const char * dw_path,
    char * dw_true_path_out_buffer,
    unsigned int dw_true_path_bufferlen,
    unsigned int dw_groupnumber,
    Dwarf_Handler dw_errhand,
    Dwarf_Ptr dw_errarg,
    Dwarf_Debug * dw_dbg,
    char ** dw_dl_path_array,
    unsigned int dw_dl_path_array_size,
    unsigned char * dw_dl_path_source,
    Dwarf_Error * dw_error )
```

Initialization following GNU debuglink section data.

Sets the true-path with DWARF if there is appropriate debuglink data available.

In case DW_DLV_ERROR returned be sure to call dwarf_dealloc_error even though the returned Dwarf_Debug is NULL.

Parameters

<i>dw_path</i>	Pass in the path to the object file to open.
<i>dw_true_path_out_buffer</i>	Pass in NULL or the name of a string buffer.
<i>dw_true_path_bufferlen</i>	Pass in the length in bytes of the buffer.
<i>dw_groupnumber</i>	The value passed in should be DW_GROUPNUMBER_ANY unless one wishes to other than a standard group.
<i>dw_errhand</i>	Pass in NULL, normally. If non-null one wishes libdwarf to call this error handling function (which you must write) instead of passing meaningful values to the dw_error argument.
<i>dw_errarg</i>	Pass in NULL, normally. If dw_errorhand is non-null, then this value (a pointer or integer that means something to you) is passed to the dw_errhand function in case that is helpful to you.
<i>dw_dbg</i>	On success, *dw_dbg is set to a pointer to a new Dwarf_Debug structure to be used in calls to libdwarf functions.
<i>dw_dl_path_array</i>	debuglink processing allows a user-specified set of file paths and this argument allows one to specify these. Pass in a pointer to array of pointers to strings which you, the caller, have filled in. The strings should be alternate paths (see the GNU debuglink documentation.)
<i>dw_dl_path_array_size</i>	Specify the size of the dw_dl_path_array.
<i>dw_dl_path_source</i>	returns DW_PATHSOURCE_basic or other such value so the caller can know how the true-path was resolved.
<i>dw_error</i>	In case return is DW_DLV_ERROR dw_error is set to point to the error details.

Returns

DW_DLV_OK etc.

[Details on separate DWARF object access](#)

See also

[Using dwarf_init_path_dl\(\)](#)

9.7.3.7 dwarf_init_path_dl_a()

```
DW_API int dwarf_init_path_dl_a (
    const char * dw_path,
    char * dw_true_path_out_buffer,
    unsigned int dw_true_path_bufferlen,
    unsigned int dw_groupnumber,
    unsigned int dw_universalnumber,
    Dwarf_Handler dw_errhand,
    Dwarf_Ptr dw_errarg,
    Dwarf_Debug * dw_dbg,
    char ** dw_dl_path_array,
    unsigned int dw_dl_path_array_size,
    unsigned char * dw_dl_path_source,
    Dwarf_Error * dw_error )
```

Initialization based on path with debuglink.

This identical to [dwarf_init_path_dl\(\)](#) except that it adds a new argument, `dw_universalnumber`, with which you can specify which object in a Mach-O universal binary you wish to open.

It is always safe and appropriate to pass zero as the `dw_universalnumber`. Elf and PE and (non-universal) Mach-O object files ignore the value of `dw_universalnumber`.

Mach-O objects do not contain or use debuglink data.

9.7.3.8 dwarf_object_finish()

```
DW_API int dwarf_object_finish (
    Dwarf_Debug dw_dbg )
```

Used to close the object_init `dw_dbg`.

Close the `dw_dbg` opened by [dwarf_object_init_b\(\)](#).

Parameters

<code>dw_dbg</code>	Must be an open <code>Dwarf_Debug</code> opened by dwarf_object_init_b() . The init call <code>dw_obj</code> data is not freed by the call to <code>dwarf_object_finish</code> .
---------------------	--

Returns

The return value `DW_DLV_OK` etc is useless, one could possibly report it somehow. Callers usually ignore the return value.

9.7.3.9 dwarf_object_init_b()

```
DW_API int dwarf_object_init_b (
    Dwarf_Obj_Access_Interface_a * dw_obj,
    Dwarf_Handler dw_errhand,
    Dwarf_Ptr dw_errarg,
```

```

    unsigned int dw_groupnumber,
    Dwarf_Debug * dw_dbg,
    Dwarf_Error * dw_error )

```

Used to access DWARF information in memory or in an object format unknown to libdwarf.

In case DW_DLV_ERROR returned be sure to call dwarf_dealloc_error even though the returned Dwarf_Debug is NULL.

Since libdwarf is not reading the object directly in this case it us up to the code actually reading the object to check the object file for format and do sufficient format-specific checks for correctness and return DW_DLV_ERROR if object checks fail.

See also

userobjread <src/bin/dwarfexample/jitreader.c>

Parameters

<i>dw_obj</i>	A data structure filled out by the caller so libdwarf can access DWARF data not in a supported object file format.
<i>dw_errhand</i>	Pass in NULL normally.
<i>dw_groupnumber</i>	The value passed in should be DW_GROUPNUMBER_ANY unless one wishes to other than a standard group (quite unlikely for this interface).
<i>dw_dbg</i>	On success, *dw_dbg is set to a pointer to a new Dwarf_Debug structure to be used in calls to libdwarf functions.
<i>dw_error</i>	In case return is DW_DLV_ERROR dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc.

9.7.3.10 dwarf_set_tied_dbg()

```

DW_API int dwarf_set_tied_dbg (
    Dwarf_Debug dw_split_dbg,
    Dwarf_Debug dw_tied_dbg,
    Dwarf_Error * dw_error )

```

Use with split dwarf.

In libdwarf usage the object file being reported on [a] is opened with [dwarf_init_path\(\)](#) or the like. If that object file [a] is a split-dwarf object then important data needed to report all of what is in the object file [a] needs an open Dwarf_Debug on the base object file [b] (usually the base executable object). Here we call that executable object file [b] the *tied* object.

See DWARF5 Appendix F.

Parameters

<i>dw_split_dbg</i>	Pass in an open dbg, on a split-dwarf object file with (normally) lots of DWARF but no executable code.
<i>dw_tied_dbg</i>	Pass in an open dbg on an executable (we call it a <i>tied</i> dbg here) which has minimal DWARF (to save space in the executable).
<i>dw_error</i>	In case return is DW_DLV_ERROR dw_error is set to point to the error details. <small>Generated by Doxygen</small>

Returns

DW_DLV_OK etc.

See also

[Attaching a tied dbg](#)

[Detaching a tied dbg](#)

9.8 Compilation Unit (CU) Access

Functions

- DW_API int [dwarf_next_cu_header_e](#) (Dwarf_Debug dw_dbg, Dwarf_Bool dw_is_info, Dwarf_Die *dw_cu↵_die, Dwarf_Unsigned *dw_cu_header_length, Dwarf_Half *dw_version_stamp, Dwarf_Off *dw_abbrev↵_offset, Dwarf_Half *dw_address_size, Dwarf_Half *dw_length_size, Dwarf_Half *dw_extension_size, Dwarf_Sig8 *dw_type_signature, Dwarf_Unsigned *dw_typeoffset, Dwarf_Unsigned *dw_next_cu_header↵_offset, Dwarf_Half *dw_header_cu_type, Dwarf_Error *dw_error)
Return information on the next CU header(e).
- DW_API int [dwarf_next_cu_header_d](#) (Dwarf_Debug dw_dbg, Dwarf_Bool dw_is_info, Dwarf_Unsigned *dw_cu_header_length, Dwarf_Half *dw_version_stamp, Dwarf_Off *dw_abbrev_offset, Dwarf_Half *dw↵_address_size, Dwarf_Half *dw_length_size, Dwarf_Half *dw_extension_size, Dwarf_Sig8 *dw_type↵_signature, Dwarf_Unsigned *dw_typeoffset, Dwarf_Unsigned *dw_next_cu_header_offset, Dwarf_Half *dw_header_cu_type, Dwarf_Error *dw_error)
Return information on the next CU header(d).
- DW_API int [dwarf_siblingof_c](#) (Dwarf_Die dw_die, Dwarf_Die *dw_return_siblingdie, Dwarf_Error *dw_error)
Return the next sibling DIE.
- DW_API int [dwarf_siblingof_b](#) (Dwarf_Debug dw_dbg, Dwarf_Die dw_die, Dwarf_Bool dw_is_info, Dwarf_Die *dw_return_siblingdie, Dwarf_Error *dw_error)
Return the first DIE or the next sibling DIE.
- DW_API int [dwarf_cu_header_basics](#) (Dwarf_Die dw_die, Dwarf_Half *dw_version, Dwarf_Bool *dw_is_info, Dwarf_Bool *dw_is_dwo, Dwarf_Half *dw_offset_size, Dwarf_Half *dw_address_size, Dwarf_Half *dw↵_extension_size, Dwarf_Sig8 **dw_signature, Dwarf_Off *dw_offset_of_length, Dwarf_Unsigned *dw_total↵_byte_length, Dwarf_Error *dw_error)
Return some CU-relative facts.
- DW_API int [dwarf_child](#) (Dwarf_Die dw_die, Dwarf_Die *dw_return_childdie, Dwarf_Error *dw_error)
Return the child DIE, if any. The child may be the first of a list of sibling DIEs.
- DW_API void [dwarf_dealloc_die](#) (Dwarf_Die dw_die)
Deallocate (free) a DIE.
- DW_API int [dwarf_die_from_hash_signature](#) (Dwarf_Debug dw_dbg, Dwarf_Sig8 *dw_hash_sig, const char *dw_sig_type, Dwarf_Die *dw_returned_CU_die, Dwarf_Error *dw_error)
Return a CU DIE given a has signature.
- DW_API int [dwarf_offdie_b](#) (Dwarf_Debug dw_dbg, Dwarf_Off dw_offset, Dwarf_Bool dw_is_info, Dwarf_Die *dw_return_die, Dwarf_Error *dw_error)
Return DIE given global (not CU-relative) offset.
- DW_API int [dwarf_find_die_given_sig8](#) (Dwarf_Debug dw_dbg, Dwarf_Sig8 *dw_ref, Dwarf_Die *dw_die ↵_out, Dwarf_Bool *dw_is_info, Dwarf_Error *dw_error)
Return a DIE given a Dwarf_Sig8 hash.
- DW_API Dwarf_Bool [dwarf_get_die_infotypes_flag](#) (Dwarf_Die dw_die)
Return the is_info flag.

9.8.1 Detailed Description

9.8.2 Function Documentation

9.8.2.1 dwarf_child()

```
DW_API int dwarf_child (
    Dwarf_Die dw_die,
    Dwarf_Die * dw_return_chiiddie,
    Dwarf_Error * dw_error )
```

Return the child DIE, if any. The child may be the first of a list of sibling DIEs.

Parameters

<i>dw_die</i>	We will return the first child of this DIE.
<i>dw_return_chiiddie</i>	Returns the first child through the pointer. For subsequent dies siblings of the first, use dwarf_siblingof_c() .
<i>dw_error</i>	The usual Dwarf_Error*.

Returns

Returns DW_DLV_OK etc. Returns DW_DLV_NO_ENTRY if dw_die has no children.

See also

[Using dwarf_child\(\)](#)

9.8.2.2 dwarf_cu_header_basics()

```
DW_API int dwarf_cu_header_basics (
    Dwarf_Die dw_die,
    Dwarf_Half * dw_version,
    Dwarf_Bool * dw_is_info,
    Dwarf_Bool * dw_is_dwo,
    Dwarf_Half * dw_offset_size,
    Dwarf_Half * dw_address_size,
    Dwarf_Half * dw_extension_size,
    Dwarf_Sig8 ** dw_signature,
    Dwarf_Off * dw_offset_of_length,
    Dwarf_Unsigned * dw_total_byte_length,
    Dwarf_Error * dw_error )
```

Return some CU-relative facts.

Any Dwarf_Die will work. The values returned through the pointers are about the CU for a DIE

Parameters

<i>dw_die</i>	Some open Dwarf_Die.
<i>dw_version</i>	Returns the DWARF version: 2,3,4, or 5

Parameters

<i>dw_is_info</i>	Returns non-zero if the CU is .debug_info. Returns zero if the CU is .debug_types (DWARF4).
<i>dw_is_dwo</i>	Returns non-zero if the CU is a dwo/dwp object and zero if it is a standard object.
<i>dw_offset_size</i>	Returns offset size, 4 and 8 are possible.
<i>dw_address_size</i>	Almost always returns 4 or 8. Could be 2 in unusual circumstances.
<i>dw_extension_size</i>	The sum of <i>dw_offset_size</i> and <i>dw_extension_size</i> are the count of the initial bytes of the CU. Standard lengths are 4 and 12. For 1990's SGI objects the length could be 8.
<i>dw_signature</i>	Returns a pointer to an 8 byte signature.
<i>dw_offset_of_length</i>	Returns the section offset of the initial byte of the CU.
<i>dw_total_byte_length</i>	Returns the total length of the CU including the length field and the content of the CU.
<i>dw_error</i>	The usual Dwarf_Error*.

Returns

Returns DW_DLV_OK etc.

9.8.2.3 dwarf_dealloc_die()

```
DW_API void dwarf_dealloc_die (
    Dwarf_Die dw_die )
```

Deallocate (free) a DIE.

Parameters

<i>dw_die</i>	Frees (deallocs) memory associated with this Dwarf_Die.
---------------	---

DIEs not freed explicitly will be freed by [dwarf_finish\(\)](#).

9.8.2.4 dwarf_die_from_hash_signature()

```
DW_API int dwarf_die_from_hash_signature (
    Dwarf_Debug dw_dbg,
    Dwarf_Sig8 * dw_hash_sig,
    const char * dw_sig_type,
    Dwarf_Die * dw_returned_CU_die,
    Dwarf_Error * dw_error )
```

Return a CU DIE given a has signature.

Parameters

<i>dw_dbg</i>	
<i>dw_hash_sig</i>	A pointer to an 8 byte signature to be looked up. in .debug_names.
<i>dw_sig_type</i>	Valid type requests are "cu" and "tu"
<i>dw_returned_CU_die</i>	Returns the found CU DIE if one is found.
<i>dw_error</i>	The usual Dwarf_Error*.

Returns

DW_DLV_OK means dw_returned_CU_die was set. DW_DLV_NO_ENTRY means the signature could not be found.

9.8.2.5 dwarf_find_die_given_sig8()

```
DW_API int dwarf_find_die_given_sig8 (
    Dwarf_Debug dw_dbg,
    Dwarf_Sig8 * dw_ref,
    Dwarf_Die * dw_die_out,
    Dwarf_Bool * dw_is_info,
    Dwarf_Error * dw_error )
```

Return a DIE given a Dwarf_Sig8 hash.

Returns DIE and is_info flag if it finds the hash signature of a DIE. Often will be the CU DIE of DW_UT_split_type or DW_UT_type CU.

Parameters

<i>dw_dbg</i>	The applicable Dwarf_Debug
<i>dw_ref</i>	A pointer to a Dwarf_Sig8 struct whose content defines what is being searched for.
<i>dw_die_out</i>	If found, this returns the found DIE itself.
<i>dw_is_info</i>	If found, this returns section (.debug_is_info or .debug_is_types).
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.8.2.6 dwarf_get_die_infotypes_flag()

```
DW_API Dwarf_Bool dwarf_get_die_infotypes_flag (
    Dwarf_Die dw_die )
```

Return the is_info flag.

So client software knows if a DIE is in debug_info or (DWARF4-only) debug_types.

Parameters

<i>dw_die</i>	The DIE being queried.
---------------	------------------------

Returns

If non-zero the flag means the DIE is in .debug_info. Otherwise it means the DIE is in .debug_types.

9.8.2.7 dwarf_next_cu_header_d()

```
DW_API int dwarf_next_cu_header_d (
    Dwarf_Debug dw_dbg,
    Dwarf_Bool dw_is_info,
    Dwarf_Unsigned * dw_cu_header_length,
    Dwarf_Half * dw_version_stamp,
    Dwarf_Off * dw_abbrev_offset,
    Dwarf_Half * dw_address_size,
    Dwarf_Half * dw_length_size,
    Dwarf_Half * dw_extension_size,
    Dwarf_Sig8 * dw_type_signature,
    Dwarf_Unsigned * dw_typeoffset,
    Dwarf_Unsigned * dw_next_cu_header_offset,
    Dwarf_Half * dw_header_cu_type,
    Dwarf_Error * dw_error )
```

Return information on the next CU header(d)

This is the version to use for linking against libdwarf v0.8.0 and earlier (and it also works for later versions).

Replace all uses of [dwarf_next_cu_header_d\(\)](#) and use [dwarf_next_cu_header_e](#) instead.

Assuming you continue to use [dwarf_next_cu_header_d\(\)](#) read the following carefully.

The library keeps track of where it is in the object file following a call to [dwarf_next_cu_header_d\(\)](#) and it knows (see next paragraph) how to interpret [dwarf_siblingof_b\(dw_dbg,NULL,dw_is_info, &cu_die,...\)](#).

In order to read the DIE tree of the CU this records information in the `dw_dbg` data and after a successful call to [dwarf_next_cu_header_d\(\)](#) only an immediate call to [dwarf_siblingof_b\(dw_dbg,NULL,dw_is_info, &cu_die,...\)](#) is guaranteed to return the correct DIE (a Compilation Unit DIE).

Avoid any call to libdwarf between a successful call to [dwarf_next_cu_header_d\(\)](#) and [dwarf_siblingof_b\(dw_dbg,↔ NULL,dw_is_info, &cu_die,...\)](#) to ensure the intended and correct Dwarf_Die is returned.

See also

[Example walking CUs\(d\)](#)

All arguments are the same as [dwarf_next_cu_header_e\(\)](#) except that there is no `dw_cu_die` argument in [dwarf_next_cu_header_d\(\)](#).

9.8.2.8 dwarf_next_cu_header_e()

```
DW_API int dwarf_next_cu_header_e (
    Dwarf_Debug dw_dbg,
    Dwarf_Bool dw_is_info,
    Dwarf_Die * dw_cu_die,
    Dwarf_Unsigned * dw_cu_header_length,
    Dwarf_Half * dw_version_stamp,
    Dwarf_Off * dw_abbrev_offset,
    Dwarf_Half * dw_address_size,
    Dwarf_Half * dw_length_size,
    Dwarf_Half * dw_extension_size,
    Dwarf_Sig8 * dw_type_signature,
    Dwarf_Unsigned * dw_typeoffset,
```

```
Dwarf_Unsigned * dw_next_cu_header_offset,
Dwarf_Half * dw_header_cu_type,
Dwarf_Error * dw_error )
```

Return information on the next CU header(e).

New in v0.9.0 November 2023.

[dwarf_next_cu_header_e\(\)](#) is preferred over [dwarf_next_cu_header_d\(\)](#) as the latter requires a second (immediate) step to access the CU-DIE of the CU.

With the CU-DIE returned by [dwarf_next_cu_header_e\(\)](#) one calls [dwarf_child\(\)](#) first (the CU-DIE has no siblings) and then one calls [dwarf_siblingof_c\(\)](#) and [dwarf_child\(\)](#) appropriately to descend the tree of DIEs.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_is_info</i>	Pass in TRUE if reading through .debug_info Pass in FALSE if reading through DWARF4 .debug_types.
<i>dw_cu_die</i>	Pass in a pointer to a Dwarf_Die. the call sets the passed-in pointer to be a Compilation Unit Die for use with dwarf_child() or any other call requiring a Dwarf_Die argument.
<i>dw_cu_header_length</i>	Returns the length of the just-read CU header.
<i>dw_version_stamp</i>	Returns the version number (2 to 5) of the CU header just read.
<i>dw_abbrev_offset</i>	Returns the .debug_abbrev offset from the the CU header just read.
<i>dw_address_size</i>	Returns the address size specified for this CU, usually either 4 or 8.
<i>dw_length_size</i>	Returns the offset size (the length of the size field from the header) specified for this CU, either 4 or 4.
<i>dw_extension_size</i>	If the section is standard 64bit DWARF then this value is 4. Else the value is zero.
<i>dw_type_signature</i>	If the CU is DW_UT_skeleton DW_UT_split_compile, DW_UT_split_type or DW_UT_type this is the type signature from the CU_header compiled into this field.
<i>dw_typeoffset</i>	For DW_UT_split_type or DW_UT_type this is the type offset from the CU header.
<i>dw_next_cu_header_offset</i>	The offset in the section of the next CU (unless there is a compiler bug this is rarely of interest).
<i>dw_header_cu_type</i>	Returns DW_UT_compile, or other DW_UT value.
<i>dw_error</i>	In case return is DW_DLV_ERROR dw_error is set to point to the error details.

Returns

Returns DW_DLV_OK on success. Returns DW_DLV_NO_ENTRY if all CUs have been read.

See also

[Example walking CUs\(e\)](#)

9.8.2.9 dwarf_offdie_b()

```
DW_API int dwarf_offdie_b (
    Dwarf_Debug dw_dbg,
```

```

Dwarf_Off dw_offset,
Dwarf_Bool dw_is_info,
Dwarf_Die * dw_return_die,
Dwarf_Error * dw_error )

```

Return DIE given global (not CU-relative) offset.

This works whether or not the target section has had [dwarf_next_cu_header_d\(\)](#) applied, the CU the offset exists in has been seen at all, or the target offset is one libdwarf has seen before.

Parameters

<i>dw_dbg</i>	The applicable Dwarf_Debug
<i>dw_offset</i>	The global offset of the DIE in the appropriate section.
<i>dw_is_info</i>	Pass TRUE if the target is .debug_info. Pass FALSE if the target is .debug_types.
<i>dw_return_die</i>	On success this returns a DIE pointer to the found DIE.
<i>dw_error</i>	The usual Dwarf_Error*.

Returns

DW_DLX_OK means *dw_returned_die* was found DW_DLX_NO_ENTRY is only possible if the offset is to a null DIE, and that is very unusual. Otherwise expect DW_DLX_ERROR.

See also

[Using dwarf_offdie_b\(\)](#)

9.8.2.10 dwarf_siblingof_b()

```

DW_API int dwarf_siblingof_b (
    Dwarf_Debug dw_dbg,
    Dwarf_Die dw_die,
    Dwarf_Bool dw_is_info,
    Dwarf_Die * dw_return_siblingdie,
    Dwarf_Error * dw_error )

```

Return the first DIE or the next sibling DIE.

This function follows [dwarf_next_cu_header_d\(\)](#) to return the CU-DIE that [dwarf_next_cu_header_d\(\)](#) implies but does not reveal.

Aside from the special case required use of [dwarf_siblingof_b\(\)](#) immediately following [dwarf_next_cu_header_d\(\)](#), [dwarf_siblingof_c\(\)](#) is the faster function.

This function will eventually be deprecated.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug one is operating on.
<i>dw_die</i>	Immediately after calling dwarf_next_cu_header_d pass in NULL to retrieve the CU DIE. Or pass in a known DIE and this will retrieve the next sibling in the chain.
<i>dw_is_info</i>	Pass TRUE or FALSE to match the applicable dwarf_next_cu_header_d call.
<i>dw_return_siblingdie</i>	The DIE returned through the pointer.
<i>dw_error</i>	The usual error information, if any.

Returns

Returns DW_DLV_OK etc.

See also

example4

dwarf_get_die_infotypes

9.8.2.11 dwarf_siblingof_c()

```
DW_API int dwarf_siblingof_c (
    Dwarf_Die dw_die,
    Dwarf_Die * dw_return_siblingdie,
    Dwarf_Error * dw_error )
```

Return the next sibling DIE.

Parameters

<i>dw_die</i>	Pass in a known DIE and this will retrieve the next sibling in the chain.
<i>dw_return_siblingdie</i>	The DIE returned through the pointer.
<i>dw_error</i>	The usual error information, if any.

Returns

Returns DW_DLV_OK etc.

See also

example4

dwarf_get_die_infotypes

9.9 Debugging Information Entry (DIE) content**Functions**

- DW_API int [dwarf_die_abbrev_global_offset](#) ([Dwarf_Die](#) dw_die, [Dwarf_Off](#) *dw_abbrev_offset, [Dwarf_Unsigned](#) *dw_abbrev_count, [Dwarf_Error](#) *dw_error)
Return the abbrev section offset of a DIE's abbrevs.
- DW_API int [dwarf_tag](#) ([Dwarf_Die](#) dw_die, [Dwarf_Half](#) *dw_return_tag, [Dwarf_Error](#) *dw_error)
Get TAG value of DIE.
- DW_API int [dwarf_dieoffset](#) ([Dwarf_Die](#) dw_die, [Dwarf_Off](#) *dw_return_offset, [Dwarf_Error](#) *dw_error)
Return the global section offset of the DIE.
- DW_API int [dwarf_debug_addr_index_to_addr](#) ([Dwarf_Die](#) dw_die, [Dwarf_Unsigned](#) dw_index, [Dwarf_Addr](#) *dw_return_addr, [Dwarf_Error](#) *dw_error)
Extract address given address index. DWARF5.
- DW_API [Dwarf_Bool](#) [dwarf_addr_form_is_indexed](#) (int dw_form)

- Informes if a DW_FORM is an indexed form.*

• DW_API int [dwarf_CU_dieoffset_given_die](#) (Dwarf_Die dw_die, Dwarf_Off *dw_return_offset, Dwarf_Error *dw_error)

Return the CU DIE offset given any DIE.
- DW_API int [dwarf_get_cu_die_offset_given_cu_header_offset_b](#) (Dwarf_Debug dw_dbg, Dwarf_Off dw_in↵_cu_header_offset, Dwarf_Bool dw_is_info, Dwarf_Off *dw_out_cu_die_offset, Dwarf_Error *dw_error)

Return the CU DIE section offset given CU header offset.
- DW_API int [dwarf_die_CU_offset](#) (Dwarf_Die dw_die, Dwarf_Off *dw_return_offset, Dwarf_Error *dw_error)

returns the CU relative offset of the DIE.
- DW_API int [dwarf_die_CU_offset_range](#) (Dwarf_Die dw_die, Dwarf_Off *dw_return_CU_header_offset, Dwarf_Off *dw_return_CU_length_bytes, Dwarf_Error *dw_error)

Return the offset length of the entire CU of a DIE.
- DW_API int [dwarf_attr](#) (Dwarf_Die dw_die, Dwarf_Half dw_attrnum, Dwarf_Attribute *dw_returned_attr, Dwarf_Error *dw_error)

Given DIE and attribute number return a Dwarf_attribute.
- DW_API int [dwarf_die_text](#) (Dwarf_Die dw_die, Dwarf_Half dw_attrnum, char **dw_ret_name, Dwarf_Error *dw_error)

Given DIE and attribute number return a string.
- DW_API int [dwarf_diename](#) (Dwarf_Die dw_die, char **dw_diename, Dwarf_Error *dw_error)

Return the string from a DW_AT_name attribute.
- DW_API Dwarf_Unsigned [dwarf_die_abbrev_code](#) (Dwarf_Die dw_die)

Return the DIE abbrev code.
- DW_API int [dwarf_die_abbrev_children_flag](#) (Dwarf_Die dw_die, Dwarf_Half *dw_ab_has_child)

Return TRUE if the DIE has children.
- DW_API int [dwarf_validate_die_sibling](#) (Dwarf_Die dw_sibling, Dwarf_Off *dw_offset)

Validate a sibling DIE.
- DW_API int [dwarf_hasattr](#) (Dwarf_Die dw_die, Dwarf_Half dw_attrnum, Dwarf_Bool *dw_returned_bool, Dwarf_Error *dw_error)

Tells whether a DIE has a particular attribute.
- DW_API int [dwarf_offset_list](#) (Dwarf_Debug dw_dbg, Dwarf_Off dw_offset, Dwarf_Bool dw_is_info, Dwarf_Off **dw_offbuf, Dwarf_Unsigned *dw_offcount, Dwarf_Error *dw_error)

Return an array of DIE children offsets.
- DW_API int [dwarf_get_die_address_size](#) (Dwarf_Die dw_die, Dwarf_Half *dw_addr_size, Dwarf_Error *dw↵_error)

Get the address size applying to a DIE.
- DW_API int [dwarf_die_offsets](#) (Dwarf_Die dw_die, Dwarf_Off *dw_global_offset, Dwarf_Off *dw_local↵_offset, Dwarf_Error *dw_error)

Return section and CU-local offsets of a DIE.
- DW_API int [dwarf_get_version_of_die](#) (Dwarf_Die dw_die, Dwarf_Half *dw_version, Dwarf_Half *dw↵_offset_size)

Get the version and offset size.
- DW_API int [dwarf_lowpc](#) (Dwarf_Die dw_die, Dwarf_Addr *dw_returned_addr, Dwarf_Error *dw_error)

Return the DW_AT_low_pc value.
- DW_API int [dwarf_highpc_b](#) (Dwarf_Die dw_die, Dwarf_Addr *dw_return_addr, Dwarf_Half *dw_return↵_form, enum Dwarf_Form_Class *dw_return_class, Dwarf_Error *dw_error)

Return the DW_AT_hipc address value.
- DW_API int [dwarf_dietype_offset](#) (Dwarf_Die dw_die, Dwarf_Off *dw_return_offset, Dwarf_Bool *dw_is_info, Dwarf_Error *dw_error)

Return the offset from the DW_AT_type attribute.
- DW_API int [dwarf_bytesize](#) (Dwarf_Die dw_die, Dwarf_Unsigned *dw_returned_size, Dwarf_Error *dw↵_error)

Return the value of the attribute DW_AT_byte_size.

- DW_API int [dwarf_bitsize](#) ([Dwarf_Die](#) dw_die, [Dwarf_Unsigned](#) *dw_returned_size, [Dwarf_Error](#) *dw_error)
Return the value of the attribute DW_AT_bitsize.
- DW_API int [dwarf_bitoffset](#) ([Dwarf_Die](#) dw_die, [Dwarf_Half](#) *dw_attrnum, [Dwarf_Unsigned](#) *dw_returned_offset, [Dwarf_Error](#) *dw_error)
Return the bit offset attribute of a DIE.
- DW_API int [dwarf_srclang](#) ([Dwarf_Die](#) dw_die, [Dwarf_Unsigned](#) *dw_returned_lang, [Dwarf_Error](#) *dw_error)
Return the value of the DW_AT_language attribute.
- DW_API int [dwarf_srclanglname](#) ([Dwarf_Die](#) dw_die, [Dwarf_Unsigned](#) *dw_returned_lname, [Dwarf_Error](#) *dw_error)
Return the value of the DW_AT_language_name attribute.
- DW_API int [dwarf_srclanglname_version](#) ([Dwarf_Die](#) dw_die, const char *dw_returned_verstring, [Dwarf_Error](#) *dw_error)
Return the value of the DW_AT_language_version attribute.
- DW_API int [dwarf_language_version_data](#) ([Dwarf_Unsigned](#) dw_lname_name, int *dw_default_lower_bound, const char **dw_version_string)
Return values associated with DW_AT_language_name.
- DW_API int [dwarf_language_version_string](#) ([Dwarf_Unsigned](#) dw_lname_name, int *dw_default_lower_bound, const char **dw_version_string)
dwarf_language_version_string is obsolete.
- DW_API int [dwarf_lvn_name_direct](#) ([Dwarf_Unsigned](#) dw_lv_lang, [Dwarf_Unsigned](#) dw_lv_ver, const char **dw_ret_version_name, const char **dw_ret_version_scheme)
Return language version name.
- DW_API int [dwarf_lvn_name](#) ([Dwarf_Die](#) dw_die, const char **dw_ret_version_name, const char **dw_ret_version_scheme)
Return values associated with DW_AT_language_version.
- DW_API int [dwarf_lvn_table_entry](#) ([Dwarf_Unsigned](#) dw_lvn_index, [Dwarf_Unsigned](#) *dw_lvn_language_name, [Dwarf_Unsigned](#) *dw_lvn_language_version, const char **dw_lvn_language_version_scheme, const char **dw_lvn_language_version_name)
Return values from the DWARF6 language version standard.
- DW_API int [dwarf_arrayorder](#) ([Dwarf_Die](#) dw_die, [Dwarf_Unsigned](#) *dw_returned_order, [Dwarf_Error](#) *dw_error)
Return the value of the DW_AT_ordering attribute.

9.9.1 Detailed Description

This is the main interface to attributes of a DIE.

9.9.2 Function Documentation

9.9.2.1 dwarf_addr_form_is_indexed()

```
DW_API Dwarf_Bool dwarf_addr_form_is_indexed (
    int dw_form )
```

Informs if a DW_FORM is an indexed form.

Reading a CU DIE with DW_AT_low_pc an indexed value can be problematic as several different FORMs are indexed. Some in DWARF5 others being extensions to DWARF4 and DWARF5. Indexed forms interact with DW_AT_addr_base in a DIE making this a very relevant distinction.

9.9.2.2 dwarf_arrayorder()

```
DW_API int dwarf_arrayorder (
    Dwarf_Die dw_die,
    Dwarf_Unsigned * dw_returned_order,
    Dwarf_Error * dw_error )
```

Return the value of the DW_AT_ordering attribute.

Parameters

<i>dw_die</i>	The DIE of interest.
<i>dw_returned_order</i>	On success returns the ordering value. For example DW_ORD_row_major
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.3 dwarf_attr()

```
DW_API int dwarf_attr (
    Dwarf_Die dw_die,
    Dwarf_Half dw_attrnum,
    Dwarf_Attribute * dw_returned_attr,
    Dwarf_Error * dw_error )
```

Given DIE and attribute number return a Dwarf_attribute.

Returns DW_DLV_NO_ENTRY if the DIE has no attribute dw_attrnum.

Parameters

<i>dw_die</i>	The DIE of interest.
<i>dw_attrnum</i>	An attribute number, for example DW_AT_name.
<i>dw_returned_attr</i>	On success a Dwarf_Attribute pointer is returned and it should eventually be deallocated.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.4 dwarf_bitoffset()

```
DW_API int dwarf_bitoffset (
    Dwarf_Die dw_die,
    Dwarf_Half * dw_attrnum,
    Dwarf_Unsigned * dw_returned_offset,
    Dwarf_Error * dw_error )
```

Return the bit offset attribute of a DIE.

If the attribute is DW_AT_data_bit_offset (DWARF4, DWARF5) the returned bit offset has one meaning. If the attribute is DW_AT_bit_offset (DWARF2, DWARF3) the meaning is quite different.

Parameters

<i>dw_die</i>	The DIE of interest.
<i>dw_attrnum</i>	If successful, returns the number of the attribute (DW_AT_data_bit_offset or DW_AT_bit_offset)
<i>dw_returned_offset</i>	If successful, returns the bit offset value.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.5 dwarf_bitsize()

```
DW_API int dwarf_bitsize (
    Dwarf_Die dw_die,
    Dwarf_Unsigned * dw_returned_size,
    Dwarf_Error * dw_error )
```

Return the value of the attribute DW_AT_bitsize.

Parameters

<i>dw_die</i>	The DIE of interest.
<i>dw_returned_size</i>	If successful, returns the size through the pointer.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.6 dwarf_bytesize()

```
DW_API int dwarf_bytesize (
    Dwarf_Die dw_die,
    Dwarf_Unsigned * dw_returned_size,
    Dwarf_Error * dw_error )
```

Return the value of the attribute DW_AT_byte_size.

Parameters

<i>dw_die</i>	The DIE of interest.
<i>dw_returned_size</i>	If successful, returns the size through the pointer.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.7 dwarf_CU_dieoffset_given_die()

```
DW_API int dwarf_CU_dieoffset_given_die (
    Dwarf_Die dw_die,
    Dwarf_Off * dw_return_offset,
    Dwarf_Error * dw_error )
```

Return the CU DIE offset given any DIE.

Returns the global debug_info section offset of the CU DIE in the CU containing the given_die (the passed in DIE can be any DIE).

This does not identify whether the section is .debug_info or .debug_types, use [dwarf_get_die_infotypes_flag\(\)](#) to determine the section.

See also

[dwarf_get_cu_die_offset_given_cu_header_offset_b](#)
[Using dwarf_offset_given_die\(\)](#)

Parameters

<i>dw_die</i>	The DIE being queried.
<i>dw_return_offset</i>	Returns the section offset of the CU DIE for dw_die.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.8 dwarf_debug_addr_index_to_addr()

```
DW_API int dwarf_debug_addr_index_to_addr (
    Dwarf_Die dw_die,
    Dwarf_Unsigned dw_index,
    Dwarf_Addr * dw_return_addr,
    Dwarf_Error * dw_error )
```

Extract address given address index. DWARF5.

Useful for checking for compiler/linker errors in the creation of DWARF5.

Parameters

<i>dw_die</i>	The DIE of interest
<i>dw_index</i>	An index into .debug_addr. This will look first for .debug_addr in the dbg object DIE and if not there will look in the tied object if that is available.
<i>dw_return_addr</i>	On success the address is returned through the pointer.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.9 dwarf_die_abbrev_children_flag()

```
DW_API int dwarf_die_abbrev_children_flag (
    Dwarf_Die dw_die,
    Dwarf_Half * dw_ab_has_child )
```

Return TRUE if the DIE has children.

Parameters

<i>dw_die</i>	A valid DIE pointer (not NULL).
<i>dw_ab_has_child</i>	Sets TRUE though the pointer if the DIE has children. Otherwise sets FALSE.

Returns

Returns TRUE if the DIE has a child DIE. Else returns FALSE.

9.9.2.10 dwarf_die_abbrev_code()

```
DW_API Dwarf_Unsigned dwarf_die_abbrev_code (
    Dwarf_Die dw_die )
```

Return the DIE abbrev code.

The Abbrev code for a DIE is a positive integer assigned by the compiler within a particular CU. For .debug_names abbreviations the situation is conceptually similar. The code values are arbitrary but compilers are motivated to make them small so the object size is as small as possible.

Returns the abbrev code of the die. Cannot fail.

Parameters

<i>dw_die</i>	The DIE of interest.
---------------	----------------------

Returns

The abbrev code. of the DIE.

9.9.2.11 dwarf_die_abbrev_global_offset()

```
DW_API int dwarf_die_abbrev_global_offset (
    Dwarf_Die dw_die,
    Dwarf_Off * dw_abbrev_offset,
```

```
Dwarf_Unsigned * dw_abbrev_count,
Dwarf_Error * dw_error )
```

Return the abbrev section offset of a DIE's abbrevs.

So we can associate a DIE's abbreviations with the contents the abbreviations section. Useful for detailed printing and analysis of abbreviations.

Parameters

<i>dw_die</i>	The DIE of interest
<i>dw_abbrev_offset</i>	On success is set to the global offset in the .debug_abbrev section of the abbreviations for the DIE.
<i>dw_abbrev_count</i>	On success is set to the count of abbreviations in the .debug_abbrev section of the abbreviations for the DIE.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.12 dwarf_die_CU_offset()

```
DW_API int dwarf_die_CU_offset (
    Dwarf_Die dw_die,
    Dwarf_Off * dw_return_offset,
    Dwarf_Error * dw_error )
```

returns the CU relative offset of the DIE.

See also

[dwarf_CU_dieoffset_given_die](#)

This does not identify whether the section is .debug_info or .debug_types, use [dwarf_get_die_infotypes_flag\(\)](#) to determine the section.

Parameters

<i>dw_die</i>	The DIE being queried.
<i>dw_return_offset</i>	Returns the CU relative offset of this DIE.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.13 dwarf_die_CU_offset_range()

```
DW_API int dwarf_die_CU_offset_range (
    Dwarf_Die dw_die,
```

```

Dwarf_Off * dw_return_CU_header_offset,
Dwarf_Off * dw_return_CU_length_bytes,
Dwarf_Error * dw_error )

```

Return the offset length of the entire CU of a DIE.

This does not identify whether the section is `.debug_info` or `.debug_types`, use `dwarf_get_die_infotypes_flag()` to determine the section.

Parameters

<i>dw_die</i>	The DIE being queried.
<i>dw_return_CU_header_offset</i>	On success returns the section offset of the CU this DIE is in.
<i>dw_return_CU_length_bytes</i>	On success returns the CU length of the CU this DIE is in, including the CU length, header, and all DIEs.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.14 dwarf_die_offsets()

```

DW_API int dwarf_die_offsets (
    Dwarf_Die dw_die,
    Dwarf_Off * dw_global_offset,
    Dwarf_Off * dw_local_offset,
    Dwarf_Error * dw_error )

```

Return section and CU-local offsets of a DIE.

This does not identify whether the section is `.debug_info` or `.debug_types`, use `dwarf_get_die_infotypes_flag()` to determine the section.

Parameters

<i>dw_die</i>	The DIE of interest.
<i>dw_global_offset</i>	On success returns the offset of the DIE in its section.
<i>dw_local_offset</i>	On success returns the offset of the DIE within its CU.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.15 dwarf_die_text()

```

DW_API int dwarf_die_text (
    Dwarf_Die dw_die,

```



```
Dwarf_Half dw_attrnum,
char ** dw_ret_name,
Dwarf_Error * dw_error )
```

Given DIE and attribute number return a string.

Returns DW_DLV_NO_ENTRY if the DIE has no attribute dw_attrnum.

Parameters

<i>dw_die</i>	The DIE of interest.
<i>dw_attrnum</i>	An attribute number, for example DW_AT_name.
<i>dw_ret_name</i>	On success a pointer to the string is returned. Do not free the string. Many attributes allow various forms that directly or indirectly contain strings and this returns the string.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.16 dwarf_diename()

```
DW_API int dwarf_diename (
    Dwarf_Die dw_die,
    char ** dw_diename,
    Dwarf_Error * dw_error )
```

Return the string from a DW_AT_name attribute.

Returns DW_DLV_NO_ENTRY if the DIE has no attribute DW_AT_name

Parameters

<i>dw_die</i>	The DIE of interest.
<i>dw_diename</i>	On success a pointer to the string is returned. Do not free the string. Various forms directly or indirectly contain strings and this follows all of them to their string.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.17 dwarf_dieoffset()

```
DW_API int dwarf_dieoffset (
    Dwarf_Die dw_die,
    Dwarf_Off * dw_return_offset,
    Dwarf_Error * dw_error )
```

Return the global section offset of the DIE.

Parameters

<i>dw_die</i>	The DIE of interest
<i>dw_return_offset</i>	On success the offset refers to the section of the DIE itself, which may be <code>.debug_offset</code> or <code>.debug_types</code> .
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.18 dwarf_dietype_offset()

```
DW_API int dwarf_dietype_offset (
    Dwarf_Die dw_die,
    Dwarf_Off * dw_return_offset,
    Dwarf_Bool * dw_is_info,
    Dwarf_Error * dw_error )
```

Return the offset from the DW_AT_type attribute.

The offset returned is a global offset from the DW_AT_type of the DIE passed in. If this CU is DWARF4 the offset could be in `.debug_types`, otherwise it is in `.debug_info`. Check the section of the DIE to know which it is, [dwarf_cu_header_basics\(\)](#) will return that.

Added pointer argument to return the section the offset applies to. December 2022.

Parameters

<i>dw_die</i>	The DIE of interest.
<i>dw_return_offset</i>	If successful, returns the offset through the pointer.
<i>dw_is_info</i>	If successful, set to TRUE if the <code>dw_return_offset</code> is in <code>.debug_info</code> and FALSE if the <code>dw_return_offset</code> is in <code>.debug_types</code> .
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.19 dwarf_get_cu_die_offset_given_cu_header_offset_b()

```
DW_API int dwarf_get_cu_die_offset_given_cu_header_offset_b (
    Dwarf_Debug dw_dbg,
    Dwarf_Off dw_in_cu_header_offset,
    Dwarf_Bool dw_is_info,
    Dwarf_Off * dw_out_cu_die_offset,
    Dwarf_Error * dw_error )
```

Return the CU DIE section offset given CU header offset.

Returns the CU DIE global offset if one knows the CU header global offset.

See also

[dwarf_CU_dieoffset_given_die](#)

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_in_cu_header_offset</i>	The CU header offset.
<i>dw_is_info</i>	If TRUE the CU header offset is in .debug_info. Otherwise the CU header offset is in .debug_types.
<i>dw_out_cu_die_offset</i>	The CU DIE offset returned through this pointer.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.20 dwarf_get_die_address_size()

```
DW_API int dwarf_get_die_address_size (
    Dwarf_Die dw_die,
    Dwarf_Half * dw_addr_size,
    Dwarf_Error * dw_error )
```

Get the address size applying to a DIE.

Parameters

<i>dw_die</i>	The DIE of interest.
<i>dw_addr_size</i>	On success, returns the address size that applies to dw_die. Normally 4 or 8.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.21 dwarf_get_version_of_die()

```
DW_API int dwarf_get_version_of_die (
    Dwarf_Die dw_die,
    Dwarf_Half * dw_version,
    Dwarf_Half * dw_offset_size )
```

Get the version and offset size.

The values returned apply to the CU this DIE belongs to. This is useful as preparation for calling [dwarf_get_form_class](#)

Parameters

<i>dw_die</i>	The DIE of interest.
<i>dw_version</i>	Returns the version of the CU this DIE is contained in. Standard version numbers are 2 through 5.
<i>dw_offset_size</i>	Returns the offset_size (4 or 8) of the CU this DIE is contained in.

Returns

On success, returns DW_DLV_OK. If *dw_die* is null or its contents are corrupted returns DW_DLV_ERROR and there is nothing useful returned. Never returns DW_DLV_NO_ENTRY.

9.9.2.22 dwarf_hasattr()

```
DW_API int dwarf_hasattr (
    Dwarf_Die dw_die,
    Dwarf_Half dw_attrnum,
    Dwarf_Bool * dw_returned_bool,
    Dwarf_Error * dw_error )
```

Tells whether a DIE has a particular attribute.

Parameters

<i>dw_die</i>	The DIE of interest.
<i>dw_attrnum</i>	The attribute number we are asking about, DW_AT_name for example.
<i>dw_returned_bool</i>	On success is set TRUE if <i>dw_die</i> has <i>dw_attrnum</i> and FALSE otherwise.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Never returns DW_DLV_NO_ENTRY. Returns DW_DLV_OK unless there is an error, in which case it returns DW_DLV_ERROR and sets *dw_error* to the error details.

9.9.2.23 dwarf_highpc_b()

```
DW_API int dwarf_highpc_b (
    Dwarf_Die dw_die,
    Dwarf_Addr * dw_return_addr,
    Dwarf_Half * dw_return_form,
    enum Dwarf_Form_Class * dw_return_class,
    Dwarf_Error * dw_error )
```

Return the DW_AT_highpc address value.

This is accessing the DW_AT_highpc attribute. Calculating the high pc involves elements which we don't describe here, but which are shown in the example. See the DWARF5 standard.

See also

[Reading high pc from a DIE.](#)

Parameters

<i>dw_die</i>	The DIE of interest.
<i>dw_return_addr</i>	On success returns the high-pc address for this DIE. If the high-pc is a not DW_FORM_addr and is a non-indexed constant form one must add the value of the DW_AT_low_pc to this to get the true high-pc value as the value returned is an unsigned offset of the associated low-pc value.
<i>dw_return_form</i>	On success returns the actual FORM for this attribute. Needed for certain cases to calculate the true dw_return_addr;
<i>dw_return_class</i>	On success returns the FORM CLASS for this attribute. Needed for certain cases to calculate the true dw_return_addr;
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.24 dwarf_language_version_data()

```
DW_API int dwarf_language_version_data (
    Dwarf_Unsigned dw_lname_name,
    int * dw_default_lower_bound,
    const char ** dw_version_string )
```

Return values associated with DW_AT_language_name.

Returns the value of a the default-lower-bound and a string defining the interpretation of the DWARF6 version from the DW_AT_language_version attribute. Replaces [dwarf_language_version_string\(\)](#).

Parameters

<i>dw_lname_name</i>	Pass in a DW_LNAME value, for example DW_LNAME_C (0x0003).
<i>dw_default_lower_bound.</i>	On success returns the language code (normally only found on a CU DIE). For example DW_LNAME_C has a default lower bound of zero (0) that will be returned through the pointer.
<i>dw_version_scheme</i>	On success, return the version scheme, For DW_LNAME_C the string returned through the pointer would be "YYYYMM". If there is no version scheme defined, return a NULL through the pointer. Never dealloc or free() the string returned through dw_version_scheme as it is a static constant string.

Returns

Returns DW_DLV_OK or the dw_lang_name is unknown, returns DW_DLV_NO_ENTRY. Never returns DW_DLV_ERROR;

9.9.2.25 dwarf_language_version_string()

```
DW_API int dwarf_language_version_string (
    Dwarf_Unsigned dw_lname_name,
```

```
int * dw_default_lower_bound,
const char ** dw_version_string )
```

`dwarf_language_version_string` is obsolete.

OBSOLETE NAME. Do Not use `dwarf_language_version_string()` use `dwarf_language_version_data()`.

9.9.2.26 dwarf_lowpc()

```
DW_API int dwarf_lowpc (
    Dwarf_Die dw_die,
    Dwarf_Addr * dw_returned_addr,
    Dwarf_Error * dw_error )
```

Return the DW_AT_low_pc value.

Parameters

<code>dw_die</code>	The DIE of interest.
<code>dw_returned_addr</code>	On success returns, through the pointer, the address DW_AT_low_pc defines.
<code>dw_error</code>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.27 dwarf_lvn_name()

```
DW_API int dwarf_lvn_name (
    Dwarf_Die dw_die,
    const char ** dw_ret_version_name,
    const char ** dw_ret_version_scheme )
```

Return values associated with DW_AT_language_version.

New in version 2.2.0 July 2025

Given any valid DIE for a Compilation Unit returns the value of a the CU_DIE name of the DWARF6 DW_AT_language_version as a string, as "C++98" for example. And the string defining the format of the language version, for example "YYYYMM" if DW_LNAME_C. Never free or dealloc the returned string, it is static memory @param dw_die Pass in any valid open Dwarf_Die for the compilation unit of interest. @param dw_reg_version_name On success returns the language version name string through the pointer. Never dealloc or free the string, it points to static memory. @param dw_ret_version_scheme On success, return the version scheme, For DW_LNAME_C the string returned through the pointer would be "YYYYMM". If there is no version scheme defined, return a NULL through the pointer. Never dealloc or free() the string returned through dw_version_scheme as it is a static constant string.

Returns

Returns DW_DLV_OK or the dw_lang_name is unknown, returns DW_DLV_NO_ENTRY. Never returns DW_DLV_ERROR;

9.9.2.28 dwarf_lvn_name_direct()

```
DW_API int dwarf_lvn_name_direct (
    Dwarf_Unsigned dw_lv_lang,
    Dwarf_Unsigned dw_lv_ver,
    const char ** dw_ret_version_name,
    const char ** dw_ret_version_scheme )
```

Return language version name.

New in version 2.2.0 July 2025

Returns the value of a the name of the DWARF6 DW_AT_language_version as a string, as "C++98" for example. And the string defining the format of the language version, for example "YYYYMM" if DW_LNAME_C. Never free or dealloc the returned string, it is static memory @param dw_lv_lang Pass in a DW_LNAME value, for example DW_LNAME_C (0x0003). @param Pass in the language version, for example 201103 (meaning C++ 11). @param dw_ret_version_name On success, return the name of the version, "C++11" for example. Never free or dealloc the string. @param dw_reg_version_scheme On success, returns For DW_LNAME_C the string returned through the pointer would be "YYYYMM". If there is no version scheme defined, return a NULL through the pointer. Never dealloc or free() the string returned through dw_version_scheme as it is a static constant string.

Returns

Returns DW_DLV_OK or the dw_lang_name is unknown, returns DW_DLV_NO_ENTRY. Never returns DW_DLV_ERROR;

9.9.2.29 dwarf_lvn_table_entry()

```
DW_API int dwarf_lvn_table_entry (
    Dwarf_Unsigned dw_lvn_index,
    Dwarf_Unsigned * dw_lvn_language_name,
    Dwarf_Unsigned * dw_lvn_language_version,
    const char ** dw_lvn_language_version_scheme,
    const char ** dw_lvn_language_version_name )
```

Return values from the DWARF6 language version standard.

New in version 2.2.0 July 2025

Primarily used by dwarfdump. This enables access to the instances of DWARF6 language version table known to this version of libdwarf. None of the strings returned through pointers should be dealloc-d or free-d, they are static strings.

Parameters

<i>dw_lvn_index</i>	To see all table entries, pass in the index of a table entry, beginning with 0, and call again with subsequent numbers until the function returns DW_DLV_NO_ENTRY (meaning there are no more entries). The index has no intrinsic meaning.
<i>dw_lvn_language_name</i>	On success, the function returns the language name through the pointer. For example, a value like DW_LNAME_C.
<i>dw_lvn_language_version</i>	On success, the function returns the language version through the pointer. For example a number such as for C: 199901.
<i>dw_lvn_language_version_scheme</i>	On success, the function returns a pointer to a string identifying the format of the language version through the pointer. For example "YYYYMM" for C.
<small>Generated by Doxygen</small> <i>dw_lvn_language_version_name</i>	On success, the function returns a pointer to a string for C. identifying the name of the language version through the pointer. For example: "C99".

Returns

9.9.2.30 dwarf_offset_list()

```
DW_API int dwarf_offset_list (
    Dwarf_Debug dw_dbg,
    Dwarf_Off dw_offset,
    Dwarf_Bool dw_is_info,
    Dwarf_Off ** dw_offbuf,
    Dwarf_Unsigned * dw_offcount,
    Dwarf_Error * dw_error )
```

Return an array of DIE children offsets.

Given a DIE section offset and dw_is_info, returns an array of DIE global [section] offsets of the children of DIE.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_offset</i>	A DIE offset.
<i>dw_is_info</i>	If TRUE says to use the offset in .debug_info. Else use the offset in .debug_types.
<i>dw_offbuf</i>	A pointer to an array of children DIE global [section] offsets is returned through the pointer.
<i>dw_offcount</i>	The number of elements in dw_offbuf. If the DIE has no children it could be zero, in which case dw_offbuf and dw_offcount are not touched.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc. DW_DLV_NO_ENTRY means there are no children of the DIE, hence no list of child offsets.

On successful return, use dwarf_dealloc(dbg, dw_offbuf, DW_DLA_UARRAY); to dealloc the allocated space.

See also

[Using dwarf_offset_list\(\)](#)

9.9.2.31 dwarf_srclang()

```
DW_API int dwarf_srclang (
    Dwarf_Die dw_die,
    Dwarf_Unsigned * dw_returned_lang,
    Dwarf_Error * dw_error )
```

Return the value of the DW_AT_language attribute.

Returns DWARF5 DW_LANG language name. The DW_LANG value returned lets one access the LANG name as a string with [dwarf_get_LANG_name\(\)](#)

To access DW_LNAME names (in DWARF5 or later) see [dwarf_srclangname\(\)](#). To get the DW_LNAME as a string, call [dwarf_get_LNAME_name\(\)](#).

DWARF5 and earlier

The DIE should be a CU DIE.

Parameters

<i>dw_die</i>	The DIE of interest.
<i>dw_returned_lang</i>	On success returns the language code (normally only found on a CU DIE). For example DW_LANG_C (0x0002).
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLX_OK etc.

9.9.2.32 dwarf_srclanglname()

```
DW_API int dwarf_srclanglname (
    Dwarf_Die dw_die,
    Dwarf_Unsigned * dw_returned_lname,
    Dwarf_Error * dw_error )
```

Return the value of the DW_AT_language_name attribute.

New in v2.1.0 July 2025.

Returns a DWARF6 DW_AT_language_name name. The DW_LNAME value returned lets one access the LNAME name as a string with [dwarf_get_LNAME_name\(\)](#) Also see [dwarf_language_version_data\(\)](#) for valued based on DW_LNAME names.

To access DW_LANG names (in DWARF5 or earlier) see [dwarf_srclang\(\)](#).

Parameters

<i>dw_die</i>	The DIE of interest, normally a CU_DIE.
<i>dw_returned_lname</i>	On success returns the language name (code) (normally only found on a CU DIE). For example DW_LNAME_C (0x0003).
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLX_OK etc.

9.9.2.33 dwarf_srclanglname_version()

```
DW_API int dwarf_srclanglname_version (
    Dwarf_Die dw_die,
    const char * dw_returned_verstring,
    Dwarf_Error * dw_error )
```

Return the value of the DW_AT_language_version attribute.

New in v2.1.0 July 2025.

Finds the DW_AT_language_version of the DIE if one is present.

The DIE should be a CU DIE.

Parameters

<i>dw_die</i>	The DIE of interest.
<i>dw_returned</i>	verstring On success returns the language version string from a DW_AT_language_version attributes (normally only found on a CU DIE). For example DW_LNAME_C would return a pointer to "YYYYMM" Never free or dealloc the string returned through dw_returned_verstring, it is in static memory.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.34 dwarf_tag()

```
DW_API int dwarf_tag (
    Dwarf_Die dw_die,
    Dwarf_Half * dw_return_tag,
    Dwarf_Error * dw_error )
```

Get TAG value of DIE.

Parameters

<i>dw_die</i>	The DIE of interest
<i>dw_return_tag</i>	On success, set to the DW_TAG value of the DIE.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.9.2.35 dwarf_validate_die_sibling()

```
DW_API int dwarf_validate_die_sibling (
    Dwarf_Die dw_sibling,
    Dwarf_Off * dw_offset )
```

Validate a sibling DIE.

This is used by dwarfdump (when dwarfdump is checking for valid DWARF) to try to catch a corrupt DIE tree.

This does not identify whether the section is .debug_info or .debug_types, use [dwarf_get_die_infotypes_flag\(\)](#) to determine the section.

See also

[using dwarf_validate_die_sibling](#)

Parameters

<i>dw_sibling</i>	Pass in a DIE returned by dwarf_siblingof_b() .
<i>dw_offset</i>	Set to zero through the pointer.

Returns

Returns DW_DLV_OK if the sibling is at an appropriate place in the section. Otherwise it returns DW_DLV_↔ ERROR indicating the DIE tree is corrupt.

9.10 DIE Attribute and Attribute-Form Details

Functions

- DW_API int [dwarf_attrlist](#) ([Dwarf_Die](#) dw_die, [Dwarf_Attribute](#) **dw_attrbuf, [Dwarf_Signed](#) *dw_attrcount, [Dwarf_Error](#) *dw_error)
Gets the full list of attributes.
- DW_API int [dwarf_hasform](#) ([Dwarf_Attribute](#) dw_attr, [Dwarf_Half](#) dw_form, [Dwarf_Bool](#) *dw_returned_bool, [Dwarf_Error](#) *dw_error)
Sets TRUE if a Dwarf_Attribute has the indicated FORM.
- DW_API int [dwarf_whatform](#) ([Dwarf_Attribute](#) dw_attr, [Dwarf_Half](#) *dw_returned_final_form, [Dwarf_Error](#) *dw_error)
Return the form of the Dwarf_Attribute.
- DW_API int [dwarf_whatform_direct](#) ([Dwarf_Attribute](#) dw_attr, [Dwarf_Half](#) *dw_returned_initial_form, [Dwarf_Error](#) *dw_error)
Return the initial form of the Dwarf_Attribute.
- DW_API int [dwarf_whatattr](#) ([Dwarf_Attribute](#) dw_attr, [Dwarf_Half](#) *dw_returned_attrnum, [Dwarf_Error](#) *dw_↔_error)
Return the attribute number of the Dwarf_Attribute.
- DW_API int [dwarf_formref](#) ([Dwarf_Attribute](#) dw_attr, [Dwarf_Off](#) *dw_return_offset, [Dwarf_Bool](#) *dw_is_info, [Dwarf_Error](#) *dw_error)
Retrieve the CU-relative offset of a reference.
- DW_API int [dwarf_global_formref_b](#) ([Dwarf_Attribute](#) dw_attr, [Dwarf_Off](#) *dw_return_offset, [Dwarf_Bool](#) *dw_offset_is_info, [Dwarf_Error](#) *dw_error)
Return the section-relative offset of a Dwarf_Attribute.
- DW_API int [dwarf_global_formref](#) ([Dwarf_Attribute](#) dw_attr, [Dwarf_Off](#) *dw_return_offset, [Dwarf_Error](#) *dw_↔_error)
Same as dwarf_global_formref_b except...
- DW_API int [dwarf_formsig8](#) ([Dwarf_Attribute](#) dw_attr, [Dwarf_Sig8](#) *dw_returned_sig_bytes, [Dwarf_Error](#) *dw_error)
Return an 8 byte reference form for DW_FORM_ref_sig8.
- DW_API int [dwarf_formsig8_const](#) ([Dwarf_Attribute](#) dw_attr, [Dwarf_Sig8](#) *dw_returned_sig_bytes, [Dwarf_Error](#) *dw_error)
Return an 8 byte reference form for DW_FORM_data8.
- DW_API int [dwarf_formaddr](#) ([Dwarf_Attribute](#) dw_attr, [Dwarf_Addr](#) *dw_returned_addr, [Dwarf_Error](#) *dw_↔_error)
Return the address when the attribute has form address.
- DW_API int [dwarf_get_debug_addr_index](#) ([Dwarf_Attribute](#) dw_attr, [Dwarf_Unsigned](#) *dw_return_index, [Dwarf_Error](#) *dw_error)
Get the addr index of a Dwarf_Attribute.

- DW_API int [dwarf_formflag](#) (Dwarf_Attribute dw_attr, Dwarf_Bool *dw_returned_bool, Dwarf_Error *dw_↵
error)
Return the flag value of a flag form.
- DW_API int [dwarf_formudata](#) (Dwarf_Attribute dw_attr, Dwarf_Unsigned *dw_returned_val, Dwarf_Error
*dw_error)
Return an unsigned value.
- DW_API int [dwarf_formsdata](#) (Dwarf_Attribute dw_attr, Dwarf_Signed *dw_returned_val, Dwarf_Error *dw_↵
_error)
Return a signed value.
- DW_API int [dwarf_formdata16](#) (Dwarf_Attribute dw_attr, Dwarf_Form_Data16 *dw_returned_val,
Dwarf_Error *dw_error)
Return a 16 byte Dwarf_Form_Data16 value.
- DW_API int [dwarf_formblock](#) (Dwarf_Attribute dw_attr, Dwarf_Block **dw_returned_block, Dwarf_Error
*dw_error)
Return an allocated filled-in Form_Block.
- DW_API int [dwarf_formstring](#) (Dwarf_Attribute dw_attr, char **dw_returned_string, Dwarf_Error *dw_error)
Return a pointer to a string.
- DW_API int [dwarf_get_debug_str_index](#) (Dwarf_Attribute dw_attr, Dwarf_Unsigned *dw_return_index,
Dwarf_Error *dw_error)
Return a string index.
- DW_API int [dwarf_formexprloc](#) (Dwarf_Attribute dw_attr, Dwarf_Unsigned *dw_return_exprlen, Dwarf_Ptr
*dw_block_ptr, Dwarf_Error *dw_error)
Return a pointer-to and length-of a block of data.
- DW_API enum [Dwarf_Form_Class](#) [dwarf_get_form_class](#) (Dwarf_Half dw_version, Dwarf_Half dw_attrnum,
Dwarf_Half dw_offset_size, Dwarf_Half dw_form)
Return the FORM_CLASS applicable. Four pieces of information are necessary to get the correct FORM_CLASS.
- DW_API int [dwarf_attr_offset](#) (Dwarf_Die dw_die, Dwarf_Attribute dw_attr, Dwarf_Off *dw_return_offset,
Dwarf_Error *dw_error)
Return the offset of an attribute in its section.
- DW_API int [dwarf_uncompress_integer_block_a](#) (Dwarf_Debug dw_dbg, Dwarf_Unsigned dw_input_↵
length_in_bytes, void *dw_input_block, Dwarf_Unsigned *dw_value_count, Dwarf_Signed **dw_value_↵
array, Dwarf_Error *dw_error)
*Uncompress a block of sleb numbers It's not much of a compression so not much of an uncompression. Developed
by Sun Microsystems and it is unclear if it was ever used.*
- DW_API void [dwarf_dealloc_uncompressed_block](#) (Dwarf_Debug dw_dbg, void *dw_value_array)
Dealloc what dwarf_uncompress_integer_block_a allocated.
- DW_API int [dwarf_convert_to_global_offset](#) (Dwarf_Attribute dw_attr, Dwarf_Off dw_offset, Dwarf_Off *dw_↵
_return_offset, Dwarf_Error *dw_error)
Convert local offset to global offset.
- DW_API void [dwarf_dealloc_attribute](#) (Dwarf_Attribute dw_attr)
Dealloc a Dwarf_Attribute When this call returns the dw_attr is a stale pointer.
- DW_API int [dwarf_discr_list](#) (Dwarf_Debug dw_dbg, Dwarf_Small *dw_blockpointer, Dwarf_Unsigned dw_↵
_blocklen, Dwarf_Disc_Head *dw_dsc_head_out, Dwarf_Unsigned *dw_dsc_array_length_out, Dwarf_Error
*dw_error)
Return an array of discriminant values.
- DW_API int [dwarf_discr_entry_u](#) (Dwarf_Disc_Head dw_dsc, Dwarf_Unsigned dw_entrynum, Dwarf_Half
*dw_out_type, Dwarf_Unsigned *dw_out_discr_low, Dwarf_Unsigned *dw_out_discr_high, Dwarf_Error
*dw_error)
Access a single unsigned discriminant list entry.
- DW_API int [dwarf_discr_entry_s](#) (Dwarf_Disc_Head dw_dsc, Dwarf_Unsigned dw_entrynum, Dwarf_Half
*dw_out_type, Dwarf_Signed *dw_out_discr_low, Dwarf_Signed *dw_out_discr_high, Dwarf_Error *dw_↵
_error)
Access to a single signed discriminant list entry.

9.10.1 Detailed Description

Access to the details of DIEs

9.10.2 Function Documentation

9.10.2.1 dwarf_attr_offset()

```
DW_API int dwarf_attr_offset (
    Dwarf_Die dw_die,
    Dwarf_Attribute dw_attr,
    Dwarf_Off * dw_return_offset,
    Dwarf_Error * dw_error )
```

Return the offset of an attribute in its section.

Parameters

<i>dw_die</i>	The DIE of interest.
<i>dw_attr</i>	A Dwarf_Attribute of interest in this DIE
<i>dw_return_offset</i>	The offset is in .debug_info if the DIE is there. The offset is in .debug_types if the DIE is there.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLX_OK if it succeeds. DW_DLX_NO_ENTRY is impossible.

9.10.2.2 dwarf_attrlist()

```
DW_API int dwarf_attrlist (
    Dwarf_Die dw_die,
    Dwarf_Attribute ** dw_attrbuf,
    Dwarf_Signed * dw_attrcount,
    Dwarf_Error * dw_error )
```

Gets the full list of attributes.

Parameters

<i>dw_die</i>	The DIE from which to pull attributes.
<i>dw_attrbuf</i>	The pointer is set to point to an array of Dwarf_Attribute (pointers to attribute data). This array must eventually be deallocated.
<i>dw_attrcount</i>	The number of entries in the array of pointers. There is no null-pointer to terminate the list, use this count.
<i>dw_error</i>	A place to return error details.

Returns

If it returns DW_DLV_ERROR and dw_error is non-null it creates an Dwarf_Error and places it in this argument. Usually returns DW_DLV_OK.

See also

[Using dwarf_attrlist\(\)](#)

[Using dwarf_attrlist\(\)](#)

9.10.2.3 dwarf_convert_to_global_offset()

```
DW_API int dwarf_convert_to_global_offset (
    Dwarf_Attribute dw_attr,
    Dwarf_Off dw_offset,
    Dwarf_Off * dw_return_offset,
    Dwarf_Error * dw_error )
```

Convert local offset to global offset.

Uses the DW_FORM of the attribute to determine if the dw_offset is local, and if so, adds the CU base offset to adjust dw_offset.

Parameters

<i>dw_attr</i>	The attribute the local offset was extracted from.
<i>dw_offset</i>	The global offset of the attribute.
<i>dw_return_offset</i>	The returned section (global) offset.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds. Returns DW_DLV_ERROR if the dw_attr form is not an offset form (for example, DW_FORM_ref_udata).

9.10.2.4 dwarf_dealloc_attribute()

```
DW_API void dwarf_dealloc_attribute (
    Dwarf_Attribute dw_attr )
```

Dealloc a Dwarf_Attribute When this call returns the dw_attr is a stale pointer.

Parameters

<i>dw_attr</i>	The attribute to dealloc.
----------------	---------------------------

9.10.2.5 dwarf_dealloc_uncompressed_block()

```
DW_API void dwarf_dealloc_uncompressed_block (
    Dwarf_Debug dw_dbg,
    void * dw_value_array )
```

Dealloc what dwarf_uncompress_integer_block_a allocated.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest
<i>dw_value_array</i>	The array was called an array of Dwarf_Signed. We dealloc all of it without needing dw_value_count.

9.10.2.6 dwarf_discr_entry_s()

```
DW_API int dwarf_discr_entry_s (
    Dwarf_Dsc_Head dw_dsc,
    Dwarf_Unsigned dw_entrynum,
    Dwarf_Half * dw_out_type,
    Dwarf_Signed * dw_out_discr_low,
    Dwarf_Signed * dw_out_discr_high,
    Dwarf_Error * dw_error )
```

Access to a single signed discriminant list entry.

The same as dwarf_discr_entry_u except here the values are signed.

9.10.2.7 dwarf_discr_entry_u()

```
DW_API int dwarf_discr_entry_u (
    Dwarf_Dsc_Head dw_dsc,
    Dwarf_Unsigned dw_entrynum,
    Dwarf_Half * dw_out_type,
    Dwarf_Unsigned * dw_out_discr_low,
    Dwarf_Unsigned * dw_out_discr_high,
    Dwarf_Error * dw_error )
```

Access a single unsigned discriminant list entry.

It is up to the caller to know whether the discriminant values are signed or unsigned (therefore to know whether this or dwarf_discr_entry_s. should be called)

Parameters

<i>dw_dsc</i>	The Dwarf_Dsc_Head applicable.
<i>dw_entrynum</i>	Valid values are zero to dw_dsc_array_length_out-1
<i>dw_out_type</i>	On success is set to either DW_DSC_label or DW_DSC_range through the pointer.
<i>dw_out_discr_low</i>	On success set to the lowest in this discriminant range
<i>dw_out_discr_high</i>	On success set to the highest in this discriminant range
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.10.2.8 dwarf_discr_list()

```
DW_API int dwarf_discr_list (
    Dwarf_Debug dw_dbg,
    Dwarf_Small * dw_blockpointer,
    Dwarf_Unsigned dw_blocklen,
    Dwarf_Dsc_Head * dw_dsc_head_out,
    Dwarf_Unsigned * dw_dsc_array_length_out,
    Dwarf_Error * dw_error )
```

Return an array of discriminant values.

This applies if a DW_TAG_variant has one of the DW_FORM_block forms.

See also

[dwarf_formblock](#)

For an example of use and dealloc:

See also

[Using dwarf_discr_list\(\)](#)

Parameters

<i>dw_dbg</i>	The applicable Dwarf_Debug
<i>dw_blockpointer</i>	The bl_data value from a Dwarf_Block.
<i>dw_blocklen</i>	The bl_len value from a Dwarf_Block.
<i>dw_dsc_head_out</i>	On success returns a pointer to an array of discriminant values in an opaque struct.
<i>dw_dsc_array_length_out</i>	On success returns the number of entries in the dw_dsc_head_out array.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.10.2.9 dwarf_formaddr()

```
DW_API int dwarf_formaddr (
    Dwarf_Attribute dw_attr,
    Dwarf_Addr * dw_returned_addr,
    Dwarf_Error * dw_error )
```

Return the address when the attribute has form address.

There are several address forms, some of them indexed.

Parameters

<i>dw_attr</i>	The Dwarf_Attribute of interest.
<i>dw_returned_addr</i>	On success this set through the pointer to the address in the attribute.
<i>dw_error</i>	A place to return error details.

Returns

On success returns DW_DLV_OK sets *dw_returned_addr* . If attribute is passed in NULL or the attribute is badly broken or the address cannot be retrieved the call returns DW_DLV_ERROR. Never returns DW_DLV_NO_ENTRY.

9.10.2.10 dwarf_formblock()

```
DW_API int dwarf_formblock (
    Dwarf_Attribute dw_attr,
    Dwarf_Block ** dw_returned_block,
    Dwarf_Error * dw_error )
```

Return an allocated filled-in Form_Block.

It is an error if the DW_FORM in the attribute is not a block form. DW_FORM_block2 is an example of a block form.

See also

[Dwarf_Block](#)

[Using dwarf_discr_list\(\)](#)

Parameters

<i>dw_attr</i>	The Dwarf_Attribute of interest.
<i>dw_returned_block</i>	Allocates a Dwarf_Block and returns a pointer to the filled-in block.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds. Never returns DW_DLV_NO_ENTRY.

9.10.2.11 dwarf_formdata16()

```
DW_API int dwarf_formdata16 (
    Dwarf_Attribute dw_attr,
    Dwarf_Form_Data16 * dw_returned_val,
    Dwarf_Error * dw_error )
```

Return a 16 byte Dwarf_Form_Data16 value.

We just store the bytes in a struct, we have no 16 byte integer type. It is an error if the FORM is not DW_FORM_data16

See also

[Dwarf_Form_Data16](#)

Parameters

<i>dw_attr</i>	The Dwarf_Attribute of interest.
<i>dw_returned_val</i>	Copies the 16 byte value into the pointed to area.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds. Never returns DW_DLV_NO_ENTRY.

9.10.2.12 dwarf_formexprloc()

```
DW_API int dwarf_formexprloc (
    Dwarf_Attribute dw_attr,
    Dwarf_Unsigned * dw_return_exprlen,
    Dwarf_Ptr * dw_block_ptr,
    Dwarf_Error * dw_error )
```

Return a pointer-to and length-of a block of data.

Parameters

<i>dw_attr</i>	The Dwarf_Attribute of interest.
<i>dw_return_exprlen</i>	Returns the length in bytes of the block if it succeeds.
<i>dw_block_ptr</i>	Returns a pointer to the first byte of the block of data if it succeeds.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds. If the attribute form is not DW_FORM_exprloc it returns DW_DLV_ERROR and sets dw_error to point to the error details.

9.10.2.13 dwarf_formflag()

```
DW_API int dwarf_formflag (
    Dwarf_Attribute dw_attr,
    Dwarf_Bool * dw_returned_bool,
    Dwarf_Error * dw_error )
```

Return the flag value of a flag form.

It is an error if the FORM is not a flag form.

Parameters

<i>dw_attr</i>	The Dwarf_Attribute of interest.
<i>dw_returned_bool</i>	Returns either TRUE or FALSE through the pointer.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds. Never returns DW_DLV_NO_ENTRY.

9.10.2.14 dwarf_formref()

```
DW_API int dwarf_formref (
    Dwarf_Attribute dw_attr,
    Dwarf_Off * dw_return_offset,
    Dwarf_Bool * dw_is_info,
    Dwarf_Error * dw_error )
```

Retrieve the CU-relative offset of a reference.

The DW_FORM of the attribute must be one of a small set of local reference forms: DW_FORM_ref<n> or DW_FORM_↔_udata.

Parameters

<i>dw_attr</i>	The Dwarf_Attribute of interest.
<i>dw_return_offset</i>	Returns the CU-relative offset through the pointer.
<i>dw_is_info</i>	Returns a flag through the pointer. TRUE if the offset is in .debug_info, FALSE if it is in .debug_types
<i>dw_error</i>	A place to return error details.

Returns

Returns DW_DLV_OK and sets dw_returned_attrnum If attribute is passed in NULL or the attribute is badly broken or the FORM of this attribute is not one of the small set of local references the call returns DW_DLV↔_ERROR. Never returns DW_DLV_NO_ENTRY;

9.10.2.15 dwarf_formsdata()

```
DW_API int dwarf_formsdata (
    Dwarf_Attribute dw_attr,
    Dwarf_Signed * dw_returned_val,
    Dwarf_Error * dw_error )
```

Return a signed value.

The form must be a signed integral type. It is an error otherwise.

Parameters

<i>dw_attr</i>	The Dwarf_Attribute of interest.
<i>dw_returned_val</i>	On success returns the signed value through the pointer.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds. Never returns DW_DLV_NO_ENTRY.

9.10.2.16 dwarf_formsig8()

```
DW_API int dwarf_formsig8 (
    Dwarf_Attribute dw_attr,
    Dwarf_Sig8 * dw_returned_sig_bytes,
    Dwarf_Error * dw_error )
```

Return an 8 byte reference form for DW_FORM_ref_sig8.

Parameters

<i>dw_attr</i>	The Dwarf_Attribute of interest.
<i>dw_returned_sig_bytes</i>	On success returns DW_DLV_OK and copies the 8 bytes into dw_returned_sig_bytes.
<i>dw_error</i>	A place to return error details.

Returns

On success returns DW_DLV_OK and copies the 8 bytes into dw_returned_sig_bytes. If attribute is passed in NULL or the attribute is badly broken the call returns DW_DLV_ERROR. If the dw_attr has a form other than DW_FORM_ref_sig8 the function returns DW_DLV_NO_ENTRY

9.10.2.17 dwarf_formsig8_const()

```
DW_API int dwarf_formsig8_const (
    Dwarf_Attribute dw_attr,
    Dwarf_Sig8 * dw_returned_sig_bytes,
    Dwarf_Error * dw_error )
```

Return an 8 byte reference form for DW_FORM_data8.

Parameters

<i>dw_attr</i>	The Dwarf_Attribute of interest.
<i>dw_returned_sig_bytes</i>	On success Returns DW_DLV_OK and copies the 8 bytes into dw_returned_sig_bytes.
<i>dw_error</i>	A place to return error details.

Returns

On success returns DW_DLV_OK and copies the 8 bytes into dw_returned_sig_bytes. If attribute is passed in NULL or the attribute is badly broken the call returns DW_DLV_ERROR. If the dw_attr has a form other than DW_FORM_data8 the function returns DW_DLV_NO_ENTRY

9.10.2.18 dwarf_formstring()

```
DW_API int dwarf_formstring (
    Dwarf_Attribute dw_attr,
```

```
char ** dw_returned_string,
Dwarf_Error * dw_error )
```

Return a pointer to a string.

Parameters

<i>dw_attr</i>	The Dwarf_Attribute of interest.
<i>dw_returned_string</i>	On success puts a pointer to a string existing in an appropriate DWARF section into <i>dw_returned_string</i> . Never free() or dealloc the returned string.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.10.2.19 dwarf_formudata()

```
DW_API int dwarf_formudata (
    Dwarf_Attribute dw_attr,
    Dwarf_Unsigned * dw_returned_val,
    Dwarf_Error * dw_error )
```

Return an unsigned value.

The form can be an unsigned or signed integral type but if it is a signed type the value must be non-negative. It is an error otherwise.

Parameters

<i>dw_attr</i>	The Dwarf_Attribute of interest.
<i>dw_returned_val</i>	On success returns the unsigned value through the pointer.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds. Never returns DW_DLV_NO_ENTRY.

9.10.2.20 dwarf_get_debug_addr_index()

```
DW_API int dwarf_get_debug_addr_index (
    Dwarf_Attribute dw_attr,
    Dwarf_Unsigned * dw_return_index,
    Dwarf_Error * dw_error )
```

Get the addr index of a Dwarf_Attribute.

So a consumer can get the index when the object with the actual .debug_addr section is elsewhere (Debug Fission). Or if the caller just wants the index. Only call it when you know it should have an index address FORM such as DW_FORM_addrx1 or one of the GNU address index forms.

Parameters

<i>dw_attr</i>	The Dwarf_Attribute of interest.
<i>dw_return_index</i>	If successful it returns the index through the pointer.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds. Never returns DW_DLV_NO_ENTRY.

9.10.2.21 dwarf_get_debug_str_index()

```
DW_API int dwarf_get_debug_str_index (
    Dwarf_Attribute dw_attr,
    Dwarf_Unsigned * dw_return_index,
    Dwarf_Error * dw_error )
```

Return a string index.

Parameters

<i>dw_attr</i>	The Dwarf_Attribute of interest.
<i>dw_return_index</i>	If the form is a string index form (for example DW_FORM_strx) the string index value is returned via the pointer.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds. If the attribute form is not one of the string index forms it returns DW_DLV_ERROR and sets dw_error to point to the error details.

9.10.2.22 dwarf_get_form_class()

```
DW_API enum Dwarf_Form_Class dwarf_get_form_class (
    Dwarf_Half dw_version,
    Dwarf_Half dw_attrnum,
    Dwarf_Half dw_offset_size,
    Dwarf_Half dw_form )
```

Return the FORM_CLASS applicable. Four pieces of information are necessary to get the correct FORM_CLASS.

Parameters

<i>dw_version</i>	The CU's DWARF version. Standard numbers are 2,3,4, or 5.
<i>dw_attrnum</i>	For example DW_AT_name
<i>dw_offset_size</i>	The offset size applicable to the compilation unit relevant to the attribute and form.
<i>dw_form</i>	The FORM number, for example DW_FORM_data4

Returns

Returns a form class, for example DW_FORM_CLASS_CONSTANT. The FORM_CLASS names are mentioned (for example as 'address' in Table 2.3 of DWARF5) but are not assigned formal names & numbers in the standard.

9.10.2.23 dwarf_global_formref()

```
DW_API int dwarf_global_formref (
    Dwarf_Attribute dw_attr,
    Dwarf_Off * dw_return_offset,
    Dwarf_Error * dw_error )
```

Same as dwarf_global_formref_b except...

See also

[dwarf_global_formref_b](#)

This is the same, except there is no dw_offset_is_info pointer so in the case of DWARF4 and DW_FORM_ref_sig8 it is not possible to determine which section the offset applies to!

9.10.2.24 dwarf_global_formref_b()

```
DW_API int dwarf_global_formref_b (
    Dwarf_Attribute dw_attr,
    Dwarf_Off * dw_return_offset,
    Dwarf_Bool * dw_offset_is_info,
    Dwarf_Error * dw_error )
```

Return the section-relative offset of a Dwarf_Attribute.

The target section of the returned offset can be in various sections depending on the FORM. Only a DW_FORM_ref_sig8 can change the returned offset of a .debug_info DIE via a lookup into .debug_types by changing dw_offset_is_info to FALSE (DWARF4).

The caller must determine the target section from the FORM.

Parameters

<i>dw_attr</i>	The Dwarf_Attribute of interest.
<i>dw_return_offset</i>	Returns the CU-relative offset through the pointer.
<i>dw_offset_is_info</i>	For references to DIEs this informs whether the target DIE (the target the offset refers to) is in .debug_info or .debug_types. For non-DIE targets this field is not meaningful. Refer to the attribute FORM to determine the target section of the offset.
<i>dw_error</i>	A place to return error details.

Returns

Returns DW_DLV_OK and sets dw_return_offset and dw_offset_is_info. If attribute is passed in NULL or the attribute is badly broken or the FORM of this attribute is not one of the many reference types the call returns DW_DLV_ERROR. Never returns DW_DLV_NO_ENTRY;

9.10.2.25 dwarf_hasform()

```
DW_API int dwarf_hasform (
    Dwarf_Attribute dw_attr,
    Dwarf_Half dw_form,
    Dwarf_Bool * dw_returned_bool,
    Dwarf_Error * dw_error )
```

Sets TRUE if a Dwarf_Attribute has the indicated FORM.

Parameters

<i>dw_attr</i>	The Dwarf_Attribute of interest.
<i>dw_form</i>	The DW_FORM you are asking about, DW_FORM_strp for example.
<i>dw_returned_bool</i>	The pointer passed in must be a valid non-null pointer to a Dwarf_Bool. On success, sets the value to TRUE or FALSE.
<i>dw_error</i>	A place to return error details.

Returns

Returns DW_DLV_OK and sets dw_returned_bool. If attribute is passed in NULL or the attribute is badly broken the call returns DW_DLV_ERROR. Never returns DW_DLV_NO_ENTRY;

9.10.2.26 dwarf_uncompress_integer_block_a()

```
DW_API int dwarf_uncompress_integer_block_a (
    Dwarf_Debug dw_dbg,
    Dwarf_Unsigned dw_input_length_in_bytes,
    void * dw_input_block,
    Dwarf_Unsigned * dw_value_count,
    Dwarf_Signed ** dw_value_array,
    Dwarf_Error * dw_error )
```

Uncompress a block of sleb numbers It's not much of a compression so not much of an uncompression. Developed by Sun Microsystems and it is unclear if it was ever used.

See also

[dwarf_dealloc_uncompressed_block](#)

9.10.2.27 dwarf_whatattr()

```
DW_API int dwarf_whatattr (
    Dwarf_Attribute dw_attr,
    Dwarf_Half * dw_returned_attrnum,
    Dwarf_Error * dw_error )
```

Return the attribute number of the Dwarf_Attribute.

Parameters

<i>dw_attr</i>	The Dwarf_Attribute of interest.
<i>dw_returned_attrnum</i>	The attribute number of the attribute is returned through the pointer. For example, DW_AT_name
<i>dw_error</i>	A place to return error details.

Returns

Returns DW_DLV_OK and sets dw_returned_attrnum If attribute is passed in NULL or the attribute is badly broken the call returns DW_DLV_ERROR. Never returns DW_DLV_NO_ENTRY;

9.10.2.28 dwarf_whatform()

```
DW_API int dwarf_whatform (
    Dwarf_Attribute dw_attr,
    Dwarf_Half * dw_returned_final_form,
    Dwarf_Error * dw_error )
```

Return the form of the Dwarf_Attribute.

Parameters

<i>dw_attr</i>	The Dwarf_Attribute of interest.
<i>dw_returned_final_form</i>	The form of the item is returned through the pointer. If the base form is DW_FORM_indirect the function resolves the final form and returns that final form.
<i>dw_error</i>	A place to return error details.

Returns

Returns DW_DLV_OK and sets dw_returned_final_form If attribute is passed in NULL or the attribute is badly broken the call returns DW_DLV_ERROR. Never returns DW_DLV_NO_ENTRY;

9.10.2.29 dwarf_whatform_direct()

```
DW_API int dwarf_whatform_direct (
    Dwarf_Attribute dw_attr,
    Dwarf_Half * dw_returned_initial_form,
    Dwarf_Error * dw_error )
```

Return the initial form of the Dwarf_Attribute.

Parameters

<i>dw_attr</i>	The Dwarf_Attribute of interest.
<i>dw_returned_initial_form</i>	The form of the item is returned through the pointer. If the base form is DW_FORM_indirect the value set is DW_FORM_indirect.
<i>dw_error</i>	A place to return error details.

Returns

Returns DW_DLV_OK and sets dw_returned_initial_form. If attribute is passed in NULL or the attribute is badly broken the call returns DW_DLV_ERROR. Never returns DW_DLV_NO_ENTRY;

9.11 Line Table For a CU

Functions

- DW_API int [dwarf_srcfiles](#) ([Dwarf_Die](#) dw_cu_die, char ***dw_srcfiles, [Dwarf_Signed](#) *dw_filecount, [Dwarf_Error](#) *dw_error)
The list of source files from the line table header.
- DW_API int [dwarf_srclines_b](#) ([Dwarf_Die](#) dw_cudie, [Dwarf_Unsigned](#) *dw_version_out, [Dwarf_Small](#) *dw_table_count, [Dwarf_Line_Context](#) *dw_linecontext, [Dwarf_Error](#) *dw_error)
Initialize Dwarf_Line_Context for line table access.
- DW_API int [dwarf_srclines_from_linecontext](#) ([Dwarf_Line_Context](#) dw_linecontext, [Dwarf_Line](#) **dw_linebuf, [Dwarf_Signed](#) *dw_linecount, [Dwarf_Error](#) *dw_error)
Access source lines from line context.
- DW_API int [dwarf_srclines_two_level_from_linecontext](#) ([Dwarf_Line_Context](#) dw_context, [Dwarf_Line](#) **dw_linebuf, [Dwarf_Signed](#) *dw_linecount, [Dwarf_Line](#) **dw_linebuf_actuals, [Dwarf_Signed](#) *dw_linecount_actuals, [Dwarf_Error](#) *dw_error)
Returns line table counts and data.
- DW_API void [dwarf_srclines_dealloc_b](#) ([Dwarf_Line_Context](#) dw_context)
Dealloc the memory allocated by dwarf_srclines_b.
- DW_API int [dwarf_srclines_table_offset](#) ([Dwarf_Line_Context](#) dw_context, [Dwarf_Unsigned](#) *dw_offset, [Dwarf_Error](#) *dw_error)
Return the srclines table offset.
- DW_API int [dwarf_srclines_comp_dir](#) ([Dwarf_Line_Context](#) dw_context, const char **dw_compilation_directory, [Dwarf_Error](#) *dw_error)
Compilation Directory name for the CU.
- DW_API int [dwarf_srclines_subprog_count](#) ([Dwarf_Line_Context](#) dw_context, [Dwarf_Signed](#) *dw_count, [Dwarf_Error](#) *dw_error)
Subprog count: Part of the two-level line table extension.
- DW_API int [dwarf_srclines_subprog_data](#) ([Dwarf_Line_Context](#) dw_context, [Dwarf_Signed](#) dw_index, const char **dw_name, [Dwarf_Unsigned](#) *dw_decl_file, [Dwarf_Unsigned](#) *dw_decl_line, [Dwarf_Error](#) *dw_error)
Retrieve data from the line table subprog array.
- DW_API int [dwarf_srclines_files_indexes](#) ([Dwarf_Line_Context](#) dw_context, [Dwarf_Signed](#) *dw_baseindex, [Dwarf_Signed](#) *dw_count, [Dwarf_Signed](#) *dw_endindex, [Dwarf_Error](#) *dw_error)
Return values easing indexing line table file numbers. Count is the real count of files array entries. Since DWARF 2,3,4 are zero origin indexes and DWARF5 and later are one origin, this function replaces dwarf_srclines_files_count().
- DW_API int [dwarf_srclines_files_data_b](#) ([Dwarf_Line_Context](#) dw_context, [Dwarf_Signed](#) dw_index, const char **dw_name, [Dwarf_Unsigned](#) *dw_directory_index, [Dwarf_Unsigned](#) *dw_last_mod_time, [Dwarf_Unsigned](#) *dw_file_length, [Dwarf_Form_Data16](#) **dw_md5ptr, [Dwarf_Error](#) *dw_error)
Access data for each line table file.
- DW_API int [dwarf_srclines_include_dir_count](#) ([Dwarf_Line_Context](#) dw_line_context, [Dwarf_Signed](#) *dw_count, [Dwarf_Error](#) *dw_error)
Return the number of include directories in the Line Table.
- DW_API int [dwarf_srclines_include_dir_data](#) ([Dwarf_Line_Context](#) dw_line_context, [Dwarf_Signed](#) dw_index, const char **dw_name, [Dwarf_Error](#) *dw_error)
Return the include directories in the Line Table.
- DW_API int [dwarf_srclines_version](#) ([Dwarf_Line_Context](#) dw_line_context, [Dwarf_Unsigned](#) *dw_version, [Dwarf_Small](#) *dw_table_count, [Dwarf_Error](#) *dw_error)

The DWARF version number of this compile-unit.

- DW_API int `dwarf_linebeginstatement` (Dwarf_Line dw_line, Dwarf_Bool *dw_returned_bool, Dwarf_Error *dw_error)

Read Line beginstatement register.

- DW_API int `dwarf_lineendsequence` (Dwarf_Line dw_line, Dwarf_Bool *dw_returned_bool, Dwarf_Error *dw_error)

Read Line endsequence register flag.

- DW_API int `dwarf_lineno` (Dwarf_Line dw_line, Dwarf_Unsigned *dw_returned_lineno, Dwarf_Error *dw_error)

Read Line line register.

- DW_API int `dwarf_line_srcfileno` (Dwarf_Line dw_line, Dwarf_Unsigned *dw_returned_filenum, Dwarf_Error *dw_error)

Read Line file register.

- DW_API int `dwarf_line_is_addr_set` (Dwarf_Line dw_line, Dwarf_Bool *dw_is_addr_set, Dwarf_Error *dw_error)

Is the Dwarf_Line address from DW_LNS_set_address? This is not a line register, but it is a flag set by the library in each Dwarf_Line, and it is derived from reading the line table.

- DW_API int `dwarf_lineaddr` (Dwarf_Line dw_line, Dwarf_Addr *dw_returned_addr, Dwarf_Error *dw_error)

Return the address of the Dwarf_Line.

- DW_API int `dwarf_lineoff_b` (Dwarf_Line dw_line, Dwarf_Unsigned *dw_returned_lineoffset, Dwarf_Error *dw_error)

Return a column number through the pointer.

- DW_API int `dwarf_linesrc` (Dwarf_Line dw_line, char **dw_returned_name, Dwarf_Error *dw_error)

Return the file name applicable to the Dwarf_Line.

- DW_API int `dwarf_lineblock` (Dwarf_Line dw_line, Dwarf_Bool *dw_returned_bool, Dwarf_Error *dw_error)

Return the basic_block line register.

- DW_API int `dwarf_prologue_end_etc` (Dwarf_Line dw_line, Dwarf_Bool *dw_prologue_end, Dwarf_Bool *dw_epilogue_begin, Dwarf_Unsigned *dw_isa, Dwarf_Unsigned *dw_discriminator, Dwarf_Error *dw_error)

Return various line table registers in one call.

- DW_API int `dwarf_linelogical` (Dwarf_Line dw_line, Dwarf_Unsigned *dw_returned_logical, Dwarf_Error *dw_error)

Experimental Two-level logical Row Number Experimental two level line tables. Not explained here. When reading from an actuals table, `dwarf_line_logical()` returns the logical row number for the line.

- DW_API int `dwarf_linecontext` (Dwarf_Line dw_line, Dwarf_Unsigned *dw_returned_context, Dwarf_Error *dw_error)

Experimental Two-level line tables call contexts Experimental two level line tables. Not explained here. When reading from a logicals table, `dwarf_linecontext()` returns the logical row number corresponding the the calling context for an inlined call.

- DW_API int `dwarf_line_subprogn` (Dwarf_Line, Dwarf_Unsigned *, Dwarf_Error *)

Two-level line tables get subprogram number Experimental two level line tables. Not explained here. When reading from a logicals table, `dwarf_line_subprogn()` returns the index in the subprograms table of the inlined subprogram. Currently this always returns zero through the pointer as the relevant field is never updated from the default of zero.

- DW_API int `dwarf_line_subprog` (Dwarf_Line, char **, char **, Dwarf_Unsigned *, Dwarf_Error *)

Two-level line tables get subprog, file, line Experimental two level line tables. Not explained here. When reading from a logicals table, `dwarf_line_subprog()` returns the name of the inlined subprogram, its declaration filename, and its declaration line number, if available.

- DW_API int `dwarf_check_lineheader_b` (Dwarf_Die dw_cu_die, int *dw_errcount_out, Dwarf_Error *dw_error)

Access to detailed line table header issues.

- DW_API int `dwarf_print_lines` (Dwarf_Die dw_cu_die, Dwarf_Error *dw_error, int *dw_errorcount_out)

Print line information in great detail.

- DW_API struct `Dwarf_Printf_Callback_Info_s` `dwarf_register_printf_callback` (Dwarf_Debug dw_dbg, struct `Dwarf_Printf_Callback_Info_s` *dw_callbackinfo)

For line details this records callback details.

9.11.1 Detailed Description

Access to all the line table details.

9.11.2 Function Documentation

9.11.2.1 dwarf_check_lineheader_b()

```
DW_API int dwarf_check_lineheader_b (
    Dwarf_Die dw_cu_die,
    int * dw_errcount_out,
    Dwarf_Error * dw_error )
```

Access to detailed line table header issues.

Lets the caller get detailed messages about some compiler errors we detect. Calls back, the caller should do something with the messages (likely just print them). The lines passed back already have newlines.

See also

dwarf_check_lineheader(b)
[Dwarf_Printf_Callback_Info_s](#)

Parameters

<i>dw_cu_die</i>	The CU DIE of interest
<i>dw_error</i>	If DW_DLV_ERROR this shows one error encountered.
<i>dw_errcount_out</i>	Returns the count of detected errors through the pointer.

Returns

DW_DLV_OK etc.

9.11.2.2 dwarf_line_is_addr_set()

```
DW_API int dwarf_line_is_addr_set (
    Dwarf_Line dw_line,
    Dwarf_Bool * dw_is_addr_set,
    Dwarf_Error * dw_error )
```

Is the Dwarf_Line address from DW_LNS_set_address? This is not a line register, but it is a flag set by the library in each Dwarf_Line, and it is derived from reading the line table.

Parameters

<i>dw_line</i>	The Dwarf_Line of interest.
<i>dw_is_addr_set</i>	On success it sets the flag to TRUE or FALSE.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.3 dwarf_line_srcfileno()

```
DW_API int dwarf_line_srcfileno (
    Dwarf_Line dw_line,
    Dwarf_Unsigned * dw_returned_filenum,
    Dwarf_Error * dw_error )
```

Read Line file register.

[Line Table Registers](#)**Parameters**

<i>dw_line</i>	The Dwarf_Line of interest.
<i>dw_returned_filenum</i>	On success it sets the value to the file number from the Dwarf_Line file register
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.4 dwarf_lineaddr()

```
DW_API int dwarf_lineaddr (
    Dwarf_Line dw_line,
    Dwarf_Addr * dw_returned_addr,
    Dwarf_Error * dw_error )
```

Return the address of the Dwarf_Line.

[Line Table Registers](#)**Parameters**

<i>dw_line</i>	The Dwarf_Line of interest.
<i>dw_returned_addr</i>	On success it sets the value to the value of the address register in the Dwarf_Line.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.5 dwarf_linebeginstatement()

```
DW_API int dwarf_linebeginstatement (
    Dwarf_Line dw_line,
```

```
Dwarf_Bool * dw_returned_bool,
Dwarf_Error * dw_error )
```

Read Line beginstatement register.

Line Table Registers

Parameters

<i>dw_line</i>	The Dwarf_Line of interest.
<i>dw_returned_bool</i>	On success it sets the value TRUE (if the dw_line has the is_stmt register set) and FALSE if is_stmt is not set.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.6 dwarf_lineblock()

```
DW_API int dwarf_lineblock (
    Dwarf_Line dw_line,
    Dwarf_Bool * dw_returned_bool,
    Dwarf_Error * dw_error )
```

Return the basic_block line register.

Line Table Registers

Parameters

<i>dw_line</i>	The Dwarf_Line of interest.
<i>dw_returned_bool</i>	On success it sets the flag to TRUE or FALSE from the basic_block register in the line table.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.7 dwarf_lineendsequence()

```
DW_API int dwarf_lineendsequence (
    Dwarf_Line dw_line,
    Dwarf_Bool * dw_returned_bool,
    Dwarf_Error * dw_error )
```

Read Line endsequence register flag.

Line Table Registers

Parameters

<i>dw_line</i>	The Dwarf_Line of interest.
<i>dw_returned_bool</i>	On success it sets the value TRUE (if the dw_line has the end_sequence register set) and FALSE if end_sequence is not set.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.8 dwarf_lineno()

```
DW_API int dwarf_lineno (
    Dwarf_Line dw_line,
    Dwarf_Unsigned * dw_returned_linenum,
    Dwarf_Error * dw_error )
```

Read Line line register.

[Line Table Registers](#)

Parameters

<i>dw_line</i>	The Dwarf_Line of interest.
<i>dw_returned_linenum</i>	On success it sets the value to the line number from the Dwarf_Line line register
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.9 dwarf_lineoff_b()

```
DW_API int dwarf_lineoff_b (
    Dwarf_Line dw_line,
    Dwarf_Unsigned * dw_returned_lineoffset,
    Dwarf_Error * dw_error )
```

Return a column number through the pointer.

[Line Table Registers](#)

Parameters

<i>dw_line</i>	The Dwarf_Line of interest.
<i>dw_returned_lineoffset</i>	On success it sets the value to the column register from the Dwarf_Line.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.10 dwarf_linesrc()

```
DW_API int dwarf_linesrc (
    Dwarf_Line dw_line,
    char ** dw_returned_name,
    Dwarf_Error * dw_error )
```

Return the file name applicable to the Dwarf_Line.

[Line Table Registers](#)**Parameters**

<i>dw_line</i>	The Dwarf_Line of interest.
<i>dw_returned_name</i>	On success it reads the file register and finds constructs a file name from a directory and filename there and and returns a pointer to that string through the pointer. It is necessary to deallocate the returned string with <code>dwarf_dealloc(dbg, lsrc_filename, DW_DLA_STRING)</code> ; (Older versions of this function incorrectly said not to free() or dwarf_dealloc() .)
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

See also

[Using dwarf_srclines_b\(\) and linecontext](#)

9.11.2.11 dwarf_print_lines()

```
DW_API int dwarf_print_lines (
    Dwarf_Die dw_cu_die,
    Dwarf_Error * dw_error,
    int * dw_errorcount_out )
```

Print line information in great detail.

`dwarf_print_lines` lets the caller prints line information for a CU in great detail. Does not use `printf`. Instead it calls back to the application using a function pointer once per line-to-print. The lines passed back already have any needed newlines.

`dwarfdump` uses this function for verbose printing of line table data.

Failing to call the [dwarf_register_printf_callback\(\)](#) function will prevent the lines from being passed back but such omission is not an error. The same function, but focused on checking for errors is [dwarf_check_lineheader_b\(\)](#).

See also

[Dwarf_Printf_Callback_Info_s](#)

Parameters

<i>dw_cu_die</i>	The CU DIE of interest
<i>dw_error</i>	
<i>dw_errorcount_out</i>	

Returns

DW_DLV_OK etc.

9.11.2.12 `dwarf_prologue_end_etc()`

```
DW_API int dwarf_prologue_end_etc (
    Dwarf_Line dw_line,
    Dwarf_Bool * dw_prologue_end,
    Dwarf_Bool * dw_epilogue_begin,
    Dwarf_Unsigned * dw_isa,
    Dwarf_Unsigned * dw_discriminator,
    Dwarf_Error * dw_error )
```

Return various line table registers in one call.

Line Table Registers

Parameters

<i>dw_line</i>	The Dwarf_Line of interest.
<i>dw_prologue_end</i>	On success it sets the flag to TRUE or FALSE from the prologue_end register in the line table.
<i>dw_epilogue_begin</i>	On success it sets the flag to TRUE or FALSE from the epilogue_begin register in the line table.
<i>dw_isa</i>	On success it sets the value to the value of from the isa register in the line table.
<i>dw_discriminator</i>	On success it sets the value to the value of from the discriminator register in the line table.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.13 `dwarf_register_printf_callback()`

```
DW_API struct Dwarf_Printf_Callback_Info_s dwarf_register_printf_callback (
    Dwarf_Debug dw_dbg,
    struct Dwarf_Printf_Callback_Info_s * dw_callbackinfo )
```

For line details this records callback details.

Not usually needed. It is a way to check (while using the library) what callback data is in use or to update that callback data.

See also

[Dwarf_Printf_Callback_Info_s](#)

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_callbackinfo</i>	If non-NULL pass in a pointer to your instance of struct Dwarf_Printf_Callback_Info_s with all the fields filled in.

Returns

If *dw_callbackinfo* NULL it returns a copy of the current [Dwarf_Printf_Callback_Info_s](#) for *dw_dbg*. Otherwise it returns the previous contents of the struct.

9.11.2.14 dwarf_srcfiles()

```
DW_API int dwarf_srcfiles (
    Dwarf_Die dw_cu_die,
    char *** dw_srcfiles,
    Dwarf_Signed * dw_filecount,
    Dwarf_Error * dw_error )
```

The list of source files from the line table header.

The array returned by this function applies to a single compilation unit (CU).

The returned array is indexed from 0 (zero) to *dw_filecount*-1 when the function returns DW_DLV_OK.

In referencing the array via a file-number from a **DW_AT_decl_file** or **DW_AT_call_file** attribute one needs to know if the CU is DWARF5 or not.

Line Table Version numbers match compilation unit version numbers except that an experimental line table with line table version 0xfe06 has sometimes been used with DWARF4.

For DWARF5:

The file-number from a **DW_AT_decl_file** or **DW_AT_call_file** is the proper index into the array of string pointers.

For DWARF2,3,4, including experimental line table version 0xfe06 and a file-number from a **DW_AT_decl_file** or **DW_AT_call_file**:

1. If the file-number is zero there is no file name to find.
2. Otherwise subtract one(1) from the file-number and use the new value as the index into the array of string pointers.

The name strings returned are each assembled in the following way by [dwarf_srcfiles\(\)](#):

1. The file number denotes a name in the line table header.
2. If the name is not a full path (i.e. not starting with / in posix/linux/Macos) then prepend the appropriate directory string from the line table header.
3. If the name is still not a full path then prepend the content of the DW_AT_comp_dir attribute of the CU DIE.

To retrieve the line table version call [dwarf_srclines_b\(\)](#) and [dwarf_srclines_version\(\)](#).

See also

[Using dwarf_srclines_b\(\)](#)

Parameters

<i>dw_cu_die</i>	The CU DIE in this CU.
<i>dw_srcfiles</i>	On success allocates an array of pointers to strings and for each such, computes the fullest path possible given the CU DIE data for each file name listed in the line table header.
<i>dw_filecount</i>	On success returns the number of entries in the array of pointers to strings. The number returned is non-negative.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds. If there is no `.debug_line[.dwo]` returns DW_DLV_NO_ENTRY.

See also

[Using dwarf_srcfiles\(\)](#)

9.11.2.15 dwarf_srclines_b()

```
DW_API int dwarf_srclines_b (
    Dwarf_Die dw_cudie,
    Dwarf_Unsigned * dw_version_out,
    Dwarf_Small * dw_table_count,
    Dwarf_Line_Context * dw_linecontext,
    Dwarf_Error * dw_error )
```

Initialize Dwarf_Line_Context for line table access.

Returns Dwarf_Line_Context pointer, needed for access to line table data. Returns the line table version number (needed to use [dwarf_srcfiles\(\)](#) properly).

See also

[Using dwarf_srclines_b\(\)](#)

[Using dwarf_srclines_b\(\) and linecontext](#)

Parameters

<i>dw_cudie</i>	The Compilation Unit (CU) DIE of interest.
<i>dw_version_out</i>	The DWARF Line Table version number (Standard: 2,3,4, or 5) Version 0xf006 is an experimental (two-level) line table.
<i>dw_table_count</i>	Zero or one means this is a normal DWARF line table. Two means this is an experimental two-level line table.
<i>dw_linecontext</i>	On success sets the pointer to point to an opaque structure usable for further queries.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.16 dwarf_srclines_comp_dir()

```
DW_API int dwarf_srclines_comp_dir (
    Dwarf_Line_Context dw_context,
    const char ** dw_compilation_directory,
    Dwarf_Error * dw_error )
```

Compilation Directory name for the CU.

Do not free() or dealloc the string, it is in a dwarf section.

Parameters

<i>dw_context</i>	The Line Context of interest.
<i>dw_compilation_directory</i>	On success returns a pointer to a string identifying the compilation directory of the CU.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLX_OK if it succeeds.

9.11.2.17 dwarf_srclines_dealloc_b()

```
DW_API void dwarf_srclines_dealloc_b (
    Dwarf_Line_Context dw_context )
```

Dealloc the memory allocated by dwarf_srclines_b.

The way to deallocate (free) a Dwarf_Line_Context

Parameters

<i>dw_context</i>	The context to be deallocated (freed). On return the pointer passed in is stale and calling applications should zero the pointer.
-------------------	---

9.11.2.18 dwarf_srclines_files_data_b()

```
DW_API int dwarf_srclines_files_data_b (
    Dwarf_Line_Context dw_context,
    Dwarf_Signed dw_index_in,
    const char ** dw_name,
    Dwarf_Unsigned * dw_directory_index,
    Dwarf_Unsigned * dw_last_mod_time,
    Dwarf_Unsigned * dw_file_length,
    Dwarf_Form_Data16 ** dw_md5ptr,
    Dwarf_Error * dw_error )
```

Access data for each line table file.

Has the md5ptr field so cases where DW_LNCT_MD5 is present can return pointer to the MD5 value. With DWARF 5 index starts with 0. [dwarf_srclines_files_indexes\(\)](#) makes indexing through the files easy.

See also

[dwarf_srclines_files_indexes](#)

Using [dwarf_srclines_b\(\)](#)

Parameters

<i>dw_context</i>	The line context of interest.
<i>dw_index_in</i>	The entry of interest. Callers should index as <i>dw_baseindex</i> through <i>dw_endindex</i> -1.
<i>dw_name</i>	If <i>dw_name</i> non-null on success returns The file name in the line table header through the pointer.
<i>dw_directory_index</i>	If <i>dw_directory_index</i> non-null on success returns the directory number in the line table header through the pointer.
<i>dw_last_mod_time</i>	If <i>dw_last_mod_time</i> non-null on success returns the directory last modification date/time through the pointer.
<i>dw_file_length</i>	If <i>dw_file_length</i> non-null on success returns the file length recorded in the line table through the pointer.
<i>dw_md5ptr</i>	If <i>dw_md5ptr</i> non-null on success returns a pointer to the 16byte MD5 hash of the file through the pointer. If there is no md5 value present it returns 0 through the pointer.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.19 dwarf_srclines_files_indexes()

```
DW_API int dwarf_srclines_files_indexes (
    Dwarf_Line_Context dw_context,
    Dwarf_Signed * dw_baseindex,
    Dwarf_Signed * dw_count,
    Dwarf_Signed * dw_endindex,
    Dwarf_Error * dw_error )
```

Return values easing indexing line table file numbers. Count is the real count of files array entries. Since DWARF 2,3,4 are zero origin indexes and DWARF5 and later are one origin, this function replaces [dwarf_srclines_files_count\(\)](#).

Parameters

<i>dw_context</i>	The line context of interest.
<i>dw_baseindex</i>	On success returns the base index of valid file indexes. With DWARF2,3,4 the value is 1. With DWARF5 the value is 0.
<i>dw_count</i>	On success returns the real count of entries.
<i>dw_endindex</i>	On success returns value such that callers should index as <i>dw_baseindex</i> through <i>dw_endindex</i> -1.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

See also

[Using dwarf_srclines_b\(\)](#)

9.11.2.20 dwarf_srclines_from_linecontext()

```
DW_API int dwarf_srclines_from_linecontext (
    Dwarf_Line_Context dw_linecontext,
    Dwarf_Line ** dw_linebuf,
    Dwarf_Signed * dw_linecount,
    Dwarf_Error * dw_error )
```

Access source lines from line context.

Provides access to Dwarf_Line data from a Dwarf_Line_Context on a standard line table.

Parameters

<i>dw_linecontext</i>	The line context of interest.
<i>dw_linebuf</i>	On success returns an array of pointers to Dwarf_Line.
<i>dw_linecount</i>	On success returns the count of entries in dw_linebuf. If dw_linecount is returned as zero this is a line table with no lines.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.21 dwarf_srclines_include_dir_count()

```
DW_API int dwarf_srclines_include_dir_count (
    Dwarf_Line_Context dw_line_context,
    Dwarf_Signed * dw_count,
    Dwarf_Error * dw_error )
```

Return the number of include directories in the Line Table.

Parameters

<i>dw_line_context</i>	The line context of interest.
<i>dw_count</i>	On success returns the count of directories. How to use this depends on the line table version number.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

See also

[dwarf_srclines_include_dir_data](#)

9.11.2.22 dwarf_srclines_include_dir_data()

```
DW_API int dwarf_srclines_include_dir_data (
    Dwarf_Line_Context dw_line_context,
    Dwarf_Signed dw_index,
    const char ** dw_name,
    Dwarf_Error * dw_error )
```

Return the include directories in the Line Table.

Parameters

<i>dw_line_context</i>	The line context of interest.
<i>dw_index</i>	Pass in an index to the line context list of include directories. If the line table is version 2,3, or 4, the valid indexes are 1 through dw_count. If the line table is version 5 the valid indexes are 0 through dw_count-1.
<i>dw_name</i>	On success it returns a pointer to a directory name. Do not free/deallocate the string.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

See also

[dwarf_srclines_include_dir_count](#)

9.11.2.23 dwarf_srclines_subprog_count()

```
DW_API int dwarf_srclines_subprog_count (
    Dwarf_Line_Context dw_context,
    Dwarf_Signed * dw_count,
    Dwarf_Error * dw_error )
```

Subprog count: Part of the two-level line table extension.

A non-standard table. The actual meaning of subprog count left undefined here.

Parameters

<i>dw_context</i>	The Dwarf_Line_Context of interest.
<i>dw_count</i>	On success returns the two-level line table subprogram array size in this line context.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.24 dwarf_srclines_subprog_data()

```
DW_API int dwarf_srclines_subprog_data (
    Dwarf_Line_Context dw_context,
    Dwarf_Signed dw_index,
    const char ** dw_name,
    Dwarf_Unsigned * dw_decl_file,
    Dwarf_Unsigned * dw_decl_line,
    Dwarf_Error * dw_error )
```

Retrieve data from the line table subprog array.

A non-standard table. Not defined here.

Parameters

<i>dw_context</i>	The Dwarf_Line_Context of interest.
<i>dw_index</i>	The item to retrieve. Valid indexes are 1 through dw_count.
<i>dw_name</i>	On success returns a pointer to the subprog name.
<i>dw_decl_file</i>	On success returns a file number through the pointer.
<i>dw_decl_line</i>	On success returns a line number through the pointer.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.25 dwarf_srclines_table_offset()

```
DW_API int dwarf_srclines_table_offset (
    Dwarf_Line_Context dw_context,
    Dwarf_Unsigned * dw_offset,
    Dwarf_Error * dw_error )
```

Return the srclines table offset.

The offset is in the relevant .debug_line or .debug_line.dwo section (and in a split dwarf package file includes the base line table offset).

Parameters

<i>dw_context</i>	
<i>dw_offset</i>	On success returns the section offset of the dw_context.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.11.2.26 dwarf_srclines_two_level_from_linecontext()

```
DW_API int dwarf_srclines_two_level_from_linecontext (
    Dwarf_Line_Context dw_context,
    Dwarf_Line ** dw_linebuf,
    Dwarf_Signed * dw_linecount,
    Dwarf_Line ** dw_linebuf_actuais,
    Dwarf_Signed * dw_linecount_actuais,
    Dwarf_Error * dw_error )
```

Returns line table counts and data.

Works for DWARF2,3,4,5 and for experimental two-level line tables. A single level table will have *linebuf_actuais and *linecount_actuais set to 0.

Two-level line tables are non-standard and not documented further. For standard (one-level) tables, it will return the single table through dw_linebuf, and the value returned through dw_linecount_actuais will be 0.

People not using these two-level tables should dwarf_srclines_from_linecontext instead.

9.11.2.27 dwarf_srclines_version()

```
DW_API int dwarf_srclines_version (
    Dwarf_Line_Context dw_line_context,
    Dwarf_Unsigned * dw_version,
    Dwarf_Small * dw_table_count,
    Dwarf_Error * dw_error )
```

The DWARF version number of this compile-unit.

The .debug_lines[.dwo] table count informs about the line table version and the type of line table involved.

Meaning of the value returned via dw_table_count:

- 0 The table is a header with no lines.
- 1 The table is a standard line table.
- 2 The table is an experimental line table.

Parameters

<i>dw_line_context</i>	The Line Context of interest.
<i>dw_version</i>	On success, returns the line table version through the pointer.
<i>dw_table_count</i>	On success, returns the tablecount through the pointer. If the table count is zero the line table is a header with no lines. If the table count is 1 this is a standard line table. If the table count is this is an experimental two-level line table.
<i>dw_error</i>	The usual error pointer.

Returns

DW_DLV_OK if it succeeds.

9.12 Ranges: code addresses in DWARF3-4

Functions

- DW_API int [dwarf_get_ranges_b](#) ([Dwarf_Debug](#) dw_dbg, [Dwarf_Off](#) dw_rangesoffset, [Dwarf_Die](#) dw_die, [Dwarf_Off](#) *dw_return_realoffset, [Dwarf_Ranges](#) **dw_rangesbuf, [Dwarf_Signed](#) *dw_rangecount, [Dwarf_Unsigned](#) *dw_bytecount, [Dwarf_Error](#) *dw_error)
Access to code ranges from a CU or just reading through the raw .debug_ranges section.
- DW_API void [dwarf_dealloc_ranges](#) ([Dwarf_Debug](#) dw_dbg, [Dwarf_Ranges](#) *dw_rangesbuf, [Dwarf_Signed](#) dw_rangecount)
Dealloc the array dw_rangesbuf.
- DW_API int [dwarf_get_ranges_baseaddress](#) ([Dwarf_Debug](#) dw_dbg, [Dwarf_Die](#) dw_die, [Dwarf_Bool](#) *dw_known_base, [Dwarf_Unsigned](#) *dw_baseaddress, [Dwarf_Bool](#) *dw_at_ranges_offset_present, [Dwarf_Unsigned](#) *dw_at_ranges_offset, [Dwarf_Error](#) *dw_error)
Find ranges base address.

9.12.1 Detailed Description

In DWARF3 and DWARF4 the DW_AT_ranges attribute provides an offset into the .debug_ranges section, which contains code address ranges.

See also

[Dwarf_Ranges](#)

DWARF3 and DWARF4. DW_AT_ranges with an unsigned constant FORM (DWARF3) or DW_FORM_sec_offset(↵ DWARF4).

9.12.2 Function Documentation

9.12.2.1 dwarf_dealloc_ranges()

```
DW_API void dwarf_dealloc_ranges (
    Dwarf_Debug dw_dbg,
    Dwarf_Ranges * dw_rangesbuf,
    Dwarf_Signed dw_rangecount )
```

Dealloc the array dw_rangesbuf.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_rangesbuf</i>	The dw_rangesbuf pointer returned by dwarf_get_ranges_b
<i>dw_rangecount</i>	The dw_rangecount returned by dwarf_get_ranges_b

9.12.2.2 dwarf_get_ranges_b()

```
DW_API int dwarf_get_ranges_b (
    Dwarf_Debug dw_dbg,
    Dwarf_Off dw_rangesoffset,
    Dwarf_Die dw_die,
    Dwarf_Off * dw_return_realoffset,
    Dwarf_Ranges ** dw_rangesbuf,
    Dwarf_Signed * dw_rangecount,
    Dwarf_Unsigned * dw_bytecount,
    Dwarf_Error * dw_error )
```

Access to code ranges from a CU or just reading through the raw .debug_ranges section.

Adds return of the dw_realoffset to accommodate DWARF4 GNU split-dwarf, where the ranges could be in the tieddbg (meaning the real executable, a.out, not in a dwp). DWARF4 split-dwarf is an extension, not standard DWARF4.

If printing all entries in the section pass in an initial dw_rangesoffset of zero and dw_die of NULL. Then increment dw_rangesoffset by dw_bytecount and call again to get the next batch of ranges. With a specific option dwarfdump can do this. This not a normal thing to do!

See also

[Example getting .debug_ranges data](#)

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest
<i>dw_rangesoffset</i>	The offset to read from in the section.
<i>dw_die</i>	Pass in the DIE whose DW_AT_ranges brought us to ranges.
<i>dw_return_realoffset</i>	The actual offset in the section actually read. In a tieddbg dwp DWARF4 extension object the base offset is added to dw_rangesoffset and returned here.
<i>dw_rangesbuf</i>	A pointer to an array of structs is returned here. The struct contents are the raw values in the section.
<i>dw_rangecount</i>	The count of structs in the array is returned here.
<i>dw_bytecount</i>	The number of bytes in the .debug_ranges section applying to the returned array. This makes possible just marching through the section by offset.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.12.2.3 dwarf_get_ranges_baseaddress()

```
DW_API int dwarf_get_ranges_baseaddress (
    Dwarf_Debug dw_dbg,
    Dwarf_Die dw_die,
    Dwarf_Bool * dw_known_base,
    Dwarf_Unsigned * dw_baseaddress,
```

```
Dwarf_Bool * dw_at_ranges_offset_present,
Dwarf_Unsigned * dw_at_ranges_offset,
Dwarf_Error * dw_error )
```

Find ranges base address.

The function allows callers to calculate actual address from .debug_ranges data in a simple and efficient way by returning the CU DIE ranges baseaddress.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_die</i>	Pass in any non-null valid Dwarf_Die to find the applicable .debug_ranges base address. The dw_die need not be a CU-DIE. A null dw_die is allowed.
<i>dw_known_base</i>	if dw_die is non-null and there is a known base address for the CU DIE that (a DW_at_low_pc in the CU DIE) dw_known_base will be set TRUE, Otherwise the value FALSE will be returned through dw_known_base.
<i>dw_baseaddress</i>	if dw_known_base is returned as TRUE then dw_baseaddress will be set with the correct pc value. Otherwise zero will be set through dw_baseaddress.
<i>dw_at_ranges_offset_present</i>	Set to 1 (TRUE) if dw_die has the attribute DW_AT_ranges.
<i>dw_at_ranges_offset</i>	Set to the value of DW_AT_ranges attribute of dw_die if dw_at_ranges_offset_present was set to TRUE. The offset is of the beginning of the .debug_ranges section range lists applying to this DIE.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK or DW_DLV_ERROR. Never returns DW_DLV_NO_ENTRY.

9.13 Rnglists: code addresses in DWARF5

Functions

- DW_API int [dwarf_rnglists_get_rle_head](#) (Dwarf_Attribute dw_attr, Dwarf_Half dw_theform, Dwarf_Unsigned dw_index_or_offset_value, Dwarf_Rnglists_Head *dw_head_out, Dwarf_Unsigned *dw_count_of_entries↵_in_head, Dwarf_Unsigned *dw_global_offset_of_rle_set, Dwarf_Error *dw_error)
Get Access to DWARF5 rnglists.
- DW_API int [dwarf_get_rnglists_entry_fields_a](#) (Dwarf_Rnglists_Head dw_head, Dwarf_Unsigned dw↵_entrynum, unsigned int *dw_entrylen, unsigned int *dw_rle_value_out, Dwarf_Unsigned *dw_raw1, Dwarf_Unsigned *dw_raw2, Dwarf_Bool *dw_debug_addr_unavailable, Dwarf_Unsigned *dw_cooked1, Dwarf_Unsigned *dw_cooked2, Dwarf_Error *dw_error)
Access rnglist entry details.
- DW_API void [dwarf_dealloc_rnglists_head](#) (Dwarf_Rnglists_Head dw_head)
Dealloc a Dwarf_Rnglists_Head.
- DW_API int [dwarf_load_rnglists](#) (Dwarf_Debug dw_dbg, Dwarf_Unsigned *dw_rnglists_count, Dwarf_Error *dw_error)
Loads all .debug_rnglists headers.
- DW_API int [dwarf_get_rnglist_offset_index_value](#) (Dwarf_Debug dw_dbg, Dwarf_Unsigned dw_context↵_index, Dwarf_Unsigned dw_offsetentry_index, Dwarf_Unsigned *dw_offset_value_out, Dwarf_Unsigned *dw_global_offset_value_out, Dwarf_Error *dw_error)
Retrieve the section offset of a rnglist.

- DW_API int `dwarf_get_rnglist_head_basics` (`Dwarf_Rnglists_Head` dw_head, `Dwarf_Unsigned` *dw↵_rle_count, `Dwarf_Unsigned` *dw_rnglists_version, `Dwarf_Unsigned` *dw_rnglists_index_returned, `Dwarf_Unsigned` *dw_bytes_total_in_rle, `Dwarf_Half` *dw_offset_size, `Dwarf_Half` *dw_address↵_size, `Dwarf_Half` *dw_segment_selector_size, `Dwarf_Unsigned` *dw_overall_offset_of_this_context, `Dwarf_Unsigned` *dw_total_length_of_this_context, `Dwarf_Unsigned` *dw_offset_table_offset, `Dwarf_Unsigned` *dw_offset_table_entrycount, `Dwarf_Bool` *dw_rnglists_base_present, `Dwarf_Unsigned` *dw_rnglists↵_base, `Dwarf_Bool` *dw_rnglists_base_address_present, `Dwarf_Unsigned` *dw_rnglists_base_address, `Dwarf_Bool` *dw_rnglists_debug_addr_base_present, `Dwarf_Unsigned` *dw_rnglists_debug_addr_base, `Dwarf_Error` *dw_error)

Access to internal data on rngelists.

- DW_API int `dwarf_get_rnglist_context_basics` (`Dwarf_Debug` dw_dbg, `Dwarf_Unsigned` dw_index, `Dwarf_Unsigned` *dw_header_offset, `Dwarf_Small` *dw_offset_size, `Dwarf_Small` *dw_extension_size, unsigned int *dw_version, `Dwarf_Small` *dw_address_size, `Dwarf_Small` *dw_segment_selector_size, `Dwarf_Unsigned` *dw_offset_entry_count, `Dwarf_Unsigned` *dw_offset_of_offset_array, `Dwarf_Unsigned` *dw_offset_of_first_rangeentry, `Dwarf_Unsigned` *dw_offset_past_last_rangeentry, `Dwarf_Error` *dw_error)

Access to rnglists header data.

- DW_API int `dwarf_get_rnglist_rle` (`Dwarf_Debug` dw_dbg, `Dwarf_Unsigned` dw_contextnumber, `Dwarf_Unsigned` dw_entry_offset, `Dwarf_Unsigned` dw_endoffset, unsigned int *dw_entrylen, unsigned int *dw_entry_kind, `Dwarf_Unsigned` *dw_entry_operand1, `Dwarf_Unsigned` *dw_entry_operand2, `Dwarf_Error` *dw_error)

Access to raw rnglists range data.

9.13.1 Detailed Description

Used in DWARF5 to define valid address ranges for code.

DW_FORM_rnglistx or DW_AT_ranges with DW_FORM_sec_offset

9.13.2 Function Documentation

9.13.2.1 dwarf_dealloc_rnglists_head()

```
DW_API void dwarf_dealloc_rnglists_head (
    Dwarf_Rnglists_Head dw_head )
```

Dealloc a Dwarf_Rnglists_Head.

Parameters

<code>dw_head</code>	dealloc all the memory associated with dw_head. The caller should then immediately set the pointer to zero/NULL as it is stale.
----------------------	---

9.13.2.2 dwarf_get_rnglist_context_basics()

```
DW_API int dwarf_get_rnglist_context_basics (
    Dwarf_Debug dw_dbg,
    Dwarf_Unsigned dw_index,
    Dwarf_Unsigned * dw_header_offset,
    Dwarf_Small * dw_offset_size,
    Dwarf_Small * dw_extension_size,
```

```

unsigned int * dw_version,
Dwarf_Small * dw_address_size,
Dwarf_Small * dw_segment_selector_size,
Dwarf_Unsigned * dw_offset_entry_count,
Dwarf_Unsigned * dw_offset_of_offset_array,
Dwarf_Unsigned * dw_offset_of_first_rangeentry,
Dwarf_Unsigned * dw_offset_past_last_rangeentry,
Dwarf_Error * dw_error )

```

Access to rnglists header data.

This returns, independent of any DIES or CUs information on the .debug_rnglists headers present in the section.

We do not document the details here. See the DWARF5 standard.

Enables printing of details about the Range List Table Headers, one header per call. Index starting at 0. Returns DW_DLV_NO_ENTRY if index is too high for the table. A .debug_rnglists section may contain any number of Range List Table Headers with their details.

9.13.2.3 dwarf_get_rnglist_head_basics()

```

DW_API int dwarf_get_rnglist_head_basics (
    Dwarf_Rnglists_Head dw_head,
    Dwarf_Unsigned * dw_rle_count,
    Dwarf_Unsigned * dw_rnglists_version,
    Dwarf_Unsigned * dw_rnglists_index_returned,
    Dwarf_Unsigned * dw_bytes_total_in_rle,
    Dwarf_Half * dw_offset_size,
    Dwarf_Half * dw_address_size,
    Dwarf_Half * dw_segment_selector_size,
    Dwarf_Unsigned * dw_overall_offset_of_this_context,
    Dwarf_Unsigned * dw_total_length_of_this_context,
    Dwarf_Unsigned * dw_offset_table_offset,
    Dwarf_Unsigned * dw_offset_table_entrycount,
    Dwarf_Bool * dw_rnglists_base_present,
    Dwarf_Unsigned * dw_rnglists_base,
    Dwarf_Bool * dw_rnglists_base_address_present,
    Dwarf_Unsigned * dw_rnglists_base_address,
    Dwarf_Bool * dw_rnglists_debug_addr_base_present,
    Dwarf_Unsigned * dw_rnglists_debug_addr_base,
    Dwarf_Error * dw_error )

```

Access to internal data on rngelists.

Returns detailed data from a Dwarf_Rnglists_Head Since this is primarily internal data we don't describe the details of the returned fields here.

9.13.2.4 dwarf_get_rnglist_offset_index_value()

```

DW_API int dwarf_get_rnglist_offset_index_value (
    Dwarf_Debug dw_dbg,
    Dwarf_Unsigned dw_context_index,
    Dwarf_Unsigned dw_offsetentry_index,
    Dwarf_Unsigned * dw_offset_value_out,
    Dwarf_Unsigned * dw_global_offset_value_out,
    Dwarf_Error * dw_error )

```

Retrieve the section offset of a rnglist.

Can be used to access raw rnglist data. Not used by most callers. See DWARF5 Section 7.28 Range List Table
Page 242

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_context_index</i>	Begin this at zero.
<i>dw_offsetentry_index</i>	Begin this at zero.
<i>dw_offset_value_out</i>	On success returns the rangelist entry offset within the rangelist set.
<i>dw_global_offset_value_out</i>	On success returns the rangelist entry offset within rnglist section.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc. If there are no rnglists at all, or if one of the above index values is too high to be valid it returns DW_DLV_NO_ENTRY.

9.13.2.5 dwarf_get_rnglist_rle()

```
DW_API int dwarf_get_rnglist_rle (
    Dwarf_Debug dw_dbg,
    Dwarf_Unsigned dw_contextnumber,
    Dwarf_Unsigned dw_entry_offset,
    Dwarf_Unsigned dw_endoffset,
    unsigned int * dw_entrylen,
    unsigned int * dw_entry_kind,
    Dwarf_Unsigned * dw_entry_operand1,
    Dwarf_Unsigned * dw_entry_operand2,
    Dwarf_Error * dw_error )
```

Access to raw rnglists range data.

Describes the actual raw data recorded in a particular range entry.

We do not describe all these fields for now, the raw values are mostly useful for people debugging compiler-generated DWARF.

9.13.2.6 dwarf_get_rnglists_entry_fields_a()

```
DW_API int dwarf_get_rnglists_entry_fields_a (
    Dwarf_Rnglists_Head dw_head,
    Dwarf_Unsigned dw_entrynum,
    unsigned int * dw_entrylen,
    unsigned int * dw_rle_value_out,
    Dwarf_Unsigned * dw_raw1,
    Dwarf_Unsigned * dw_raw2,
    Dwarf_Bool * dw_debug_addr_unavailable,
    Dwarf_Unsigned * dw_cooked1,
    Dwarf_Unsigned * dw_cooked2,
    Dwarf_Error * dw_error )
```

Access rnglist entry details.

See also

[Accessing rnglists section](#)

Parameters

<i>dw_head</i>	The Dwarf_Rnglists_Head of interest.
<i>dw_entrynum</i>	Valid values are 0 through dw_count_of_entries_in_head-1.
<i>dw_entrylen</i>	On success returns the length in bytes of this individual entry.
<i>dw_rle_value_out</i>	On success returns the RLE value of the entry, such as DW_RLE_startx_endx. This determines which of dw_raw1 and dw_raw2 contain meaningful data.
<i>dw_raw1</i>	On success returns a value directly recorded in the rangelist entry if that applies to this rle.
<i>dw_raw2</i>	On success returns a value directly recorded in the rangelist entry if that applies to this rle.
<i>dw_debug_addr_unavailable</i>	On success returns a flag. If the .debug_addr section is required but absent or unavailable the flag is set to TRUE. Otherwise sets the flag FALSE.
<i>dw_cooked1</i>	On success returns (if appropriate) the dw_raw1 value turned into a valid address.
<i>dw_cooked2</i>	On success returns (if appropriate) the dw_raw2 value turned into a valid address. Ignore the value if dw_debug_addr_unavailable is set.
<i>dw_error</i>	The usual error detail return pointer. Ignore the value if dw_debug_addr_unavailable is set.

Returns

Returns DW_DLV_OK etc.

9.13.2.7 dwarf_load_rnglists()

```
DW_API int dwarf_load_rnglists (
    Dwarf_Debug dw_dbg,
    Dwarf_Unsigned * dw_rnglists_count,
    Dwarf_Error * dw_error )
```

Loads all .debug_rnglists headers.

Loads all the rnglists headers and returns DW_DLV_NO_ENTRY if the section is missing or empty. Intended to be done quite early. It is automatically done if anything needing CU or DIE information is called, so it is not necessary for you to call this in any normal situation.

See also

[Accessing accessing raw rnglist](#)

Doing it more than once is never necessary or harmful. There is no deallocation call made visible, deallocation happens when [dwarf_finish\(\)](#) is called.

Parameters

<i>dw_dbg</i>	
<i>dw_rnglists_count</i>	On success it returns the number of rnglists headers in the section through dw_rnglists_count.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc. If the section does not exist the function returns DW_DLV_OK.

9.13.2.8 dwarf_rnglists_get_rle_head()

```
DW_API int dwarf_rnglists_get_rle_head (
    Dwarf_Attribute dw_attr,
    Dwarf_Half dw_theform,
    Dwarf_Unsigned dw_index_or_offset_value,
    Dwarf_Rnglists_Head * dw_head_out,
    Dwarf_Unsigned * dw_count_of_entries_in_head,
    Dwarf_Unsigned * dw_global_offset_of_rle_set,
    Dwarf_Error * dw_error )
```

Get Access to DWARF5 rnglists.

Opens a Dwarf_Rnglists_Head to access a set of DWARF5 rngelists .debug_rnglists DW_FORM_sec_offset DW↔_FORM_rnglistx (DW_AT_ranges in DWARF5).

See also

[Accessing rnglists section](#)

Parameters

<i>dw_attr</i>	The attribute referring to .debug_rnglists
<i>dw_theform</i>	The form number, DW_FORM_sec_offset or DW_FORM_rnglistx.
<i>dw_index_or_offset_value</i>	If the form is an index, pass it here. If the form is an offset, pass that here.
<i>dw_head_out</i>	On success creates a record owning the rnglists data for this attribute.
<i>dw_count_of_entries_in_head</i>	On success this is set to the number of entry in the rnglists for this attribute.
<i>dw_global_offset_of_rle_set</i>	On success set to the global offset of the rnglists in the rnglists section.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.14 Locations of data: DWARF2-DWARF5

Macros

- #define DW_LKIND_expression 0 /* DWARF2,3,4,5 */
- #define DW_LKIND_loclist 1 /* DWARF 2,3,4 */
- #define DW_LKIND_GNU_exp_list 2 /* GNU DWARF4 .dwo extension */
- #define DW_LKIND_loclists 5 /* DWARF5 loclists */
- #define DW_LKIND_unknown 99

Functions

- DW_API int [dwarf_get_loclist_c](#) (Dwarf_Attribute dw_attr, Dwarf_Loc_Head_c *dw_loclist_head, Dwarf_Unsigned *dw_locentry_count, Dwarf_Error *dw_error)

Location Lists and Expressions.

- DW_API int [dwarf_get_loclist_head_kind](#) (Dwarf_Loc_Head_c dw_loclist_head, unsigned int *dw_lkind, Dwarf_Error *dw_error)

Know what kind of location data it is.

- DW_API int [dwarf_get_locdesc_entry_d](#) (Dwarf_Loc_Head_c dw_loclist_head, Dwarf_Unsigned dw_↵ index, Dwarf_Small *dw_lle_value_out, Dwarf_Unsigned *dw_rawlowpc, Dwarf_Unsigned *dw_rawhipc, Dwarf_Bool *dw_debug_addr_unavailable, Dwarf_Addr *dw_lowpc_cooked, Dwarf_Addr *dw_hipc_cooked, Dwarf_Unsigned *dw_locexpr_op_count_out, Dwarf_Locdesc_c *dw_locentry_out, Dwarf_Small *dw_↵ loclist_source_out, Dwarf_Unsigned *dw_expression_offset_out, Dwarf_Unsigned *dw_locdesc_offset_out, Dwarf_Error *dw_error)

Retrieve the details(_d) of a location expression.

- DW_API int [dwarf_get_locdesc_entry_e](#) (Dwarf_Loc_Head_c dw_loclist_head, Dwarf_Unsigned dw_↵ index, Dwarf_Small *dw_lle_value_out, Dwarf_Unsigned *dw_rawlowpc, Dwarf_Unsigned *dw_rawhipc, Dwarf_Bool *dw_debug_addr_unavailable, Dwarf_Addr *dw_lowpc_cooked, Dwarf_Addr *dw_hipc_cooked, Dwarf_Unsigned *dw_locexpr_op_count_out, Dwarf_Unsigned *dw_lle_bytecount, Dwarf_Locdesc_c *dw_locentry_out, Dwarf_Small *dw_loclist_source_out, Dwarf_Unsigned *dw_expression_offset_out, Dwarf_Unsigned *dw_locdesc_offset_out, Dwarf_Error *dw_error)

Retrieve the details(_e) of a location expression.

- DW_API int [dwarf_get_location_op_value_c](#) (Dwarf_Locdesc_c dw_locdesc, Dwarf_Unsigned dw_↵ index, Dwarf_Small *dw_operator_out, Dwarf_Unsigned *dw_operand1, Dwarf_Unsigned *dw_operand2, Dwarf_Unsigned *dw_operand3, Dwarf_Unsigned *dw_offset_for_branch, Dwarf_Error *dw_error)

Get the raw values from a single location operation.

- DW_API int [dwarf_loclist_from_expr_c](#) (Dwarf_Debug dw_dbg, Dwarf_Ptr dw_expression_in, Dwarf_Unsigned dw_expression_length, Dwarf_Half dw_address_size, Dwarf_Half dw_offset_size, Dwarf_Half dw_dwarf_↵ version, Dwarf_Loc_Head_c *dw_loc_head, Dwarf_Unsigned *dw_listlen, Dwarf_Error *dw_error)

Generate a Dwarf_Loc_Head_c from an expression block.

- DW_API void [dwarf_dealloc_loc_head_c](#) (Dwarf_Loc_Head_c dw_head)

Dealloc (free) all memory allocated for Dwarf_Loc_Head_c.

- DW_API int [dwarf_load_loclists](#) (Dwarf_Debug dw_dbg, Dwarf_Unsigned *dw_loclists_count, Dwarf_Error *dw_error)

Load Loclists.

- DW_API int [dwarf_get_loclist_offset_index_value](#) (Dwarf_Debug dw_dbg, Dwarf_Unsigned dw_context_↵ _index, Dwarf_Unsigned dw_offsetentry_index, Dwarf_Unsigned *dw_offset_value_out, Dwarf_Unsigned *dw_global_offset_value_out, Dwarf_Error *dw_error)

Return certain loclists offsets.

- DW_API int [dwarf_get_loclist_head_basics](#) (Dwarf_Loc_Head_c dw_head, Dwarf_Small *dw_lkind, Dwarf_Unsigned *dw_lle_count, Dwarf_Unsigned *dw_loclists_version, Dwarf_Unsigned *dw_loclists_↵ _index_returned, Dwarf_Unsigned *dw_bytes_total_in_rle, Dwarf_Half *dw_offset_size, Dwarf_Half *dw_↵ _address_size, Dwarf_Half *dw_segment_selector_size, Dwarf_Unsigned *dw_overall_offset_of_this_↵ _context, Dwarf_Unsigned *dw_total_length_of_this_context, Dwarf_Unsigned *dw_offset_table_offset, Dwarf_Unsigned *dw_offset_table_entrycount, Dwarf_Bool *dw_loclists_base_present, Dwarf_Unsigned *dw_loclists_base, Dwarf_Bool *dw_loclists_base_address_present, Dwarf_Unsigned *dw_loclists_base_↵ _address, Dwarf_Bool *dw_loclists_debug_addr_base_present, Dwarf_Unsigned *dw_loclists_debug_↵ _addr_base, Dwarf_Unsigned *dw_offset_this_lle_area, Dwarf_Error *dw_error)

Return basic data about a loclists head.

- DW_API int [dwarf_get_loclist_context_basics](#) (Dwarf_Debug dw_dbg, Dwarf_Unsigned dw_index, Dwarf_Unsigned *dw_header_offset, Dwarf_Small *dw_offset_size, Dwarf_Small *dw_extension_size, unsigned int *dw_version, Dwarf_Small *dw_address_size, Dwarf_Small *dw_segment_selector_size, Dwarf_Unsigned *dw_offset_entry_count, Dwarf_Unsigned *dw_offset_of_offset_array, Dwarf_Unsigned *dw_offset_of_first_locentry, Dwarf_Unsigned *dw_offset_past_last_locentry, Dwarf_Error *dw_error)

Return basic data about a loclists context.

- DW_API int dwarf_get_loclist_lle (Dwarf_Debug dw_dbg, Dwarf_Unsigned dw_contextnumber, Dwarf_Unsigned dw_entry_offset, Dwarf_Unsigned dw_endoffset, unsigned int *dw_entrylen, unsigned int *dw_entry_kind, Dwarf_Unsigned *dw_entry_operand1, Dwarf_Unsigned *dw_entry_operand2, Dwarf_Unsigned *dw_expr_ops_blocksize, Dwarf_Unsigned *dw_expr_ops_offset, Dwarf_Small **dw_expr_opsdata, Dwarf_Error *dw_error)

Return basic data about a loclists context entry.

9.14.1 Detailed Description

9.14.2 Function Documentation

9.14.2.1 dwarf_dealloc_loc_head_c()

```
DW_API void dwarf_dealloc_loc_head_c (
    Dwarf_Loc_Head_c dw_head )
```

Dealloc (free) all memory allocated for Dwarf_Loc_Head_c.

Parameters

<i>dw_head</i>	A head pointer.
----------------	-----------------

The caller should zero the passed-in pointer on return as it is stale at that point.

9.14.2.2 dwarf_get_location_op_value_c()

```
DW_API int dwarf_get_location_op_value_c (
    Dwarf_Locdesc_c dw_locdesc,
    Dwarf_Unsigned dw_index,
    Dwarf_Small * dw_operator_out,
    Dwarf_Unsigned * dw_operand1,
    Dwarf_Unsigned * dw_operand2,
    Dwarf_Unsigned * dw_operand3,
    Dwarf_Unsigned * dw_offset_for_branch,
    Dwarf_Error * dw_error )
```

Get the raw values from a single location operation.

Parameters

<i>dw_locdesc</i>	Pass in a valid Dwarf_Locdesc_c.
<i>dw_index</i>	Pass in the operator index. zero through dw_locexpr_op_count_out-1.
<i>dw_operator_out</i>	On success returns the DW_OP operator, such as DW_OP_plus .
<i>dw_operand1</i>	On success returns the value of the operand or zero.
<i>dw_operand2</i>	On success returns the value of the operand or zero.
<i>dw_operand3</i>	On success returns the value of the operand or zero.
<i>dw_offset_for_branch</i>	On success returns The byte offset of the operator within the entire expression. Useful for checking the correctness of operators that branch..
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.14.2.3 dwarf_get_locdesc_entry_d()

```
DW_API int dwarf_get_locdesc_entry_d (
    Dwarf_Loc_Head_c dw_loclist_head,
    Dwarf_Unsigned dw_index,
    Dwarf_Small * dw_lle_value_out,
    Dwarf_Unsigned * dw_rawlowpc,
    Dwarf_Unsigned * dw_rawhipc,
    Dwarf_Bool * dw_debug_addr_unavailable,
    Dwarf_Addr * dw_lowpc_cooked,
    Dwarf_Addr * dw_hipc_cooked,
    Dwarf_Unsigned * dw_locexpr_op_count_out,
    Dwarf_Locdesc_c * dw_locentry_out,
    Dwarf_Small * dw_loclist_source_out,
    Dwarf_Unsigned * dw_expression_offset_out,
    Dwarf_Unsigned * dw_locdesc_offset_out,
    Dwarf_Error * dw_error )
```

Retrieve the details(_d) of a location expression.

Cooked value means the addresses from the location description after base values applied, so they are actual addresses. `debug_addr_unavailable` non-zero means the record from a Split Dwarf skeleton unit could not be accessed from the .dwo section or dwp object so the cooked values could not be calculated.

Parameters

<i>dw_loclist_head</i>	A loclist head pointer.
<i>dw_index</i>	Pass in an index value less than <code>dw_locentry_count</code> .
<i>dw_lle_value_out</i>	On success returns the DW_LLE value applicable, such as DW_LLE_start_end .
<i>dw_rawlowpc</i>	On success returns the first operand in the expression (if the expression has an operand).
<i>dw_rawhipc</i>	On success returns the second operand in the expression. (if the expression has a second operand).
<i>dw_debug_addr_unavailable</i>	On success returns FALSE if the data required to calculate <code>dw_lowpc_cooked</code> or <code>dw_hipc_cooked</code> was present or TRUE if some required data was missing (for example in split dwarf).
<i>dw_lowpc_cooked</i>	On success and if <code>dw_debug_addr_unavailable</code> FALSE returns the true low address.
<i>dw_hipc_cooked</i>	On success and if <code>dw_debug_addr_unavailable</code> FALSE returns the true high address.
<i>dw_locexpr_op_count_out</i>	On success returns the count of operations in the expression.
<i>dw_locentry_out</i>	On success returns a pointer to a specific location description.
<i>dw_loclist_source_out</i>	On success returns the applicable DW_LKIND value.
<i>dw_expression_offset_out</i>	On success returns the offset of the expression in the applicable section.
<i>dw_locdesc_offset_out</i>	On return sets the offset to the location description offset (if that is meaningful) or zero for simple location expressions.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.14.2.4 dwarf_get_locdesc_entry_e()

```
DW_API int dwarf_get_locdesc_entry_e (
    Dwarf_Loc_Head_c dw_loclist_head,
    Dwarf_Unsigned dw_index,
    Dwarf_Small * dw_lle_value_out,
    Dwarf_Unsigned * dw_rawlowpc,
    Dwarf_Unsigned * dw_rawhipc,
    Dwarf_Bool * dw_debug_addr_unavailable,
    Dwarf_Addr * dw_lowpc_cooked,
    Dwarf_Addr * dw_hipc_cooked,
    Dwarf_Unsigned * dw_locexpr_op_count_out,
    Dwarf_Unsigned * dw_lle_bytecount,
    Dwarf_Locdesc_c * dw_locentry_out,
    Dwarf_Small * dw_loclist_source_out,
    Dwarf_Unsigned * dw_expression_offset_out,
    Dwarf_Unsigned * dw_locdesc_offset_out,
    Dwarf_Error * dw_error )
```

Retrieve the details(_e) of a location expression.

Cooked value means the addresses from the location description after base values applied, so they are actual addresses. `debug_addr_unavailable` non-zero means the record from a Split Dwarf skeleton unit could not be accessed from the .dwo section or dwp object so the cooked values could not be calculated.

This is identical to `dwarf_get_locdesc_entry_d` except that it adds a pointer argument so the caller can know the size, in bytes, of the loclist DW_LLE operation itself.

It's used by `dwarfdump` but it is unlikely to be of interest to most callers..

9.14.2.5 dwarf_get_loclist_c()

```
DW_API int dwarf_get_loclist_c (
    Dwarf_Attribute dw_attr,
    Dwarf_Loc_Head_c * dw_loclist_head,
    Dwarf_Unsigned * dw_locentry_count,
    Dwarf_Error * dw_error )
```

Location Lists and Expressions.

This works on DWARF2 through DWARF5.

See also

[Location/expression access](#)

Parameters

<i>dw_attr</i>	The attribute must refer to a location expression or a location list, so must be DW_FORM_block, DW_FORM_exprloc, or a loclist reference form..
<i>dw_loclist_head</i>	On success returns a pointer to the created loclist head record.
<i>dw_locentry_count</i>	On success returns the count of records. For an expression it will be one.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.14.2.6 dwarf_get_loclist_context_basics()

```
DW_API int dwarf_get_loclist_context_basics (
    Dwarf_Debug dw_dbg,
    Dwarf_Unsigned dw_index,
    Dwarf_Unsigned * dw_header_offset,
    Dwarf_Small * dw_offset_size,
    Dwarf_Small * dw_extension_size,
    unsigned int * dw_version,
    Dwarf_Small * dw_address_size,
    Dwarf_Small * dw_segment_selector_size,
    Dwarf_Unsigned * dw_offset_entry_count,
    Dwarf_Unsigned * dw_offset_of_offset_array,
    Dwarf_Unsigned * dw_offset_of_first_locentry,
    Dwarf_Unsigned * dw_offset_past_last_locentry,
    Dwarf_Error * dw_error )
```

Return basic data about a loclists context.

Some of the same values as from dwarf_get_loclist_head_basics but here without any dependence on data derived from a CU context. Useful to print raw loclist data.

9.14.2.7 dwarf_get_loclist_head_basics()

```
DW_API int dwarf_get_loclist_head_basics (
    Dwarf_Loc_Head_c dw_head,
    Dwarf_Small * dw_lkind,
    Dwarf_Unsigned * dw_lle_count,
    Dwarf_Unsigned * dw_loclists_version,
    Dwarf_Unsigned * dw_loclists_index_returned,
    Dwarf_Unsigned * dw_bytes_total_in_rle,
    Dwarf_Half * dw_offset_size,
    Dwarf_Half * dw_address_size,
    Dwarf_Half * dw_segment_selector_size,
    Dwarf_Unsigned * dw_overall_offset_of_this_context,
    Dwarf_Unsigned * dw_total_length_of_this_context,
    Dwarf_Unsigned * dw_offset_table_offset,
    Dwarf_Unsigned * dw_offset_table_entrycount,
    Dwarf_Bool * dw_loclists_base_present,
    Dwarf_Unsigned * dw_loclists_base,
```

```

Dwarf_Bool * dw_loclists_base_address_present,
Dwarf_Unsigned * dw_loclists_base_address,
Dwarf_Bool * dw_loclists_debug_addr_base_present,
Dwarf_Unsigned * dw_loclists_debug_addr_base,
Dwarf_Unsigned * dw_offset_this_lle_area,
Dwarf_Error * dw_error )

```

Return basic data about a loclists head.

Used by dwarfdump to print basic data from the data generated to look at a specific loclist context as returned by dwarf_loclists_index_get_lle_head() or dwarf_loclists_offset_get_lle_head. Here we know there was a Dwarf↵_Attribute so additional things are known as compared to calling dwarf_get_loclist_context_basics See DWARF5 Section 7.20 Location List Table page 243.

9.14.2.8 dwarf_get_loclist_head_kind()

```

DW_API int dwarf_get_loclist_head_kind (
    Dwarf_Loc_Head_c dw_loclist_head,
    unsigned int * dw_lkind,
    Dwarf_Error * dw_error )

```

Know what kind of location data it is.

Parameters

<i>dw_loclist_head</i>	Pass in a loclist head pointer.
<i>dw_lkind</i>	On success returns the loclist kind through the pointer. For example DW_LKIND_expression.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.14.2.9 dwarf_get_loclist_lle()

```

DW_API int dwarf_get_loclist_lle (
    Dwarf_Debug dw_dbg,
    Dwarf_Unsigned dw_contextnumber,
    Dwarf_Unsigned dw_entry_offset,
    Dwarf_Unsigned dw_endoffset,
    unsigned int * dw_entrylen,
    unsigned int * dw_entry_kind,
    Dwarf_Unsigned * dw_entry_operand1,
    Dwarf_Unsigned * dw_entry_operand2,
    Dwarf_Unsigned * dw_expr_ops_blocksize,
    Dwarf_Unsigned * dw_expr_ops_offset,
    Dwarf_Small ** dw_expr_opsdata,
    Dwarf_Error * dw_error )

```

Return basic data about a loclists context entry.

Useful to print raw loclist data.

9.14.2.10 dwarf_get_loclist_offset_index_value()

```
DW_API int dwarf_get_loclist_offset_index_value (
    Dwarf_Debug dw_dbg,
    Dwarf_Unsigned dw_context_index,
    Dwarf_Unsigned dw_offsetentry_index,
    Dwarf_Unsigned * dw_offset_value_out,
    Dwarf_Unsigned * dw_global_offset_value_out,
    Dwarf_Error * dw_error )
```

Return certain loclists offsets.

Useful with the DWARF5 .debug_loclists section.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_context_index</i>	Pass in the loclists context index.
<i>dw_offsetentry_index</i>	Pass in the offset array index.
<i>dw_offset_value_out</i>	On success returns the offset value at offset table[dw_offsetentry_index], an offset local to this context.
<i>dw_global_offset_value_out</i>	On success returns the same offset value but with the offset of the table added in to form a section offset.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc. If one of the indexes passed in is out of range it returns DW_DLV_NO_ENTRY.

9.14.2.11 dwarf_load_loclists()

```
DW_API int dwarf_load_loclists (
    Dwarf_Debug dw_dbg,
    Dwarf_Unsigned * dw_loclists_count,
    Dwarf_Error * dw_error )
```

Load Loclists.

This loads raw .debug_loclists (DWARF5). It is unlikely you have a reason to use this function. If CUs or DIES have been referenced in any way loading is already done. A duplicate loading attempt returns DW_DLV_OK immediately, returning dw_loclists_count filled in and does nothing else.

Doing it more than once is never necessary or harmful. There is no deallocation call made visible, deallocation happens when [dwarf_finish\(\)](#) is called.

Parameters

<i>dw_dbg</i>	The applicable Dwarf_Debug.
<i>dw_loclists_count</i>	On success, returns the number of DWARF5 loclists contexts in the section, whether this is the first or a duplicate load.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK if it loaded successfully or if it is a duplicate load. If no .debug_loclists present returns DW_DLV_NO_ENTRY.

9.14.2.12 dwarf_loclist_from_expr_c()

```
DW_API int dwarf_loclist_from_expr_c (
    Dwarf_Debug dw_dbg,
    Dwarf_Ptr dw_expression_in,
    Dwarf_Unsigned dw_expression_length,
    Dwarf_Half dw_address_size,
    Dwarf_Half dw_offset_size,
    Dwarf_Half dw_dwarf_version,
    Dwarf_Loc_Head_c * dw_loc_head,
    Dwarf_Unsigned * dw_listlen,
    Dwarf_Error * dw_error )
```

Generate a Dwarf_Loc_Head_c from an expression block.

Useful if you have an expression block (from somewhere), do not have a Dwarf_Attribute available, and wish to deal with the expression.

See also

[Reading a location expression](#)

Parameters

<i>dw_dbg</i>	The applicable Dwarf_Debug
<i>dw_expression_in</i>	Pass in a pointer to the expression bytes.
<i>dw_expression_length</i>	Pass in the length, in bytes, of the expression.
<i>dw_address_size</i>	Pass in the applicable address_size.
<i>dw_offset_size</i>	Pass in the applicable offset size.
<i>dw_dwarf_version</i>	Pass in the applicable dwarf version.
<i>dw_loc_head</i>	On success returns a pointer to a dwarf location head record for use in getting to the details of the expression.
<i>dw_listlen</i>	On success, sets the listlen to one.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.15 .debug_addr access: DWARF5**Functions**

- DW_API int [dwarf_debug_addr_table](#) (Dwarf_Debug dw_dbg, Dwarf_Unsigned dw_section_offset, Dwarf_Debug_Addr_Table *dw_table_header, Dwarf_Unsigned *dw_length, Dwarf_Half *dw_version, Dwarf_Small *dw_address_size, Dwarf_Unsigned *dw_at_addr_base, Dwarf_Unsigned *dw_entry_count, Dwarf_Unsigned *dw_next_table_offset, Dwarf_Error *dw_error)

Return a .debug_addr table.

- DW_API int `dwarf_debug_addr_by_index` (`Dwarf_Debug_Addr_Table` dw_dat, `Dwarf_Unsigned` dw_entry_index, `Dwarf_Unsigned` *dw_address, `Dwarf_Error` *dw_error)

Return .debug_addr address given table index.

- DW_API void `dwarf_dealloc_debug_addr_table` (`Dwarf_Debug_Addr_Table` dw_dat)
dealloc (free) a Dwarf_Attr_Table record.

9.15.1 Detailed Description

Reading just the .debug_addr section.

These functions solely useful for reading that section. It seems unlikely you would have a reason to call these. The functions getting attribute values use the section when appropriate without using these functions.

9.15.2 Function Documentation

9.15.2.1 dwarf_dealloc_debug_addr_table()

```
DW_API void dwarf_dealloc_debug_addr_table (
    Dwarf_Debug_Addr_Table dw_dat )
```

dealloc (free) a Dwarf_Attr_Table record.

Parameters

<i>dw_dat</i>	Pass in a valid Dwarf_Debug_Addr_Table pointer. Does nothing if the dw_dat field is NULL.
---------------	---

9.15.2.2 dwarf_debug_addr_by_index()

```
DW_API int dwarf_debug_addr_by_index (
    Dwarf_Debug_Addr_Table dw_dat,
    Dwarf_Unsigned dw_entry_index,
    Dwarf_Unsigned * dw_address,
    Dwarf_Error * dw_error )
```

Return .debug_addr address given table index.

Parameters

<i>dw_dat</i>	Pass in a Dwarf_Debug_Addr_Table pointer.
<i>dw_entry_index</i>	Pass in a Dwarf_Debug_Addr_Table index to an address. If out of the valid range 0 through dw_entry_count-1 the function returns DW_DLV_NO_ENTRY.
<i>dw_address</i>	Returns an address in the program through the pointer.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc. If the `dw_section_offset` passed in is out of range it returns DW_DLV_NO_ENTRY. If it returns DW_DLV_ERROR only `dw_error` is set, `dw_address` is not set.

9.15.2.3 dwarf_debug_addr_table()

```
DW_API int dwarf_debug_addr_table (
    Dwarf_Debug dw_dbg,
    Dwarf_Unsigned dw_section_offset,
    Dwarf_Debug_Addr_Table * dw_table_header,
    Dwarf_Unsigned * dw_length,
    Dwarf_Half * dw_version,
    Dwarf_Small * dw_address_size,
    Dwarf_Unsigned * dw_at_addr_base,
    Dwarf_Unsigned * dw_entry_count,
    Dwarf_Unsigned * dw_next_table_offset,
    Dwarf_Error * dw_error )
```

Return a .debug_addr table.

Allocates and returns a pointer to a Dwarf_Debug_Addr_Table as well as the contents of the record.

Other than `dw_debug` and `dw_error` and `dw_table_header` a NULL passed in as a pointer argument means the return value will not be set through the pointer, so a caller can pass NULL for return values of no immediate interest.

It is only intended to enable printing of the simple DWARF5 .debug_addr section (by `dwarfdump`).

When emitting DWARF4, gcc may emit a GNU-specified .debug_addr format. If some CU has been opened then this call will work, but the single table will have all the entries for all CUs.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_section_offset</i>	Pass in the section offset of a table header. Start with zero. If the passed-in offset is past the last byte of the table the function returns DW_DLV_NO_ENTRY.
<i>dw_table_header</i>	On success Returns a pointer to a Dwarf_Debug_Addr_Table for use with <code>dwarf_get_attr_by_index()</code> .
<i>dw_length</i>	On success Returns the length in bytes of this contribution to .debug_addr from the table header, including the table length field and the array of addresses.
<i>dw_version</i>	On success returns the version number, which should be 5.
<i>dw_address_size</i>	On success returns the address size of the address entries in this table.
<i>dw_at_addr_base</i>	On success returns the value that will appear in some DW_AT_addr_base attribute.
<i>dw_entry_count</i>	On success returns the number of table entries in this table instance.
<i>dw_next_table_offset</i>	On success returns the offset of the next table in the section. Use the offset returned in the next call to this function.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc. If the `dw_section_offset` passed in is out of range it returns DW_DLV_NO_ENTRY. If it returns DW_DLV_ERROR only `dw_error` is set, none of the other return values are set through the pointers.

9.16 Macro Access: DWARF5

Functions

- DW_API int [dwarf_get_macro_context](#) (Dwarf_Die dw_die, Dwarf_Unsigned *dw_version_out, Dwarf_Macro_Context *dw_macro_context, Dwarf_Unsigned *dw_macro_unit_offset_out, Dwarf_Unsigned *dw_macro_ops_count_out, Dwarf_Unsigned *dw_macro_ops_data_length_out, Dwarf_Error *dw_error)
DWARF5 .debug_macro access via Dwarf_Die.
- DW_API int [dwarf_get_macro_context_by_offset](#) (Dwarf_Die dw_die, Dwarf_Unsigned dw_offset, Dwarf_Unsigned *dw_version_out, Dwarf_Macro_Context *dw_macro_context, Dwarf_Unsigned *dw_macro_ops_count_out, Dwarf_Unsigned *dw_macro_ops_data_length, Dwarf_Error *dw_error)
DWARF5 .debug_macro access via Dwarf_Die and an offset.
- DW_API int [dwarf_macro_context_total_length](#) (Dwarf_Macro_Context dw_context, Dwarf_Unsigned *dw_mac_total_len, Dwarf_Error *dw_error)
Return a macro context total length.
- DW_API void [dwarf_dealloc_macro_context](#) (Dwarf_Macro_Context dw_mc)
Dealloc a macro context.
- DW_API int [dwarf_macro_context_head](#) (Dwarf_Macro_Context dw_mc, Dwarf_Half *dw_version, Dwarf_Unsigned *dw_mac_offset, Dwarf_Unsigned *dw_mac_len, Dwarf_Unsigned *dw_mac_header_len, unsigned int *dw_flags, Dwarf_Bool *dw_has_line_offset, Dwarf_Unsigned *dw_line_offset, Dwarf_Bool *dw_has_offset_size_64, Dwarf_Bool *dw_has_operands_table, Dwarf_Half *dw_opcode_count, Dwarf_Error *dw_error)
Access the internal details of a Dwarf_Macro_Context.
- DW_API int [dwarf_macro_operands_table](#) (Dwarf_Macro_Context dw_mc, Dwarf_Half dw_index, Dwarf_Half *dw_opcode_number, Dwarf_Half *dw_operand_count, const Dwarf_Small **dw_operand_array, Dwarf_Error *dw_error)
Access to the details of the opcode operands table.
- DW_API int [dwarf_get_macro_op](#) (Dwarf_Macro_Context dw_macro_context, Dwarf_Unsigned dw_op_number, Dwarf_Unsigned *dw_op_start_section_offset, Dwarf_Half *dw_macro_operator, Dwarf_Half *dw_forms_count, const Dwarf_Small **dw_formcode_array, Dwarf_Error *dw_error)
Access macro operation details of a single operation.
- DW_API int [dwarf_get_macro_defundef](#) (Dwarf_Macro_Context dw_macro_context, Dwarf_Unsigned dw_op_number, Dwarf_Unsigned *dw_line_number, Dwarf_Unsigned *dw_index, Dwarf_Unsigned *dw_offset, Dwarf_Half *dw_forms_count, const char **dw_macro_string, Dwarf_Error *dw_error)
Get Macro defundef.
- DW_API int [dwarf_get_macro_startend_file](#) (Dwarf_Macro_Context dw_macro_context, Dwarf_Unsigned dw_op_number, Dwarf_Unsigned *dw_line_number, Dwarf_Unsigned *dw_name_index_to_line_tab, const char **dw_src_file_name, Dwarf_Error *dw_error)
Get Macro start end.
- DW_API int [dwarf_get_macro_import](#) (Dwarf_Macro_Context dw_macro_context, Dwarf_Unsigned dw_op_number, Dwarf_Unsigned *dw_target_offset, Dwarf_Error *dw_error)
Get Macro import.

9.16.1 Detailed Description

Reading the .debug_macro section.

See also

[Reading .debug_macro data \(DWARF5\)](#) An example reading .debug_macro

9.16.2 Function Documentation

9.16.2.1 dwarf_dealloc_macro_context()

```
DW_API void dwarf_dealloc_macro_context (
    Dwarf_Macro_Context dw_mc )
```

Dealloc a macro context.

Parameters

<i>dw_mc</i>	A pointer to the macro context of interest. On return the caller should zero the pointer as the pointer is then stale.
--------------	--

9.16.2.2 dwarf_get_macro_context()

```
DW_API int dwarf_get_macro_context (
    Dwarf_Die dw_die,
    Dwarf_Unsigned * dw_version_out,
    Dwarf_Macro_Context * dw_macro_context,
    Dwarf_Unsigned * dw_macro_unit_offset_out,
    Dwarf_Unsigned * dw_macro_ops_count_out,
    Dwarf_Unsigned * dw_macro_ops_data_length_out,
    Dwarf_Error * dw_error )
```

DWARF5 .debug_macro access via Dwarf_Die.

See also

[Reading .debug_macro data \(DWARF5\)](#)

Parameters

<i>dw_die</i>	The CU DIE of interest.
<i>dw_version_out</i>	On success returns the macro context version (5)
<i>dw_macro_context</i>	On success returns a pointer to a macro context which allows access to the context content.
<i>dw_macro_unit_offset_out</i>	On success returns the offset of the macro context.
<i>dw_macro_ops_count_out</i>	On success returns the number of macro operations in the context.
<i>dw_macro_ops_data_length_out</i>	On success returns the length in bytes of the operations in the context.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc. If no .debug_macro section exists for the CU it returns DW_DLV_NO_ENTRY.

9.16.2.3 dwarf_get_macro_context_by_offset()

```
DW_API int dwarf_get_macro_context_by_offset (
    Dwarf_Die dw_die,
```

```

Dwarf_Unsigned dw_offset,
Dwarf_Unsigned * dw_version_out,
Dwarf_Macro_Context * dw_macro_context,
Dwarf_Unsigned * dw_macro_ops_count_out,
Dwarf_Unsigned * dw_macro_ops_data_length,
Dwarf_Error * dw_error )

```

DWARF5 .debug_macro access via Dwarf_Die and an offset.

Parameters

<i>dw_die</i>	The CU DIE of interest.
<i>dw_offset</i>	The offset in the section to begin reading.
<i>dw_version_out</i>	On success returns the macro context version (5)
<i>dw_macro_context</i>	On success returns a pointer to a macro context which allows access to the context content.
<i>dw_macro_ops_count_out</i>	On success returns the number of macro operations in the context.
<i>dw_macro_ops_data_length</i>	On success returns the length in bytes of the macro context, starting at the offset of the first byte of the context.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc. If no .debug_macro section exists for the CU it returns DW_DLV_NO_ENTRY. If the dw_offset is outside the section it returns DW_DLV_ERROR.

9.16.2.4 dwarf_get_macro_defundef()

```

DW_API int dwarf_get_macro_defundef (
    Dwarf_Macro_Context dw_macro_context,
    Dwarf_Unsigned dw_op_number,
    Dwarf_Unsigned * dw_line_number,
    Dwarf_Unsigned * dw_index,
    Dwarf_Unsigned * dw_offset,
    Dwarf_Half * dw_forms_count,
    const char ** dw_macro_string,
    Dwarf_Error * dw_error )

```

Get Macro defundef.

To extract the value portion of a macro define:

See also

[dwarf_find_macro_value_start](#)

Parameters

<i>dw_macro_context</i>	The macro context of interest.
<i>dw_op_number</i>	valid values are 0 through dw_macro_ops_count_out-1. The op number must be for a def/undef.
<i>dw_line_number</i>	The line number in the user source for this define/undef

Parameters

<i>dw_index</i>	On success if the macro is an strx form the value returned is the string index in the record, otherwise zero is returned.
<i>dw_offset</i>	On success if the macro is an strp or sup form the value returned is the string offset in the appropriate section, otherwise zero is returned.
<i>dw_forms_count</i>	On success the value 2 is returned.
<i>dw_macro_string</i>	On success a pointer to a null-terminated string is returned. Do not dealloc or free this string.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc. It is an error if operator *dw_op_number* is not a DW_MACRO_define, DW_MACRO_undef, DW_MACRO_define_strp, DW_MACRO_undef_strp, DW_MACRO_undef_sup, DW_MACRO_undef_strx, or DW_MACRO_define_strx,

9.16.2.5 dwarf_get_macro_import()

```
DW_API int dwarf_get_macro_import (
    Dwarf_Macro_Context dw_macro_context,
    Dwarf_Unsigned dw_op_number,
    Dwarf_Unsigned * dw_target_offset,
    Dwarf_Error * dw_error )
```

Get Macro import.

Parameters

<i>dw_macro_context</i>	The macro context of interest.
<i>dw_op_number</i>	Valid values are 0 through <i>dw_macro_ops_count_out</i> -1.
<i>dw_target_offset</i>	Returns the offset in the imported section.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc. It is an error if the operator is not DW_MACRO_import or DW_MACRO_import_sup.

9.16.2.6 dwarf_get_macro_op()

```
DW_API int dwarf_get_macro_op (
    Dwarf_Macro_Context dw_macro_context,
    Dwarf_Unsigned dw_op_number,
    Dwarf_Unsigned * dw_op_start_section_offset,
    Dwarf_Half * dw_macro_operator,
    Dwarf_Half * dw_forms_count,
    const Dwarf_Small ** dw_formcode_array,
    Dwarf_Error * dw_error )
```

Access macro operation details of a single operation.

Useful for printing basic data about the operation.

Parameters

<i>dw_macro_context</i>	The macro context of interest.
<i>dw_op_number</i>	valid values are 0 through <i>dw_macro_ops_count_out</i> -1.
<i>dw_op_start_section_offset</i>	On success returns the section offset of this operator.
<i>dw_macro_operator</i>	On success returns the the macro operator itself, for example DW_MACRO_define.
<i>dw_forms_count</i>	On success returns the number of forms in the formcode array.
<i>dw_formcode_array</i>	On success returns a pointer to the formcode array of operand forms.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.16.2.7 dwarf_get_macro_startend_file()

```
DW_API int dwarf_get_macro_startend_file (
    Dwarf_Macro_Context dw_macro_context,
    Dwarf_Unsigned dw_op_number,
    Dwarf_Unsigned * dw_line_number,
    Dwarf_Unsigned * dw_name_index_to_line_tab,
    const char ** dw_src_file_name,
    Dwarf_Error * dw_error )
```

Get Macro start end.

Parameters

<i>dw_macro_context</i>	The macro context of interest.
<i>dw_op_number</i>	Valid values are 0 through <i>dw_macro_ops_count_out</i> -1. The op number must be for a start/end.
<i>dw_line_number</i>	If end_file nothing is returned here. If start_file on success returns the line number of the source line of the include directive.
<i>dw_name_index_to_line_tab</i>	If end_file nothing is returned here. If start_file on success returns the file name index in the line table file names table.
<i>dw_src_file_name</i>	If end_file nothing is returned here. If start_file on success returns a pointer to the null-terminated source file name. Do not free or dealloc this string.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc. It is an error if the operator is not DW_MACRO_start_file or DW_MACRO_end_file.

9.16.2.8 dwarf_macro_context_head()

```
DW_API int dwarf_macro_context_head (
    Dwarf_Macro_Context dw_mc,
    Dwarf_Half * dw_version,
    Dwarf_Unsigned * dw_mac_offset,
```

```

Dwarf_Unsigned * dw_mac_len,
Dwarf_Unsigned * dw_mac_header_len,
unsigned int * dw_flags,
Dwarf_Bool * dw_has_line_offset,
Dwarf_Unsigned * dw_line_offset,
Dwarf_Bool * dw_has_offset_size_64,
Dwarf_Bool * dw_has_operands_table,
Dwarf_Half * dw_opcode_count,
Dwarf_Error * dw_error )

```

Access the internal details of a Dwarf_Macro_Context.

Not described in detail here. See DWARF5 Standard Section 6.3.1 Macro Information Header page 166.

9.16.2.9 dwarf_macro_context_total_length()

```

DW_API int dwarf_macro_context_total_length (
    Dwarf_Macro_Context dw_context,
    Dwarf_Unsigned * dw_mac_total_len,
    Dwarf_Error * dw_error )

```

Return a macro context total length.

Parameters

<i>dw_context</i>	A pointer to the macro context of interest.
<i>dw_mac_total_len</i>	On success returns the length in bytes of the macro context.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.16.2.10 dwarf_macro_operands_table()

```

DW_API int dwarf_macro_operands_table (
    Dwarf_Macro_Context dw_mc,
    Dwarf_Half dw_index,
    Dwarf_Half * dw_opcode_number,
    Dwarf_Half * dw_operand_count,
    const Dwarf_Small ** dw_operand_array,
    Dwarf_Error * dw_error )

```

Access to the details of the opcode operands table.

Not of much interest to most libdwarf users.

Parameters

<i>dw_mc</i>	The macro context of interest.
<i>dw_index</i>	The opcode operands table index. 0 through dw_opcode_count-1.
<i>dw_opcode_number</i>	On success returns the opcode number in the table.
<i>dw_operand_count</i>	On success returns the number of forms for that dw_index.
<i>dw_operand_array</i>	On success returns the array of op operand forms
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.17 Macro Access: DWARF2-4

Functions

- DW_API char * [dwarf_find_macro_value_start](#) (char *dw_macro_string)
Return a pointer to the value part of a macro.
- DW_API int [dwarf_get_macro_details](#) ([Dwarf_Debug](#) dw_dbg, [Dwarf_Off](#) dw_macro_offset, [Dwarf_Unsigned](#) dw_maximum_count, [Dwarf_Signed](#) *dw_entry_count, [Dwarf_Macro_Details](#) **dw_details, [Dwarf_Error](#) *dw_error)
Getting .debug_macinfo macro details.

9.17.1 Detailed Description

Reading the .debug_macinfo section.

The section is rarely used since it takes a lot of disk space. DWARF5 has much more compact macro data (in section .debug_macro).

For an example see

See also

[Reading .debug_macinfo \(DWARF2-4\)](#) An example reading .debug_macinfo

9.17.2 Function Documentation

9.17.2.1 dwarf_find_macro_value_start()

```
DW_API char * dwarf_find_macro_value_start (
    char * dw_macro_string )
```

Return a pointer to the value part of a macro.

This function Works for all versions, DWARF2-DWARF5

Parameters

<i>dw_macro_string</i>	The macro string passed in should be properly formatted with a name, a space, and then the value portion (whether a function-like macro or not function-like).
------------------------	--

Returns

On success it returns a pointer to the value portion of the macro. On failure it returns a pointer to a NUL byte (so a zero-length string).

9.17.2.2 dwarf_get_macro_details()

```
DW_API int dwarf_get_macro_details (
    Dwarf_Debug dw_dbg,
    Dwarf_Off dw_macro_offset,
    Dwarf_Unsigned dw_maximum_count,
    Dwarf_Signed * dw_entry_count,
    Dwarf_Macro_Details ** dw_details,
    Dwarf_Error * dw_error )
```

Getting .debug_machinfo macro details.

[An example calling this function](#)

See also

[Reading .debug_machinfo \(DWARF2-4\)](#)

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_macro_offset</i>	The offset in the section you wish to start from.
<i>dw_maximum_count</i>	Pass in a count to ensure we will not allocate an excessive amount (guarding against a
<i>dw_entry_count</i>	On success returns a count of the macro operations in a CU macro set.
<i>dw_details</i>	On success returns a pointer to an array of struct DW_Macro_Details_s .
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.18 Stack Frame Access

Typedefs

- typedef int(* [dwarf_iterate_fde_callback_function_type](#)) ([Dwarf_Regtable3](#) *dw_reg_table, [Dwarf_Addr](#) dw↵_row_pc, [Dwarf_Bool](#) dw_has_more_rows, [Dwarf_Addr](#) dw_subsequent_pc, void *dw_user_data)

Functions

- DW_API int [dwarf_get_fde_list](#) ([Dwarf_Debug](#) dw_dbg, [Dwarf_Cie](#) **dw_cie_data, [Dwarf_Signed](#) *dw_cie↵_element_count, [Dwarf_Fde](#) **dw_fde_data, [Dwarf_Signed](#) *dw_fde_element_count, [Dwarf_Error](#) *dw↵_error)

Get lists of .debug_frame FDEs and CIEs.
- DW_API int [dwarf_get_fde_list_eh](#) ([Dwarf_Debug](#) dw_dbg, [Dwarf_Cie](#) **dw_cie_data, [Dwarf_Signed](#) *dw↵_cie_element_count, [Dwarf_Fde](#) **dw_fde_data, [Dwarf_Signed](#) *dw_fde_element_count, [Dwarf_Error](#) *dw_error)

Get lists of .eh_frame FDEs and CIEs.
- DW_API void [dwarf_dealloc_fde_cie_list](#) ([Dwarf_Debug](#) dw_dbg, [Dwarf_Cie](#) *dw_cie_data, [Dwarf_Signed](#) dw_cie_element_count, [Dwarf_Fde](#) *dw_fde_data, [Dwarf_Signed](#) dw_fde_element_count)

Release storage associated with FDE and CIE arrays.

- DW_API int `dwarf_get_fde_range` (Dwarf_Fde dw_fde, Dwarf_Addr *dw_low_pc, Dwarf_Unsigned *dw_func_length, Dwarf_Small **dw_fde_bytes, Dwarf_Unsigned *dw_fde_byte_length, Dwarf_Off *dw_cie_offset, Dwarf_Signed *dw_cie_index, Dwarf_Off *dw_fde_offset, Dwarf_Error *dw_error)

Return the FDE data for a single FDE.

- DW_API int `dwarf_get_fde_exception_info` (Dwarf_Fde dw_fde, Dwarf_Signed *dw_offset_into_exception_tables, Dwarf_Error *dw_error)

IRIX only access to C++ destructor tables.

- DW_API int `dwarf_get_cie_of_fde` (Dwarf_Fde dw_fde, Dwarf_Cie *dw_cie_returned, Dwarf_Error *dw_error)

Given FDE get CIE.

- DW_API int `dwarf_get_cie_info_b` (Dwarf_Cie dw_cie, Dwarf_Unsigned *dw_bytes_in_cie, Dwarf_Small *dw_version, char **dw_augmenter, Dwarf_Unsigned *dw_code_alignment_factor, Dwarf_Signed *dw_data_alignment_factor, Dwarf_Half *dw_return_address_register_rule, Dwarf_Small **dw_initial_instructions, Dwarf_Unsigned *dw_initial_instructions_length, Dwarf_Half *dw_offset_size, Dwarf_Error *dw_error)

Given a CIE get access to its content.

- DW_API int `dwarf_get_cie_index` (Dwarf_Cie dw_cie, Dwarf_Signed *dw_index, Dwarf_Error *dw_error)

Return CIE index given CIE.

- DW_API int `dwarf_get_fde_instr_bytes` (Dwarf_Fde dw_fde, Dwarf_Small **dw_outinstrs, Dwarf_Unsigned *dw_outlen, Dwarf_Error *dw_error)

Return length and pointer to access frame instructions.

- DW_API int `dwarf_iterate_fde_all_regs3` (Dwarf_Fde dw_fde, Dwarf_Regtable3 *dw_reg_table, dwarf_iterate_fde_callback_func dw_callback, void *dw_callback_user_data, Dwarf_Error *dw_error)

Iterate all rows for a given FDE. Invokes a provided callback function for each row. Iteration continues until all rows have been visited.

- DW_API int `dwarf_get_fde_info_for_all_regs3_b` (Dwarf_Fde dw_fde, Dwarf_Addr dw_pc_requested, Dwarf_Regtable3 *dw_reg_table, Dwarf_Addr *dw_row_pc, Dwarf_Bool *dw_has_more_rows, Dwarf_Addr *dw_subsequent_pc, Dwarf_Error *dw_error)

Return information on frame registers at a given pc value.

- DW_API int `dwarf_get_fde_info_for_all_regs3` (Dwarf_Fde dw_fde, Dwarf_Addr dw_pc_requested, Dwarf_Regtable3 *dw_reg_table, Dwarf_Addr *dw_row_pc, Dwarf_Error *dw_error)

Return information on frame registers at a given pc value.

- DW_API int `dwarf_get_fde_info_for_reg3_c` (Dwarf_Fde dw_fde, Dwarf_Half dw_table_column, Dwarf_Addr dw_pc_requested, Dwarf_Small *dw_value_type, Dwarf_Unsigned *dw_offset_relevant, Dwarf_Unsigned *dw_register, Dwarf_Signed *dw_offset, Dwarf_Block *dw_block_content, Dwarf_Addr *dw_row_pc_out, Dwarf_Bool *dw_has_more_rows, Dwarf_Addr *dw_subsequent_pc, Dwarf_Error *dw_error)

Return details about a particular pc and register.

- DW_API int `dwarf_get_fde_info_for_reg3_b` (Dwarf_Fde dw_fde, Dwarf_Half dw_table_column, Dwarf_Addr dw_pc_requested, Dwarf_Small *dw_value_type, Dwarf_Unsigned *dw_offset_relevant, Dwarf_Unsigned *dw_register, Dwarf_Unsigned *dw_offset, Dwarf_Block *dw_block_content, Dwarf_Addr *dw_row_pc_out, Dwarf_Bool *dw_has_more_rows, Dwarf_Addr *dw_subsequent_pc, Dwarf_Error *dw_error)

Return details about a particular pc and register.

- DW_API int `dwarf_get_fde_info_for_cfa_reg3_c` (Dwarf_Fde dw_fde, Dwarf_Addr dw_pc_requested, Dwarf_Small *dw_value_type, Dwarf_Unsigned *dw_offset_relevant, Dwarf_Unsigned *dw_register, Dwarf_Signed *dw_offset, Dwarf_Block *dw_block, Dwarf_Addr *dw_row_pc_out, Dwarf_Bool *dw_has_more_rows, Dwarf_Addr *dw_subsequent_pc, Dwarf_Error *dw_error)

Get the value of the CFA for a particular pc value.

- DW_API int `dwarf_get_fde_info_for_cfa_reg3_b` (Dwarf_Fde dw_fde, Dwarf_Addr dw_pc_requested, Dwarf_Small *dw_value_type, Dwarf_Unsigned *dw_offset_relevant, Dwarf_Unsigned *dw_register, Dwarf_Unsigned *dw_offset, Dwarf_Block *dw_block, Dwarf_Addr *dw_row_pc_out, Dwarf_Bool *dw_has_more_rows, Dwarf_Addr *dw_subsequent_pc, Dwarf_Error *dw_error)

Get the value of the CFA for a particular pc value.

- DW_API int `dwarf_get_fde_for_die` (Dwarf_Debug dw_dbg, Dwarf_Die dw_subr_die, Dwarf_Fde *dw_returned_fde, Dwarf_Error *dw_error)

Get the fde given DW_AT_MIPS_fde in a DIE.

- DW_API int [dwarf_get_fde_n](#) (Dwarf_Fde *dw_fde_data, Dwarf_Unsigned dw_fde_index, Dwarf_Fde *dw_↵_returned_fde, Dwarf_Error *dw_error)

Retrieve an FDE from an FDE table.

- DW_API int [dwarf_get_fde_at_pc](#) (Dwarf_Fde *dw_fde_data, Dwarf_Addr dw_pc_of_interest, Dwarf_Fde *dw_↵_returned_fde, Dwarf_Addr *dw_lopc, Dwarf_Addr *dw_hipc, Dwarf_Error *dw_error)

Retrieve an FDE given a pc.

- DW_API int [dwarf_get_cie_augmentation_data](#) (Dwarf_Cie dw_cie, Dwarf_Small **dw_augdata, Dwarf_Unsigned *dw_augdata_len, Dwarf_Error *dw_error)

Return .eh_frame CIE augmentation data.

- DW_API int [dwarf_get_fde_augmentation_data](#) (Dwarf_Fde dw_fde, Dwarf_Small **dw_augdata, Dwarf_Unsigned *dw_augdata_len, Dwarf_Error *dw_error)

Return .eh_frame FDE augmentation data.

- DW_API int [dwarf_expand_frame_instructions](#) (Dwarf_Cie dw_cie, Dwarf_Small *dw_instructionspointer, Dwarf_Unsigned dw_length_in_bytes, Dwarf_Frame_Instr_Head *dw_head, Dwarf_Unsigned *dw_instr_↵count, Dwarf_Error *dw_error)

Expands CIE or FDE instructions for detailed examination. Called for CIE initial instructions and FDE instructions. Call [dwarf_get_fde_instr_bytes\(\)](#) or [dwarf_get_cie_info_b\(\)](#) to get the initial instruction bytes and instructions byte count you wish to expand.

- DW_API int [dwarf_get_frame_instruction](#) (Dwarf_Frame_Instr_Head dw_head, Dwarf_Unsigned dw_↵instr_index, Dwarf_Unsigned *dw_instr_offset_in_instrs, Dwarf_Small *dw_cfa_operation, const char **dw_fields_description, Dwarf_Unsigned *dw_u0, Dwarf_Unsigned *dw_u1, Dwarf_Signed *dw_s0, Dwarf_Signed *dw_s1, Dwarf_Unsigned *dw_code_alignment_factor, Dwarf_Signed *dw_data_alignment_↵_factor, Dwarf_Block *dw_expression_block, Dwarf_Error *dw_error)

Return information about a single instruction Fields_description means a sequence of up to three letters including u,s,r,c,d,b, terminated by NUL byte. It is a string but we test individual bytes instead of using string compares. Do not free any of the returned values.

- DW_API int [dwarf_get_frame_instruction_a](#) (Dwarf_Frame_Instr_Head dw_, Dwarf_Unsigned dw_↵instr_index, Dwarf_Unsigned *dw_instr_offset_in_instrs, Dwarf_Small *dw_cfa_operation, const char **dw_fields_description, Dwarf_Unsigned *dw_u0, Dwarf_Unsigned *dw_u1, Dwarf_Unsigned *dw_u2, Dwarf_Signed *dw_s0, Dwarf_Signed *dw_s1, Dwarf_Unsigned *dw_code_alignment_factor, Dwarf_Signed *dw_data_alignment_factor, Dwarf_Block *dw_expression_block, Dwarf_Error *dw_error)

Expands CIE or FDE instructions for detailed examination. Called for CIE initial instructions and FDE instructions. This is the same as [dwarf_get_frame_instruction\(\)](#) except that it adds a dw_u2 field which contains an address-space identifier if the letter a appears in dw_fields_description. The dw_u2 field is non-standard and only applies to Heterogeneous Debugging frame instructions defined by LLVM (DW_CFA_LLVM_def_aspace_cfa and DW_CFA_↵LLVM_def_aspace_cfa_sf)

- DW_API void [dwarf_dealloc_frame_instr_head](#) (Dwarf_Frame_Instr_Head dw_head)

Deallocates the frame instruction data in dw_head.

- DW_API int [dwarf_fde_section_offset](#) (Dwarf_Debug dw_dbg, Dwarf_Fde dw_in_fde, Dwarf_Off *dw_fde_off, Dwarf_Off *dw_cie_off, Dwarf_Error *dw_error)

Return FDE and CIE offsets from debugging info.

- DW_API int [dwarf_cie_section_offset](#) (Dwarf_Debug dw_dbg, Dwarf_Cie dw_in_cie, Dwarf_Off *dw_cie_off, Dwarf_Error *dw_error)

Use to print CIE offsets from debugging info.

- DW_API Dwarf_Half [dwarf_set_frame_rule_table_size](#) (Dwarf_Debug dw_dbg, Dwarf_Half dw_value)

Frame Rule Table Size *Invariants for setting frame registers* .

- DW_API Dwarf_Half [dwarf_set_frame_rule_initial_value](#) (Dwarf_Debug dw_dbg, Dwarf_Half dw_value)

Frame Rule Initial Value.

- DW_API Dwarf_Half [dwarf_set_frame_cfa_value](#) (Dwarf_Debug dw_dbg, Dwarf_Half dw_value)

Frame CFA Column *Invariants for setting frame registers* .

- DW_API Dwarf_Half [dwarf_set_frame_same_value](#) (Dwarf_Debug dw_dbg, Dwarf_Half dw_value)

Frame Same Value Default *Invariants for setting frame registers* .

- DW_API Dwarf_Half [dwarf_set_frame_undefined_value](#) (Dwarf_Debug dw_dbg, Dwarf_Half dw_value)

Frame Undefined Value Default *Invariants for setting frame registers* .

9.18.1 Detailed Description

Use to access DWARF2-5 `.debug_frame` and GNU `.eh_frame` sections. Does not evaluate frame instructions, but provides detailed data so it is possible do that yourself.

9.18.2 Typedef Documentation

9.18.2.1 `dwarf_iterate_fde_callback_function_type`

`dwarf_iterate_fde_callback_function_type`

Used as a function pointer to a user-written callback function. This provides the register table for a row address. See [dwarf_iterate_fde_all_regs3\(\)](#).

Parameters

<i>dw_reg_table</i>	The register table for address data.
<i>dw_row_pc</i>	The address for the row the callback is reporting.
<i>dw_has_more_rows</i>	If non-zero means there are more rows in the current FDE.
<i>dw_subsequent_pc</i>	The pc address of the next row in the current FDE.
<i>dw_user_data</i>	Passes your callback a pointer to space you allocated

Returns

Return `DW_DLV_OK` if the data is valid. If a serious error of some kind return `DW_DLV_ERROR`.

9.18.3 Function Documentation

9.18.3.1 `dwarf_cie_section_offset()`

```
DW_API int dwarf_cie_section_offset (
    Dwarf_Debug dw_dbg,
    Dwarf_Cie dw_in_cie,
    Dwarf_Off * dw_cie_off,
    Dwarf_Error * dw_error )
```

Use to print CIE offsets from debugging info.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest
<i>dw_in_cie</i>	Pass in the CIE of interest.
<i>dw_cie_off</i>	On success returns the section offset of the CIE.
<i>dw_error</i>	Error return details

Returns

Returns `DW_DLV_OK` etc.

9.18.3.2 dwarf_dealloc_fde_cie_list()

```
DW_API void dwarf_dealloc_fde_cie_list (
    Dwarf_Debug dw_dbg,
    Dwarf_Cie * dw_cie_data,
    Dwarf_Signed dw_cie_element_count,
    Dwarf_Fde * dw_fde_data,
    Dwarf_Signed dw_fde_element_count )
```

Release storage associated with FDE and CIE arrays.

Applies to .eh_frame and .debug_frame lists.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug used in the list setup.
<i>dw_cie_data</i>	As returned from the list setup call.
<i>dw_cie_element_count</i>	
<i>dw_fde_data</i>	As returned from the list setup call.
<i>dw_fde_element_count</i>	As returned from the list setup call.

On return the pointers passed in *dw_cie_data* and *dw_fde_data* should be zeroed by the caller as they are then stale pointers.

9.18.3.3 dwarf_dealloc_frame_instr_head()

```
DW_API void dwarf_dealloc_frame_instr_head (
    Dwarf_Frame_Instr_Head dw_head )
```

Deallocates the frame instruction data in *dw_head*.

Parameters

<i>dw_head</i>	A head pointer. Frees all data created by dwarf_expand_frame_instructions() and makes the head pointer stale. The caller should set to NULL.
----------------	--

9.18.3.4 dwarf_expand_frame_instructions()

```
DW_API int dwarf_expand_frame_instructions (
    Dwarf_Cie dw_cie,
    Dwarf_Small * dw_instructionspointer,
    Dwarf_Unsigned dw_length_in_bytes,
    Dwarf_Frame_Instr_Head * dw_head,
    Dwarf_Unsigned * dw_instr_count,
    Dwarf_Error * dw_error )
```

Expands CIE or FDE instructions for detailed examination. Called for CIE initial instructions and FDE instructions. Call [dwarf_get_fde_instr_bytes\(\)](#) or [dwarf_get_cie_info_b\(\)](#) to get the initial instruction bytes and instructions byte count you wish to expand.

Combined with [dwarf_get_frame_instruction\(\)](#) or [dwarf_get_frame_instruction_a\(\)](#) (the second is like the first but adds an argument for LLVM address space numbers) it enables detailed access to frame instruction fields for evaluation or printing.

Free allocated memory with [dwarf_dealloc_frame_instr_head\(\)](#).

See also

[Using dwarf_expand_frame_instructions](#)

Parameters

<i>dw_cie</i>	The cie relevant to the instructions.
<i>dw_instructionspointer</i>	points to the instructions
<i>dw_length_in_bytes</i>	byte length of the instruction sequence.
<i>dw_head</i>	The address of an allocated <i>dw_head</i>
<i>dw_instr_count</i>	Returns the number of instructions in the byte stream
<i>dw_error</i>	Error return details

Returns

On success returns DW_DLV_OK

9.18.3.5 dwarf_fde_section_offset()

```
DW_API int dwarf_fde_section_offset (
    Dwarf_Debug dw_dbg,
    Dwarf_Fde dw_in_fde,
    Dwarf_Off * dw_fde_off,
    Dwarf_Off * dw_cie_off,
    Dwarf_Error * dw_error )
```

Return FDE and CIE offsets from debugging info.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest
<i>dw_in_fde</i>	Pass in the FDE of interest.
<i>dw_fde_off</i>	On success returns the section offset of the FDE.
<i>dw_cie_off</i>	On success returns the section offset of the CIE.
<i>dw_error</i>	Error return details

Returns

Returns DW_DLV_OK etc.

9.18.3.6 dwarf_get_cie_augmentation_data()

```
DW_API int dwarf_get_cie_augmentation_data (
    Dwarf_Cie dw_cie,
```

```
Dwarf_Small ** dw_augdata,
Dwarf_Unsigned * dw_augdata_len,
Dwarf_Error * dw_error )
```

Return `.eh_frame` CIE augmentation data.

GNU `.eh_frame` CIE augmentation information. See Linux Standard Base Core Specification version 3.0 .

See also

<https://gcc.gnu.org/legacy-ml/gcc/2003-12/msg01168.html>

Parameters

<i>dw_cie</i>	The CIE of interest.
<i>dw_augdata</i>	On success returns a pointer to the augmentation data.
<i>dw_augdata_len</i>	On success returns the length in bytes of the augmentation data.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns `DW_DLV_OK` etc. If the augmentation data length is zero it returns `DW_DLV_NO_ENTRY`.

9.18.3.7 dwarf_get_cie_index()

```
DW_API int dwarf_get_cie_index (
    Dwarf_Cie dw_cie,
    Dwarf_Signed * dw_index,
    Dwarf_Error * dw_error )
```

Return CIE index given CIE.

Parameters

<i>dw_cie</i>	Pass in the CIE of interest.
<i>dw_index</i>	On success, returns the index (the position of the CIE in the CIE pointer array).
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns `DW_DLV_OK` etc.

9.18.3.8 dwarf_get_cie_info_b()

```
DW_API int dwarf_get_cie_info_b (
    Dwarf_Cie dw_cie,
    Dwarf_Unsigned * dw_bytes_in_cie,
    Dwarf_Small * dw_version,
```

```

char ** dw_augmenter,
Dwarf_Unsigned * dw_code_alignment_factor,
Dwarf_Signed * dw_data_alignment_factor,
Dwarf_Half * dw_return_address_register_rule,
Dwarf_Small ** dw_initial_instructions,
Dwarf_Unsigned * dw_initial_instructions_length,
Dwarf_Half * dw_offset_size,
Dwarf_Error * dw_error )

```

Given a CIE get access to its content.

Parameters

<i>dw_cie</i>	Pass in the CIE of interest.
<i>dw_bytes_in_cie</i>	On success, returns the length of the CIE in bytes.
<i>dw_version</i>	On success, returns the CIE version number.
<i>dw_augmenter</i>	On success, returns a pointer to the augmentation string (which could be the empty string).
<i>dw_code_alignment_factor</i>	On success, returns a the code_alignment_factor used to interpret CIE/FDE operations.
<i>dw_data_alignment_factor</i>	On success, returns a the data_alignment_factor used to interpret CIE/FDE operations.
<i>dw_return_address_register_rule</i>	On success, returns a register number of the return address register.
<i>dw_initial_instructions</i>	On success, returns a pointer to the bytes of initial_instructions in the CIE.
<i>dw_initial_instructions_length</i>	On success, returns the length in bytes of the initial_instructions.
<i>dw_offset_size</i>	On success, returns the offset_size within this CIE.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.18.3.9 dwarf_get_cie_of_fde()

```

DW_API int dwarf_get_cie_of_fde (
    Dwarf_Fde dw_fde,
    Dwarf_Cie * dw_cie_returned,
    Dwarf_Error * dw_error )

```

Given FDE get CIE.

Parameters

<i>dw_fde</i>	The FDE of interest.
<i>dw_cie_returned</i>	On success returns a pointer to the applicable CIE.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.18.3.10 dwarf_get_fde_at_pc()

```
DW_API int dwarf_get_fde_at_pc (
    Dwarf_Fde * dw_fde_data,
    Dwarf_Addr dw_pc_of_interest,
    Dwarf_Fde * dw_returned_fde,
    Dwarf_Addr * dw_lopc,
    Dwarf_Addr * dw_hipc,
    Dwarf_Error * dw_error )
```

Retrieve an FDE given a pc.

Using binary search this finds the FDE that contains this dw_pc_of_interest That works because libdwarf ensures the array of FDEs is sorted by the low-pc

See also

[dwarf_get_fde_list](#)

Parameters

<i>dw_fde_data</i>	Pass in a pointer an array of fde pointers.
<i>dw_pc_of_interest</i>	The pc value of interest.
<i>dw_returned_fde</i>	On success a pointer to the applicable FDE is set through the pointer.
<i>dw_lopc</i>	On success a pointer to the low pc in dw_returned_fde is set through the pointer.
<i>dw_hipc</i>	On success a pointer to the high pc (one past the actual last byte address) in dw_returned_fde is set through the pointer.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK if the dw_pc_of_interest found in some FDE in the array. If no FDE is found containing dw_pc_of_interest DW_DLV_NO_ENTRY is returned.

9.18.3.11 dwarf_get_fde_augmentation_data()

```
DW_API int dwarf_get_fde_augmentation_data (
    Dwarf_Fde dw_fde,
    Dwarf_Small ** dw_augdata,
    Dwarf_Unsigned * dw_augdata_len,
    Dwarf_Error * dw_error )
```

Return .eh_frame FDE augmentation data.

GNU .eh_frame FDE augmentation information. See Linux Standard Base Core Specification version 3.0 .

See also

<https://gcc.gnu.org/legacy-ml/gcc/2003-12/msg01168.html>

Parameters

<i>dw_fde</i>	The FDE of interest.
<i>dw_augdata</i>	On success returns a pointer to the augmentation data.
<i>dw_augdata_len</i>	On success returns the length in bytes of the augmentation data.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc. If the augmentation data length is zero it returns DW_DLV_NO_ENTRY.

9.18.3.12 dwarf_get_fde_exception_info()

```
DW_API int dwarf_get_fde_exception_info (
    Dwarf_Fde dw_fde,
    Dwarf_Signed * dw_offset_into_exception_tables,
    Dwarf_Error * dw_error )
```

IRIX only access to C++ destructor tables.

This applies only to IRIX C++ destructor information which was never documented and is unlikely to be of interest.

9.18.3.13 dwarf_get_fde_for_die()

```
DW_API int dwarf_get_fde_for_die (
    Dwarf_Debug dw_dbg,
    Dwarf_Die dw_subr_die,
    Dwarf_Fde * dw_returned_fde,
    Dwarf_Error * dw_error )
```

Get the fde given DW_AT_MIPS_fde in a DIE.

This is essentially useless as only SGI/MIPS compilers from the 1990's had DW_AT_MIPS_fde in DW_TAG_↵ subprogram DIEs and this relies on that attribute to work.

9.18.3.14 dwarf_get_fde_info_for_all_regs3()

```
DW_API int dwarf_get_fde_info_for_all_regs3 (
    Dwarf_Fde dw_fde,
    Dwarf_Addr dw_pc_requested,
    Dwarf_Regtable3 * dw_reg_table,
    Dwarf_Addr * dw_row_pc,
    Dwarf_Error * dw_error )
```

Return information on frame registers at a given pc value.

Identical to [dwarf_get_fde_info_for_all_regs3_b\(\)](#) except that this doesn't output dw_has_more_rows and dw_↵ subsequent_pc, so [dwarf_get_fde_info_for_all_regs3_b\(\)](#) is a better choice.

If you need to iterate through all rows of the FDE, consider switching to [dwarf_get_fde_info_for_all_regs3_b\(\)](#) or [dwarf_iterate_fde_all_regs3\(\)](#).

9.18.3.15 dwarf_get_fde_info_for_all_regs3_b()

```
DW_API int dwarf_get_fde_info_for_all_regs3_b (
    Dwarf_Fde dw_fde,
    Dwarf_Addr dw_pc_requested,
    Dwarf_Regtable3 * dw_reg_table,
    Dwarf_Addr * dw_row_pc,
    Dwarf_Bool * dw_has_more_rows,
    Dwarf_Addr * dw_subsequent_pc,
    Dwarf_Error * dw_error )
```

Return information on frame registers at a given pc value.

An FDE at a given pc (code address) This function is new in October 2023 version 0.9.0. See [libdwarf.h](#) for the required condition of dw_reg_table pointer passed in.

Parameters

<i>dw_fde</i>	Pass in the FDE of interest.
<i>dw_pc_requested</i>	Pass in a pc (code) address inside that FDE.
<i>dw_reg_table</i>	Pass in the address of a Dwarf_Regtable3 struct which has been initialized with zero bits, and for which the dw_rt3_rules array has been allocated and the initialized with all zero bits. On success, returns a filled in dw_reg_table given the frame state.
<i>dw_row_pc</i>	On success returns the address of the row of frame data which may be a few counts off of the pc requested.
<i>dw_has_more_rows</i>	On success returns FALSE if there are no more rows, otherwise returns TRUE.
<i>dw_subsequent_pc</i>	On success this returns the address of the next pc for which there is a register row, making access to all the rows in sequence much more efficient than just adding 1 to a pc value.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK if the dw_pc_requested is in the FDE passed in and there is some applicable row in the table.

9.18.3.16 dwarf_get_fde_info_for_cfa_reg3_b()

```
DW_API int dwarf_get_fde_info_for_cfa_reg3_b (
    Dwarf_Fde dw_fde,
    Dwarf_Addr dw_pc_requested,
    Dwarf_Small * dw_value_type,
    Dwarf_Unsigned * dw_offset_relevant,
    Dwarf_Unsigned * dw_register,
    Dwarf_Unsigned * dw_offset,
    Dwarf_Block * dw_block,
    Dwarf_Addr * dw_row_pc_out,
    Dwarf_Bool * dw_has_more_rows,
    Dwarf_Addr * dw_subsequent_pc,
    Dwarf_Error * dw_error )
```

Get the value of the CFA for a particular pc value.

See also

[dwarf_get_fde_info_for_cfa_reg3_c](#)

This is the earlier version that returns a `dw_offset` of type `Dwarf_Unsigned`, requiring you to cast to `Dwarf_Signed` to work with the value.

9.18.3.17 dwarf_get_fde_info_for_cfa_reg3_c()

```
DW_API int dwarf_get_fde_info_for_cfa_reg3_c (
    Dwarf_Fde dw_fde,
    Dwarf_Addr dw_pc_requested,
    Dwarf_Small * dw_value_type,
    Dwarf_Unsigned * dw_offset_relevant,
    Dwarf_Unsigned * dw_register,
    Dwarf_Signed * dw_offset,
    Dwarf_Block * dw_block,
    Dwarf_Addr * dw_row_pc_out,
    Dwarf_Bool * dw_has_more_rows,
    Dwarf_Addr * dw_subsequent_pc,
    Dwarf_Error * dw_error )
```

Get the value of the CFA for a particular pc value.

See also

[dwarf_get_fde_info_for_reg3_c\(\)](#) has essentially the same return values as [dwarf_get_fde_info_for_reg3_c](#) but it refers to the CFA (which is not part of the register table) so this function has no table column argument.

New in September 2023, release 0.8.0. [dwarf_get_fde_info_for_cfa_reg3_c\(\)](#) returns `dw_offset` as a signed type. [dwarf_get_fde_info_for_cfa_reg3_b\(\)](#) returns `dw_offset` as an unsigned type, requiring the caller to cast to `Dwarf_Signed` before using the value. Both versions exist and operate properly.

If `dw_value_type == DW_EXPR_EXPRESSION` or `DW_EXPR_VALUE_EXPRESSION` `dw_offset` is not set and the caller must evaluate the expression, which usually depends on runtime frame data which cannot be calculated without a stack frame including register values (etc).

9.18.3.18 dwarf_get_fde_info_for_reg3_b()

```
DW_API int dwarf_get_fde_info_for_reg3_b (
    Dwarf_Fde dw_fde,
    Dwarf_Half dw_table_column,
    Dwarf_Addr dw_pc_requested,
    Dwarf_Small * dw_value_type,
    Dwarf_Unsigned * dw_offset_relevant,
    Dwarf_Unsigned * dw_register,
    Dwarf_Unsigned * dw_offset,
    Dwarf_Block * dw_block_content,
    Dwarf_Addr * dw_row_pc_out,
    Dwarf_Bool * dw_has_more_rows,
    Dwarf_Addr * dw_subsequent_pc,
    Dwarf_Error * dw_error )
```

Return details about a particular pc and register.

Identical to [dwarf_get_fde_info_for_reg3_c\(\)](#) except that this returns `dw_offset` as a `Dwarf_Unsigned`, which was never appropriate, and required you to cast that value to `Dwarf_Signed` to use it properly..

Please switch to using [dwarf_get_fde_info_for_reg3_c\(\)](#)

9.18.3.19 dwarf_get_fde_info_for_reg3_c()

```
DW_API int dwarf_get_fde_info_for_reg3_c (
    Dwarf_Fde dw_fde,
    Dwarf_Half dw_table_column,
    Dwarf_Addr dw_pc_requested,
    Dwarf_Small * dw_value_type,
    Dwarf_Unsigned * dw_offset_relevant,
    Dwarf_Unsigned * dw_register,
    Dwarf_Signed * dw_offset,
    Dwarf_Block * dw_block_content,
    Dwarf_Addr * dw_row_pc_out,
    Dwarf_Bool * dw_has_more_rows,
    Dwarf_Addr * dw_subsequent_pc,
    Dwarf_Error * dw_error )
```

Return details about a particular pc and register.

It is efficient to iterate across all table_columns (registers) using this function ([dwarf_get_fde_info_for_reg3_c\(\)](#)).

Or if one wants the data for all frame rows one could instead call [dwarf_iterate_fde_all_regs3\(\)](#) and index into the data it fills in and returns via a callback function you write.

If `dw_value_type == DW_EXPR_EXPRESSION` or `DW_EXPR_VALUE_EXPRESSION` `dw_offset` is not set and the caller must evaluate the expression, which usually depends on runtime frame data which cannot be calculated without a stack frame including registers (etc).

[dwarf_get_fde_info_for_reg3_c\(\)](#) is new in libdwarf 0.8.0. It corrects the incorrect type of the `dw_offset` argument in [dwarf_get_fde_info_for_reg3_b\(\)](#). Both versions operate correctly.

As of libdwarf 2.3.0 the CFA can be requested with `dw_table_column`. Previously the CFA was unavailable. By default the cfa pseudo register number is `DW_FRAME_CFA_COL` from [dwarf.h](#).

Parameters

<i>dw_fde</i>	Pass in the FDE of interest.
<i>dw_table_column</i>	Pass in the table_column, column numbers in the table are 0 through the number_of_registers-1 and the 'column' of the CFA (by default it is <code>DW_FRAME_CFA_COL</code> , but might have been set by your code using dwarf_set_frame_cfa_value()).
<i>dw_pc_requested</i>	Pass in the pc of interest within dw_fde.
<i>dw_value_type</i>	On success returns the value type, a DW_EXPR value. For example <code>DW_EXPR_EXPRESSION</code>
<i>dw_offset_relevant</i>	On success returns FALSE if the offset value is irrelevant, otherwise TRUE.
<i>dw_register</i>	On success returns a register number.
<i>dw_offset</i>	On success returns a signed register offset value when dw_value_type is <code>DW_EXPR_OFFSET</code> or <code>DW_EXPR_VAL_OFFSET</code> .
<i>dw_block_content</i>	On success returns a pointer to a block. For example, for <code>DW_EXPR_EXPRESSION</code> the block gives access to the expression bytes.
<i>dw_row_pc_out</i>	On success returns the address of the actual pc for this register at this pc.
<i>dw_has_more_rows</i>	On success returns FALSE if there are no more rows, otherwise returns TRUE.
<i>dw_subsequent_pc</i>	On success this returns the address of the next pc for which there is a register row, making access to all the rows in sequence much more efficient than just adding 1 to a pc value.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK if the dw_pc_requested is in the FDE passed in and there is a row for the pc in the table.

9.18.3.20 dwarf_get_fde_instr_bytes()

```
DW_API int dwarf_get_fde_instr_bytes (
    Dwarf_Fde dw_fde,
    Dwarf_Small ** dw_outinstrs,
    Dwarf_Unsigned * dw_outlen,
    Dwarf_Error * dw_error )
```

Return length and pointer to access frame instructions.

See also

[dwarf_expand_frame_instructions](#)

[Using dwarf_expand_frame_instructions](#)

Parameters

<i>dw_fde</i>	Pass in the FDE of interest.
<i>dw_outinstrs</i>	On success returns a pointer to the FDE instruction byte stream.
<i>dw_outlen</i>	On success returns the length of the dw_outinstrs byte stream.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.18.3.21 dwarf_get_fde_list()

```
DW_API int dwarf_get_fde_list (
    Dwarf_Debug dw_dbg,
    Dwarf_Cie ** dw_cie_data,
    Dwarf_Signed * dw_cie_element_count,
    Dwarf_Fde ** dw_fde_data,
    Dwarf_Signed * dw_fde_element_count,
    Dwarf_Error * dw_error )
```

Get lists of .debug_frame FDEs and CIEs.

See DWARF5 Section 6.4 Call Frame Information, page 171.

see [doc/checkexamples.c](#) exampleq()

See also

[Extracting fde, cie lists.](#)

The FDE array returned through dw_fde_data is sorted low-to-high by the lowest-pc in each FDE.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_cie_data</i>	On success returns a pointer to an array of pointers to CIE data.
<i>dw_cie_element_count</i>	On success returns a count of the number of elements in the <i>dw_cie_data</i> array.
<i>dw_fde_data</i>	On success returns a pointer to an array of pointers to FDE data.
<i>dw_fde_element_count</i>	On success returns a count of the number of elements in the <i>dw_fde_data</i> array. On success
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.18.3.22 dwarf_get_fde_list_eh()

```
DW_API int dwarf_get_fde_list_eh (
    Dwarf_Debug dw_dbg,
    Dwarf_Cie ** dw_cie_data,
    Dwarf_Signed * dw_cie_element_count,
    Dwarf_Fde ** dw_fde_data,
    Dwarf_Signed * dw_fde_element_count,
    Dwarf_Error * dw_error )
```

Get lists of .eh_frame FDEs and CIEs.

The arguments are identical to the previous function, the difference is the section read. The GNU-defined .eh_frame section is very similar to .debug_frame but has unique features that matter when following a stack trace.

See also

[dwarf_get_fde_list](#)

9.18.3.23 dwarf_get_fde_n()

```
DW_API int dwarf_get_fde_n (
    Dwarf_Fde * dw_fde_data,
    Dwarf_Unsigned dw_fde_index,
    Dwarf_Fde * dw_returned_fde,
    Dwarf_Error * dw_error )
```

Retrieve an FDE from an FDE table.

This is just like indexing into the FDE array but with extra checking of the pointer and index.

See also

[dwarf_get_fde_list](#)

9.18.3.24 dwarf_get_fde_range()

```
DW_API int dwarf_get_fde_range (
    Dwarf_Fde dw_fde,
    Dwarf_Addr * dw_low_pc,
    Dwarf_Unsigned * dw_func_length,
    Dwarf_Small ** dw_fde_bytes,
    Dwarf_Unsigned * dw_fde_byte_length,
    Dwarf_Off * dw_cie_offset,
    Dwarf_Signed * dw_cie_index,
    Dwarf_Off * dw_fde_offset,
    Dwarf_Error * dw_error )
```

Return the FDE data for a single FDE.

Parameters

<i>dw_fde</i>	The FDE of interest.
<i>dw_low_pc</i>	On success returns the low pc value for the function involved.
<i>dw_func_length</i>	On success returns the length of the function code in bytes.
<i>dw_fde_bytes</i>	On success returns a pointer to the bytes of the FDE.
<i>dw_fde_byte_length</i>	On success returns the length of the <i>dw_fde_bytes</i> area.
<i>dw_cie_offset</i>	On success returns the section offset of the associated CIE.
<i>dw_cie_index</i>	On success returns the CIE index of the associated CIE.
<i>dw_fde_offset</i>	On success returns the section offset of this FDE.
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns DW_DLV_OK etc.

9.18.3.25 dwarf_get_frame_instruction()

```
DW_API int dwarf_get_frame_instruction (
    Dwarf_Frame_Instr_Head dw_head,
    Dwarf_Unsigned dw_instr_index,
    Dwarf_Unsigned * dw_instr_offset_in_instrs,
    Dwarf_Small * dw_cfa_operation,
    const char ** dw_fields_description,
    Dwarf_Unsigned * dw_u0,
    Dwarf_Unsigned * dw_u1,
    Dwarf_Signed * dw_s0,
    Dwarf_Signed * dw_s1,
    Dwarf_Unsigned * dw_code_alignment_factor,
    Dwarf_Signed * dw_data_alignment_factor,
    Dwarf_Block * dw_expression_block,
    Dwarf_Error * dw_error )
```

Return information about a single instruction. Fields_description means a sequence of up to three letters including u,s,r,c,d,b, terminated by NUL byte. It is a string but we test individual bytes instead of using string compares. Do not free any of the returned values.

See also

[Using dwarf_expand_frame_instructions](#)

Parameters

<i>dw_head</i>	A head record
<i>dw_instr_index</i>	index $0 < i < \text{instr_count}$
<i>dw_instr_offset_in_instrs</i>	Returns the byte offset of this instruction within instructions.
<i>dw_cfa_operation</i>	Returns a DW_CFA opcode.
<i>dw_fields_description</i>	Returns a string. Do not free.
<i>dw_u0</i>	May be set to an unsigned value
<i>dw_u1</i>	May be set to an unsigned value
<i>dw_s0</i>	May be set to a signed value
<i>dw_s1</i>	May be set to a signed value
<i>dw_code_alignment_factor</i>	May be set by the call
<i>dw_data_alignment_factor</i>	May be set by the call
<i>dw_expression_block</i>	Pass in a pointer to a block
<i>dw_error</i>	If DW_DLV_ERROR and the argument is non-NULL, returns details about the error.

Returns

On success returns DW_DLV_OK If there is no such instruction with that index it returns DW_DLV_NO_ENTRY
On error it returns DW_DLV_ERROR and if dw_error is NULL it pushes back a pointer to a Dwarf_Error to the caller.

Frame expressions have a variety of formats and content. The dw_fields parameter is set to a pointer to a short string with some set of the letters s,u,r,d,c,b,a which enables determining exactly which values the call sets. Some examples: A s in fields[0] means s0 is a signed number.

A b somewhere in fields means the expression block passed in has been filled in.

A r in fields[1] means u1 is set to a register number.

A d in fields means data_alignment_factor is set

A c in fields means code_alignment_factor is set

An a in fields means an LLVM address space value and only exists if calling [dwarf_get_frame_instruction_a\(\)](#).

The possible frame instruction formats are:

```
" " "b" "r" "rb" "rr" "rsd" "rsda" "ru" "rua" "rud"
"sd" "u" "uc"
```

are the possible frame instruction formats.

9.18.3.26 dwarf_get_frame_instruction_a()

```
DW_API int dwarf_get_frame_instruction_a (
    Dwarf_Frame_Instr_Head dw_,
    Dwarf_Unsigned dw_instr_index,
    Dwarf_Unsigned * dw_instr_offset_in_instrs,
    Dwarf_Small * dw_cfa_operation,
    const char ** dw_fields_description,
    Dwarf_Unsigned * dw_u0,
    Dwarf_Unsigned * dw_u1,
    Dwarf_Unsigned * dw_u2,
```

```

Dwarf_Signed * dw_s0,
Dwarf_Signed * dw_s1,
Dwarf_Unsigned * dw_code_alignment_factor,
Dwarf_Signed * dw_data_alignment_factor,
Dwarf_Block * dw_expression_block,
Dwarf_Error * dw_error )

```

Expands CIE or FDE instructions for detailed examination. Called for CIE initial instructions and FDE instructions. This is the same as [dwarf_get_frame_instruction\(\)](#) except that it adds a `dw_u2` field which contains an address-space identifier if the letter `a` appears in `dw_fields_description`. The `dw_u2` field is non-standard and only applies to Heterogeneous Debugging frame instructions defined by LLVM (`DW_CFA_LLVM_def_aspace_cfa` and `DW_CFA_LLVM_def_aspace_cfa_sf`)

Where multiplication is called for (via `dw_code_alignment_factor` or `dw_data_alignment_factor`) to produce an offset there is no need to check for overflow as libdwarf has already verified there is no overflow.

The return values are the same except here we have: an `a` in `fields[2]` or `fields[3]` means `dw_u2` is an address-space identifier for the LLVM CFA instruction.

9.18.3.27 dwarf_iterate_fde_all_regs3()

```

DW_API int dwarf_iterate_fde_all_regs3 (
    Dwarf_Fde dw_fde,
    Dwarf_Regtable3 * dw_reg_table,
    dwarf_iterate_fde_callback_function_type dw_callback,
    void * dw_callback_user_data,
    Dwarf_Error * dw_error )

```

Iterate all rows for a given FDE. Invokes a provided callback function for each row. Iteration continues until all rows have been visited.

This is much more efficient than repeatedly calling [dwarf_get_fde_info_for_all_regs3_b\(\)](#) when you need to extract all rows of an FDE. See `dwarfexample/frame2.c` for an example of its use.

Parameters

<i>dw_fde</i>	Pass in the FDE of interest.
<i>dw_reg_table</i>	Pass in the address of a struct to be filled in and returned via the callback with fde row data for the current row. The struct should be all zeros. The array of struct Dwarf_Regtable_Entry3_s for register rules in the struct must have been allocated and initialized with all zero bits.
<i>dw_callback</i>	The callback that should be invoked for each row in the FDE. The register table of size <code>@dw_reg_table_size</code> is passed to the callback.
<i>dw_callback_user_data</i>	User data that is passed to the callback
<i>dw_error</i>	The usual error detail return pointer.

Returns

Returns `DW_DLV_OK` if iterations all succeeded

9.18.3.28 dwarf_set_frame_cfa_value()

```

DW_API Dwarf_Half dwarf_set_frame_cfa_value (

```

```
Dwarf_Debug dw_dbg,
Dwarf_Half dw_value )
```

Frame CFA Column [Invariants for setting frame registers](#) .

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest. If it is null or invalid, return the value zero immediately.
<i>dw_value</i>	Pass in the value to record for the library to use.

Returns

Returns the previous value.

9.18.3.29 dwarf_set_frame_rule_initial_value()

```
DW_API Dwarf_Half dwarf_set_frame_rule_initial_value (
    Dwarf_Debug dw_dbg,
    Dwarf_Half dw_value )
```

Frame Rule Initial Value.

[Invariants for setting frame registers](#)

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest. If it is null or invalid, return the value zero immediately.
<i>dw_value</i>	Pass in the value to record for the library to use.

Returns

Returns the previous value.

9.18.3.30 dwarf_set_frame_rule_table_size()

```
DW_API Dwarf_Half dwarf_set_frame_rule_table_size (
    Dwarf_Debug dw_dbg,
    Dwarf_Half dw_value )
```

Frame Rule Table Size [Invariants for setting frame registers](#) .

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest. If it is null or invalid, return the value zero immediately
<i>dw_value</i>	Pass in the value to record for the library to use. If zero, the Frame Rule Table Size is left unchanged. The library someone arbitrarily does not allow setting the number of register rules (registers) below 188 (DW_FRAME_HIGHEST_NORMAL_REGISTER in dwarf.h) and does not apply any dw_value lower than that.

Returns

Returns the previous value.

9.18.3.31 dwarf_set_frame_same_value()

```
DW_API Dwarf_Half dwarf_set_frame_same_value (
    Dwarf_Debug dw_dbg,
    Dwarf_Half dw_value )
```

Frame Same Value Default [Invariants for setting frame registers](#) .

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest. If it is null or invalid, return the value zero immediately.
<i>dw_value</i>	Pass in the value to record for the library to use.

Returns

Returns the previous value.

9.18.3.32 dwarf_set_frame_undefined_value()

```
DW_API Dwarf_Half dwarf_set_frame_undefined_value (
    Dwarf_Debug dw_dbg,
    Dwarf_Half dw_value )
```

Frame Undefined Value Default [Invariants for setting frame registers](#) .

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest. If it is null or invalid, return the value zero immediately.
<i>dw_value</i>	Pass in the value to record for the library to use.

Returns

Returns the previous value.

9.19 Abbreviations Section Details**Functions**

- DW_API int [dwarf_get_abbrev](#) ([Dwarf_Debug](#) dw_dbg, [Dwarf_Unsigned](#) dw_offset, [Dwarf_Abbrev](#) *dw_abbrev, [Dwarf_Unsigned](#) *dw_length, [Dwarf_Unsigned](#) *dw_attr_count, [Dwarf_Error](#) *dw_error)
- Reading Abbreviation Data.*
- DW_API int [dwarf_get_abbrev_tag](#) ([Dwarf_Abbrev](#) dw_abbrev, [Dwarf_Half](#) *dw_return_tag_number, [Dwarf_Error](#) *dw_error)

Get abbreviation tag.

- DW_API int `dwarf_get_abbrev_code` (Dwarf_Abbrev dw_abbrev, Dwarf_Unsigned *dw_return_code_number, Dwarf_Error *dw_error)

Get Abbreviation Code.

- DW_API int `dwarf_get_abbrev_children_flag` (Dwarf_Abbrev dw_abbrev, Dwarf_Signed *dw_return_flag, Dwarf_Error *dw_error)

Get Abbrev Children Flag.

- DW_API int `dwarf_get_abbrev_entry_b` (Dwarf_Abbrev dw_abbrev, Dwarf_Unsigned dw_idx, Dwarf_Bool dw_filter_outliers, Dwarf_Unsigned *dw_returned_attr_num, Dwarf_Unsigned *dw_returned_form, Dwarf_Signed *dw_returned_implicit_const, Dwarf_Off *dw_offset, Dwarf_Error *dw_error)

Get Abbrev Entry Details.

9.19.1 Detailed Description

Allows reading section `.debug_abbrev` independently of CUs or DIEs. Normally not done (libdwarf uses it as necessary to access DWARF DIEs and DWARF attributes) unless one is interested in the content of the section.

[About Reading Independently.](#)

9.19.2 Function Documentation

9.19.2.1 `dwarf_get_abbrev()`

```
DW_API int dwarf_get_abbrev (
    Dwarf_Debug dw_dbg,
    Dwarf_Unsigned dw_offset,
    Dwarf_Abbrev * dw_returned_abbrev,
    Dwarf_Unsigned * dw_length,
    Dwarf_Unsigned * dw_attr_count,
    Dwarf_Error * dw_error )
```

Reading Abbreviation Data.

Normally you never need to call these functions. Calls that involve DIEs do all this for you behind the scenes in the library.

This reads the data for a single abbrev code starting at `dw_offset`. Essentially, opening access to an abbreviation entry.

When libdwarf itself reads abbreviations to access DIEs the offset comes from the Compilation Unit Header `debug_abbrev_offset` field.

See also

[dwarf_next_cu_header_e](#)

Parameters

<code>dw_dbg</code>	The Dwarf_Debug of interest.
<code>dw_offset</code>	Pass in the offset where a Debug_Abbrev starts.
<code>dw_returned_abbrev</code>	On success, sets a pointer to a Dwarf_Abbrev through the pointer to allow further access.
<code>dw_length</code>	On success, returns the length of the entire abbreviation block (bytes), useful to calculate the next offset if reading the section independently of any compilation unit.
<code>dw_attr_count</code>	On success, returns the number of attributes in this abbreviation entry.
<code>dw_error</code>	On error <code>dw_error</code> is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc. If the abbreviation is a single zero byte it is a null abbreviation. DW_DLV_OK is returned.

Close the abbrev by calling dwarf_dealloc(dbg,*dw_returned_abbrev, DW_DLA_ABBREV)

9.19.2.2 dwarf_get_abbrev_children_flag()

```
DW_API int dwarf_get_abbrev_children_flag (
    Dwarf_Abbrev dw_abbrev,
    Dwarf_Signed * dw_return_flag,
    Dwarf_Error * dw_error )
```

Get Abbrev Children Flag.

Parameters

<i>dw_abbrev</i>	The Dwarf_Abbrev of interest.
<i>dw_return_flag</i>	On success returns the flag TRUE (greater than zero) if the DIE referencing the abbreviation has children, else returns FALSE (zero).
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc.

9.19.2.3 dwarf_get_abbrev_code()

```
DW_API int dwarf_get_abbrev_code (
    Dwarf_Abbrev dw_abbrev,
    Dwarf_Unsigned * dw_return_code_number,
    Dwarf_Error * dw_error )
```

Get Abbreviation Code.

Parameters

<i>dw_abbrev</i>	The Dwarf_Abbrev of interest.
<i>dw_return_code_number</i>	Returns the code for this abbreviation, a number assigned to the abbreviation and unique within the applicable CU.
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc.

9.19.2.4 dwarf_get_abbrev_entry_b()

```
DW_API int dwarf_get_abbrev_entry_b (
```

```

Dwarf_Abbrev dw_abbrev,
Dwarf_Unsigned dw_indx,
Dwarf_Bool dw_filter_outliers,
Dwarf_Unsigned * dw_returned_attr_num,
Dwarf_Unsigned * dw_returned_form,
Dwarf_Signed * dw_returned_implicit_const,
Dwarf_Off * dw_offset,
Dwarf_Error * dw_error )

```

Get Abbrev Entry Details.

Most will call with filter_outliers non-zero.

Parameters

<i>dw_abbrev</i>	The Dwarf_Abbrev of interest.
<i>dw_indx</i>	Valid dw_index values are 0 through dw_attr_count-1
<i>dw_filter_outliers</i>	Pass non-zero (TRUE) so the function will check for unreasonable abbreviation content and return DW_DLV_ERROR if such found. If zero (FALSE) passed in even a nonsensical attribute number and/or unknown DW_FORM are allowed (used by dwarfdump to report the issue(s)).
<i>dw_returned_attr_num</i>	On success returns the attribute number, such as DW_AT_name
<i>dw_returned_form</i>	On success returns the attribute FORM, such as DW_FORM_uda
<i>dw_returned_implicit_const</i>	On success, if the dw_returned_form is DW_FORM_implicit_const then dw_returned_implicit_const is the implicit const value, but if not implicit const the return value is zero..
<i>dw_offset</i>	On success returns the offset of the start of this attr/form pair in the abbreviation section.
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc. If the abbreviation code for this Dwarf_Abbrev is 0 it is a null abbreviation, the dw_indx is ignored, and the function returns DW_DLV_NO_ENTRY.

9.19.2.5 dwarf_get_abbrev_tag()

```

DW_API int dwarf_get_abbrev_tag (
    Dwarf_Abbrev dw_abbrev,
    Dwarf_Half * dw_return_tag_number,
    Dwarf_Error * dw_error )

```

Get abbreviation tag.

Parameters

<i>dw_abbrev</i>	The Dwarf_Abbrev of interest.
<i>dw_return_tag_number</i>	Returns the tag value, for example DW_TAG_compile_unit.
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc.

9.20 String Section .debug_str Details

Functions

- DW_API int [dwarf_get_str](#) ([Dwarf_Debug](#) dw_dbg, [Dwarf_Off](#) dw_offset, char **dw_string, [Dwarf_Signed](#) *dw_strlen_of_string, [Dwarf_Error](#) *dw_error)

Reading From a String Section.

9.20.1 Detailed Description

Shows just the section content in detail

9.20.2 Function Documentation

9.20.2.1 dwarf_get_str()

```
DW_API int dwarf_get_str (
    Dwarf_Debug dw_dbg,
    Dwarf_Off dw_offset,
    char ** dw_string,
    Dwarf_Signed * dw_strlen_of_string,
    Dwarf_Error * dw_error )
```

Reading From a String Section.

[Reading The String Section](#)

Parameters

<i>dw_dbg</i>	The Dwarf_Debug whose .debug_str section we want to access.
<i>dw_offset</i>	Pass in a string offset. Start at 0, and for the next call pass in dw_offset plus dw_strlen_of_string plus 1.
<i>dw_string</i>	The caller must pass in a valid pointer to a char *. On success returns a pointer to a string from offset dw_offset. Never dealloc or free this string.
<i>dw_strlen_of_string</i>	The caller must pass in a valid pointer to a Dwarf_Signed.

On success returns the strlen() of the string.

Parameters

<i>dw_error</i>	On error dw_error is set to point to the error details.
-----------------	---

Returns

The usual value: DW_DLV_OK etc. If there is no such section it returns DW_DLV_NO_ENTRY. If the dw_offset is greater than the section size, or dw_string passed in is NULL or dw_strlen_of_string is NULL the function returns DW_DLV_ERROR.

9.21 Str_Offsets section details

Functions

- DW_API int [dwarf_open_str_offsets_table_access](#) (Dwarf_Debug dw_dbg, Dwarf_Str_Offsets_Table *dw_table_data, Dwarf_Error *dw_error)
Creates access to a .debug_str_offsets table.
- DW_API int [dwarf_close_str_offsets_table_access](#) (Dwarf_Str_Offsets_Table dw_table_data, Dwarf_Error *dw_error)
Close str_offsets access, free table_data.
- DW_API int [dwarf_next_str_offsets_table](#) (Dwarf_Str_Offsets_Table dw_table_data, Dwarf_Unsigned *dw_unit_length, Dwarf_Unsigned *dw_unit_length_offset, Dwarf_Unsigned *dw_table_start_offset, Dwarf_Half *dw_entry_size, Dwarf_Half *dw_version, Dwarf_Half *dw_padding, Dwarf_Unsigned *dw_table_value_count, Dwarf_Error *dw_error)
Iterate through the offsets tables.
- DW_API int [dwarf_str_offsets_value_by_index](#) (Dwarf_Str_Offsets_Table dw_table_data, Dwarf_Unsigned dw_index_to_entry, Dwarf_Unsigned *dw_entry_value, Dwarf_Error *dw_error)
Access to an individual str offsets table entry.
- DW_API int [dwarf_str_offsets_statistics](#) (Dwarf_Str_Offsets_Table dw_table_data, Dwarf_Unsigned *dw_wasted_byte_count, Dwarf_Unsigned *dw_table_count, Dwarf_Error *dw_error)
Reports final wasted-bytes count.

9.21.1 Detailed Description

Shows just the section content in detail. Most library users will never call these, as references to this is handled by the code accessing some Dwarf_Attribute. [Reading The Str_Offsets](#)

9.21.2 Function Documentation

9.21.2.1 dwarf_close_str_offsets_table_access()

```
DW_API int dwarf_close_str_offsets_table_access (
    Dwarf_Str_Offsets_Table dw_table_data,
    Dwarf_Error * dw_error )
```

Close str_offsets access, free table_data.

See also

[Reading string offsets section data](#)

Parameters

<i>dw_table_data</i>	
<i>dw_error</i>	On error <i>dw_error</i> is set to point to the error details.

Returns

DW_DLV_OK etc. If there is no .debug_str_offsets section it returns DW_DLV_NO_ENTRY If it returns DW_DLV_ERROR there is nothing you can do except report the error and, optionally, call dwarf_dealloc_error to dealloc the error content (and then set the *dw_error* to NULL as after the dealloc the pointer is stale)..

9.21.2.2 dwarf_next_str_offsets_table()

```
DW_API int dwarf_next_str_offsets_table (
    Dwarf_Str_Offsets_Table dw_table_data,
    Dwarf_Unsigned * dw_unit_length,
    Dwarf_Unsigned * dw_unit_length_offset,
    Dwarf_Unsigned * dw_table_start_offset,
    Dwarf_Half * dw_entry_size,
    Dwarf_Half * dw_version,
    Dwarf_Half * dw_padding,
    Dwarf_Unsigned * dw_table_value_count,
    Dwarf_Error * dw_error )
```

Iterate through the offsets tables.

See also

[Reading string offsets section data](#)

Access to the tables starts at offset zero. The library progresses through the next table automatically, keeping track internally to know where it is.

Parameters

<i>dw_table_data</i>	Pass in an open Dwarf_Str_Offsets_Table.
<i>dw_unit_length</i>	On success returns a table unit_length field
<i>dw_unit_length_offset</i>	On success returns the section offset of the unit_length field.
<i>dw_table_start_offset</i>	On success returns the section offset of the array of table entries.
<i>dw_entry_size</i>	On success returns the entry size (4 or 8)
<i>dw_version</i>	On success returns the value in the version field 5.
<i>dw_padding</i>	On success returns the zero value in the padding field.
<i>dw_table_value_count</i>	On success returns the number of table entries, each of size <i>dw_entry_size</i> , in the table.
<i>dw_error</i>	On error <i>dw_error</i> is set to point to the error details.

Returns

DW_DLV_OK Returns DW_DLV_NO_ENTRY if there are no more entries.

9.21.2.3 dwarf_open_str_offsets_table_access()

```
DW_API int dwarf_open_str_offsets_table_access (
    Dwarf_Debug dw_dbg,
    Dwarf_Str_Offsets_Table * dw_table_data,
    Dwarf_Error * dw_error )
```

Creates access to a .debug_str_offsets table.

See also

[Reading string offsets section data](#)

Parameters

<i>dw_dbg</i>	Pass in the Dwarf_Debug of interest.
<i>dw_table_data</i>	On success returns a pointer to an opaque structure for use in further calls.
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

DW_DLV_OK etc. If there is no .debug_str_offsets section it returns DW_DLV_NO_ENTRY

9.21.2.4 dwarf_str_offsets_statistics()

```
DW_API int dwarf_str_offsets_statistics (
    Dwarf_Str_Offsets_Table dw_table_data,
    Dwarf_Unsigned * dw_wasted_byte_count,
    Dwarf_Unsigned * dw_table_count,
    Dwarf_Error * dw_error )
```

Reports final wasted-bytes count.

Reports the number of tables seen so far. Not very interesting.

Parameters

<i>dw_table_data</i>	Pass in the open table pointer.
<i>dw_wasted_byte_count</i>	Always returns 0 at present.
<i>dw_table_count</i>	On success returns the total number of tables seen so far in the section.
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

DW_DLV_OK etc.

9.21.2.5 dwarf_str_offsets_value_by_index()

```
DW_API int dwarf_str_offsets_value_by_index (
    Dwarf_Str_Offsets_Table dw_table_data,
```

```
Dwarf_Unsigned dw_index_to_entry,
Dwarf_Unsigned * dw_entry_value,
Dwarf_Error * dw_error )
```

Access to an individual str offsets table entry.

See also

[Reading string offsets section data](#)

Parameters

<i>dw_table_data</i>	Pass in the open table pointer.
<i>dw_index_to_entry</i>	Pass in the entry number, 0 through dw_table_value_count-1 for the active table
<i>dw_entry_value</i>	On success returns the value in that table entry, an offset into a string table.
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

DW_DLX_OK Returns DW_DLX_ERROR if dw_index_to_entry is out of the correct range.

9.22 Dwarf_Error Functions

Functions

- DW_API [Dwarf_Unsigned dwarf_errno](#) ([Dwarf_Error](#) dw_error)
What DW_DLE code does the error have?
- DW_API char * [dwarf_errmsg](#) ([Dwarf_Error](#) dw_error)
What message string is in the error?
- DW_API char * [dwarf_errmsg_by_number](#) ([Dwarf_Unsigned](#) dw_errnum)
What message string is associated with the error number.
- DW_API void [dwarf_error_creation](#) ([Dwarf_Debug](#) dw_dbg, [Dwarf_Error](#) *dw_error, char *dw_errmsg)
Creating an error. This is very rarely helpful. It lets the library user create a Dwarf_Error and associate any string with that error. Your code could then return DW_DLX_ERROR to your caller when your intent is to let your caller clean up whatever seems wrong.
- DW_API void [dwarf_dealloc_error](#) ([Dwarf_Debug](#) dw_dbg, [Dwarf_Error](#) dw_error)
Free (dealloc) an Dwarf_Error something created.

9.22.1 Detailed Description

These functions aid in understanding handling.

9.22.2 Function Documentation

9.22.2.1 dwarf_dealloc_error()

```
DW_API void dwarf_dealloc_error (
    Dwarf_Debug dw_dbg,
    Dwarf_Error dw_error )
```

Free (dealloc) an Dwarf_Error something created.

Parameters

<i>dw_dbg</i>	The relevant Dwarf_Debug pointer.
<i>dw_error</i>	A pointer to a Dwarf_Error. The pointer is then stale so you should immediately zero that pointer passed in.

9.22.2.2 dwarf_errmsg()

```
DW_API char * dwarf_errmsg (
    Dwarf_Error dw_error )
```

What message string is in the error?

Parameters

<i>dw_error</i>	The dw_error should be non-null and a valid Dwarf_Error.
-----------------	--

Returns

A string with a message related to the error.

9.22.2.3 dwarf_errmsg_by_number()

```
DW_API char * dwarf_errmsg_by_number (
    Dwarf_Unsigned dw_errornum )
```

What message string is associated with the error number.

Parameters

<i>dw_errornum</i>	The dw_error should be an integer from the DW_DLE set. For example, DW_DLE_DIE_NULL.
--------------------	--

Returns

The generic string describing that error number.

9.22.2.4 dwarf_errno()

```
DW_API Dwarf_Unsigned dwarf_errno (
    Dwarf_Error dw_error )
```

What DW_DLE code does the error have?

Parameters

<i>dw_error</i>	The dw_error should be non-null and a valid Dwarf_Error.
-----------------	--

Returns

A DW_DLE value of some kind. For example: DW_DLE_DIE_NULL.

9.22.2.5 dwarf_error_creation()

```
DW_API void dwarf_error_creation (
    Dwarf_Debug dw_dbg,
    Dwarf_Error * dw_error,
    char * dw_errmsg )
```

Creating an error. This is very rarely helpful. It lets the library user create a Dwarf_Error and associate any string with that error. Your code could then return DW_DLV_ERROR to your caller when your intent is to let your caller clean up whatever seems wrong.

Parameters

<i>dw_dbg</i>	The relevant Dwarf_Debug.
<i>dw_error</i>	a Dwarf_Error is returned through this pointer.
<i>dw_errmsg</i>	The message string you provide.

9.23 Generic dwarf_dealloc Function**Functions**

- DW_API void [dwarf_dealloc](#) ([Dwarf_Debug](#) dw_dbg, void *dw_space, [Dwarf_Unsigned](#) dw_type)
The generic dealloc (free) function. It requires you know the correct DW_DLA value to pass in, and in a few cases such as is not provided. The functions doing allocations tell you which dealloc to use.

9.23.1 Detailed Description

Works for most dealloc needed.

For easier to use versions see the following

See also

[dwarf_dealloc_attribute](#)
[dwarf_dealloc_die](#)
[dwarf_dealloc_dnames](#)
[dwarf_dealloc_error](#)
[dwarf_dealloc_fde_cie_list](#)
[dwarf_dealloc_frame_instr_head](#)
[dwarf_dealloc_macro_context](#)
[dwarf_dealloc_ranges](#)
[dwarf_dealloc_rnglists_head](#)
[dwarf_dealloc_uncompressed_block](#)
[dwarf_globals_dealloc](#)
[dwarf_gnu_index_dealloc](#)
[dwarf_loc_head_c_dealloc](#)
[dwarf_srclines_dealloc_b](#)

9.23.2 Function Documentation

9.23.2.1 dwarf_dealloc()

```
DW_API void dwarf_dealloc (
    Dwarf_Debug dw_dbg,
    void * dw_space,
    Dwarf_Unsigned dw_type )
```

The generic dealloc (free) function. It requires you know the correct DW_DLA value to pass in, and in a few cases such is not provided. The functions doing allocations tell you which dealloc to use.

Parameters

<i>dw_dbg</i>	Must be a valid open Dwarf_Debug. and must be the dw_dbg that the error was created on. If it is not the dealloc will do nothing.
<i>dw_space</i>	Must be an address returned directly by a libdwarf call that the call specifies as requiring dealloc/free. If it is not a segfault or address fault is possible.
<i>dw_type</i>	Must be a correct naming of the DW_DLA type. If it is not the dealloc will do nothing.

9.24 Access to Section .debug_sup

Functions

- DW_API int dwarf_get_debug_sup (Dwarf_Debug dw_dbg, Dwarf_Half *dw_version, Dwarf_Small *dw_is_supplementary, char **dw_filename, Dwarf_Unsigned *dw_checksum_len, Dwarf_Small **dw_checksum, Dwarf_Error *dw_error)

Return basic .debug_sup section header data.

9.24.1 Detailed Description

9.24.2 Function Documentation

9.24.2.1 dwarf_get_debug_sup()

```
DW_API int dwarf_get_debug_sup (
    Dwarf_Debug dw_dbg,
    Dwarf_Half * dw_version,
    Dwarf_Small * dw_is_supplementary,
    char ** dw_filename,
    Dwarf_Unsigned * dw_checksum_len,
    Dwarf_Small ** dw_checksum,
    Dwarf_Error * dw_error )
```

Return basic .debug_sup section header data.

This returns basic data from the header of a .debug_sup section. See DWARF5 Section 7.3.6, "DWARF Supplementary Object Files"

Other sections present should be normal DWARF5, so normal libdwarf calls should work. We have no existing examples on hand, so it is hard to know what really works.

If there is no such section it returns DW_DLV_NO_ENTRY.

9.25 Fast Access to .debug_names DWARF5

Functions

- DW_API int `dwarf_dnames_header` (`Dwarf_Debug` dw_dbg, `Dwarf_Off` dw_starting_offset, `Dwarf_Dnames_Head` *dw_dn, `Dwarf_Off` *dw_offset_of_next_table, `Dwarf_Error` *dw_error)
Open access to a .debug_names table.
- DW_API void `dwarf_dealloc_dnames` (`Dwarf_Dnames_Head` dw_dn)
Frees all the malloc data associated with dw_dn.
- DW_API int `dwarf_dnames_abbrevtable` (`Dwarf_Dnames_Head` dw_dn, `Dwarf_Unsigned` dw_index, `Dwarf_Unsigned` *dw_abbrev_offset, `Dwarf_Unsigned` *dw_abbrev_code, `Dwarf_Unsigned` *dw_abbrev_tag, `Dwarf_Unsigned` dw_array_size, `Dwarf_Half` *dw_idxattr_array, `Dwarf_Half` *dw_form_array, `Dwarf_Unsigned` *dw_idxattr_count)
Access to the abbrevs table content.
- DW_API int `dwarf_dnames_sizes` (`Dwarf_Dnames_Head` dw_dn, `Dwarf_Unsigned` *dw_comp_unit_count, `Dwarf_Unsigned` *dw_local_type_unit_count, `Dwarf_Unsigned` *dw_foreign_type_unit_count, `Dwarf_Unsigned` *dw_bucket_count, `Dwarf_Unsigned` *dw_name_count, `Dwarf_Unsigned` *dw_abbrev_table_size, `Dwarf_Unsigned` *dw_entry_pool_size, `Dwarf_Unsigned` *dw_augmentation_string_size, char **dw_augmentation_string, `Dwarf_Unsigned` *dw_section_size, `Dwarf_Half` *dw_table_version, `Dwarf_Half` *dw_offset_size, `Dwarf_Error` *dw_error)
Sizes and counts from the debug names table.
- DW_API int `dwarf_dnames_offsets` (`Dwarf_Dnames_Head` dw_dn, `Dwarf_Unsigned` *dw_header_offset, `Dwarf_Unsigned` *dw_cu_table_offset, `Dwarf_Unsigned` *dw_tu_local_offset, `Dwarf_Unsigned` *dw_foreign_tu_offset, `Dwarf_Unsigned` *dw_bucket_offset, `Dwarf_Unsigned` *dw_hashes_offset, `Dwarf_Unsigned` *dw_stringoffsets_offset, `Dwarf_Unsigned` *dw_entryoffsets_offset, `Dwarf_Unsigned` *dw_abbrev_table_offset, `Dwarf_Unsigned` *dw_entry_pool_offset, `Dwarf_Error` *dw_error)
Offsets from the debug names table.
- DW_API int `dwarf_dnames_cu_table` (`Dwarf_Dnames_Head` dw_dn, const char *dw_type, `Dwarf_Unsigned` dw_index_number, `Dwarf_Unsigned` *dw_offset, `Dwarf_Sig8` *dw_sig, `Dwarf_Error` *dw_error)
Each debug names cu list entry one at a time.
- DW_API int `dwarf_dnames_bucket` (`Dwarf_Dnames_Head` dw_dn, `Dwarf_Unsigned` dw_bucket_number, `Dwarf_Unsigned` *dw_index, `Dwarf_Unsigned` *dw_indexcount, `Dwarf_Error` *dw_error)
Access to bucket contents.
- DW_API int `dwarf_dnames_name` (`Dwarf_Dnames_Head` dw_dn, `Dwarf_Unsigned` dw_name_index, `Dwarf_Unsigned` *dw_bucket_number, `Dwarf_Unsigned` *dw_hash_value, `Dwarf_Unsigned` *dw_offset_to_debug_str, char **dw_ptrtostr, `Dwarf_Unsigned` *dw_offset_in_entrypool, `Dwarf_Unsigned` *dw_abbrev_number, `Dwarf_Half` *dw_abbrev_tag, `Dwarf_Unsigned` dw_array_size, `Dwarf_Half` *dw_idxattr_array, `Dwarf_Half` *dw_form_array, `Dwarf_Unsigned` *dw_idxattr_count, `Dwarf_Error` *dw_error)
Retrieve a name table entry.
- DW_API int `dwarf_dnames_entrypool` (`Dwarf_Dnames_Head` dw_dn, `Dwarf_Unsigned` dw_offset_in_entrypool, `Dwarf_Unsigned` *dw_abbrev_code, `Dwarf_Half` *dw_tag, `Dwarf_Unsigned` *dw_value_count, `Dwarf_Unsigned` *dw_index_of_abbrev, `Dwarf_Unsigned` *dw_offset_of_initial_value, `Dwarf_Error` *dw_error)
Return a the set of values from an entrypool entry.
- DW_API int `dwarf_dnames_entrypool_values` (`Dwarf_Dnames_Head` dw_dn, `Dwarf_Unsigned` dw_index_of_abbrev, `Dwarf_Unsigned` dw_offset_in_entrypool_of_values, `Dwarf_Unsigned` dw_arrays_length, `Dwarf_Half` *dw_array_idx_number, `Dwarf_Half` *dw_array_form, `Dwarf_Unsigned` *dw_array_of_offsets, `Dwarf_Sig8` *dw_array_of_signatures, `Dwarf_Bool` *dw_single_cu, `Dwarf_Unsigned` *dw_cu_offset, `Dwarf_Unsigned` *dw_offset_of_next_entrypool, `Dwarf_Error` *dw_error)
Return the value set defined by this entry.

9.25.1 Detailed Description

The section is new in DWARF5 and supersedes `.debug_pubnames` and `.debug_pubtypes` in DWARF2, DWARF3, and DWARF4.

The functions provide a detailed reporting of the content and structure of the table (so one can build one's own search table) but they are not particularly helpful for searching.

A new function (more than one?) would be needed for convenient searching.

9.25.2 Function Documentation

9.25.2.1 `dwarf_dealloc_dnames()`

```
DW_API void dwarf_dealloc_dnames (
    Dwarf_Dnames_Head dw_dn )
```

Frees all the malloc data associated with `dw_dn`.

Parameters

<code>dw_dn</code>	A <code>Dwarf_Dnames_Head</code> pointer. Callers should zero the pointer passed in as soon as possible after this returns as the pointer is then stale.
--------------------	--

9.25.2.2 `dwarf_dnames_abbrevtable()`

```
DW_API int dwarf_dnames_abbrevtable (
    Dwarf_Dnames_Head dw_dn,
    Dwarf_Unsigned dw_index,
    Dwarf_Unsigned * dw_abbrev_offset,
    Dwarf_Unsigned * dw_abbrev_code,
    Dwarf_Unsigned * dw_abbrev_tag,
    Dwarf_Unsigned dw_array_size,
    Dwarf_Half * dw_idxattr_array,
    Dwarf_Half * dw_form_array,
    Dwarf_Unsigned * dw_idxattr_count )
```

Access to the abbrevs table content.

Of interest mainly to debugging issues with compilers or debuggers.

Parameters

<code>dw_dn</code>	A <code>Dwarf_Dnames_Head</code> pointer.
<code>dw_index</code>	An index (starting at zero) into a table constructed of abbrev data. These indexes are derived from abbrev data and are not in the abbrev data itself.
<code>dw_abbrev_offset</code>	Returns the offset of the abbrev table entry for this names table entry.
<code>dw_abbrev_code</code>	Returns the abbrev code for the abbrev at offset <code>dw_abbrev_offset</code> .
<code>dw_abbrev_tag</code>	Returns the tag for the abbrev at offset <code>dw_abbrev_offset</code> .
<code>dw_array_size</code>	The size you allocated in each of the following two arrays.

Parameters

<i>dw_idxattr_array</i>	Pass in an array you allocated where the function returns and array of index attributes (DW_IDX) for this dw_abbrev_code. The last attribute code in the array is zero.
<i>dw_form_array</i>	Pass in an array you allocated where the function returns and array of forms for this dw_abbrev_code (paralleled to dw_idxattr_array). The last form code in the array is zero.
<i>dw_idxattr_count</i>	Returns the actual idxattribute/form count (including the terminating 0,0 pair. If the array_size passed in is less than this value the array returned is incomplete. Array entries needed. Might be larger than dw_array_size, meaning not all entries could be returned in your arrays.

Returns

Returns DW_DLV_OK on success. If the offset does not refer to a known part of the abbrev table it returns DW_DLV_NO_ENTRY. Never returns DW_DLV_ERROR.

9.25.2.3 dwarf_dnames_bucket()

```
DW_API int dwarf_dnames_bucket (
    Dwarf_Dnames_Head dw_dn,
    Dwarf_Unsigned dw_bucket_number,
    Dwarf_Unsigned * dw_index,
    Dwarf_Unsigned * dw_indexcount,
    Dwarf_Error * dw_error )
```

Access to bucket contents.

Parameters

<i>dw_dn</i>	The Dwarf_Dnames_Head of interest.
<i>dw_bucket_number</i>	Pass in a bucket number Bucket numbers start at 0.
<i>dw_index</i>	On success returns the index of the appropriate name entry. Name entry indexes start at one, a zero index means the bucket is unused.
<i>dw_indexcount</i>	On success returns the number of name entries in the bucket.
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc. An out of range dw_index_number gets a return if DW_DLV_NO_ENTRY

9.25.2.4 dwarf_dnames_cu_table()

```
DW_API int dwarf_dnames_cu_table (
    Dwarf_Dnames_Head dw_dn,
    const char * dw_type,
    Dwarf_Unsigned dw_index_number,
    Dwarf_Unsigned * dw_offset,
    Dwarf_Sig8 * dw_sig,
    Dwarf_Error * dw_error )
```

Each debug names cu list entry one at a time.

Indexes to the cu/tu/ tables start at 0.

Some values in `dw_offset` are actually offsets, such as for `DW_IDX_die_offset`. `DW_IDX_compile_unit` and `DW_IDX_type_unit` are indexes into the table specified by `dw_type` and are returned through `dw_offset` field;

Parameters

<code>dw_dn</code>	The table of interest.
<code>dw_type</code>	Pass in the type, "cu" or "tu"
<code>dw_index_number</code>	For "cu" index range is 0 through K-1 For "tu" index range is 0 through T+F-1
<code>dw_offset</code>	Zero if it cannot be determined. (check the return value!).
<code>dw_sig</code>	the Dwarf_Sig8 is filled in with a signature if the TU index is T through T+F-1
<code>dw_error</code>	On error <code>dw_error</code> is set to point to the error details.

Returns

The usual value: `DW_DLV_OK` etc.

9.25.2.5 dwarf_dnames_entrpool()

```
DW_API int dwarf_dnames_entrpool (
    Dwarf_Dnames_Head dw_dn,
    Dwarf_Unsigned dw_offset_in_entrpool,
    Dwarf_Unsigned * dw_abbrev_code,
    Dwarf_Half * dw_tag,
    Dwarf_Unsigned * dw_value_count,
    Dwarf_Unsigned * dw_index_of_abbrev,
    Dwarf_Unsigned * dw_offset_of_initial_value,
    Dwarf_Error * dw_error )
```

Return a the set of values from an entrpool entry.

Returns the basic data about an entrpool record and enables correct calling of `dwarf_dnames_entrpool_values` (see below). The two-stage approach makes it simple for callers to prepare for the number of values that will be returned by `dwarf_dnames_entrpool_values()`

Parameters

<code>dw_dn</code>	Pass in the debug names table of interest.
<code>dw_offset_in_entrpool</code>	The record offset (in the entry pool table) of the first record of IDX attributes. Starts at zero.
<code>dw_abbrev_code</code>	On success returns the abbrev code of the idx attributes for the pool entry.
<code>dw_tag</code>	On success returns the TAG of the DIE referred to by this entrpool entry.
<code>dw_value_count</code>	On success returns the number of distinct values imply by this entry.
<code>dw_index_of_abbrev</code>	On success returns the index of the abbrev index/form pairs in the abbreviation table.
<code>dw_offset_of_initial_value</code>	On success returns the entry pool offset of the sequence of bytes containing values, such as a CU index or a DIE offset.
<code>dw_error</code>	The usual error detail record

Returns

DW_DLV_OK is returned if the specified name entry exists. DW_DLV_NO_ENTRY is returned if the specified offset is outside the size of the table. DW_DLV_ERROR is returned in case of an internal error or corrupt section content.

9.25.2.6 dwarf_dnames_entrypool_values()

```
DW_API int dwarf_dnames_entrypool_values (
    Dwarf_Dnames_Head dw_dn,
    Dwarf_Unsigned dw_index_of_abbrev,
    Dwarf_Unsigned dw_offset_in_entrypool_of_values,
    Dwarf_Unsigned dw_arrays_length,
    Dwarf_Half * dw_array_idx_number,
    Dwarf_Half * dw_array_form,
    Dwarf_Unsigned * dw_array_of_offsets,
    Dwarf_Sig8 * dw_array_of_signatures,
    Dwarf_Bool * dw_single_cu,
    Dwarf_Unsigned * dw_cu_offset,
    Dwarf_Unsigned * dw_offset_of_next_entrypool,
    Dwarf_Error * dw_error )
```

Return the value set defined by this entry.

Call here after calling dwarf_dnames_entrypool to provide data to call this function correctly.

This retrieves the index attribute values that identify a names table name.

The caller allocates a set of arrays and the function fills them in. If dw_array_idx_number[n] is DW_IDX_type_hash then dw_array_of_signatures[n] contains the hash. For other IDX values dw_array_of_offsets[n] contains the value being returned.

Parameters

<i>dw_dn</i>	Pass in the debug names table of interest.
<i>dw_index_of_abbrev</i>	Pass in the abbreviation index.
<i>dw_offset_in_entrypool_of_values</i>	Pass in the offset of the values returned by dw_offset_of_initial_value above.
<i>dw_arrays_length</i>	Pass in the array length of each of the following four fields. The dw_value_count returned above is what you need to use.
<i>dw_array_idx_number</i>	Create an array of Dwarf_Half values, dw_arrays_length long, and pass a pointer to the first entry here.
<i>dw_array_form</i>	Create an array of Dwarf_Half values, dw_arrays_length long, and pass a pointer to the first entry here.
<i>dw_array_of_offsets</i>	Create an array of Dwarf_Unsigned values, dw_arrays_length long, and pass a pointer to the first entry here.
<i>dw_array_of_signatures</i>	Create an array of Dwarf_Sig8 structs, dw_arrays_length long, and pass a pointer to the first entry here.
<i>dw_offset_of_next_entrypool</i>	On success returns the offset of the next entrypool. A value here is usable in the next call to dwarf_dnames_entrypool.
<i>dw_single_cu</i>	On success, if it is a single-cu name table there is likely no DW_IDX_compile_unit. So we return TRUE via this flag in such a case.
<i>dw_cu_offset</i>	On success, for a single-cu name table with no DW_IDX_compile_unit this is set to the CU offset from that single CU-table entry.
<i>dw_error</i>	The usual error detail record

Returns

DW_DLV_OK is returned if the specified name entry exists. DW_DLV_NO_ENTRY is returned if the specified offset is outside the size of the table. DW_DLV_ERROR is returned in case of an internal error or corrupt section content.

9.25.2.7 dwarf_dnames_header()

```
DW_API int dwarf_dnames_header (
    Dwarf_Debug dw_dbg,
    Dwarf_Off dw_starting_offset,
    Dwarf_Dnames_Head * dw_dn,
    Dwarf_Off * dw_offset_of_next_table,
    Dwarf_Error * dw_error )
```

Open access to a .debug_names table.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_starting_offset</i>	Read this section starting at offset zero.
<i>dw_dn</i>	On success returns a pointer to a set of data allowing access to the table.
<i>dw_offset_of_next_table</i>	On success returns Offset just past the end of the the opened table.
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc. If there is no such table or if dw_starting_offset is past the end of the section it returns DW_DLV_NO_ENTRY.

9.25.2.8 dwarf_dnames_name()

```
DW_API int dwarf_dnames_name (
    Dwarf_Dnames_Head dw_dn,
    Dwarf_Unsigned dw_name_index,
    Dwarf_Unsigned * dw_bucket_number,
    Dwarf_Unsigned * dw_hash_value,
    Dwarf_Unsigned * dw_offset_to_debug_str,
    char ** dw_ptrtostr,
    Dwarf_Unsigned * dw_offset_in_entrypool,
    Dwarf_Unsigned * dw_abbrev_number,
    Dwarf_Half * dw_abbrev_tag,
    Dwarf_Unsigned dw_array_size,
    Dwarf_Half * dw_idxattr_array,
    Dwarf_Half * dw_form_array,
    Dwarf_Unsigned * dw_idxattr_count,
    Dwarf_Error * dw_error )
```

Retrieve a name table entry.

Retrieve the name and other data from a single name table entry.

Parameters

<i>dw_dn</i>	The table of interest.
<i>dw_name_index</i>	Pass in the desired index, start at one.
<i>dw_bucket_number</i>	On success returns a bucket number, zero if no buckets present.
<i>dw_hash_value</i>	The hash value, all zeros if no hashes present
<i>dw_offset_to_debug_str</i>	The offset to the .debug_str section string.
<i>dw_ptrtostr</i>	if dw_ptrtostr non-null returns a pointer to the applicable string here.
<i>dw_offset_in_entrypool</i>	Returns the offset in the entrypool
<i>dw_abbrev_number</i>	Returned from entrypool.
<i>dw_abbrev_tag</i>	Returned from entrypool abbrev data.
<i>dw_array_size</i>	Size of array you provide to hold DW_IDX index attribute and form numbers. Possibly 10 suffices for practical purposes.
<i>dw_idxattr_array</i>	Array space you provide, for idx attribute numbers (function will initialize it). The final entry in the array will be 0.
<i>dw_form_array</i>	Array you provide, for form numbers (function will initialize it). The final entry in the array will be 0.
<i>dw_idxattr_count</i>	Array entries needed. Might be larger than dw_array_size, meaning not all entries could be returned in your array.
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc. If the index passed in is outside the valid range returns DW_DLV_NO_ENTRY.

9.25.2.9 dwarf_dnames_offsets()

```
DW_API int dwarf_dnames_offsets (
    Dwarf_Dnames_Head dw_dn,
    Dwarf_Unsigned * dw_header_offset,
    Dwarf_Unsigned * dw_cu_table_offset,
    Dwarf_Unsigned * dw_tu_local_offset,
    Dwarf_Unsigned * dw_foreign_tu_offset,
    Dwarf_Unsigned * dw_bucket_offset,
    Dwarf_Unsigned * dw_hashes_offset,
    Dwarf_Unsigned * dw_stringoffsets_offset,
    Dwarf_Unsigned * dw_entryoffsets_offset,
    Dwarf_Unsigned * dw_abbrev_table_offset,
    Dwarf_Unsigned * dw_entry_pool_offset,
    Dwarf_Error * dw_error )
```

Offsets from the debug names table.

We do not describe these returned values, which refer to the .debug_names section.

The header offset is a section offset. The rest are offsets from the header.

See DWARF5 section 6.1.1 "Lookup By Name"

9.25.2.10 dwarf_dnames_sizes()

```
DW_API int dwarf_dnames_sizes (
    Dwarf_Dnames_Head dw_dn,
    Dwarf_Unsigned * dw_comp_unit_count,
    Dwarf_Unsigned * dw_local_type_unit_count,
    Dwarf_Unsigned * dw_foreign_type_unit_count,
    Dwarf_Unsigned * dw_bucket_count,
    Dwarf_Unsigned * dw_name_count,
    Dwarf_Unsigned * dw_abbrev_table_size,
    Dwarf_Unsigned * dw_entry_pool_size,
    Dwarf_Unsigned * dw_augmentation_string_size,
    char ** dw_augmentation_string,
    Dwarf_Unsigned * dw_section_size,
    Dwarf_Half * dw_table_version,
    Dwarf_Half * dw_offset_size,
    Dwarf_Error * dw_error )
```

Sizes and counts from the debug names table.

We do not describe these returned values. Other than for `dw_dn` and `dw_error` passing pointers you do not care about as NULL is fine. Of course no value can be returned through those passed as NULL.

Any program referencing a names table will need at least a few of these values.

See DWARF5 section 6.1.1 "Lookup By Name" particularly the graph page 139. `dw_comp_unit_count` is K(k), `dw_local_type_unit_count` is T(t), and `dw_foreign_type_unit_count` is F(f).

9.26 Fast Access to a CU given a code address

Functions

- DW_API int `dwarf_get_aranges` (Dwarf_Debug dw_dbg, Dwarf_Arange **dw_aranges, Dwarf_Signed *dw_↵
_arange_count, Dwarf_Error *dw_error)
Get access to CUs given code addresses.
- DW_API int `dwarf_get_arange` (Dwarf_Arange *dw_aranges, Dwarf_Unsigned dw_arange_count,
Dwarf_Addr dw_address, Dwarf_Arange *dw_returned_arange, Dwarf_Error *dw_error)
Find a range given a code address.
- DW_API int `dwarf_get_cu_die_offset` (Dwarf_Arange dw_arange, Dwarf_Off *dw_return_offset, Dwarf_Error
*dw_error)
Given an arange return its CU DIE offset.
- DW_API int `dwarf_get_arange_cu_header_offset` (Dwarf_Arange dw_arange, Dwarf_Off *dw_return_cu_↵
header_offset, Dwarf_Error *dw_error)
Given an arange return its CU header offset.
- DW_API int `dwarf_get_arange_info_b` (Dwarf_Arange dw_arange, Dwarf_Unsigned *dw_segment,
Dwarf_Unsigned *dw_segment_entry_size, Dwarf_Addr *dw_start, Dwarf_Unsigned *dw_length, Dwarf_Off
*dw_cu_die_offset, Dwarf_Error *dw_error)
Get the data in an arange entry.

9.26.1 Detailed Description

9.26.2 Function Documentation

9.26.2.1 dwarf_get_arange()

```
DW_API int dwarf_get_arange (
    Dwarf_Arange * dw_aranges,
    Dwarf_Unsigned dw_arange_count,
    Dwarf_Addr dw_address,
    Dwarf_Arange * dw_returned_arange,
    Dwarf_Error * dw_error )
```

Find a range given a code address.

Parameters

<i>dw_aranges</i>	Pass in a pointer to the first entry in the aranges array of pointers.
<i>dw_arange_count</i>	Pass in the <i>dw_arange_count</i> , the count for the array.
<i>dw_address</i>	Pass in the code address of interest.
<i>dw_returned_arange</i>	On success, returns the particular arange that holds that address.
<i>dw_error</i>	On error <i>dw_error</i> is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc. Returns DW_DLV_NO_ENTRY if there is no such code address present in the section.

9.26.2.2 dwarf_get_arange_cu_header_offset()

```
DW_API int dwarf_get_arange_cu_header_offset (
    Dwarf_Arange dw_arange,
    Dwarf_Off * dw_return_cu_header_offset,
    Dwarf_Error * dw_error )
```

Given an arange return its CU header offset.

Parameters

<i>dw_arange</i>	The specific arange of interest.
<i>dw_return_cu_header_offset</i>	The CU header offset (in <i>.debug_info</i>) applicable to this arange.
<i>dw_error</i>	On error <i>dw_error</i> is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc.

9.26.2.3 dwarf_get_arange_info_b()

```
DW_API int dwarf_get_arange_info_b (
    Dwarf_Arange dw_arange,
    Dwarf_Unsigned * dw_segment,
    Dwarf_Unsigned * dw_segment_entry_size,
    Dwarf_Addr * dw_start,
    Dwarf_Unsigned * dw_length,
    Dwarf_Off * dw_cu_die_offset,
    Dwarf_Error * dw_error )
```

Get the data in an arange entry.

Parameters

<i>dw_arange</i>	The specific arange of interest.
<i>dw_segment</i>	On success and if segment_entry_size is non-zero this returns the segment number from the arange.
<i>dw_segment_entry_size</i>	On success returns the segment entry size from the arange.
<i>dw_start</i>	On success returns the low address this arange refers to.
<i>dw_length</i>	On success returns the length, in bytes of the code area this arange refers to.
<i>dw_cu_die_offset</i>	On success returns the .debug_info section offset the arange refers to.
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc.

9.26.2.4 dwarf_get_aranges()

```
DW_API int dwarf_get_aranges (
    Dwarf_Debug dw_dbg,
    Dwarf_Arange ** dw_aranges,
    Dwarf_Signed * dw_arange_count,
    Dwarf_Error * dw_error )
```

Get access to CUs given code addresses.

This intended as a fast-access to tie code addresses to CU dies. The data is in the .debug_aranges section. which may appear in DWARF2,3,4, or DWARF5.

See also

[Reading an aranges section](#)

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_aranges</i>	On success returns a pointer to an array of Dwarf_Arange pointers.
<i>dw_arange_count</i>	On success returns a count of the length of the array.
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc. Returns DW_DLV_NO_ENTRY if there is no such section.

9.26.2.5 dwarf_get_cu_die_offset()

```
DW_API int dwarf_get_cu_die_offset (
    Dwarf_Arange dw_arange,
    Dwarf_Off * dw_return_offset,
    Dwarf_Error * dw_error )
```

Given an arange return its CU DIE offset.

Parameters

<i>dw_arange</i>	The specific arange of interest.
<i>dw_return_offset</i>	The CU DIE offset (in .debug_info) applicable to this arange..
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc.

9.27 Fast Access to .debug_pubnames and more.**Macros**

- #define **DW_GL_GLOBALS** 0 /* .debug_pubnames and .debug_names */
- #define **DW_GL_PUBTYPES** 1 /* .debug_pubtypes */
- #define **DW_GL_FUNCS** 2 /* .debug_funcnames */
- #define **DW_GL_TYPES** 3 /* .debug_typednames */
- #define **DW_GL_VARS** 4 /* .debug_varnames */
- #define **DW_GL_WEAKS** 5 /* .debug_weaknames */

Functions

- DW_API int **dwarf_get_globals** (Dwarf_Debug dw_dbg, Dwarf_Global **dw_globals, Dwarf_Signed *dw_↔ number_of_globals, Dwarf_Error *dw_error)
Global name space operations, .debug_pubnames access.
- DW_API int **dwarf_get_pubtypes** (Dwarf_Debug dw_dbg, Dwarf_Global **dw_pubtypes, Dwarf_Signed *dw_number_of_pubtypes, Dwarf_Error *dw_error)
Global debug types access.
- DW_API int **dwarf_globals_by_type** (Dwarf_Debug dw_dbg, int dw_requested_section, Dwarf_Global **dw_↔ _contents, Dwarf_Signed *dw_count, Dwarf_Error *dw_error)
Allocate Any Fast Access DWARF2-DWARF4.
- DW_API void **dwarf_globals_dealloc** (Dwarf_Debug dw_dbg, Dwarf_Global *dw_global_like, Dwarf_Signed dw_count)
Dealloc the Dwarf_Global data.
- DW_API int **dwarf_globname** (Dwarf_Global dw_global, char **dw_returned_name, Dwarf_Error *dw_error)

Return the name of a global-like data item.

- DW_API int [dwarf_global_die_offset](#) (Dwarf_Global dw_global, Dwarf_Off *dw_die_offset, Dwarf_Error *dw_error)

Return the DIE offset of a global data item.

- DW_API int [dwarf_global_cu_offset](#) (Dwarf_Global dw_global, Dwarf_Off *dw_cu_header_offset, Dwarf_Error *dw_error)

Return the CU header data of a global data item.

- DW_API int [dwarf_global_name_offsets](#) (Dwarf_Global dw_global, char **dw_returned_name, Dwarf_Off *dw_die_offset, Dwarf_Off *dw_cu_die_offset, Dwarf_Error *dw_error)

Return the name and offsets of a global entry.

- DW_API Dwarf_Half [dwarf_global_tag_number](#) (Dwarf_Global dw_global)

Return the DW_TAG number of a global entry.

- DW_API int [dwarf_get_globals_header](#) (Dwarf_Global dw_global, int *dw_category, Dwarf_Off *dw_offset↵_pub_header, Dwarf_Unsigned *dw_length_size, Dwarf_Unsigned *dw_length_pub, Dwarf_Unsigned *dw↵_version, Dwarf_Unsigned *dw_header_info_offset, Dwarf_Unsigned *dw_info_length, Dwarf_Error *dw↵error)

For more complete globals printing.

- DW_API int [dwarf_return_empty_pubnames](#) (Dwarf_Debug dw_dbg, int dw_flag)

A flag for dwarfdump on pubnames, pubtypes etc.

9.27.1 Detailed Description

Pubnames and Pubtypes overview

These functions each read one of a set of sections designed for fast access by name, but they are not always emitted as they each have somewhat limited and inflexible capabilities. So you may not see many of these.

All have the same set of functions with a name reflecting the specific object section involved. Only the first, of type Dwarf_Global, is documented here in full detail as the others do the same jobs just each for their applicable object section..

9.27.2 Function Documentation

9.27.2.1 dwarf_get_globals()

```
DW_API int dwarf_get_globals (
    Dwarf_Debug dw_dbg,
    Dwarf_Global ** dw_globals,
    Dwarf_Signed * dw_number_of_globals,
    Dwarf_Error * dw_error )
```

Global name space operations, .debug_pubnames access.

This accesses .debug_pubnames and .debug_names sections. Section .debug_pubnames is defined in DWARF2, DWARF3, and DWARF4. Section .debug_names is defined in DWARF5 and contains lots of information, but only the part of the wealth of information that this interface allows can be retrieved here. See [dwarf_dnames_header\(\)](#) for access to all. debug_names data.

The code here, as of 0.4.3, September 3 2022, returns data from either section.

See also

[Using dwarf_get_globals\(\)](#)

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_globals</i>	On success returns an array of pointers to opaque structs..
<i>dw_number_of_globals</i>	On success returns the number of entries in the array.
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc. Returns DW_DLV_NO_ENTRY if the section is not present.

9.27.2.2 dwarf_get_globals_header()

```
DW_API int dwarf_get_globals_header (
    Dwarf_Global dw_global,
    int * dw_category,
    Dwarf_Off * dw_offset_pub_header,
    Dwarf_Unsigned * dw_length_size,
    Dwarf_Unsigned * dw_length_pub,
    Dwarf_Unsigned * dw_version,
    Dwarf_Unsigned * dw_header_info_offset,
    Dwarf_Unsigned * dw_info_length,
    Dwarf_Error * dw_error )
```

For more complete globals printing.

For each CU represented in .debug_pubnames, etc, there is a .debug_pubnames header. For any given Dwarf_Global this returns the content of the applicable header. This does not include header information from any .debug_names headers.

The function declaration changed at version 0.6.0.

9.27.2.3 dwarf_get_pubtypes()

```
DW_API int dwarf_get_pubtypes (
    Dwarf_Debug dw_dbg,
    Dwarf_Global ** dw_pubtypes,
    Dwarf_Signed * dw_number_of_pubtypes,
    Dwarf_Error * dw_error )
```

Global debug_types access.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_pubtypes</i>	On success returns an array of pointers to opaque structs..
<i>dw_number_of_pubtypes</i>	On success returns the number of entries in the array.
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc. Returns DW_DLV_NO_ENTRY if the section is not present.

Same function name as 0.5.0 and earlier, but the data type changes to Dwarf_Global

[dwarf_get_pubtypes\(\)](#) is an alternate name for [dwarf_globals_by_type\(...,DW_GL_PUBTYPES,...\)](#).

9.27.2.4 dwarf_global_cu_offset()

```
DW_API int dwarf_global_cu_offset (
    Dwarf_Global dw_global,
    Dwarf_Off * dw_cu_header_offset,
    Dwarf_Error * dw_error )
```

Return the CU header data of a global data item.

A CU header offset is rarely useful.

Parameters

<i>dw_global</i>	The Dwarf_Global of interest.
<i>dw_cu_header_offset</i>	On success a the section-global offset of a CU header is returned.
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc.

9.27.2.5 dwarf_global_die_offset()

```
DW_API int dwarf_global_die_offset (
    Dwarf_Global dw_global,
    Dwarf_Off * dw_die_offset,
    Dwarf_Error * dw_error )
```

Return the DIE offset of a global data item.

Parameters

<i>dw_global</i>	The Dwarf_Global of interest.
<i>dw_die_offset</i>	On success a the section-global DIE offset of a data item is returned.
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc.

9.27.2.6 dwarf_global_name_offsets()

```
DW_API int dwarf_global_name_offsets (
    Dwarf_Global dw_global,
    char ** dw_returned_name,
    Dwarf_Off * dw_die_offset,
    Dwarf_Off * dw_cu_die_offset,
    Dwarf_Error * dw_error )
```

Return the name and offsets of a global entry.

Parameters

<i>dw_global</i>	The Dwarf_Global of interest.
<i>dw_returned_name</i>	On success a pointer to the name (a null-terminated string) is returned.
<i>dw_die_offset</i>	On success a the section-global DIE offset of the global with the name.
<i>dw_cu_die_offset</i>	On success a the section-global offset of the relevant CU DIE is returned.
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc.

9.27.2.7 dwarf_global_tag_number()

```
DW_API Dwarf_Half dwarf_global_tag_number (
    Dwarf_Global dw_global )
```

Return the DW_TAG number of a global entry.

Parameters

<i>dw_global</i>	The Dwarf_Global of interest.
------------------	-------------------------------

Returns

If the Dwarf_Global refers to a global from the .debug_names section the return value is the DW_TAG for the DIE in the global entry, for example DW_TAG_subprogram. In case of error or if the section for this global was not .debug_names zero is returned.

9.27.2.8 dwarf_globals_by_type()

```
DW_API int dwarf_globals_by_type (
    Dwarf_Debug dw_dbg,
    int dw_requested_section,
    Dwarf_Global ** dw_contents,
    Dwarf_Signed * dw_count,
    Dwarf_Error * dw_error )
```

Allocate Any Fast Access DWARF2-DWARF4.

This interface new in 0.6.0. Simplifies access by replace dwarf_get_pubtypes, dwarf_get_funcs, dwarf_get_types, dwarfget_vars, and dwarf_get_weak with a single set of types.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_requested_section</i>	Pass in one of the values DW_GL_GLOBALS through DW_GL_WEAKS to select the section to extract data from.
<i>dw_contents</i>	On success returns an array of pointers to opaque structs.
<i>dw_count</i>	On success returns the number of entries in the array.
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc. Returns DW_DLV_NO_ENTRY if the section is not present.

9.27.2.9 dwarf_globals_dealloc()

```
DW_API void dwarf_globals_dealloc (
    Dwarf_Debug dw_dbg,
    Dwarf_Global * dw_global_like,
    Dwarf_Signed dw_count )
```

Dealloc the Dwarf_Global data.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_global_like</i>	The array of globals/types/etc data to dealloc (free).
<i>dw_count</i>	The number of entries in the array.

9.27.2.10 dwarf_globname()

```
DW_API int dwarf_globname (
    Dwarf_Global dw_global,
    char ** dw_returned_name,
    Dwarf_Error * dw_error )
```

Return the name of a global-like data item.

Parameters

<i>dw_global</i>	The Dwarf_Global of interest.
<i>dw_returned_name</i>	On success a pointer to the name (a null-terminated string) is returned.
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

The usual value: DW_DLV_OK etc.

9.27.2.11 dwarf_return_empty_pubnames()

```
DW_API int dwarf_return_empty_pubnames (
    Dwarf_Debug dw_dbg,
    int dw_flag )
```

A flag for dwarfdump on pubnames, pubtypes etc.

Sets a flag in the dbg. Always returns DW_DLV_OK. Applies to all the sections of this kind: pubnames, pubtypes, funcs, typenames, vars, weaks. Ensures empty content (meaning no offset/name tuples, but with a header) for a CU shows up rather than being suppressed.

Primarily useful if one wants to note any pointless header data in the section.

Pubnames and Pubtypes overview

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_flag</i>	Must be the value one.

Returns

Returns DW_DLV_OK. Always.

9.28 Fast Access to GNU .debug_gnu_pubnames

Functions

- DW_API int [dwarf_get_gnu_index_head](#) (Dwarf_Debug dw_dbg, Dwarf_Bool dw_which_section, Dwarf_Gnu_Index_Head *dw_head, Dwarf_Unsigned *dw_index_block_count_out, Dwarf_Error *dw_error)
Access to .debug_gnu_pubnames or .debug_gnu_pubtypes.
- DW_API void [dwarf_gnu_index_dealloc](#) (Dwarf_Gnu_Index_Head dw_head)
Free resources of .debug_gnu_pubnames .debug_gnu_pubtypes.
- DW_API int [dwarf_get_gnu_index_block](#) (Dwarf_Gnu_Index_Head dw_head, Dwarf_Unsigned dw_number, Dwarf_Unsigned *dw_block_length, Dwarf_Half *dw_version, Dwarf_Unsigned *dw_offset_into_debug_info, Dwarf_Unsigned *dw_size_of_debug_info_area, Dwarf_Unsigned *dw_count_of_index_entries, Dwarf_Error *dw_error)
Access a particular block.
- DW_API int [dwarf_get_gnu_index_block_entry](#) (Dwarf_Gnu_Index_Head dw_head, Dwarf_Unsigned dw_blocknumber, Dwarf_Unsigned dw_entrynumber, Dwarf_Unsigned *dw_offset_in_debug_info, const char **dw_name_string, unsigned char *dw_flagbyte, unsigned char *dw_staticorglobal, unsigned char *dw_typeofentry, Dwarf_Error *dw_error)
Access a particular entry of a block.

9.28.1 Detailed Description

Section .debug_gnu_pubnames or .debug_gnu_pubtypes.

This is a section created for and used by the GNU gdb debugger to access DWARF information.

Not part of standard DWARF.

9.28.2 Function Documentation

9.28.2.1 dwarf_get_gnu_index_block()

```
DW_API int dwarf_get_gnu_index_block (
    Dwarf_Gnu_Index_Head dw_head,
    Dwarf_Unsigned dw_number,
    Dwarf_Unsigned * dw_block_length,
    Dwarf_Half * dw_version,
    Dwarf_Unsigned * dw_offset_into_debug_info,
    Dwarf_Unsigned * dw_size_of_debug_info_area,
    Dwarf_Unsigned * dw_count_of_index_entries,
    Dwarf_Error * dw_error )
```

Access a particular block.

Parameters

<i>dw_head</i>	Pass in the Dwarf_Gnu_Index_head interest.
<i>dw_number</i>	Pass in the block number of the block of interest. 0 through dw_index_block_count_out-1.
<i>dw_block_length</i>	On success set to the length of the data in this block, in bytes.
<i>dw_version</i>	On success set to the version number of the block.
<i>dw_offset_into_debug_info</i>	On success set to the offset, in .debug_info, of the data for this block. The returned offset may be outside the bounds of the actual .debug_info section, such a possibility does not cause the function to return DW_DLV_ERROR.
<i>dw_size_of_debug_info_area</i>	On success set to the size in bytes, in .debug_info, of the area this block refers to. The returned dw_dw_size_of_debug_info_area plus dw_offset_into_debug_info may be outside the bounds of the actual .debug_info section, such a possibility does not cause the function to return DW_DLV_ERROR. Use dwarf_get_section_max_offsets_d() to learn the size of .debug_info and optionally other sections as well.
<i>dw_count_of_index_entries</i>	On success set to the count of index entries in this particular block number.
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

Returns DW_DLV_OK, DW_DLV_NO_ENTRY (if the section does not exist or is empty), or, in case of an error reading the section, DW_DLV_ERROR.

9.28.2.2 dwarf_get_gnu_index_block_entry()

```
DW_API int dwarf_get_gnu_index_block_entry (
    Dwarf_Gnu_Index_Head dw_head,
    Dwarf_Unsigned dw_blocknumber,
    Dwarf_Unsigned dw_entrynumber,
    Dwarf_Unsigned * dw_offset_in_debug_info,
    const char ** dw_name_string,
    unsigned char * dw_flagbyte,
    unsigned char * dw_staticorglobal,
    unsigned char * dw_typeofentry,
    Dwarf_Error * dw_error )
```

Access a particular entry of a block.

Access to a single entry in a block.

Parameters

<i>dw_head</i>	Pass in the Dwarf_Gnu_Index_head interest.
<i>dw_blocknumber</i>	Pass in the block number of the block of interest. 0 through dw_index_block_count_out-1.
<i>dw_entrynumber</i>	Pass in the entry number of the entry of interest. 0 through dw_count_of_index_entries-1.
<i>dw_offset_in_debug_info</i>	On success set to the offset in .debug_info relevant to this entry.
<i>dw_name_string</i>	On success set to the size in bytes, in .debug_info, of the area this block refersto.
<i>dw_flagbyte</i>	On success set to the entry flag byte content.
<i>dw_staticorglobal</i>	On success set to the entry static/global letter.
<i>dw_typeofentry</i>	On success set to the type of entry.
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

Returns DW_DLV_OK, DW_DLV_NO_ENTRY (if the section does not exist or is empty), or, in case of an error reading the section, DW_DLV_ERROR.

9.28.2.3 dwarf_get_gnu_index_head()

```
DW_API int dwarf_get_gnu_index_head (
    Dwarf_Debug dw_dbg,
    Dwarf_Bool dw_which_section,
    Dwarf_Gnu_Index_Head * dw_head,
    Dwarf_Unsigned * dw_index_block_count_out,
    Dwarf_Error * dw_error )
```

Access to .debug_gnu_pubnames or .debug_gnu_pubtypes.

Call this to get access.

Parameters

<i>dw_dbg</i>	Pass in the Dwarf_Debug of interest.
<i>dw_which_section</i>	Pass in TRUE to access .debug_gnu_pubnames. Pass in FALSE to access .debug_gnu_typednames.
<i>dw_head</i>	On success, set to a pointer to a head record allowing access to all the content of the section.
<i>dw_index_block_count_out</i>	On success, set to a count of the number of blocks of data available.
<i>dw_error</i>	

Returns

Returns DW_DLV_OK, DW_DLV_NO_ENTRY (if the section does not exist or is empty), or, in case of an error reading the section, DW_DLV_ERROR.

9.28.2.4 dwarf_gnu_index_dealloc()

```
DW_API void dwarf_gnu_index_dealloc (
    Dwarf_Gnu_Index_Head dw_head )
```

Free resources of `.debug_gnu_pubnames` `.debug_gnu_pubtypes`.

Call this to deallocate all memory used by `dw_head`.

Parameters

<code>dw_head</code>	Pass in the <code>Dwarf_Gnu_Index_head</code> whose data is to be deallocated.
----------------------	--

9.29 Fast Access to Gdb Index

Functions

- DW_API int `dwarf_gdbindex_header` (`Dwarf_Debug` dw_dbg, `Dwarf_Gdbindex` *dw_gdbindexptr, `Dwarf_Unsigned` *dw_version, `Dwarf_Unsigned` *dw_cu_list_offset, `Dwarf_Unsigned` *dw_types_↵
cu_list_offset, `Dwarf_Unsigned` *dw_address_area_offset, `Dwarf_Unsigned` *dw_symbol_table_offset, `Dwarf_Unsigned` *dw_constant_pool_offset, `Dwarf_Unsigned` *dw_section_size, const char **dw_section_↵
_name, `Dwarf_Error` *dw_error)
Open access to the .gdb_index section.
- DW_API void `dwarf_dealloc_gdbindex` (`Dwarf_Gdbindex` dw_gdbindexptr)
Free (dealloc) all allocated Dwarf_Gdbindex memory It should named dwarf_dealloc_gdbindex.
- DW_API int `dwarf_gdbindex_culist_array` (`Dwarf_Gdbindex` dw_gdbindexptr, `Dwarf_Unsigned` *dw_list_↵
length, `Dwarf_Error` *dw_error)
Return the culist array length.
- DW_API int `dwarf_gdbindex_culist_entry` (`Dwarf_Gdbindex` dw_gdbindexptr, `Dwarf_Unsigned` dw_↵
entryindex, `Dwarf_Unsigned` *dw_cu_offset, `Dwarf_Unsigned` *dw_cu_length, `Dwarf_Error` *dw_error)
For a CU entry in the list return the offset and length.
- DW_API int `dwarf_gdbindex_types_culist_array` (`Dwarf_Gdbindex` dw_gdbindexptr, `Dwarf_Unsigned` *dw_↵
types_list_length, `Dwarf_Error` *dw_error)
Return the types culist array length.
- DW_API int `dwarf_gdbindex_types_culist_entry` (`Dwarf_Gdbindex` dw_gdbindexptr, `Dwarf_Unsigned` dw_↵
types_entryindex, `Dwarf_Unsigned` *dw_cu_offset, `Dwarf_Unsigned` *dw_tu_offset, `Dwarf_Unsigned` *dw_↵
_type_signature, `Dwarf_Error` *dw_error)
For a types CU entry in the list returns the offset and length.
- DW_API int `dwarf_gdbindex_addressarea` (`Dwarf_Gdbindex` dw_gdbindexptr, `Dwarf_Unsigned` *dw_↵
addressarea_list_length, `Dwarf_Error` *dw_error)
Get access to gdbindex address area.
- DW_API int `dwarf_gdbindex_addressarea_entry` (`Dwarf_Gdbindex` dw_gdbindexptr, `Dwarf_Unsigned` dw_↵
_entryindex, `Dwarf_Unsigned` *dw_low_address, `Dwarf_Unsigned` *dw_high_address, `Dwarf_Unsigned` *dw_↵
_cu_index, `Dwarf_Error` *dw_error)
Get an address area value.
- DW_API int `dwarf_gdbindex_symboltable_array` (`Dwarf_Gdbindex` dw_gdbindexptr, `Dwarf_Unsigned` *dw_↵
syntab_list_length, `Dwarf_Error` *dw_error)
Get access to the symboltable array.
- DW_API int `dwarf_gdbindex_symboltable_entry` (`Dwarf_Gdbindex` dw_gdbindexptr, `Dwarf_Unsigned` dw_↵
entryindex, `Dwarf_Unsigned` *dw_string_offset, `Dwarf_Unsigned` *dw_cu_vector_offset, `Dwarf_Error` *dw_↵
_error)
Access individual syntab entry.
- DW_API int `dwarf_gdbindex_cuvector_length` (`Dwarf_Gdbindex` dw_gdbindexptr, `Dwarf_Unsigned` dw_↵
cuvector_offset, `Dwarf_Unsigned` *dw_innercount, `Dwarf_Error` *dw_error)
Get access to a cuvector.

- DW_API int [dwarf_gdbindex_cuvector_inner_attributes](#) (Dwarf_Gdbindex dw_gdbindexptr, Dwarf_Unsigned dw_cuvector_offset_in, Dwarf_Unsigned dw_innerindex, Dwarf_Unsigned *dw_field_value, Dwarf_Error *dw_error)

Get access to a cuvector.

- DW_API int [dwarf_gdbindex_cuvector_instance_expand_value](#) (Dwarf_Gdbindex dw_gdbindexptr, Dwarf_Unsigned dw_field_value, Dwarf_Unsigned *dw_cu_index, Dwarf_Unsigned *dw_symbol_kind, Dwarf_Unsigned *dw_is_static, Dwarf_Error *dw_error)

Expand the bit fields in a cuvector entry.

- DW_API int [dwarf_gdbindex_string_by_offset](#) (Dwarf_Gdbindex dw_gdbindexptr, Dwarf_Unsigned dw_stringoffset, const char **dw_string_ptr, Dwarf_Error *dw_error)

Retrieve a symbol name from the index data.

9.29.1 Detailed Description

Section .gdb_index

This is a section created for and used by the GNU gdb debugger to access DWARF information.

Not part of standard DWARF.

See also

<https://sourceware.org/gdb/onlinedocs/gdb/Index-Section-Format.html#Index-Section-Format>

Version 8 built by gdb, so type entries are ok as is. Version 7 built by the 'gold' linker and type index entries for a CU must be derived otherwise, the type index is not correct... Earlier versions cannot be read correctly by the functions here.

The functions here make it possible to print the section content in detail, there is no search function here.

9.29.2 Function Documentation

9.29.2.1 dwarf_dealloc_gdbindex()

```
DW_API void dwarf_dealloc_gdbindex (
    Dwarf_Gdbindex dw_gdbindexptr )
```

Free (dealloc) all allocated Dwarf_Gdbindex memory It should named dwarf_dealloc_gdbindex.

Parameters

<i>dw_gdbindexptr</i>	Pass in a valid dw_gdbindexptr and on return assign zero to dw_gdbindexptr as it is stale.
-----------------------	--

9.29.2.2 dwarf_gdbindex_addressarea()

```
DW_API int dwarf_gdbindex_addressarea (
    Dwarf_Gdbindex dw_gdbindexptr,
```

```
Dwarf_Unsigned * dw_addressarea_list_length,
Dwarf_Error * dw_error )
```

Get access to gdbindex address area.

See also

[Reading gdbindex addressarea](#)

Parameters

<i>dw_gdbindexptr</i>	Pass in the Dwarf_Gdbindex pointer of interest.
<i>dw_addressarea_list_length</i>	On success returns the number of entries in the addressarea.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.3 dwarf_gdbindex_addressarea_entry()

```
DW_API int dwarf_gdbindex_addressarea_entry (
    Dwarf_Gdbindex dw_gdbindexptr,
    Dwarf_Unsigned dw_entryindex,
    Dwarf_Unsigned * dw_low_address,
    Dwarf_Unsigned * dw_high_address,
    Dwarf_Unsigned * dw_cu_index,
    Dwarf_Error * dw_error )
```

Get an address area value.

Parameters

<i>dw_gdbindexptr</i>	Pass in the Dwarf_Gdbindex pointer of interest.
<i>dw_entryindex</i>	Pass in an index, 0 through dw_addressarea_list_length-1. addressarea.
<i>dw_low_address</i>	On success returns the low address for the entry.
<i>dw_high_address</i>	On success returns the high address for the entry.
<i>dw_cu_index</i>	On success returns the index to the cu for the entry.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.4 dwarf_gdbindex_culist_array()

```
DW_API int dwarf_gdbindex_culist_array (
    Dwarf_Gdbindex dw_gdbindexptr,
```

```
Dwarf_Unsigned * dw_list_length,
Dwarf_Error * dw_error )
```

Return the culist array length.

Parameters

<i>dw_gdbindexptr</i>	Pass in the Dwarf_Gdbindex pointer of interest.
<i>dw_list_length</i>	On success returns the array length of the cu list.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.5 dwarf_gdbindex_culist_entry()

```
DW_API int dwarf_gdbindex_culist_entry (
    Dwarf_Gdbindex dw_gdbindexptr,
    Dwarf_Unsigned dw_entryindex,
    Dwarf_Unsigned * dw_cu_offset,
    Dwarf_Unsigned * dw_cu_length,
    Dwarf_Error * dw_error )
```

For a CU entry in the list return the offset and length.

Parameters

<i>dw_gdbindexptr</i>	Pass in the Dwarf_Gdbindex pointer of interest.
<i>dw_entryindex</i>	Pass in a number from 0 through dw_list_length-1. If dw_entryindex is too large for the array the function returns DW_DLV_NO_ENTRY.
<i>dw_cu_offset</i>	On success returns the CU offset for this list entry.
<i>dw_cu_length</i>	On success returns the CU length(in bytes) for this list entry.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.6 dwarf_gdbindex_cuvector_inner_attributes()

```
DW_API int dwarf_gdbindex_cuvector_inner_attributes (
    Dwarf_Gdbindex dw_gdbindexptr,
    Dwarf_Unsigned dw_cuvector_offset_in,
    Dwarf_Unsigned dw_innerindex,
    Dwarf_Unsigned * dw_field_value,
    Dwarf_Error * dw_error )
```

Get access to a cuvector.

Parameters

<i>dw_gdbindexptr</i>	Pass in the Dwarf_Gdbindex pointer of interest.
<i>dw_cuvector_offset_in</i>	Pass in the value of dw_cuvector_offset
<i>dw_innerindex</i>	Pass in the index of the CU vector in, from 0 through dw_innercount-1.
<i>dw_field_value</i>	On success returns a field of bits. To expand the bits call dwarf_gdbindex_cuvector_instance_expand_value.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.7 dwarf_gdbindex_cuvector_instance_expand_value()

```
DW_API int dwarf_gdbindex_cuvector_instance_expand_value (
    Dwarf_Gdbindex dw_gdbindexptr,
    Dwarf_Unsigned dw_field_value,
    Dwarf_Unsigned * dw_cu_index,
    Dwarf_Unsigned * dw_symbol_kind,
    Dwarf_Unsigned * dw_is_static,
    Dwarf_Error * dw_error )
```

Expand the bit fields in a cuvector entry.

Parameters

<i>dw_gdbindexptr</i>	Pass in the Dwarf_Gdbindex pointer of interest.
<i>dw_field_value</i>	Pass in the dw_field_value returned by dwarf_gdbindex_cuvector_inner_attributes.
<i>dw_cu_index</i>	On success returns the CU index from the dw_field_value
<i>dw_symbol_kind</i>	On success returns the symbol kind (see the sourceware page. Kinds are TYPE, VARIABLE, or FUNCTION).
<i>dw_is_static</i>	On success returns non-zero if the entry is a static symbol (file-local, as in C or C++), otherwise it returns non-zero and the symbol is global.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.8 dwarf_gdbindex_cuvector_length()

```
DW_API int dwarf_gdbindex_cuvector_length (
    Dwarf_Gdbindex dw_gdbindexptr,
    Dwarf_Unsigned dw_cuvector_offset,
    Dwarf_Unsigned * dw_innercount,
    Dwarf_Error * dw_error )
```

Get access to a cuvector.

See also

[Reading the gdbindex symbol table](#)

Parameters

<i>dw_gdbindexptr</i>	Pass in the Dwarf_Gdbindex pointer of interest.
<i>dw_cuvector_offset</i>	Pass in the offset, dw_cu_vector_offset.
<i>dw_innercount</i>	On success returns the number of CUs in the cuvector instance array.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.9 dwarf_gdbindex_header()

```
DW_API int dwarf_gdbindex_header (
    Dwarf_Debug dw_dbg,
    Dwarf_Gdbindex * dw_gdbindexptr,
    Dwarf_Unsigned * dw_version,
    Dwarf_Unsigned * dw_cu_list_offset,
    Dwarf_Unsigned * dw_types_cu_list_offset,
    Dwarf_Unsigned * dw_address_area_offset,
    Dwarf_Unsigned * dw_symbol_table_offset,
    Dwarf_Unsigned * dw_constant_pool_offset,
    Dwarf_Unsigned * dw_section_size,
    const char ** dw_section_name,
    Dwarf_Error * dw_error )
```

Open access to the .gdb_index section.

The section is a single table one thinks.

See also

[Reading gdbindex data](#)

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_gdbindexptr</i>	On success returns a pointer to make access to table details possible.
<i>dw_version</i>	On success returns the table version.
<i>dw_cu_list_offset</i>	On success returns the offset of the cu_list in the section.
<i>dw_types_cu_list_offset</i>	On success returns the offset of the types cu_list in the section.
<i>dw_address_area_offset</i>	On success returns the area pool offset.
<i>dw_symbol_table_offset</i>	On success returns the symbol table offset.
<i>dw_constant_pool_offset</i>	On success returns the constant pool offset.
<i>dw_section_size</i>	On success returns section size.
<i>dw_section_name</i>	On success returns section name.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc. Returns DW_DLV_NO_ENTRY if the section is absent.

9.29.2.10 dwarf_gdbindex_string_by_offset()

```
DW_API int dwarf_gdbindex_string_by_offset (
    Dwarf_Gdbindex dw_gdbindexptr,
    Dwarf_Unsigned dw_stringoffset,
    const char ** dw_string_ptr,
    Dwarf_Error * dw_error )
```

Retrieve a symbol name from the index data.

Parameters

<i>dw_gdbindexptr</i>	Pass in the Dwarf_Gdbindex pointer of interest.
<i>dw_stringoffset</i>	Pass in the string offset returned by dwarf_gdbindex_symboltable_entry
<i>dw_string_ptr</i>	On success returns a pointer to the null-terminated string.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.11 dwarf_gdbindex_symboltable_array()

```
DW_API int dwarf_gdbindex_symboltable_array (
    Dwarf_Gdbindex dw_gdbindexptr,
    Dwarf_Unsigned * dw_symtab_list_length,
    Dwarf_Error * dw_error )
```

Get access to the symboltable array.

Parameters

<i>dw_gdbindexptr</i>	Pass in the Dwarf_Gdbindex pointer of interest.
<i>dw_symtab_list_length</i>	On success returns the number of entries in the symbol table
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.12 dwarf_gdbindex_symboltable_entry()

```
DW_API int dwarf_gdbindex_symboltable_entry (
    Dwarf_Gdbindex dw_gdbindexptr,
```

```

Dwarf_Unsigned dw_entryindex,
Dwarf_Unsigned * dw_string_offset,
Dwarf_Unsigned * dw_cu_vector_offset,
Dwarf_Error * dw_error )

```

Access individual symtab entry.

Parameters

<i>dw_gdbindexptr</i>	Pass in the Dwarf_Gdbindex pointer of interest.
<i>dw_entryindex</i>	Pass in a valid index in the range 0 through dw_symtab_list_length-1. If the value is greater than dw_symtab_list_length-1 the function returns DW_DLV_NO_ENTRY;
<i>dw_string_offset</i>	On success returns the string offset in the appropriate string section.
<i>dw_cu_vector_offset</i>	On success returns the CU vector offset.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.13 dwarf_gdbindex_types_culist_array()

```

DW_API int dwarf_gdbindex_types_culist_array (
    Dwarf_Gdbindex dw_gdbindexptr,
    Dwarf_Unsigned * dw_types_list_length,
    Dwarf_Error * dw_error )

```

Return the types culist array length.

Parameters

<i>dw_gdbindexptr</i>	Pass in the Dwarf_Gdbindex pointer of interest.
<i>dw_types_list_length</i>	On success returns the array length of the types cu list.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.29.2.14 dwarf_gdbindex_types_culist_entry()

```

DW_API int dwarf_gdbindex_types_culist_entry (
    Dwarf_Gdbindex dw_gdbindexptr,
    Dwarf_Unsigned dw_types_entryindex,
    Dwarf_Unsigned * dw_cu_offset,
    Dwarf_Unsigned * dw_tu_offset,
    Dwarf_Unsigned * dw_type_signature,
    Dwarf_Error * dw_error )

```

For a types CU entry in the list returns the offset and length.

Parameters

<i>dw_gdbindexptr</i>	Pass in the Dwarf_Gdbindex pointer of interest.
<i>dw_types_entryindex</i>	Pass in a number from 0 through dw_list_length-1. If the value is greater than dw_list_length-1 the function returns DW_DLV_NO_ENTRY.
<i>dw_cu_offset</i>	On success returns the types CU offset for this list entry.
<i>dw_tu_offset</i>	On success returns the tu offset for this list entry.
<i>dw_type_signature</i>	On success returns the type unit offset for this entry if the type has a signature.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.30 Fast Access to Split Dwarf (Debug Fission)

Functions

- DW_API int [dwarf_get_xu_index_header](#) (Dwarf_Debug dw_dbg, const char *dw_section_type, Dwarf_Xu_Index_Header *dw_xuhdr, Dwarf_Unsigned *dw_version_number, Dwarf_Unsigned *dw_section_count, Dwarf_Unsigned *dw_units_count, Dwarf_Unsigned *dw_hash_slots_count, const char **dw_sect_name, Dwarf_Error *dw_error)
Access a .debug_cu_index or dw_tu_index section.
- DW_API void [dwarf_dealloc_xu_header](#) (Dwarf_Xu_Index_Header dw_xuhdr)
Dealloc (free) memory associated with dw_xuhdr.
- DW_API int [dwarf_get_xu_index_section_type](#) (Dwarf_Xu_Index_Header dw_xuhdr, const char **dw_type_name, const char **dw_sectionname, Dwarf_Error *dw_error)
Return basic information about a Dwarf_Xu_Index_Header.
- DW_API int [dwarf_get_xu_hash_entry](#) (Dwarf_Xu_Index_Header dw_xuhdr, Dwarf_Unsigned dw_index, Dwarf_Sig8 *dw_hash_value, Dwarf_Unsigned *dw_index_to_sections, Dwarf_Error *dw_error)
Get a Hash Entry.
- DW_API int [dwarf_get_xu_section_names](#) (Dwarf_Xu_Index_Header dw_xuhdr, Dwarf_Unsigned dw_column_index, Dwarf_Unsigned *dw_SECT_number, const char **dw_SECT_name, Dwarf_Error *dw_error)
get DW_SECT value for a column.
- DW_API int [dwarf_get_xu_section_offset](#) (Dwarf_Xu_Index_Header dw_xuhdr, Dwarf_Unsigned dw_row_index, Dwarf_Unsigned dw_column_index, Dwarf_Unsigned *dw_sec_offset, Dwarf_Unsigned *dw_sec_size, Dwarf_Error *dw_error)
Get row data (section data) for a row and column.
- DW_API int [dwarf_get_debugfission_for_die](#) (Dwarf_Die dw_die, Dwarf_Debug_Fission_Per_CU *dw_percu_out, Dwarf_Error *dw_error)
Get debugfission data for a Dwarf_Die.
- DW_API int [dwarf_get_debugfission_for_key](#) (Dwarf_Debug dw_dbg, Dwarf_Sig8 *dw_hash_sig, const char *dw_cu_type, Dwarf_Debug_Fission_Per_CU *dw_percu_out, Dwarf_Error *dw_error)
Given a hash signature find per-cu Fission data.

9.30.1 Detailed Description

9.30.2 Function Documentation

9.30.2.1 dwarf_dealloc_xu_header()

```
DW_API void dwarf_dealloc_xu_header (
    Dwarf_Xu_Index_Header dw_xuhdr )
```

Dealloc (free) memory associated with dw_xuhdr.

Should be named dwarf_dealloc_xuhdr instead.

Parameters

<i>dw_xuhdr</i>	Dealloc (free) all associated memory. The caller should zero the passed in value on return as it is then a stale value.
-----------------	---

9.30.2.2 dwarf_get_debugfission_for_die()

```
DW_API int dwarf_get_debugfission_for_die (
    Dwarf_Die dw_die,
    Dwarf_Debug_Fission_Per_CU * dw_percu_out,
    Dwarf_Error * dw_error )
```

Get debugfission data for a Dwarf_Die.

For any Dwarf_Die in a compilation unit, return the debug fission table data through dw_percu_out. Usually applications will pass in the CU die. Calling code should zero all of the struct [Dwarf_Debug_Fission_Per_CU_s](#) before calling this. If there is no debugfission data this returns DW_DLV_NO_ENTRY (only .dwp objects have debugfission data)

Parameters

<i>dw_die</i>	Pass in a Dwarf_Die pointer, Usually pass in a CU DIE pointer.
<i>dw_percu_out</i>	Pass in a pointer to a zeroed structure. On success the function fills in the structure.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.30.2.3 dwarf_get_debugfission_for_key()

```
DW_API int dwarf_get_debugfission_for_key (
    Dwarf_Debug dw_dbg,
    Dwarf_Sig8 * dw_hash_sig,
    const char * dw_cu_type,
```

```
Dwarf_Debug_Fission_Per_CU * dw_percu_out,  
Dwarf_Error * dw_error )
```

Given a hash signature find per-cu Fission data.

Parameters

<i>dw_dbg</i>	Pass in the Dwarf_Debug of interest.
<i>dw_hash_sig</i>	Pass in a pointer to a Dwarf_Sig8 containing a hash value of interest.
<i>dw_cu_type</i>	Pass in the type, a string. Either "cu" or "tu".
<i>dw_percu_out</i>	Pass in a pointer to a zeroed structure. On success the function fills in the structure.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.30.2.4 dwarf_get_xu_hash_entry()

```
DW_API int dwarf_get_xu_hash_entry (
    Dwarf_Xu_Index_Header dw_xuhdr,
    Dwarf_Unsigned dw_index,
    Dwarf_Sig8 * dw_hash_value,
    Dwarf_Unsigned * dw_index_to_sections,
    Dwarf_Error * dw_error )
```

Get a Hash Entry.

See also

[examplez/x](#)

Parameters

<i>dw_xuhdr</i>	Pass in an open header pointer.
<i>dw_index</i>	Pass in the index of the entry you wish. Valid index values are 0 through S-1 . If the <i>dw_index</i> passed in is outside the valid range the functionj
<i>dw_hash_value</i>	Pass in a pointer to a Dwarf_Sig8. On success the hash struct is filled in with the 8 byte hash value.
<i>dw_index_to_sections</i>	On success returns the offset/size table index for this hash entry.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK on success. If the *dw_index* passed in is outside the valid range the function it returns DW_DLV_NO_ENTRY (before version 0.7.0 it returned DW_DLV_ERROR, though nothing mentioned that). In case of error it returns DW_DLV_ERROR. If *dw_error* is non-null returns error details through *dw_error* (the usual error behavior).

9.30.2.5 dwarf_get_xu_index_header()

```
DW_API int dwarf_get_xu_index_header (
    Dwarf_Debug dw_dbg,
```

```

const char * dw_section_type,
Dwarf_Xu_Index_Header * dw_xuhdr,
Dwarf_Unsigned * dw_version_number,
Dwarf_Unsigned * dw_section_count,
Dwarf_Unsigned * dw_units_count,
Dwarf_Unsigned * dw_hash_slots_count,
const char ** dw_sect_name,
Dwarf_Error * dw_error )

```

Access a .debug_cu_index or dw_tu_index section.

These sections are in a DWARF5 package file, a file normally named with the .dwo or .dwp extension.. See DWARF5 section 7.3.5.3 Format of the CU and TU Index Sections.

Parameters

<i>dw_dbg</i>	Pass in the Dwarf_Debug of interest
<i>dw_section_type</i>	Pass in a pointer to either "cu" or "tu".
<i>dw_xuhdr</i>	On success, returns a pointer usable in further calls.
<i>dw_version_number</i>	On success returns five.
<i>dw_section_count</i>	On success returns the number of entries in the table of section counts. Referred to as N .
<i>dw_units_count</i>	On success returns the number of compilation units or type units in the index. Referred to as U .
<i>dw_hash_slots_count</i>	On success returns the number of slots in the hash table. Referred to as S .
<i>dw_sect_name</i>	On success returns a pointer to the name of the section. Do not free/dealloc the returned pointer.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc. Returns DW_DLV_NO_ENTRY if the section requested is not present.

9.30.2.6 dwarf_get_xu_index_section_type()

```

DW_API int dwarf_get_xu_index_section_type (
    Dwarf_Xu_Index_Header dw_xuhdr,
    const char ** dw_typename,
    const char ** dw_sectionname,
    Dwarf_Error * dw_error )

```

Return basic information about a Dwarf_Xu_Index_Header.

Parameters

<i>dw_xuhdr</i>	Pass in an open header pointer.
<i>dw_typename</i>	On success returns a pointer to the immutable string "tu" or "cu". Do not free.
<i>dw_sectionname</i>	On success returns a pointer to the section name in the object file. Do not free.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.30.2.7 dwarf_get_xu_section_names()

```
DW_API int dwarf_get_xu_section_names (
    Dwarf_Xu_Index_Header dw_xuhdr,
    Dwarf_Unsigned dw_column_index,
    Dwarf_Unsigned * dw_SECT_number,
    const char ** dw_SECT_name,
    Dwarf_Error * dw_error )
```

get DW_SECT value for a column.

See also

[Reading Split Dwarf \(Debug Fission\) data](#)

Parameters

<i>dw_xuhdr</i>	Pass in an open header pointer.
<i>dw_column_index</i>	The section names are in row zero of the table so we do not mention the row number at all. Pass in the column of the entry you wish. Valid <i>dw_column_index</i> values are 0 through N-1 .
<i>dw_SECT_number</i>	On success returns DW_SECT_INFO or other section id as appears in <i>dw_column_index</i> .
<i>dw_SECT_name</i>	On success returns a pointer to the string with the section name.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.30.2.8 dwarf_get_xu_section_offset()

```
DW_API int dwarf_get_xu_section_offset (
    Dwarf_Xu_Index_Header dw_xuhdr,
    Dwarf_Unsigned dw_row_index,
    Dwarf_Unsigned dw_column_index,
    Dwarf_Unsigned * dw_sec_offset,
    Dwarf_Unsigned * dw_sec_size,
    Dwarf_Error * dw_error )
```

Get row data (section data) for a row and column.

The section offset represents a base offset for the section the row data refers to. DWARF6 Section 7.3.5.3 page 193.

Parameters

<i>dw_xuhdr</i>	Pass in an open header pointer.
<i>dw_row_index</i>	Pass in a row number , 1 through U
<i>dw_column_index</i>	Pass in a column number , 0 through N-1
<i>dw_sec_offset</i>	On success returns the section offset of the section whose name <code>dwarf_get_xu_section_names</code> returns.
<i>dw_sec_size</i>	On success returns the section size of the section whose name <code>dwarf_get_xu_section_names</code> returns. If the returned section size is zero then this column makes no contribution to the dwp object file and the <code>dw_sec_size</code> and <code>dw_sec_offset</code> should be ignored.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.31 Access GNU .gnu_debuglink, build-id.

Functions

- DW_API int `dwarf_gnu_debuglink` (`Dwarf_Debug` dw_dbg, char **dw_debuglink_path_returned, unsigned char **dw_crc_returned, char **dw_debuglink_fullpath_returned, unsigned int *dw_debuglink_path_length_returned, unsigned int *dw_buildid_type_returned, char **dw_buildid_owner_name_returned, unsigned char **dw_buildid_returned, unsigned int *dw_buildid_length_returned, char ***dw_paths_returned, unsigned int *dw_paths_length_returned, `Dwarf_Error` *dw_error)
Find a separated DWARF object file.
- DW_API int `dwarf_suppress_debuglink_crc` (int dw_suppress)
Suppressing crc calculations.
- DW_API int `dwarf_add_debuglink_global_path` (`Dwarf_Debug` dw_dbg, const char *dw_pathname, `Dwarf_Error` *dw_error)
Adding debuglink global paths.
- DW_API int `dwarf_crc32` (`Dwarf_Debug` dw_dbg, unsigned char *dw_crcbuf, `Dwarf_Error` *dw_error)
Crc32 used for debuglink crc calculation.
- DW_API unsigned int `dwarf_basic_crc32` (const unsigned char *dw_buf, unsigned long dw_len, unsigned int dw_init)
Public interface to the real crc calculation.

9.31.1 Detailed Description

When DWARF sections are in a different object than the executable or a normal shared object. The special GNU section provides a way to name the object file with DWARF.

libdwarf will attempt to use this data to find the object file with DWARF.

Has nothing to do with split-dwarf/debug-fission.

9.31.2 Function Documentation

9.31.2.1 dwarf_add_debuglink_global_path()

```
DW_API int dwarf_add_debuglink_global_path (
    Dwarf_Debug dw_dbg,
    const char * dw_pathname,
    Dwarf_Error * dw_error )
```

Adding debuglink global paths.

Used inside src/bin/dwarfexample/dwdebuglink.c so we can show all that is going on. The following has the explanation for how debuglink and global paths interact:

See also

<https://sourceware.org/gdb/onlinedocs/gdb/Separate-Debug-Files.html>

Parameters

<i>dw_dbg</i>	Pass in the Dwarf_Debug of interest.
<i>dw_pathname</i>	Pass in a pathname to add to the list of global paths used by debuglink.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.31.2.2 dwarf_basic_crc32()

```
DW_API unsigned int dwarf_basic_crc32 (
    const unsigned char * dw_buf,
    unsigned long dw_len,
    unsigned int dw_init )
```

Public interface to the real crc calculation.

It is unlikely this is useful. The calculation will not produce a return matching that of Linux/Macos if the compiler implements unsigned int or signed int as 16 bits long.

The caller must guarantee that dw_buf is non-null and pointing to dw_len bytes of readable memory. If dw_buf is NULL then 0 is immediately returned and there is no indication of error.

Parameters

<i>dw_buf</i>	Pass in a pointer to some bytes on which the crc calculation as done in debuglink is to be done.
<i>dw_len</i>	Pass in the length in bytes of dw_buf.
<i>dw_init</i>	Pass in the initial 32 bit value, zero is the right choice.

Returns

Returns an int (assumed 32 bits int!) with the calculated crc.

9.31.2.3 dwarf_crc32()

```
DW_API int dwarf_crc32 (
    Dwarf_Debug dw_dbg,
    unsigned char * dw_crcbuf,
    Dwarf_Error * dw_error )
```

Crc32 used for debuglink crc calculation.

Caller passes pointer to array of 4 unsigned char provided by the caller and if this returns DW_DLV_OK that array is filled in.

Callers must guarantee dw_crcbuf points to at least 4 bytes of writable memory. Passing in a null dw_crcbuf results in an immediate return of DW_DLV_NO_ENTRY and the pointer is not used.

Parameters

<i>dw_dbg</i>	Pass in an open dw_dbg. When you attempted to open it, and it succeeded then pass the it via the Dwarf_Debug The function reads the file into memory and performs a crc calculation.
<i>dw_crcbuf</i>	Pass in a pointer to a 4 byte area to hold the returned crc, on success the function puts the 4 bytes there.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.31.2.4 dwarf_gnu_debuglink()

```
DW_API int dwarf_gnu_debuglink (
    Dwarf_Debug dw_dbg,
    char ** dw_debuglink_path_returned,
    unsigned char ** dw_crc_returned,
    char ** dw_debuglink_fullpath_returned,
    unsigned int * dw_debuglink_path_length_returned,
    unsigned int * dw_buildid_type_returned,
    char ** dw_buildid_owner_name_returned,
    unsigned char ** dw_buildid_returned,
    unsigned int * dw_buildid_length_returned,
    char *** dw_paths_returned,
    unsigned int * dw_paths_length_returned,
    Dwarf_Error * dw_error )
```

Find a separated DWARF object file.

.gnu_debuglink and/or the section .note.gnu.build-id.

Unless something is odd and you want to know details of the two sections you will not need this function.

See also

<https://sourceware.org/gdb/onlinedocs/gdb/Separate-Debug-Files.html>
[Using GNU debuglink data](#)

If no debuglink then name_returned, crc_returned and debuglink_path_returned will get set 0 through the pointers.

If no .note.gnu.build-id then buildid_length_returned, and buildid_returned will be set 0 through the pointers.

In most cases output arguments can be passed as zero and the function will simply not return data through such arguments. Useful if you only care about some of the data potentially returned.

If dw_debuglink_fullpath_returned is set by the call the space allocated must be freed by the caller with free(dw_↵ debuglink_fullpath_returned).

if dw_debuglink_paths_returned is set by the call the space allocated must be free by the caller with free(dw_↵ debuglink_paths_returned).

[dwarf_finish\(\)](#) will not free strings dw_debuglink_fullpath_returned or dw_debuglink_paths_returned.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_debuglink_path_returned</i>	On success returns a pointer to a path in the debuglink section. Do not free!
<i>dw_crc_returned</i>	On success returns a pointer to a 4 byte area through the pointer.
<i>dw_debuglink_fullpath_returned</i>	On success returns a pointer to a full path computed from debuglink data of a correct path to a file with DWARF sections. Free this string when no longer of interest.
<i>dw_debuglink_path_length_returned</i>	On success returns the strlen() of dw_debuglink_fullpath_returned .
<i>dw_buildid_type_returned</i>	On success returns a pointer to integer with a type code. See the buildid definition.
<i>dw_buildid_owner_name_returned</i>	On success returns a pointer to the owner name from the buildid section. Do not free this.
<i>dw_buildid_returned</i>	On success returns a pointer to a sequence of bytes containing the buildid.
<i>dw_buildid_length_returned</i>	On success this is set to the length of the set of bytes pointed to by dw_buildid_returned .
<i>dw_paths_returned</i>	On success sets a pointer to an array of pointers to strings, each with a global path. These strings must be freed by the caller, dwarf_finish() will not free these strings. Call free(dw_paths_returned).
<i>dw_paths_length_returned</i>	On success returns the length of the array of string pointers dw_paths_returned points at.
<i>dw_error</i>	The usual pointer to return error details.

Returns

Returns DW_DLV_OK etc.

9.31.2.5 dwarf_suppress_debuglink_crc()

```
DW_API int dwarf_suppress_debuglink_crc (
    int dw_suppress )
```

Suppressing crc calculations.

The `.gnu_debuglink` section contains a compilation-system created crc (4 byte) value. If `dwarf_init_path[_dl]()` is called such a section can result in the reader/consumer calculating the crc value of a different object file. Which on a large object file could seem slow. See https://en.wikipedia.org/wiki/Cyclic_redundancy_check

When one is confident that any `debug_link` file found is the appropriate one one can call `dwarf_suppress_debuglink_crc` with a non-zero argument and any `dwarf_init_path[_dl]` call will skip debuglink crc calculations and just assume the crc would match whenever it applies. This is a global flag, applies to all Dwarf_Debug opened after the call in the program execution.

Does not apply to the `.note.gnu.buildid` section as that section never implies the reader/consumer needs to do a crc calculation.

Parameters

<code>dw_suppress</code>	Pass in 1 to suppress future calculation of crc values to verify a debuglink target is correct. So use only when you know this is safe. Pass in 0 to ensure future <code>dwarf_init_path[_dl]</code> calls compute debuglink CRC values as required.
--------------------------	--

Returns

Returns the previous value of the global flag.

[Details on separate DWARF object access](#)

9.32 Harmless Error recording

Macros

- `#define DW_HARMLESS_ERROR_CIRCULAR_LIST_DEFAULT_SIZE 4`
Default size of the libdwarf-internal circular list.

Functions

- DW_API int `dwarf_get_harmless_error_list` (Dwarf_Debug dw_dbg, unsigned int dw_count, const char **dw_errmsg_ptrs_array, unsigned int *dw_newerr_count)
Get the harmless error count and content.
- DW_API unsigned int `dwarf_set_harmless_error_list_size` (Dwarf_Debug dw_dbg, unsigned int dw_maxcount)
The size of the circular list of strings libdwarf holds internally may be set and reset as needed. If it is shortened excess messages are simply dropped. It returns the previous size. If zero passed in the size is unchanged and it simply returns the current size.
- DW_API int `dwarf_set_harmless_errors_enabled` (Dwarf_Debug dw_dbg, int dw_v)
Enable or disable libdwarf tracking of harmless errors. Harmless errors are used by tools like dwarfdump. Disabling harmless errors can improve performance by avoiding string copies. Defaults to enabled.
- DW_API void `dwarf_insert_harmless_error` (Dwarf_Debug dw_dbg, char *dw_newerror)
Harmless Error Insertion is only for testing.

9.32.1 Detailed Description

The harmless error list is a fixed size circular buffer of errors we note but which do not stop us from processing the object. Created so dwarfdump or other tools can report such inconsequential errors without causing anything to stop early.

You can change the list size from the default of DW_HARMLESS_ERROR_CIRCULAR_LIST_DEFAULT_SIZE at any time for a Dwarf_Debug dbg.

Harmless error data is dealloc'd by `dwarf_finish()`.

9.32.2 Function Documentation

9.32.2.1 dwarf_get_harmless_error_list()

```
DW_API int dwarf_get_harmless_error_list (
    Dwarf_Debug dw_dbg,
    unsigned int dw_count,
    const char ** dw_errmsg_ptrs_array,
    unsigned int * dw_newerr_count )
```

Get the harmless error count and content.

User code supplies size of array of pointers `dw_errmsg_ptrs_array` in count and the array of pointers (the pointers themselves need not be initialized). The pointers returned in the array of pointers are invalidated by ANY call to libdwarf. Use them before making another libdwarf call! The array of string pointers passed in always has a final null pointer, so if there are N pointers and M actual strings, then MIN(M,N-1) pointers are set to point to error strings. The array of pointers to strings always terminates with a NULL pointer. Do not free the strings. Every string is null-terminated.

Each call empties the error list (discarding all current entries). and fills in your array

Parameters

<i>dw_dbg</i>	The applicable Dwarf_Debug.
<i>dw_count</i>	The number of string buffers. If count is passed as zero no elements of the array are touched.
<i>dw_errmsg_ptrs_array</i>	A pointer to a user-created array of pointer to const char.
<i>dw_newerr_count</i>	If non-NULL the count of harmless errors pointers since the last call is returned through the pointer. If dw_count is greater than zero the first dw_count of the pointers in the user-created array point to null-terminated strings. Do not free the strings. print or copy the strings before any other libdwarf call.

Returns

Returns DW_DLV_NO_ENTRY if no harmless errors were noted so far. Returns DW_DLV_OK if there are harmless errors. Never returns DW_DLV_ERROR.

If DW_DLV_NO_ENTRY is returned none of the arguments other than dw_dbg are touched or used.

9.32.2.2 dwarf_insert_harmless_error()

```
DW_API void dwarf_insert_harmless_error (
    Dwarf_Debug dw_dbg,
    char * dw_newerror )
```

Harmless Error Insertion is only for testing.

Useful for testing the harmless error mechanism.

Parameters

<i>dw_dbg</i>	Pass in an open Dwarf_Debug
<i>dw_newerror</i>	Pass in a string whose content the function inserts as a harmless error (which dwarf_get_harmless_error_list will retrieve).

9.32.2.3 dwarf_set_harmless_error_list_size()

```
DW_API unsigned int dwarf_set_harmless_error_list_size (
    Dwarf_Debug dw_dbg,
    unsigned int dw_maxcount )
```

The size of the circular list of strings libdwarf holds internally may be set and reset as needed. If it is shortened excess messages are simply dropped. It returns the previous size. If zero passed in the size is unchanged and it simply returns the current size.

Parameters

<i>dw_dbg</i>	The applicable Dwarf_Debug.
<i>dw_maxcount</i>	Set the new internal buffer count to a number greater than zero.

Returns

returns the current size of the internal circular buffer if dw_maxcount is zero. If dw_maxcount is greater than zero the internal array is adjusted to hold that many and the previous number of harmless errors possible in the circular buffer is returned.

9.32.2.4 dwarf_set_harmless_errors_enabled()

```
DW_API int dwarf_set_harmless_errors_enabled (
    Dwarf_Debug dw_dbg,
    int dw_v )
```

Enable or disable libdwarf tracking of harmless errors. Harmless errors are used by tools like dwarfdump. Disabling harmless errors can improve performance by avoiding string copies. Defaults to enabled.

Parameters

<i>dw_dbg</i>	Pass in an open Dwarf_Debug
<i>dw_v</i>	If zero passed in, harmless errors will not be tracked and libdwarf will run somewhat faster. If non-zero passed in libdwarf will resume or continue tracking harmless errors.

Returns

Returns the previous version of the flag.

9.33 Names DW_TAG_member etc as strings

Functions

- DW_API int **dwarf_get_ACCESS_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_ACCESS_name
- DW_API int **dwarf_get_ADDR_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_ADDR_name
- DW_API int **dwarf_get_AT_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_AT_name
- DW_API int **dwarf_get_ATCF_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_AT_name
- DW_API int **dwarf_get_ATE_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_ATE_name
- DW_API int **dwarf_get_CC_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_CC_name
- DW_API int **dwarf_get_CFA_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_CFA_name
- DW_API int **dwarf_get_children_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_children_name - historic misspelling.
- DW_API int **dwarf_get_CHILDREN_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_CHILDREN_name
- DW_API int **dwarf_get_DEFAULTED_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_DEFAULTED_name
- DW_API int **dwarf_get_DS_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_DS_name
- DW_API int **dwarf_get_DSC_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_DSC_name
- DW_API int **dwarf_get_GNUIKIND_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_GNUIKIND_name - libdwarf invention
- DW_API int **dwarf_get_EH_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_EH_name
- DW_API int **dwarf_get_END_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_END_name
- DW_API int **dwarf_get_FORM_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_FORM_name
- DW_API int **dwarf_get_FRAME_name** (unsigned int dw_val_in, const char **dw_s_out)
This is a set of register names.
- DW_API int **dwarf_get_GNUIVIS_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_GNUIVIS_name - a libdwarf invention
- DW_API int **dwarf_get_ID_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_ID_name
- DW_API int **dwarf_get_IDX_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_IDX_name
- DW_API int **dwarf_get_INL_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_INL_name

- DW_API int **dwarf_get_ISA_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_ISA_name
- DW_API int **dwarf_get_LANG_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_LANG_name
- DW_API int **dwarf_get_LLE_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_LLE_name
- DW_API int **dwarf_get_LLEX_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_LLEX_name - a GNU extension.
- DW_API int **dwarf_get_LNAME_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_LNAME
- DW_API int **dwarf_get_LNCT_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_LNCT_name
- DW_API int **dwarf_get_LNE_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_LNE_name
- DW_API int **dwarf_get_LNS_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_LNS_name
- DW_API int **dwarf_get_MACINFO_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_MACINFO_name
- DW_API int **dwarf_get_MACRO_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_MACRO_name
- DW_API int **dwarf_get_OP_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_OP_name
- DW_API int **dwarf_get_ORD_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_ORD_name
- DW_API int **dwarf_get_RLE_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_RLE_name
- DW_API int **dwarf_get_SECT_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_SECT_name
- DW_API int **dwarf_get_TAG_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_TAG_name
- DW_API int **dwarf_get_UT_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_UT_name
- DW_API int **dwarf_get_VIRTUALITY_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_VIRTUALITY_name
- DW_API int **dwarf_get_VIS_name** (unsigned int dw_val_in, const char **dw_s_out)
dwarf_get_VIS_name
- DW_API int **dwarf_get_FORM_CLASS_name** (enum [Dwarf_Form_Class](#) dw_fc, const char **dw_s_out)
dwarf_get_FORM_CLASS_name is for a libdwarf extension. Not defined by the DWARF standard though the concept is defined in the standard. It seemed essential to invent it for libdwarf to report correctly.

9.33.1 Detailed Description

Given a value you know is one of a particular name category in DWARF2 or later, call the appropriate function and on finding the name it returns DW_DLV_OK and sets the identifier for the value through a pointer. On success these functions return the string corresponding to **dw_val_in** passed in through the pointer **dw_s_out** and the value returned is DW_DLV_OK.

The strings returned on success are in static storage and must not be freed.

These functions are generated from information in [dwarf.h](#), not hand coded functions.

If DW_DLV_NO_ENTRY is returned the **dw_val_in** is not known and ***s_out** is not set. This is unusual.

DW_DLV_ERROR is never returned.

The example referred to offers the suggested way to use functions like these.

See also

[Retrieving tag,attribute,etc names](#)

9.33.2 Function Documentation

9.33.2.1 dwarf_get_EH_name()

```
DW_API int dwarf_get_EH_name (
    unsigned int dw_val_in,
    const char ** dw_s_out )
```

dwarf_get_EH_name

So we can report this GNU extension sensibly.

9.33.2.2 dwarf_get_FORM_CLASS_name()

```
DW_API int dwarf_get_FORM_CLASS_name (
    enum Dwarf_Form_Class dw_fc,
    const char ** dw_s_out )
```

dwarf_get_FORM_CLASS_name is for a libdwarf extension. Not defined by the DWARF standard though the concept is defined in the standard. It seemed essential to invent it for libdwarf to report correctly.

See DWARF5 Table 2.3, Classes of Attribute Value page 23. Earlier DWARF versions have a similar table.

9.33.2.3 dwarf_get_FRAME_name()

```
DW_API int dwarf_get_FRAME_name (
    unsigned int dw_val_in,
    const char ** dw_s_out )
```

This is a set of register names.

The set of register names is unlikely to match your register set, but perhaps this is better than no name.

9.33.2.4 dwarf_get_GNUKIND_name()

```
DW_API int dwarf_get_GNUKIND_name (
    unsigned int dw_val_in,
    const char ** dw_s_out )
```

dwarf_get_GNUKIND_name - libdwarf invention

So we can report things GNU extensions sensibly.

9.33.2.5 dwarf_get_GNUIVIS_name()

```
DW_API int dwarf_get_GNUIVIS_name (
    unsigned int dw_val_in,
    const char ** dw_s_out )
```

dwarf_get_GNUIVIS_name - a libdwarf invention

So we report a GNU extension sensibly.

9.33.2.6 dwarf_get_LLEX_name()

```
DW_API int dwarf_get_LLEX_name (
    unsigned int dw_val_in,
    const char ** dw_s_out )
```

dwarf_get_LLEX_name - a GNU extension.

The name is a libdwarf invention for the GNU extension. So we report a GNU extension sensibly.

9.33.2.7 dwarf_get_MACINFO_name()

```
DW_API int dwarf_get_MACINFO_name (
    unsigned int dw_val_in,
    const char ** dw_s_out )
```

dwarf_get_MACINFO_name

Used in DWARF2-DWARF4

9.33.2.8 dwarf_get_MACRO_name()

```
DW_API int dwarf_get_MACRO_name (
    unsigned int dw_val_in,
    const char ** dw_s_out )
```

dwarf_get_MACRO_name

Used in DWARF5

9.34 Object Sections Data

Functions

- DW_API int [dwarf_get_die_section_name](#) (Dwarf_Debug dw_dbg, Dwarf_Bool dw_is_info, const char **dw_sec_name, Dwarf_Error *dw_error)
Get the real name a DIE section.
- DW_API int [dwarf_get_die_section_name_b](#) (Dwarf_Die dw_die, const char **dw_sec_name, Dwarf_Error *dw_error)
Get the real name of a DIE section.
- DW_API int [dwarf_get_macro_section_name](#) (Dwarf_Debug dw_dbg, const char **dw_sec_name_out, Dwarf_Error *dw_err)
Get the real name of a .debug_macro section.
- DW_API int [dwarf_get_real_section_name](#) (Dwarf_Debug dw_dbg, const char *dw_std_section_name, const char **dw_actual_sec_name_out, Dwarf_Small *dw_marked_zcompressed, Dwarf_Small *dw_marked_zlib_compressed, Dwarf_Small *dw_marked_shf_compressed, Dwarf_Unsigned *dw_compressed_length, Dwarf_Unsigned *dw_uncompressed_length, Dwarf_Error *dw_error)
Get the real name of a section.
- DW_API int [dwarf_get_frame_section_name](#) (Dwarf_Debug dw_dbg, const char **dw_section_name_out, Dwarf_Error *dw_error)
Get .debug_frame section name.
- DW_API int [dwarf_get_frame_section_name_eh_gnu](#) (Dwarf_Debug dw_dbg, const char **dw_section_name_out, Dwarf_Error *dw_error)
Get GNU .eh_frame section name.
- DW_API int [dwarf_get_aranges_section_name](#) (Dwarf_Debug dw_dbg, const char **dw_section_name_out, Dwarf_Error *dw_error)
Get .debug_aranges section name The usual arguments.
- DW_API int [dwarf_get_ranges_section_name](#) (Dwarf_Debug dw_dbg, const char **dw_section_name_out, Dwarf_Error *dw_error)
Get .debug_ranges section name The usual arguments and return values.
- DW_API int [dwarf_get_offset_size](#) (Dwarf_Debug dw_dbg, Dwarf_Half *dw_offset_size, Dwarf_Error *dw_error)
Get offset size as defined by the object.
- DW_API int [dwarf_get_address_size](#) (Dwarf_Debug dw_dbg, Dwarf_Half *dw_addr_size, Dwarf_Error *dw_error)
Get the address size as defined by the object.
- DW_API int [dwarf_get_string_section_name](#) (Dwarf_Debug dw_dbg, const char **dw_section_name_out, Dwarf_Error *dw_error)
Get the string table section name The usual arguments and return values.
- DW_API int [dwarf_get_line_section_name](#) (Dwarf_Debug dw_dbg, const char **dw_section_name_out, Dwarf_Error *dw_error)
Get the line table section name The usual arguments and return values.
- DW_API int [dwarf_get_line_section_name_from_die](#) (Dwarf_Die dw_die, const char **dw_section_name_out, Dwarf_Error *dw_error)
Get the line table section name.
- DW_API int [dwarf_get_section_info_by_name_a](#) (Dwarf_Debug dw_dbg, const char *dw_section_name, Dwarf_Addr *dw_section_addr, Dwarf_Unsigned *dw_section_size, Dwarf_Unsigned *dw_section_flags, Dwarf_Unsigned *dw_section_offset, Dwarf_Error *dw_error)
Given a section name, get its size, address, etc.
- DW_API int [dwarf_get_section_info_by_name](#) (Dwarf_Debug dw_dbg, const char *dw_section_name, Dwarf_Addr *dw_section_addr, Dwarf_Unsigned *dw_section_size, Dwarf_Error *dw_error)
Given a section name, get its size and address.

- DW_API int [dwarf_get_section_info_by_index_a](#) (Dwarf_Debug dw_dbg, int dw_section_index, const char **dw_section_name, Dwarf_Addr *dw_section_addr, Dwarf_Unsigned *dw_section_size, Dwarf_Unsigned *dw_section_flags, Dwarf_Unsigned *dw_section_offset, Dwarf_Error *dw_error)
Given a section index, get its size and address, etc.
- DW_API int [dwarf_get_section_info_by_index](#) (Dwarf_Debug dw_dbg, int dw_section_index, const char **dw_section_name, Dwarf_Addr *dw_section_addr, Dwarf_Unsigned *dw_section_size, Dwarf_Error *dw_error)
Given a section index, get its size and address.
- DW_API int [dwarf_machine_architecture_a](#) (Dwarf_Debug dw_dbg, Dwarf_Small *dw_ftype, Dwarf_Small *dw_obj_pointersize, Dwarf_Bool *dw_obj_is_big_endian, Dwarf_Unsigned *dw_obj_machine, Dwarf_Unsigned *dw_obj_type, Dwarf_Unsigned *dw_obj_flags, Dwarf_Small *dw_path_source, Dwarf_Unsigned *dw_ub_offset, Dwarf_Unsigned *dw_ub_count, Dwarf_Unsigned *dw_ub_index, Dwarf_Unsigned *dw_comdat_groupnumber)
Get basic object information from Dwarf_Debug.
- DW_API int [dwarf_machine_architecture](#) (Dwarf_Debug dw_dbg, Dwarf_Small *dw_ftype, Dwarf_Small *dw_obj_pointersize, Dwarf_Bool *dw_obj_is_big_endian, Dwarf_Unsigned *dw_obj_machine, Dwarf_Unsigned *dw_obj_flags, Dwarf_Small *dw_path_source, Dwarf_Unsigned *dw_ub_offset, Dwarf_Unsigned *dw_ub_count, Dwarf_Unsigned *dw_ub_index, Dwarf_Unsigned *dw_comdat_groupnumber)
Get basic object information original version.
- DW_API Dwarf_Unsigned [dwarf_get_section_count](#) (Dwarf_Debug dw_dbg)
Get section count (of object file sections).
- DW_API int [dwarf_get_section_max_offsets_d](#) (Dwarf_Debug dw_dbg, Dwarf_Unsigned *dw_debug_info_size, Dwarf_Unsigned *dw_debug_abbrev_size, Dwarf_Unsigned *dw_debug_line_size, Dwarf_Unsigned *dw_debug_loc_size, Dwarf_Unsigned *dw_debug_aranges_size, Dwarf_Unsigned *dw_debug_macinfo_size, Dwarf_Unsigned *dw_debug_pubnames_size, Dwarf_Unsigned *dw_debug_str_size, Dwarf_Unsigned *dw_debug_frame_size, Dwarf_Unsigned *dw_debug_ranges_size, Dwarf_Unsigned *dw_debug_pubtypes_size, Dwarf_Unsigned *dw_debug_types_size, Dwarf_Unsigned *dw_debug_macro_size, Dwarf_Unsigned *dw_debug_str_offsets_size, Dwarf_Unsigned *dw_debug_sup_size, Dwarf_Unsigned *dw_debug_cu_index_size, Dwarf_Unsigned *dw_debug_tu_index_size, Dwarf_Unsigned *dw_debug_names_size, Dwarf_Unsigned *dw_debug_loclists_size, Dwarf_Unsigned *dw_debug_rnglists_size)
Get section sizes for many sections.

9.34.1 Detailed Description

These functions are not often used. They give access to section- and objectfile-related information, and that sort of information is not generally needed to understand DWARF content..

Section name access. Because names sections such as .debug_info might end with .dwo or be .zdebug or might not.

String pointers returned via these functions must not be freed, the strings are statically declared.

For non-Elf the name reported will be as if it were Elf sections. For example, not the names Macos puts in its object sections (which the Macos reader translates).

These calls returning selected object header {machine architecture, flags} and section {offset, flags} data are not of interest to most library callers: [dwarf_machine_architecture\(\)](#), [dwarf_get_section_info_by_index_a\(\)](#), and [dwarf_get_section_info_by_name_a\(\)](#).

The simple calls will not be documented in full detail here.

9.34.2 Function Documentation

9.34.2.1 dwarf_get_address_size()

```
DW_API int dwarf_get_address_size (
    Dwarf_Debug dw_dbg,
    Dwarf_Half * dw_addr_size,
    Dwarf_Error * dw_error )
```

Get the address size as defined by the object.

This is not from DWARF information, it is from object file headers.

9.34.2.2 dwarf_get_die_section_name()

```
DW_API int dwarf_get_die_section_name (
    Dwarf_Debug dw_dbg,
    Dwarf_Bool dw_is_info,
    const char ** dw_sec_name,
    Dwarf_Error * dw_error )
```

Get the real name a DIE section.

dw_is_info

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest
<i>dw_is_info</i>	We do not pass in a DIE, so we have to pass in TRUE for for .debug_info, or if DWARF4 .debug_types pass in FALSE.
<i>dw_sec_name</i>	On success returns a pointer to the actual section name in the object file. Do not free the string.
<i>dw_error</i>	The usual error argument to report error details.

Returns

DW_DLV_OK etc.

9.34.2.3 dwarf_get_die_section_name_b()

```
DW_API int dwarf_get_die_section_name_b (
    Dwarf_Die dw_die,
    const char ** dw_sec_name,
    Dwarf_Error * dw_error )
```

Get the real name of a DIE section.

The same as **dwarf_get_die_section_name** except we have a DIE so do not need **dw_is_info** as a argument.

9.34.2.4 dwarf_get_frame_section_name()

```
DW_API int dwarf_get_frame_section_name (
    Dwarf_Debug dw_dbg,
    const char ** dw_section_name_out,
    Dwarf_Error * dw_error )
```

Get .debug_frame section name.

Returns

returns DW_DLV_OK if the .debug_frame exists

9.34.2.5 dwarf_get_frame_section_name_eh_gnu()

```
DW_API int dwarf_get_frame_section_name_eh_gnu (
    Dwarf_Debug dw_dbg,
    const char ** dw_section_name_out,
    Dwarf_Error * dw_error )
```

Get GNU .eh_frame section name.

Returns

Returns DW_DLV_OK if the .debug_frame is present Returns DW_DLV_NO_ENTRY if it is not present.

9.34.2.6 dwarf_get_line_section_name_from_die()

```
DW_API int dwarf_get_line_section_name_from_die (
    Dwarf_Die dw_die,
    const char ** dw_section_name_out,
    Dwarf_Error * dw_error )
```

Get the line table section name.

Parameters

<i>dw_die</i>	Pass in a Dwarf_Die pointer.
<i>dw_section_name_out</i>	On success returns the section name, usually some .debug_info* name but in DWARF4 could be a .debug_types* name.
<i>dw_error</i>	On error returns the usual error pointer.

Returns

Returns DW_DLV_OK etc.

9.34.2.7 dwarf_get_offset_size()

```
DW_API int dwarf_get_offset_size (
    Dwarf_Debug dw_dbg,
```

```
Dwarf_Half * dw_offset_size,
Dwarf_Error * dw_error )
```

Get offset size as defined by the object.

This is not from DWARF information, it is from object file headers.

9.34.2.8 dwarf_get_real_section_name()

```
DW_API int dwarf_get_real_section_name (
    Dwarf_Debug dw_dbg,
    const char * dw_std_section_name,
    const char ** dw_actual_sec_name_out,
    Dwarf_Small * dw_marked_zcompressed,
    Dwarf_Small * dw_marked_zlib_compressed,
    Dwarf_Small * dw_marked_shf_compressed,
    Dwarf_Unsigned * dw_compressed_length,
    Dwarf_Unsigned * dw_uncompressed_length,
    Dwarf_Error * dw_error )
```

Get the real name of a section.

If the object has section groups only the sections in the group in dw_dbg will be found.

Whether .zdebug or ZLIB or SHF_COMPRESSED is the marker there is just one uncompress algorithm (zlib) for all three cases.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_std_section_name</i>	Pass in a standard section name, such as .debug_info or .debug_info.dwo .
<i>dw_actual_sec_name_out</i>	On success returns the actual section name from the object file.
<i>dw_marked_zcompressed</i>	On success returns TRUE if the original section name ends in .zdebug
<i>dw_marked_zlib_compressed</i>	On success returns TRUE if the section has the ZLIB string at the front of the section.
<i>dw_marked_shf_compressed</i>	On success returns TRUE if the section flag (Elf SHF_COMPRESSED) is marked as compressed.
<i>dw_compressed_length</i>	On success if the section was compressed it returns the original section length in the object file.
<i>dw_uncompressed_length</i>	On success if the section was compressed this returns the uncompressed length of the object section.
<i>dw_error</i>	On error returns the error usual details.

Returns

The usual DW_DLV_OK etc. If the section is not relevant to this Dwarf_Debug or is not in the object file at all, returns DW_DLV_NO_ENTRY

9.34.2.9 dwarf_get_section_count()

```
DW_API Dwarf_Unsigned dwarf_get_section_count (
    Dwarf_Debug dw_dbg )
```

Get section count (of object file sections).

Return the section count. Returns 0 if the `dw_dbg` argument is improper in any way.

Parameters

<code>dw_dbg</code>	Pass in a valid Dwarf_Debug of interest.
---------------------	--

Returns

Returns the count of sections in the object file or zero.

9.34.2.10 dwarf_get_section_info_by_index()

```
DW_API int dwarf_get_section_info_by_index (
    Dwarf_Debug dw_dbg,
    int dw_section_index,
    const char ** dw_section_name,
    Dwarf_Addr * dw_section_addr,
    Dwarf_Unsigned * dw_section_size,
    Dwarf_Error * dw_error )
```

Given a section index, get its size and address.

See [dwarf_get_section_info_by_index_a\(\)](#) for the newest version which returns additional values.

Fields and meanings in [dwarf_get_section_info_by_index\(\)](#) are the same as in [dwarf_get_section_info_by_index_a\(\)](#) except that the arguments `dw_section_flags` and `dw_section_offset` are missing here.

9.34.2.11 dwarf_get_section_info_by_index_a()

```
DW_API int dwarf_get_section_info_by_index_a (
    Dwarf_Debug dw_dbg,
    int dw_section_index,
    const char ** dw_section_name,
    Dwarf_Addr * dw_section_addr,
    Dwarf_Unsigned * dw_section_size,
    Dwarf_Unsigned * dw_section_flags,
    Dwarf_Unsigned * dw_section_offset,
    Dwarf_Error * dw_error )
```

Given a section index, get its size and address, etc.

See [dwarf_get_section_info_by_index\(\)](#) for the older and still current version.

Any of the pointers `dw_section_addr`, `dw_section_size`, `dw_section_flags`, and `dw_section_offset` may be passed in as zero and those will be ignored by the function.

Parameters

<code>dw_dbg</code>	The Dwarf_Debug of interest.
---------------------	------------------------------

Parameters

<i>dw_section_index</i>	Pass in an index, 0 through N-1 where N is the count returned from <code>dwarf_get_section_count</code> . As an index type <code>-int-</code> works in practice, but should really be <code>Dwarf_Unsigned</code> .
<i>dw_section_name</i>	On success returns a pointer to the section name as it appears in the object file.
<i>dw_section_addr</i>	On success returns the section address as defined by an object header.
<i>dw_section_size</i>	On success returns the section size as defined by an object header.
<i>dw_section_flags</i>	On success returns the section flags as defined by an object header. The flag meaning depends on which object format is being read and the meaning is defined by the object format. In PE object files this field is called Characteristics . We hope it is of some use.
<i>dw_section_offset</i>	On success returns the section offset as defined by an object header. The offset meaning is supposedly an object file offset but the meaning depends on the object file type(!). We hope it is of some use.
<i>dw_error</i>	On error returns the usual error pointer.

Returns

Returns `DW_DLV_OK` etc.

9.34.2.12 `dwarf_get_section_info_by_name()`

```
DW_API int dwarf_get_section_info_by_name (
    Dwarf_Debug dw_dbg,
    const char * dw_section_name,
    Dwarf_Addr * dw_section_addr,
    Dwarf_Unsigned * dw_section_size,
    Dwarf_Error * dw_error )
```

Given a section name, get its size and address.

See [dwarf_get_section_info_by_name_a\(\)](#) for the newest version which returns additional values.

Fields and meanings in [dwarf_get_section_info_by_name\(\)](#) are the same as in [dwarf_get_section_info_by_name_a\(\)](#) except that the arguments `dw_section_flags` and `dw_section_offset` are missing here.

9.34.2.13 `dwarf_get_section_info_by_name_a()`

```
DW_API int dwarf_get_section_info_by_name_a (
    Dwarf_Debug dw_dbg,
    const char * dw_section_name,
    Dwarf_Addr * dw_section_addr,
    Dwarf_Unsigned * dw_section_size,
    Dwarf_Unsigned * dw_section_flags,
    Dwarf_Unsigned * dw_section_offset,
    Dwarf_Error * dw_error )
```

Given a section name, get its size, address, etc.

New in v0.9.0 November 2023.

This is not often used and is completely unnecessary for most to call.

See [dwarf_get_section_info_by_name\(\)](#) for the older and still current version.

Any of the pointers `dw_section_addr`, `dw_section_size`, `dw_section_flags`, and `dw_section_offset` may be passed in as zero and those will be ignored by the function.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_section_name</i>	Pass in a pointer to a section name. It must be an exact match to the real section name.
<i>dw_section_addr</i>	On success returns the section address as defined by an object header.
<i>dw_section_size</i>	On success returns the section size as defined by an object header.
<i>dw_section_flags</i>	On success returns the section flags as defined by an object header. The flag meaning depends on which object format is being read and the meaning is defined by the object format. We hope it is of some use. In PE object files this field is called Characteristics .
<i>dw_section_offset</i>	On success returns the section offset as defined by an object header. The offset meaning is supposedly an object file offset but the meaning depends on the object file type(!). We hope it is of some use.
<i>dw_error</i>	On error returns the usual error pointer.

Returns

Returns DW_DLV_OK etc.

9.34.2.14 dwarf_get_section_max_offsets_d()

```
DW_API int dwarf_get_section_max_offsets_d (
    Dwarf_Debug dw_dbg,
    Dwarf_Unsigned * dw_debug_info_size,
    Dwarf_Unsigned * dw_debug_abbrev_size,
    Dwarf_Unsigned * dw_debug_line_size,
    Dwarf_Unsigned * dw_debug_loc_size,
    Dwarf_Unsigned * dw_debug_aranges_size,
    Dwarf_Unsigned * dw_debug_macinfo_size,
    Dwarf_Unsigned * dw_debug_pubnames_size,
    Dwarf_Unsigned * dw_debug_str_size,
    Dwarf_Unsigned * dw_debug_frame_size,
    Dwarf_Unsigned * dw_debug_ranges_size,
    Dwarf_Unsigned * dw_debug_pubtypes_size,
    Dwarf_Unsigned * dw_debug_types_size,
    Dwarf_Unsigned * dw_debug_macro_size,
    Dwarf_Unsigned * dw_debug_str_offsets_size,
    Dwarf_Unsigned * dw_debug_sup_size,
    Dwarf_Unsigned * dw_debug_cu_index_size,
    Dwarf_Unsigned * dw_debug_tu_index_size,
    Dwarf_Unsigned * dw_debug_names_size,
    Dwarf_Unsigned * dw_debug_loclists_size,
    Dwarf_Unsigned * dw_debug_rnglists_size )
```

Get section sizes for many sections.

The list of sections is incomplete and the argument list is ... too long ... making this an unusual function

Originally a hack so clients could verify offsets. Added so that one can detect broken offsets (which happened in an IRIX executable larger than 2GB with MIPSpro 7.3.1.3 toolchain.).

Parameters

<i>dw_dbg</i>	Pass in a valid Dwarf_Debug of interest.
---------------	--

Returns

If the `dw_dbg` is non-null it returns `DW_DLV_OK`. If `dw_dbg` is NULL it returns `DW_DLV_NO_ENTRY`.

9.34.2.15 `dwarf_machine_architecture()`

```
DW_API int dwarf_machine_architecture (
    Dwarf_Debug dw_dbg,
    Dwarf_Small * dw_ftype,
    Dwarf_Small * dw_obj_pointersize,
    Dwarf_Bool * dw_obj_is_big_endian,
    Dwarf_Unsigned * dw_obj_machine,
    Dwarf_Unsigned * dw_obj_flags,
    Dwarf_Small * dw_path_source,
    Dwarf_Unsigned * dw_ub_offset,
    Dwarf_Unsigned * dw_ub_count,
    Dwarf_Unsigned * dw_ub_index,
    Dwarf_Unsigned * dw_comdat_groupnumber )
```

Get basic object information original version.

Identical to `dwarf_machine_architecture_a()` except that this older version does not have the `dw_obj_type` argument so it cannot return the Elf `e_type` value..

9.34.2.16 `dwarf_machine_architecture_a()`

```
DW_API int dwarf_machine_architecture_a (
    Dwarf_Debug dw_dbg,
    Dwarf_Small * dw_ftype,
    Dwarf_Small * dw_obj_pointersize,
    Dwarf_Bool * dw_obj_is_big_endian,
    Dwarf_Unsigned * dw_obj_machine,
    Dwarf_Unsigned * dw_obj_type,
    Dwarf_Unsigned * dw_obj_flags,
    Dwarf_Small * dw_path_source,
    Dwarf_Unsigned * dw_ub_offset,
    Dwarf_Unsigned * dw_ub_count,
    Dwarf_Unsigned * dw_ub_index,
    Dwarf_Unsigned * dw_comdat_groupnumber )
```

Get basic object information from `Dwarf_Debug`.

Not all the fields here are relevant for all object types, and the `dw_obj_machine` and `dw_obj_flags` have ABI-defined values which have nothing to do with DWARF.

This version added December 2024 with an additional argument: `dw_obj_type`.

`dw_ub_offset`, `dw_ub_count`, `dw_ub_index` only apply to `DW_FTYPE_APPLEUNIVERSAL`.

`dw_comdat_groupnumber` only applies to `DW_FTYPE_ELF`.

Other than `dw_dbg` one can pass in NULL for any pointer parameter whose value is not of interest.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_ftype</i>	Pass in a pointer. On success the value pointed to will be set to the applicable DW_FTYPE value (see libdwwarf.h).
<i>dw_obj_pointersize</i>	Pass in a pointer. On success the value pointed to will be set to the applicable pointer size, which is almost always either 4 or 8.
<i>dw_obj_is_big_endian</i>	Pass in a pointer. On success the value pointed to will be set to either 1 (the object being read is big-endian) or 0 (the object being read is little-endian).
<i>dw_obj_machine</i>	Pass in a pointer. On success the value pointed to will be set to a value that the specific ABI uses for the machine-architecture the object file says it is for.
<i>dw_obj_type</i>	Pass in a pointer. On success the value pointed to will be set to a value that the specific ABI uses for the machine-architecture the object file says it is for (for ELF is elf header e_type).
<i>dw_obj_flags</i>	Pass in a pointer. On success the value pointed to will be set to a value that the specific ABI uses for a header record flags word (in a PE object the flags word is called Characteristics).
<i>dw_path_source</i>	Pass in a pointer. On success the value pointed to will be set to a value that libdwwarf sets to a DW_PATHSOURCE value indicating what caused the file path.
<i>dw_ub_offset</i>	Pass in a pointer. On success if the value of dw_ftype is DW_FTYPE_APPLEUNIVERSAL the returned value will be set to the count (in all other cases, the value is set to 0)
<i>dw_ub_count</i>	Pass in a pointer. On success if the value of dw_ftype is DW_FTYPE_APPLEUNIVERSAL the returned value will be set to the number of object files in the binary (in all other cases, the value is set to 0)
<i>dw_ub_index</i>	Pass in a pointer. On success if the value of dw_ftype is DW_FTYPE_APPLEUNIVERSAL the returned value will be set to the number of the specific object from the universal-binary, usable values are 0 through dw_ub_count-1. (in all other cases, the value is set to 0)
<i>dw_comdat_groupnumber</i>	Pass in a pointer. On success if the value of dw_ftype is DW_FTYPE_ELF the returned value will be the comdat group being referenced. (in all other cases, the value is set to 0)

Returns

Returns DW_DLV_NO_ENTRY if the Dwarf_Debug passed in is null or stale. Otherwise returns DW_DLV_OK and non-null return-value pointers will have meaningful data.

9.35 Section Groups Objectfile Data

Functions

- DW_API int [dwarf_sec_group_sizes](#) (Dwarf_Debug dw_dbg, Dwarf_Unsigned *dw_section_count_out, Dwarf_Unsigned *dw_group_count_out, Dwarf_Unsigned *dw_selected_group_out, Dwarf_Unsigned *dw_map_entry_count_out, Dwarf_Error *dw_error)

Get Section Groups data counts.

- DW_API int [dwarf_sec_group_map](#) (Dwarf_Debug dw_dbg, Dwarf_Unsigned dw_map_entry_count, Dwarf_Unsigned *dw_group_numbers_array, Dwarf_Unsigned *dw_sec_numbers_array, const char **dw_sec_names_array, Dwarf_Error *dw_error)

Return a map between group numbers and section numbers.

9.35.1 Detailed Description

Section Groups are defined in the extended Elf ABI and are seen in relocatable Elf object files, not executables or shared objects.

[Section Groups Overview](#)

9.35.2 Function Documentation

9.35.2.1 dwarf_sec_group_map()

```
DW_API int dwarf_sec_group_map (
    Dwarf_Debug dw_dbg,
    Dwarf_Unsigned dw_map_entry_count,
    Dwarf_Unsigned * dw_group_numbers_array,
    Dwarf_Unsigned * dw_sec_numbers_array,
    const char ** dw_sec_names_array,
    Dwarf_Error * dw_error )
```

Return a map between group numbers and section numbers.

This map shows all the groups in the object file and shows which object sections go with which group.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_map_entry_count</i>	Pass in the dw_map_entry_count_out from dwarf_sec_group_sizes
<i>dw_group_numbers_array</i>	Pass in an array of Dwarf_Unsigned with dw_map_entry_count entries. Zero the data before the call here. On success returns a list of group numbers.
<i>dw_sec_numbers_array</i>	Pass in an array of Dwarf_Unsigned with dw_map_entry_count entries. Zero the data before the call here. On success returns a list of section numbers.
<i>dw_sec_names_array</i>	Pass in an array of const char * with dw_map_entry_count entries. Zero the data before the call here. On success returns a list of section names.
<i>dw_error</i>	The usual error details pointer.

Returns

On success returns DW_DLV_OK

9.35.2.2 dwarf_sec_group_sizes()

```
DW_API int dwarf_sec_group_sizes (
    Dwarf_Debug dw_dbg,
    Dwarf_Unsigned * dw_section_count_out,
    Dwarf_Unsigned * dw_group_count_out,
    Dwarf_Unsigned * dw_selected_group_out,
    Dwarf_Unsigned * dw_map_entry_count_out,
    Dwarf_Error * dw_error )
```

Get Section Groups data counts.

Allows callers to find out what groups (dwo or COMDAT) are in the object and how much to allocate so one can get the group-section map data.

This is relevant for Debug Fission. If an object file has both .dwo sections and non-dwo sections or it has Elf COMDAT GROUP sections this becomes important.

Section Groups Overview

Parameters

<i>dw_dbg</i>	Pass in the Dwarf_Debug of interest.
<i>dw_section_count_out</i>	On success returns the number of DWARF sections in the object file. Can sometimes be many more than are of interest.
<i>dw_group_count_out</i>	On success returns the number of groups. Though usually one, it can be much larger.
<i>dw_selected_group_out</i>	On success returns the groupnumber that applies to this specific open Dwarf_Debug.
<i>dw_map_entry_count_out</i>	On success returns the count of record allocations needed to call dwarf_sec_group_map successfully. dw_map_entry_count_out will be less than or equal to dw_section_count_out.
<i>dw_error</i>	The usual error details pointer.

Returns

On success returns DW_DLV_OK

9.36 LEB Encode and Decode

Functions

- DW_API int **dwarf_encode_leb128** ([Dwarf_Unsigned](#) dw_val, int *dw_nbytes, char *dw_space, int dw_splen)
- DW_API int **dwarf_encode_signed_leb128** ([Dwarf_Signed](#) dw_val, int *dw_nbytes, char *dw_space, int dw_splen)
- DW_API int **dwarf_decode_leb128** (char *dw_leb, [Dwarf_Unsigned](#) *dw_leblen, [Dwarf_Unsigned](#) *dw_outval, char *dw_endptr)
- DW_API int **dwarf_decode_signed_leb128** (char *dw_leb, [Dwarf_Unsigned](#) *dw_leblen, [Dwarf_Signed](#) *dw_outval, char *dw_endptr)

9.36.1 Detailed Description

These are LEB/ULEB reading and writing functions heavily used inside libdwarf.

While the DWARF Standard does not mention allowing extra insignificant trailing bytes in a ULEB these functions allow a few such for compilers using extras for alignment in DWARF.

9.37 Miscellaneous Functions

Functions

- DW_API const char * [dwarf_package_version](#) (void)
Return the version string in the library.
- DW_API int [dwarf_set_stringcheck](#) (int dw_stringcheck)
Turn off libdwarf checks of strings.
- DW_API int [dwarf_set_reloc_application](#) (int dw_apply)
*Set libdwarf response to *.rela relocations.*
- DW_API void [dwarf_record_cmdline_options](#) (Dwarf_Cmdline_Options dw_dd_options)
Tell libdwarf to add verbosity to Line Header errors By default the flag in the struct argument is zero. dwarfdump uses this when -v used on dwarfdump.
- DW_API int [dwarf_set_de_alloc_flag](#) (int dw_v)
Eliminate libdwarf tracking of allocations Independent of any Dwarf_Debug and applicable to all whenever the setting is changed. Defaults to non-zero.
- DW_API int [dwarf_library_allow_dup_attr](#) (int dw_v)
Eliminate libdwarf checking attribute duplication.
- DW_API Dwarf_Small [dwarf_set_default_address_size](#) (Dwarf_Debug dw_dbg, Dwarf_Small dw_value)
Set the address size on a Dwarf_Debug.
- DW_API int [dwarf_get_universalbinary_count](#) (Dwarf_Debug dw_dbg, Dwarf_Unsigned *dw_current_index, Dwarf_Unsigned *dw_available_count)
Retrieve universal binary index.

Variables

- DW_API void(*) (void *, const void *, unsigned long) [dwarf_get_endian_copy_function](#) (Dwarf_Debug dw_dbg)
Get a pointer to the applicable swap/noswap function.
- DW_API Dwarf_Cmdline_Options [dwarf_cmdline_options](#)

9.37.1 Detailed Description

9.37.2 Function Documentation

9.37.2.1 dwarf_get_universalbinary_count()

```
DW_API int dwarf_get_universalbinary_count (
    Dwarf_Debug dw_dbg,
    Dwarf_Unsigned * dw_current_index,
    Dwarf_Unsigned * dw_available_count )
```

Retrieve universal binary index.

For Mach-O universal binaries this returns relevant information.

For non-universal binaries (Mach-O, Elf, or PE) the values are not meaningful, so the function returns DW_DLV_NO_ENTRY..

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_current_index</i>	If dw_current_index is passed in non-null the function returns the universal-binary index of the current object (which came from a universal binary).
<i>dw_available_count</i>	If dw_current_index is passed in non-null the function returns the count of binaries in the universal binary.

Returns

Returns DW_DLV_NO_ENTRY if the object file is not from a Mach-O universal binary. Returns DW_DLV_↵ NO_ENTRY if dw_dbg is passed in NULL. Never returns DW_DLV_ERROR.

9.37.2.2 dwarf_library_allow_dup_attr()

```
DW_API int dwarf_library_allow_dup_attr (
    int dw_v )
```

Eliminate libdwf checking attribute duplication.

Independent of any Dwarf_Debug, this sets a global flag in libdwf and is applicable to all whenever the setting is changed. Defaults to zero so by default libdwf does check every set of abbreviations for duplicate attributes.

DWARF5 Sec 2.2 Attribute Types Each attribute value is characterized by an attribute name. No more than one attribute with a given name may appear in any debugging information entry. Essentially the same wording is in Sec 2.2 of DWARF2, DWARF3 and DWARF4.

Do not call this with non-zero dw_v unless you really want the library to avoid this basic DWARF-correctness check.

Since

{0.12.0}

Parameters

<i>dw↵_v</i>	If non-zero passed in libdwf will avoid the checks and will not return errors for an abbreviation list with duplicate attributes.
--------------	---

Returns

Returns the previous version of the flag.

9.37.2.3 dwarf_package_version()

```
DW_API const char * dwarf_package_version (
    void )
```

Return the version string in the library.

An example: "0.3.0" which is a Semantic Version identifier. Before September 2021 the version string was a date, for example "20210528", which is in ISO date format. See DW_LIBDWARF_VERSION DW_LIBDWARF_VERSION↵ _MAJOR DW_LIBDWARF_VERSION_MINOR DW_LIBDWARF_VERSION_MICRO

Returns

The Package Version built into libdwarf.so or libdwarf.a

9.37.2.4 dwarf_record_cmdline_options()

```
DW_API void dwarf_record_cmdline_options (
    Dwarf_Cmdline_Options dw_dd_options )
```

Tell libdwarf to add verbosity to Line Header errors By default the flag in the struct argument is zero. dwarfdump uses this when -v used on dwarfdump.

See also

[dwarf_register_printf_callback](#)

Parameters

<i>dw_dd_options</i>	The structure has one flag, and if the flag is nonzero and there is an error in reading a line table header the function passes back detail error messages via dwarf_register_printf_callback.
----------------------	--

9.37.2.5 dwarf_set_de_alloc_flag()

```
DW_API int dwarf_set_de_alloc_flag (
    int dw_v )
```

Eliminate libdwarf tracking of allocations Independent of any Dwarf_Debug and applicable to all whenever the setting is changed. Defaults to non-zero.

Parameters

<i>dw↔ _v</i>	If zero passed in libdwarf will run somewhat faster and library memory allocations will not all be tracked and dwarf_finish() will be unable to free/dealloc some things. User code can do the necessary deallocs (as documented), but the normal guarantee that libdwarf will clean up is revoked. If non-zero passed in libdwarf will resume or continue tracking allocations
-------------------	---

Returns

Returns the previous version of the flag.

9.37.2.6 dwarf_set_default_address_size()

```
DW_API Dwarf_Small dwarf_set_default_address_size (
    Dwarf_Debug dw_dbg,
    Dwarf_Small dw_value )
```

Set the address size on a Dwarf_Debug.

DWARF information CUs and other section DWARF headers define a CU-specific address size, but this Dwarf_↔ Debug value is used when other address size information does not exist, for example in a DWARF2 CIE or FDE.

Parameters

<i>dw_dbg</i>	The Dwarf_Debug of interest.
<i>dw_value</i>	Sets the address size for the Dwarf_Debug to a non-zero value. The default address size is derived from headers in the object file. Values larger than the size of Dwarf_Addr are not set. If zero passed the default is not changed.

Returns

Returns the last set address size.

9.37.2.7 dwarf_set_reloc_application()

```
DW_API int dwarf_set_reloc_application (
    int dw_apply )
```

Set libdwarf response to *.rela relocations.

dw_apply defaults to 1 and means apply all '.rela' relocations on reading in a dwarf object section of such relocations. Best to just ignore this function It applies to all Dwarf_Debug open and all opened later in this library instance.

Parameters

<i>dw_apply</i>	Pass in a zero to turn off reading and applying of .rela relocations, which will likely break reading of .o object files but probably will not break reading executables or shared objects. Pass in non zero (it is really just an 8 bit value, so use a small value) to turn off inspecting .rela sections.
-----------------	--

Returns

Returns the previous value of the apply flag.

9.37.2.8 dwarf_set_stringcheck()

```
DW_API int dwarf_set_stringcheck (
    int dw_stringcheck )
```

Turn off libdwarf checks of strings.

Zero is the default and means do all string length validity checks. It applies to all Dwarf_Debug open and all opened later in this library instance.

Parameters

<i>dw_stringcheck</i>	Pass in a small non-zero value to turn off all libdwarf string validity checks. It speeds up libdwarf, but...is dangerous and voids all promises the library will not segfault.
-----------------------	---

Returns

Returns the previous value of this flag.

9.37.3 Variable Documentation**9.37.3.1 dwarf_get_endian_copy_function**

```
DW_API void(*) (void *, const void *, unsigned long) dwarf_get_endian_copy_function(Dwarf_Debug
dw_dbg) (
    Dwarf_Debug dw_dbg )
```

Get a pointer to the applicable swap/noswap function.

the function pointer returned enables libdwarf users to use the same 64bit/32bit/16bit word copy as libdwarf does internally for the Dwarf_Debug passed in. The function makes it possible for libdwarf to read either endianness.

Parameters

<i>dw_dbg</i>	Pass in a pointer to the applicable Dwarf_Debug.
---------------	--

Returns

a pointer to a copy function. If the object file referred to and the libdwarf reading that file are the same endianness the function returned will, when called, do a simple memcpy, effectively, while otherwise it would do a byte-swapping copy. It seems unlikely this will be useful to most library users. To call the copy function returned the first argument must be a pointer to the target word and the second must be a pointer to the input word. The third argument is the length to be copied and it must be 2,4,or 8.

9.38 Determine Object Type of a File**Functions**

- DW_API int **dwarf_object_detector_path_b** (const char *dw_path, char *dw_outpath_buffer, unsigned long dw_outpathlen, char **dw_gl_pathnames, unsigned int dw_gl_pathcount, unsigned int *dw_ftype, unsigned int *dw_endian, unsigned int *dw_offsetsize, Dwarf_Unsigned *dw_filesize, unsigned char *dw_pathsource, int *dw_errcode)
- DW_API int **dwarf_object_detector_path_dSYM** (const char *dw_path, char *dw_outpath, unsigned long dw_outpath_len, char **dw_gl_pathnames, unsigned int dw_gl_pathcount, unsigned int *dw_ftype, unsigned int *dw_endian, unsigned int *dw_offsetsize, Dwarf_Unsigned *dw_filesize, unsigned char *dw_pathsource, int *dw_errcode)
- DW_API int **dwarf_object_detector_fd** (int dw_fd, unsigned int *dw_ftype, unsigned int *dw_endian, unsigned int *dw_offsetsize, Dwarf_Unsigned *dw_filesize, int *dw_errcode)

9.38.1 Detailed Description

This group of functions are unlikely to be called by your code unless your code needs to know the basic data about an object file without actually opening a Dwarf_Debug.

These are crucial for libdwarf itself. The dw_ftype returned is one of DW_FTYPE_ELF, DW_FTYPE_PE, DW_FTYPE_MACH_O, or DW_FTYPE_APPLEUNIVERSAL.

These are not meant to deal with a specific binary inside a MacOS Universal Binary (DW_FTYPE_APPLEUNIVERSAL).

9.39 Section allocation: malloc or mmap

Functions

- DW_API enum [Dwarf_Sec_Alloc_Pref](#) [dwarf_set_load_preference](#) (enum [Dwarf_Sec_Alloc_Pref](#) [dw_load_preference](#))
Set/Retrieve section allocation preference.
- DW_API int [dwarf_get_mmap_count](#) ([Dwarf_Debug](#) [dw_dbg](#), [Dwarf_Unsigned](#) *[dw_mmap_count](#), [Dwarf_Unsigned](#) *[dw_mmap_size](#), [Dwarf_Unsigned](#) *[dw_malloc_count](#), [Dwarf_Unsigned](#) *[dw_malloc_size](#))
Retrieve count of mmap/malloc sections.

9.39.1 Detailed Description

Functions related to the choice of malloc/read or mmap for object section memory allocation.

The default allocation preference is malloc().

The shell environment variable DWARF_WHICH_ALLOC is also involved at runtime but it only applies to reading Elf object files.. If the value is 'malloc' then use of read/malloc is preferred. If the value is 'mmap' then use of mmap is preferred (Example: 'export DWARF_WHICH_ALLOC=mmap'). Otherwise, the environment value is checked and ignored.

If present and valid this environment variable takes precedence over [dwarf_set_load_preference\(\)](#).

9.39.2 Function Documentation

9.39.2.1 dwarf_get_mmap_count()

```
DW_API int dwarf_get_mmap_count (
    Dwarf_Debug dw_dbg,
    Dwarf_Unsigned * dw_mmap_count,
    Dwarf_Unsigned * dw_mmap_size,
    Dwarf_Unsigned * dw_malloc_count,
    Dwarf_Unsigned * dw_malloc_size )
```

Retrieve count of mmap/malloc sections.

Since

{0.12.0}

Note that compressed section contents will be expanded into a malloc/read section in all cases.

Parameters

<i>dw_dbg</i>	A valid open Dwarf_Debug.
<i>dw_mmap_count</i>	On success the number of sections allocated with mmap is returned. If null passed in the argument is ignored.
<i>dw_mmap_size</i>	On success the size total in bytes of sections allocated with mmap is returned. If null passed in the argument is ignored.
<i>dw_malloc_count</i>	On success the number of sections read/allocated with read/malloc is returned. If null passed in the argument is ignored. On success the number of sections allocated with malloc/read is returned.
<i>dw_malloc_size</i>	On success the total size in bytes of sections with malloc/read is returned. If null passed in the argument is ignored. On success the number of sections read/allocated with

Returns

On success returns DW_DLV_OK and sets the counts and total size through the respective non-null pointer arguments. If dw_dbg is invalid or NULL the function returns DW_DLV_ERROR. Never returns DW_DLV_↔NO_ENTRY.

9.39.2.2 dwarf_set_load_preference()

```
DW_API enum Dwarf_Sec_Alloc_Pref dwarf_set_load_preference (
    enum Dwarf_Sec_Alloc_Pref dw_load_preference )
```

Set/Retrieve section allocation preference.

Since

{0.12.0}

By default object file sections are loaded using malloc and read (Dwarf_Alloc_Malloc). This works everywhere and works well on all but gigantic object files.

The preference of Dwarf_Alloc_Mmap does not guarantee mmap will be used for object section data, but does cause mmap() to be used when possible.

In 0.12.0 mmap() is only usable on Elf object files.

dw_load_preference is one of Dwarf_Alloc_Malloc (1) Dwarf_Alloc_Mmap (2)

Must be called before calling a dwarf_init*() to be effective in a dwarf_init*(). The value is remembered for subsequent dwarf_init*() in the library runtime being executed.

Parameters

<i>dw_load_preference</i>	If passed in Dwarf_Alloc_Mmap then future calls to any dwarf_init*() function will use mmap to load object sections if possible. If passed in Dwarf_Alloc_Malloc then future calls to any dwarf_init*() function will use mmap to load sections. Any other value passed in dw_load_preference is ignored.
---------------------------	---

Returns

Always returns the value before dw_load_preference applied, of this runtime global preference.

9.40 Using dwarf_init_path()

Example of a libdwarf initialization call.

Example of a libdwarf initialization call.

An example calling dwarf_init_path() and dwarf_finish()

Parameters

<i>path</i>	Path to an object we wish to open.
<i>groupnumber</i>	Desired groupnumber. Use DW_DW_GROUPNUMBER_ANY unless you have reason to do otherwise.

Returns

Returns the applicable result. DW_DLV_OK etc.

```

*/
int exampleinit(const char *path, unsigned groupnumber)
{
    static char true_pathbuf[FILENAME_MAX];
    unsigned tpathlen = FILENAME_MAX;
    Dwarf_Handler errhand = 0;
    Dwarf_Ptr errarg = 0;
    Dwarf_Error error = 0;
    Dwarf_Debug dbg = 0;
    int res = 0;

    res = dwarf_init_path(path, true_pathbuf,
        tpathlen, groupnumber, errhand,
        errarg, &dbg, &error);
    if (res == DW_DLV_ERROR) {
        /* Necessary call even though dbg is null!
           This avoids a memory leak. */
        dwarf_dealloc_error(dbg, error);
        return res;
    }
    if (res == DW_DLV_NO_ENTRY) {
        /* Nothing we can do */
        return res;
    }
    printf("The file we actually opened is %s\n",
        true_pathbuf);
    /* Call libdwarf functions here */
    dwarf_finish(dbg);
    return DW_DLV_OK;
}

```

9.41 Using dwarf_init_path_dl()

Example focused on GNU debuglink data.

Example focused on GNU debuglink data.

In case GNU debuglink data is followed the true_pathbuf content will not match path. The path actually used is copied to true_path_out.

In the case of MacOS dSYM the true_path_out may not match path.

If debuglink data is missing from the Elf executable or shared-object (ie, it is a normal object!) or unusable by libdwarf or true_path_buffer len is zero or true_path_out_buffer is zero libdwarf accepts the path given as the object to report on, no debuglink or dSYM processing will be used.

See also

<https://sourceware.org/gdb/onlinedocs/gdb/Separate-Debug-Files.html>

An example calling `dwarf_init_path_dl()` and `dwarf_finish()`

Parameters

<i>path</i>	Path to an object we wish to open.
<i>groupnumber</i>	Desired groupnumber. Use DW_DW_GROUPNUMBER_ANY unless you have reason to do otherwise.
<i>error</i>	A pointer we can use to record error details.

Returns

Returns the applicable result. DW_DLV_OK etc.

```

*/
int exampleinit_dl(const char *path, unsigned groupnumber,
Dwarf_Error *error)
{
    static char true_pathbuf[FILENAME_MAX];
    static const char *glpath[3] = {
        "/usr/local/debug",
        "/usr/local/private/debug",
        "/usr/local/libdwarf/debug"
    };
    unsigned tpathlen = FILENAME_MAX;
    Dwarf_Handler errhand = 0;
    Dwarf_Ptr errarg = 0;
    Dwarf_Debug dbg = 0;
    int res = 0;
    unsigned char path_source = 0;

    res = dwarf_init_path_dl(path, true_pathbuf,
        tpathlen, groupnumber, errhand,
        errarg, &dbg,
        (char **)glpath,
        3,
        &path_source,
        error);
    if (res == DW_DLV_ERROR) {
        /* We are not returning dbg, so we must do:
           dwarf_dealloc_error(dbg, *error);
           here to free the error details. */
        dwarf_dealloc_error(dbg, *error);
        *error = 0;
        return res;
    }
    if (res == DW_DLV_NO_ENTRY) {
        return res;
    }
    printf("The file we actually opened is %s\n",
        true_pathbuf);
    /* Call libdwarf functions here */
    dwarf_finish(dbg);
    return res;
}

```

9.42 Using dwarf_attrlist()

Example showing [dwarf_attrlist\(\)](#)

Example showing [dwarf_attrlist\(\)](#)

Parameters

<i>somedie</i>	Pass in any valid relevant DIE pointer.
<i>error</i>	An error pointer we can use.

Returns

Return DW_DLV_OK (etc).

```

*/
int example1(Dwarf_Die somedie, Dwarf_Error *error)
{
    Dwarf_Debug dbg = 0;
    Dwarf_Signed atcount;
    Dwarf_Attribute *atlist;
    Dwarf_Signed i = 0;
    int errv;

    errv = dwarf_attrlist(somedie, &atlist, &atcount, error);
    if (errv != DW_DLV_OK) {
        return errv;
    }
    for (i = 0; i < atcount; ++i) {
        Dwarf_Half attrnum = 0;
        const char *attrname = 0;

        /* use atlist[i], likely calling
         libdwarf functions and likely
         returning DW_DLV_ERROR if
         what you call gets DW_DLV_ERROR */
        errv = dwarf_whatattr(atlist[i], &attrnum, error);
        if (errv != DW_DLV_OK) {
            /* Something really bad happened. */
            return errv;
        }
        dwarf_get_AT_name(attrnum, &attrname);
        printf("Attribute[%ld], value %u name %s\n",
            (long int)i, attrnum, attrname);
        dwarf_dealloc_attribute(atlist[i]);
        atlist[i] = 0;
    }
    dwarf_dealloc(dbg, atlist, DW_DLA_LIST);
    return DW_DLV_OK;
}

```

9.43 Attaching a tied dbg

Example attaching base dbg to a split-DWARF object.

Example attaching base dbg to a split-DWARF object.

See DWARF5 Appendix F on Split-DWARF.

By libdwarf convention, open the split Dwarf_Debug using a dwarf_init call. Then open the executable as the tied object. Then call [dwarf_set_tied_dbg\(\)](#) so the library can look for relevant data in the tied-dbg (the executable).

With split dwarf your libdwarf calls after the the initial open are done against the split Dwarf_Dbg and libdwarf automatically looks in the tied dbg when and as appropriate. the tied_dbg can be detached too, see [example3](#) link, though you must call [dwarf_finish\(\)](#) on the detached dw_tied_dbg, the library will not do that for you.

Parameters

<i>split_dbg</i>	
<i>tied_dbg</i>	
<i>error</i>	

Returns

Returns DW_DLV_OK or DW_DLV_ERROR or DW_DLV_NO_ENTRY to the caller.

```

*/
int example2(Dwarf_Debug split_dbg, Dwarf_Debug tied_dbg,
    Dwarf_Error *error)

```

```

{
    int res = 0;

    /* The caller should have opened dbg
       on the split-dwarf object/dwp,
       an object with DWARF, but no executable
       code.
       And it should have opened tieddbg on the
       runnable shared object or executable. */
    res = dwarf_set_tied_dbg(split_dbg,tied_dbg,error);
    /* Let the caller (who initialized the dbg
       values) deal with doing dwarf_finish()
       /
    return res;
}

```

9.44 Detaching a tied dbg

Example detaching a tied (executable) dbg.

Example detaching a tied (executable) dbg.

See DWARF5 Appendix F on Split-DWARF.

With split dwarf your libdwarf calls after than the initial open are done against the split Dwarf_Dbg and libdwarf automatically looks in the open tied dbg when and as appropriate. the tied-dbg can be detached too, see example3 link, though you must call [dwarf_finish\(\)](#) on the detached dw_tied_dbg, the library will not do that for you..

```

*/
int example3(Dwarf_Debug split_dbg,Dwarf_Error *error)
{
    int res = 0;
    res = dwarf_set_tied_dbg(split_dbg,NULL,error);
    if (res != DW_DLV_OK) {
        /* Something went wrong*/
        return res;
    }
    return res;
}

```

9.45 Examining Section Group data

Example accessing Section Group data.

Example accessing Section Group data.

With split dwarf your libdwarf calls after than the initial open are done against the base Dwarf_Dbg and libdwarf automatically looks in the open tied dbg when and as appropriate. the tied-dbg can be detached too, see example3 link, though you must call [dwarf_finish\(\)](#) on the detached dw_tied_dbg, the library will not do that for you..

Section groups apply to Elf COMDAT groups too.

```

*/
void examplesecgroup(Dwarf_Debug dbg)
{
    int res = 0;
    Dwarf_Unsigned section_count = 0;
    Dwarf_Unsigned group_count;
    Dwarf_Unsigned selected_group = 0;
    Dwarf_Unsigned group_map_entry_count = 0;
    Dwarf_Unsigned *sec_nums = 0;
    Dwarf_Unsigned *group_nums = 0;
    const char ** sec_names = 0;
    Dwarf_Error error = 0;
    Dwarf_Unsigned i = 0;

    res = dwarf_sec_group_sizes(dbg,&section_count,
        &group_count,&selected_group, &group_map_entry_count,
        &error);
    if (res != DW_DLV_OK) {

```

```

        /* Something is badly wrong*/
        return;
    }
    /* In an object without split-dwarf sections
    or COMDAT sections we now have
    selected_group == 1. */
    sec_nums = calloc(group_map_entry_count, sizeof(Dwarf_Unsigned));
    if (!sec_nums) {
        /* FAIL. out of memory */
        return;
    }
    group_nums = calloc(group_map_entry_count, sizeof(Dwarf_Unsigned));
    if (!group_nums) {
        free(group_nums);
        /* FAIL. out of memory */
        return;
    }
    sec_names = calloc(group_map_entry_count, sizeof(char*));
    if (!sec_names) {
        free(group_nums);
        free(sec_nums);
        /* FAIL. out of memory */
        return;
    }

    res = dwarf_sec_group_map(dbg, group_map_entry_count,
                             group_nums, sec_nums, sec_names, &error);
    if (res != DW_DLV_OK) {
        /* FAIL. Something badly wrong. */
        free(sec_names);
        free(group_nums);
        free(sec_nums);
    }
    for (i = 0; i < group_map_entry_count; ++i) {
        /* Now do something with
        group_nums[i], sec_nums[i], sec_names[i] */
    }
    /* The strings are in Elf data.
    Do not free() the strings themselves.*/
    free(sec_names);
    free(group_nums);
    free(sec_nums);
}

```

9.46 Using dwarf_siblingof_c()

Example accessing a DIE sibling.

Example accessing a DIE sibling.

Access to each DIE on a sibling list. This is the preferred form as it is slightly more efficient than [dwarf_siblingof_b\(\)](#).

```

*/
int example4c(Dwarf_Die in_die,
             Dwarf_Error *error)
{
    Dwarf_Die return_sib = 0;
    int res = 0;

    /* in_die must be a valid Dwarf_Die */
    res = dwarf_siblingof_c(in_die, &return_sib, error);
    if (res == DW_DLV_OK) {
        /* Use return_sib here. */
        dwarf_dealloc_die(return_sib);
        /* return_sib is no longer usable for anything, we
        ensure we do not use it accidentally with: */
        return_sib = 0;
        return res;
    }
    return res;
}

```

9.47 Using dwarf_siblingof_b()

Example accessing a DIE sibling.

Example accessing a DIE sibling.

Access to each DIE on a sibling list This is the older form, required after `dwarf_next_cu_header_d()`.

Better to use `dwarf_next_cu_header_e()` and `dwarf_siblingof_c()`.

```

*/
int example4b(Dwarf_Debug dbg, Dwarf_Die in_die,
             Dwarf_Bool is_info,
             Dwarf_Error *error)
{
    Dwarf_Die return_sib = 0;
    int res = 0;

    /* in_die might be NULL following a call
       to dwarf_next_cu_header_d()
       or a valid Dwarf_Die */
    res = dwarf_siblingof_b(dbg, in_die, is_info, &return_sib, error);
    if (res == DW_DLV_OK) {
        /* Use return_sib here. */
        dwarf_dealloc_die(return_sib);
        /* return_sib is no longer usable for anything, we
           ensure we do not use it accidentally with: */
        return_sib = 0;
        return res;
    }
    return res;
}

```

9.48 Using dwarf_child()

Example accessing a DIE child.

Example accessing a DIE child.

If the DIE has children (for example inner scopes in a function or members of a struct) this retrieves the DIE which appears first. The child itself may have its own sibling chain.

```

*/
void example5(Dwarf_Die in_die)
{
    Dwarf_Die return_kid = 0;
    Dwarf_Error error = 0;
    int res = 0;

    res = dwarf_child(in_die, &return_kid, &error);
    if (res == DW_DLV_OK) {
        /* Use return_kid here. */
        dwarf_dealloc_die(return_kid);
        /* The original form of dealloc still works
           dwarf_dealloc(dbg, return_kid, DW_DLA_DIE);
           /
        /* return_kid is no longer usable for anything, we
           ensure we do not use it accidentally with: */
        return_kid = 0;
    }
}

```

9.49 using dwarf_validate_die_sibling

Example of a DIE tree validation.

Example of a DIE tree validation.

Here we show how one uses `dwarf_validate_die_sibling()`. DwarfDump uses this function as a part of its validation of DIE trees.

It is not something you need to use. But one must use it in a specific pattern for it to work properly.

`dwarf_validate_die_sibling()` depends on data set by `dwarf_child()` preceeding `dwarf_siblingof_b()` . `dwarf_child()` records a little bit of information invisibly in the Dwarf_Debug data.

```

*/
int example_sibvalid(Dwarf_Debug dbg,
    Dwarf_Die in_die,
    Dwarf_Error*error)
{
    int      cres = DW_DLV_OK;
    int      sibres = DW_DLV_OK;
    Dwarf_Die die = 0;
    Dwarf_Die sibdie = 0;
    Dwarf_Die child = 0;
    Dwarf_Bool is_info = dwarf_get_die_infotypes_flag(die);

    die = in_die;
    for ( ; die ; die = sibdie) {
        int vres = 0;
        Dwarf_Unsigned offset = 0;

        /* Maybe print something you extract from the DIE */
        cres = dwarf_child(die,&child,error);
        if (cres == DW_DLV_ERROR) {
            if (die != in_die) {
                dwarf_dealloc_die(die);
            }
            printf("dwarf_child ERROR\n");
            return DW_DLV_ERROR;
        }
        if (cres == DW_DLV_OK) {
            int lres = 0;

            child = 0;
            lres = example_sibvalid(dbg,child,error);
            if (lres == DW_DLV_ERROR) {
                if (die != in_die) {
                    dwarf_dealloc_die(die);
                }
                dwarf_dealloc_die(child);
                printf("example_sibvalid ERROR\n");
                return lres;
            }
        }
        sibdie = 0;
        sibres = dwarf_siblingof_b(dbg,die,is_info,
            &sibdie,error);
        if (sibres == DW_DLV_ERROR) {
            if (die != in_die) {
                dwarf_dealloc_die(die);
            }
            if (child) {
                dwarf_dealloc_die(child);
            }
            printf("dwarf_siblingof_b ERROR\n");
            return DW_DLV_ERROR;
        }
        if (sibres == DW_DLV_NO_ENTRY) {
            if (die != in_die) {
                dwarf_dealloc_die(die);
            }
            if (child) {
                dwarf_dealloc_die(child);
            }
            return DW_DLV_OK;
        }
        vres = dwarf_validate_die_sibling(sibdie,&offset);
        if (vres == DW_DLV_ERROR) {
            if (die != in_die) {
                dwarf_dealloc_die(die);
            }
            if (child) {
                dwarf_dealloc_die(child);
            }
            dwarf_dealloc_die(sibdie);
            printf("Invalid sibling DIE\n");
            return DW_DLV_ERROR;
        }
        /* loop again */
        if (die != in_die) {
            dwarf_dealloc_die(die);
        }
        die = 0;
    }
    return DW_DLV_OK;
}

```

9.50 Example walking CUs(e)

Example examining CUs looking for specific items(e).

Example examining CUs looking for specific items(e).

Loops through as many CUs as needed, stops and returns once a CU provides the desired data.

Assumes certain functions you write to remember the aspect of CUs that matter to you so once found in a cu my_needed_data_exists() or some other function of yours can identify the correct record.

Depending on your goals in examining the DIE tree it may be helpful to maintain a DIE stack of active DIEs, pushing and popping as you make your way through the DIE levels.

We assume that on a serious error we will give up (for simplicity here).

We assume the caller to examplecuhdre() will know what to retrieve (when we return DW_DLV_OK from examplecuhdree() and that myrecords points to a record with all the data needed by my_needed_data_exists() and recorded by myrecord_data_for_die().

```

*/

struct myrecords_struct *myrecords;
void myrecord_data_for_die(struct myrecords_struct *myrecords_data,
    Dwarf_Die d)
{
    /* do something */
    /* avoid compiler warnings */
    (void)myrecords_data;
    (void)d;
}
int my_needed_data_exists(struct myrecords_struct *myrecords_data)
{
    /* do something */
    /* avoid compiler warnings */
    (void)myrecords_data;
    return DW_DLV_OK;
}

/* Loop on DIE tree. */
static void
record_die_and_siblings_e(Dwarf_Debug dbg, Dwarf_Die in_die,
    int is_info, int in_level,
    struct myrecords_struct *myrec,
    Dwarf_Error *error)
{
    int res = DW_DLV_OK;
    Dwarf_Die cur_die=in_die;
    Dwarf_Die child = 0;

    myrecord_data_for_die(myrec,in_die);

    /* Loop on a list of siblings */
    for (;;) {
        Dwarf_Die sib_die = 0;

        /* Depending on your goals, the in_level,
        and the DW_TAG of cur_die, you may want
        to skip the dwarf_child call. We descend
        the DWARF-standard way of depth-first. */
        res = dwarf_child(cur_die,&child,error);
        if (res == DW_DLV_ERROR) {
            printf("Error in dwarf_child , level %d \n",in_level);
            exit(EXIT_FAILURE);
        }
        if (res == DW_DLV_OK) {
            record_die_and_siblings_e(dbg,child,is_info,
                in_level+1,myrec,error);
            /* No longer need 'child' die. */
            dwarf_dealloc(dbg,child,DW_DLA_DIE);
            child = 0;
        }
        /* res == DW_DLV_NO_ENTRY or DW_DLV_OK */
        res = dwarf_siblingof_c(cur_die,&sib_die,error);
        if (res == DW_DLV_ERROR) {
            exit(EXIT_FAILURE);
        }
        if (res == DW_DLV_NO_ENTRY) {

```

```

        /* Done at this level. */
        break;
    }
    /* res == DW_DLV_OK */
    if (cur_die != in_die) {
        dwarf_dealloc(dbg, cur_die, DW_DLA_DIE);
        cur_die = 0;
    }
    cur_die = sib_die;
    myrecord_data_for_die(myrec, sib_die);
}
return;
}

/* Assuming records properly initialized for your use. */
int examplecuhdre(Dwarf_Debug dbg,
    struct myrecords_struct *myrec,
    Dwarf_Error *error)
{
    Dwarf_Unsigned abbrev_offset = 0;
    Dwarf_Half address_size = 0;
    Dwarf_Half version_stamp = 0;
    Dwarf_Half offset_size = 0;
    Dwarf_Half extension_size = 0;
    Dwarf_Sig8 signature;
    Dwarf_Unsigned typeoffset = 0;
    Dwarf_Unsigned next_cu_header = 0;
    Dwarf_Half header_cu_type = 0;
    Dwarf_Bool is_info = TRUE;
    int res = 0;

    while(!my_needed_data_exists(myrec)) {
        Dwarf_Die cu_die = 0;
        Dwarf_Unsigned cu_header_length = 0;

        memset(&signature, 0, sizeof(signature));
        res = dwarf_next_cu_header_e(dbg, is_info,
            &cu_die,
            &cu_header_length,
            &version_stamp, &abbrev_offset,
            &address_size, &offset_size,
            &extension_size, &signature,
            &typeoffset, &next_cu_header,
            &header_cu_type, error);
        if (res == DW_DLV_ERROR) {
            return res;
        }
        if (res == DW_DLV_NO_ENTRY) {
            if (is_info == TRUE) {
                /* Done with .debug_info, now check for
                 * .debug_types. */
                is_info = FALSE;
                continue;
            }
            /* No more CUs to read! Never found
             * what we were looking for in either
             * .debug_info or .debug_types. */
            return res;
        }
        /* We have the cu_die .
         * New in v0.9.0 because the connection of
         * the CU_DIE to the CU header is clear
         * in the argument list.
         */
        record_die_and_siblings_e(dbg, cu_die, is_info,
            0, myrec, error);
        dwarf_dealloc_die(cu_die);
    }
    /* Found what we looked for */
    return DW_DLV_OK;
}

```

9.51 Example walking CUs(d)

Example accessing all CUs looking for specific items(d).

Example accessing all CUs looking for specific items(d).

Loops through as many CUs as needed, stops and returns once a CU provides the desired data.

Assumes certain functions you write to remember the aspect of CUs that matter to you so once found in a cu my_↵ needed_data_exists() or some other function of yours can identify the correct record. (Possibly a DIE global offset. Remember to note if each DIE has is_info TRUE or FALSE so libdwarf can find the DIE properly.)

Depending on your goals in examining the DIE tree it may be helpful to maintain a DIE stack of active DIEs, pushing and popping as you make your way through the DIE levels.

We assume that on a serious error we will give up (for simplicity here).

We assume the caller to examplecuhdrd() will know what to retrieve (when we return DW_DLV_OK from examplecuhdrd() and that myrecords points to a record with all the data needed by my_needed_data_exists() and recorded by myrecord_data_for_die().

```

*/
struct myrecords_struct *myrecords;
void myrecord_data_for_die(struct myrecords_struct *myrecords,
    Dwarf_Die d);
int my_needed_data_exists(struct myrecords_struct *myrecords);

/* Loop on DIE tree. */
static void
record_die_and_siblingsd(Dwarf_Debug dbg, Dwarf_Die in_die,
    int is_info, int in_level,
    struct myrecords_struct *myrec,
    Dwarf_Error *error)
{
    int res = DW_DLV_OK;
    Dwarf_Die cur_die=in_die;
    Dwarf_Die child = 0;

    myrecord_data_for_die(myrec,in_die);

    /* Loop on a list of siblings */
    for (;;) {
        Dwarf_Die sib_die = 0;

        /* Depending on your goals, the in_level,
           and the DW_TAG of cur_die, you may want
           to skip the dwarf_child call. */
        res = dwarf_child(cur_die,&child,error);
        if (res == DW_DLV_ERROR) {
            printf("Error in dwarf_child , level %d \n",in_level);
            exit(EXIT_FAILURE);
        }
        if (res == DW_DLV_OK) {
            record_die_and_siblingsd(dbg,child,is_info,
                in_level+1,myrec,error);
            /* No longer need 'child' die. */
            dwarf_dealloc(dbg,child,DW_DLA_DIE);
            child = 0;
        }
        /* res == DW_DLV_NO_ENTRY or DW_DLV_OK */
        res = dwarf_siblingof_b(dbg,cur_die,is_info,&sib_die,error);
        if (res == DW_DLV_ERROR) {
            exit(EXIT_FAILURE);
        }
        if (res == DW_DLV_NO_ENTRY) {
            /* Done at this level. */
            break;
        }
        /* res == DW_DLV_OK */
        if (cur_die != in_die) {
            dwarf_dealloc(dbg,cur_die,DW_DLA_DIE);
            cur_die = 0;
        }
        cur_die = sib_die;
        myrecord_data_for_die(myrec,sib_die);
    }
    return;
}

/* Assuming records properly initialized for your use. */
int examplecuhdrd(Dwarf_Debug dbg,
    struct myrecords_struct *myrec,
    Dwarf_Error *error)
{
    Dwarf_Unsigned abbrev_offset = 0;
    Dwarf_Half address_size = 0;
    Dwarf_Half version_stamp = 0;
    Dwarf_Half offset_size = 0;
    Dwarf_Half extension_size = 0;
    Dwarf_Sig8 signature;
    Dwarf_Unsigned typeoffset = 0;

```

```

Dwarf_Unsigned next_cu_header = 0;
Dwarf_Half     header_cu_type = 0;
Dwarf_Bool     is_info = TRUE;
int            res = 0;

while(!my_needed_data_exists(myrec)) {
    Dwarf_Die no_die = 0;
    Dwarf_Die cu_die = 0;
    Dwarf_Unsigned cu_header_length = 0;

    memset(&signature, 0, sizeof(signature));
    res = dwarf_next_cu_header_d(dbg, is_info, &cu_header_length,
                                &version_stamp, &abbrev_offset,
                                &address_size, &offset_size,
                                &extension_size, &signature,
                                &typeoffset, &next_cu_header,
                                &header_cu_type, error);
    if (res == DW_DLV_ERROR) {
        return res;
    }
    if (res == DW_DLV_NO_ENTRY) {
        if (is_info == TRUE) {
            /* Done with .debug_info, now check for
             * .debug_types. */
            is_info = FALSE;
            continue;
        }
        /* No more CUs to read! Never found
         * what we were looking for in either
         * .debug_info or .debug_types. */
        return res;
    }
    /* The CU will have a single sibling, a cu_die.
     * It is essential to call this right after
     * a call to dwarf_next_cu_header_d() because
     * there is no explicit connection provided to
     * dwarf_siblingof_b(), which returns a DIE
     * from whatever CU was last accessed by
     * dwarf_next_cu_header_d()!
     * The lack of explicit connection was a
     * design mistake in the API (made in 1992). */

    res = dwarf_siblingof_b(dbg, no_die, is_info,
                            &cu_die, error);
    if (res == DW_DLV_ERROR) {
        return res;
    }
    if (res == DW_DLV_NO_ENTRY) {
        /* Impossible */
        exit(EXIT_FAILURE);
    }
    record_die_and_siblingsd(dbg, cu_die, is_info,
                             0, myrec, error);
    dwarf_dealloc_die(cu_die);
}
/* Found what we looked for */
return DW_DLV_OK;
}

```

9.52 Using dwarf_offdie_b()

Example accessing a DIE by its offset.

Example accessing a DIE by its offset.

```

/*
int example6(Dwarf_Debug dbg, Dwarf_Off die_offset,
             Dwarf_Bool is_info,
             Dwarf_Error *error)
{
    Dwarf_Die return_die = 0;
    int res = 0;

    res = dwarf_offdie_b(dbg, die_offset, is_info, &return_die, error);
    if (res != DW_DLV_OK) {
        /* res could be NO ENTRY or ERROR, so no
         * dealloc necessary. */
        return res;
    }
    /* Use return_die here. */
    dwarf_dealloc_die(return_die);
}

```

```

    /* return_die is no longer usable for anything, we
       ensure we do not use it accidentally
       though a bit silly here given the return_die
       goes out of scope... */
    return_die = 0;
    return res;
}

```

9.53 Using dwarf_offset_given_die()

Example finding the section offset of a DIE.

Example finding the section offset of a DIE.

Here finding the offset of a CU-DIE.

```

/*
int example7(Dwarf_Debug dbg, Dwarf_Die in_die,
             Dwarf_Bool is_info,
             Dwarf_Error * error)
{
    int res = 0;
    Dwarf_Off cudieoff = 0;
    Dwarf_Die cudie = 0;

    res = dwarf_CU_dieoffset_given_die(in_die, &cudieoff, error);
    if (res != DW_DLV_OK) {
        /* FAIL */
        return res;
    }
    res = dwarf_offdie_b(dbg, cudieoff, is_info, &cudie, error);
    if (res != DW_DLV_OK) {
        /* FAIL */
        return res;
    }
    /* do something with cu_die */
    dwarf_dealloc_die(cudie);
    return res;
}

```

9.54 Using dwarf_attrlist()

Example Calling dwarf_attrlist()

Example Calling dwarf_attrlist()

```

/*
int example8(Dwarf_Debug dbg, Dwarf_Die somedie, Dwarf_Error *error)
{
    Dwarf_Signed atcount = 0;
    Dwarf_Attribute *atlist = 0;
    int errv = 0;
    Dwarf_Signed i = 0;

    errv = dwarf_attrlist(somedie, &atlist, &atcount, error);
    if (errv != DW_DLV_OK) {
        return errv;
    }
    for (i = 0; i < atcount; ++i) {
        /* use atlist[i] */
        dwarf_dealloc_attribute(atlist[i]);
        atlist[i] = 0;
    }
    dwarf_dealloc(dbg, atlist, DW_DLA_LIST);
    return DW_DLV_OK;
}

```

9.55 Using dwarf_offset_list()

Example using dwarf_offset_list.

Example using dwarf_offset_list.

An example calling dwarf_offset_list

Parameters

<i>dbg</i>	the Dwarf_Debug of interest
<i>dieoffset</i>	The section offset of a Dwarf_Die
<i>is_info</i>	Pass in TRUE if the dieoffset is for the .debug_info section, else pass in FALSE meaning the dieoffset is for the DWARF4 .debug_types section.
<i>error</i>	The usual error detail return.

Returns

Returns DW_DLV_OK etc

```

/*
int exampleoffset_list(Dwarf_Debug dbg, Dwarf_Off dieoffset,
    Dwarf_Bool is_info, Dwarf_Error * error)
{
    Dwarf_Unsigned offcnt = 0;
    Dwarf_Off *offbuf = 0;
    int errv = 0;
    Dwarf_Unsigned i = 0;

    errv = dwarf_offset_list(dbg, dieoffset, is_info,
        &offbuf, &offcnt, error);
    if (errv != DW_DLV_OK) {
        return errv;
    }
    for (i = 0; i < offcnt; ++i) {
        /* use offbuf[i] */
        /* No need to free the offbuf entry, it
           is just an offset value. */
    }
    dwarf_dealloc(dbg, offbuf, DW_DLA_LIST);
    return DW_DLV_OK;
}

```

9.56 Documenting Form_Block

Example documents Form_Block content.

Example documents Form_Block content.

Used with certain location information functions, a frame expression function, expanded frame instructions, and DW_FORM_block<> functions and more.

See also

[dwarf_formblock](#)

[Dwarf_Block_s](#)

```

struct Dwarf_Block_s fields {
    Dwarf_Unsigned bl_len;
        Length of block bl_data points at

    Dwarf_Ptr      bl_data;
        Uninterpreted data bytes

    Dwarf_Small    bl_from_loclist;
        See libdwarf.h DW_LKIND, defaults to
        DW_LKIND_expression and except in certain
        location expressions the field is ignored.

    Dwarf_Unsigned bl_section_offset;
        Section offset of what bl_data points to

```


9.57 Using dwarf_discr_list()

Example using dwarf_discr_list, dwarf_formblock.

Example using dwarf_discr_list, dwarf_formblock.

An example calling dwarf_get_form_class, dwarf_discr_list, and dwarf_formblock. and the dwarf_deallocs applicable.

See also

[dwarf_discr_list](#)

[dwarf_get_form_class](#)

[dwarf_formblock](#)

Parameters

<i>dw_dbg</i>	The applicable Dwarf_Debug
<i>dw_die</i>	The applicable Dwarf_Die
<i>dw_attr</i>	The applicable Dwarf_Attribute
<i>dw_attrnum, The</i>	attribute number passed in to shorten this example a bit.
<i>dw_isunsigned, The</i>	attribute number passed in to shorten this example a bit.
<i>dw_theform, The</i>	form number passed in to shorten this example a bit.
<i>dw_error</i>	The usual error pointer.

Returns

Returns DW_DLV_OK etc

```

/*
int example_discr_list(Dwarf_Debug dbg,
    Dwarf_Die die,
    Dwarf_Attribute attr,
    Dwarf_Half attrnum,
    Dwarf_Bool isunsigned,
    Dwarf_Half theform,
    Dwarf_Error *error)
{
    /* The example here assumes that
       attribute attr is a DW_AT_discr_list.
       isunsigned should be set from the signedness
       of the parent of 'die' per DWARF rules for
       DW_AT_discr_list. */
    enum Dwarf_Form_Class fc = DW_FORM_CLASS_UNKNOWN;
    Dwarf_Half version = 0;
    Dwarf_Half offset_size = 0;
    int wres = 0;

    wres = dwarf_get_version_of_die(die, &version, &offset_size);
    if (wres != DW_DLV_OK) {
        /* FAIL */
        return wres;
    }
    fc = dwarf_get_form_class(version, attrnum, offset_size, theform);
    if (fc == DW_FORM_CLASS_BLOCK) {
        int fres = 0;
        Dwarf_Block *tempb = 0;
        fres = dwarf_formblock(attr, &tempb, error);
        if (fres == DW_DLV_OK) {
            Dwarf_Disc_Head h = 0;
            Dwarf_Unsigned u = 0;
            Dwarf_Unsigned arraycount = 0;
            int sres = 0;

            sres = dwarf_discr_list(dbg,
                (Dwarf_Small *)tempb->bl_data,
                tempb->bl_len,

```

```

        &h,&arraycount,error);
    if (sres == DW_DLV_NO_ENTRY) {
        /* Nothing here. */
        dwarf_dealloc(dbg, tempb, DW_DLA_BLOCK);
        return sres;
    }
    if (sres == DW_DLV_ERROR) {
        /* FAIL . */
        dwarf_dealloc(dbg, tempb, DW_DLA_BLOCK);
        return sres ;
    }
    for (u = 0; u < arraycount; u++) {
        int u2res = 0;
        Dwarf_Half dtype = 0;
        Dwarf_Signed dlow = 0;
        Dwarf_Signed dhigh = 0;
        Dwarf_Unsigned ulow = 0;
        Dwarf_Unsigned uhigh = 0;

        if (isunsigned) {
            u2res = dwarf_discr_entry_u(h,u,
                &dtype,&ulow,&uhigh,error);
        } else {
            u2res = dwarf_discr_entry_s(h,u,
                &dtype,&dlow,&dhigh,error);
        }
        if (u2res == DW_DLV_ERROR) {
            /* Something wrong */
            dwarf_dealloc(dbg,h,DW_DLA_DSC_HEAD);
            dwarf_dealloc(dbg, tempb, DW_DLA_BLOCK);
            return u2res ;
        }
        if (u2res == DW_DLV_NO_ENTRY) {
            /* Impossible. u < arraycount. */
            dwarf_dealloc(dbg,h,DW_DLA_DSC_HEAD);
            dwarf_dealloc(dbg, tempb, DW_DLA_BLOCK);
            return u2res;
        }
        /* Do something with dtype, and whichever
           of ulow, uhigh,dlow,dhigh got set.
           Probably save the values somewhere.
           Simple casting of dlow to ulow (or vice versa)
           will not get the right value due to the nature
           of LEB values. Similarly for uhigh, dhigh.
           One must use the right call.  */
    }
    dwarf_dealloc(dbg,h,DW_DLA_DSC_HEAD);
    dwarf_dealloc(dbg, tempb, DW_DLA_BLOCK);
}
return DW_DLV_OK;
}

```

9.58 Location/expression access

Example using DWARF2-5 loclists and loc-expressions.

Example using DWARF2-5 loclists and loc-expressions.

Valid for DWARF2 and later DWARF.

This example simply *assumes* the attribute has a form which relates to location lists or location expressions. Use [dwarf_get_form_class\(\)](#) to determine if this attribute fits. Use [dwarf_get_version_of_die\(\)](#) to help get the data you need.

See also

[dwarf_get_form_class](#)

[dwarf_get_version_of_die](#)

[Reading a location expression](#)

```

*/
int example_loclistcv5(Dwarf_Attribute someattr,
    Dwarf_Error *error)
{
    Dwarf_Unsigned lcount = 0;
    Dwarf_Loc_Head_c loclist_head = 0;
    int lres = 0;

    lres = dwarf_get_loclist_c(someattr, &loclist_head,
        &lcount, error);
    if (lres == DW_DLV_OK) {
        Dwarf_Unsigned i = 0;

        /* Before any return remember to call
           dwarf_loc_head_c_dealloc(loclist_head); */
        for (i = 0; i < lcount; ++i) {
            Dwarf_Small loclist_lkind = 0;
            Dwarf_Small lle_value = 0;
            Dwarf_Unsigned rawval1 = 0;
            Dwarf_Unsigned rawval2 = 0;
            Dwarf_Bool debug_addr_unavailable = FALSE;
            Dwarf_Addr lopc = 0;
            Dwarf_Addr hipc = 0;
            Dwarf_Unsigned loclist_expr_op_count = 0;
            Dwarf_Locdesc_c locdesc_entry = 0;
            Dwarf_Unsigned expression_offset = 0;
            Dwarf_Unsigned locdesc_offset = 0;

            lres = dwarf_get_locdesc_entry_d(loclist_head,
                i,
                &lle_value,
                &rawval1, &rawval2,
                &debug_addr_unavailable,
                &lopc, &hipc,
                &loclist_expr_op_count,
                &locdesc_entry,
                &loclist_lkind,
                &expression_offset,
                &locdesc_offset,
                error);
            if (lres == DW_DLV_OK) {
                Dwarf_Unsigned j = 0;
                int opres = 0;
                Dwarf_Small op = 0;

                for (j = 0; j < loclist_expr_op_count; ++j) {
                    Dwarf_Unsigned opd1 = 0;
                    Dwarf_Unsigned opd2 = 0;
                    Dwarf_Unsigned opd3 = 0;
                    Dwarf_Unsigned offsetforbranch = 0;

                    opres = dwarf_get_location_op_value_c(
                        locdesc_entry, j, &op,
                        &opd1, &opd2, &opd3,
                        &offsetforbranch,
                        error);
                    if (opres == DW_DLV_OK) {
                        /* Do something with the operators.
                           Usually you want to use opd1,2,3
                           as appropriate. Calculations
                           involving base addresses etc
                           have already been incorporated
                           in opd1,2,3. */
                    } else {
                        dwarf_dealloc_loc_head_c(loclist_head);
                        /*Something is wrong. */
                        return opres;
                    }
                }
            } else {
                /* Something is wrong. Do something. */
                dwarf_dealloc_loc_head_c(loclist_head);
                return lres;
            }
        }
    }
    /* Always call dwarf_loc_head_c_dealloc()
       to free all the memory associated with loclist_head. */
    dwarf_dealloc_loc_head_c(loclist_head);
    loclist_head = 0;
    return lres;
}

```

9.59 Reading a location expression

Example getting details of a location expression.

Example getting details of a location expression.

See also

[Location/expression access](#)

```

*/
int example_locexpr(Dwarf_Debug dbg, Dwarf_Ptr expr_bytes,
    Dwarf_Unsigned expr_len,
    Dwarf_Half addr_size,
    Dwarf_Half offset_size,
    Dwarf_Half version,
    Dwarf_Error*error)
{
    Dwarf_Loc_Head_c head = 0;
    Dwarf_Locdesc_c locentry = 0;
    int res2 = 0;
    Dwarf_Unsigned rawlopc = 0;
    Dwarf_Unsigned rawhipc = 0;
    Dwarf_Bool debug_addr_unavail = FALSE;
    Dwarf_Unsigned lopc = 0;
    Dwarf_Unsigned hipc = 0;
    Dwarf_Unsigned ulistlen = 0;
    Dwarf_Unsigned ulocentry_count = 0;
    Dwarf_Unsigned section_offset = 0;
    Dwarf_Unsigned locdesc_offset = 0;
    Dwarf_Small lle_value = 0;
    Dwarf_Small loclist_source = 0;
    Dwarf_Unsigned i = 0;

    res2 = dwarf_loclist_from_expr_c(dbg,
        expr_bytes, expr_len,
        addr_size,
        offset_size,
        version,
        &head,
        &ulistlen,
        error);
    if (res2 != DW_DLV_OK) {
        return res2;
    }
    /* These are a location expression, not loclist.
       So we just need the 0th entry. */
    res2 = dwarf_get_locdesc_entry_d(head,
        0, /* Data from 0th because it is a loc expr,
           there is no list */
        &lle_value,
        &rawlopc, &rawhipc, &debug_addr_unavail, &lopc, &hipc,
        &ulocentry_count, &locentry,
        &loclist_source, &section_offset, &locdesc_offset,
        error);
    if (res2 == DW_DLV_ERROR) {
        dwarf_dealloc_loc_head_c(head);
        return res2;
    } else if (res2 == DW_DLV_NO_ENTRY) {
        dwarf_dealloc_loc_head_c(head);
        return res2;
    }
    /* ASSERT: ulistlen == 1 */
    for (i = 0; i < ulocentry_count; ++i) {
        Dwarf_Small op = 0;
        Dwarf_Unsigned opd1 = 0;
        Dwarf_Unsigned opd2 = 0;
        Dwarf_Unsigned opd3 = 0;
        Dwarf_Unsigned offsetforbranch = 0;

        res2 = dwarf_get_location_op_value_c(locentry,
            i, &op, &opd1, &opd2, &opd3,
            &offsetforbranch,
            error);
        /* Do something with the expression operator and operands */
        if (res2 != DW_DLV_OK) {
            dwarf_dealloc_loc_head_c(head);
            return res2;
        }
    }
    dwarf_dealloc_loc_head_c(head);
    return DW_DLV_OK;
}

```

9.60 Using dwarf_srclines_b()

Example using [dwarf_srclines_b\(\)](#)

Example using [dwarf_srclines_b\(\)](#)

An example calling dwarf_srclines_b

dwarf_srclines_dealloc_b dwarf_srclines_from_linecontext dwarf_srclines_files_indexes dwarf_srclines_files_↔
data_b dwarf_srclines_two_level_from_linecontext

Parameters

<i>path</i>	Path to an object we wish to open.
<i>error</i>	Allows passing back error details to the caller.

Returns

Return DW_DLV_OK etc.

```

*/
int examplec(Dwarf_Die cu_die, Dwarf_Error *error)
{
    /* EXAMPLE: DWARF2-DWARF5 access. */
    Dwarf_Line *linebuf = 0;
    Dwarf_Signed linecount = 0;
    Dwarf_Line *linebuf_actuall = 0;
    Dwarf_Signed linecount_actuall = 0;
    Dwarf_Line_Context line_context = 0;
    Dwarf_Small table_count = 0;
    Dwarf_Unsigned lineversion = 0;
    int sres = 0;
    /* ... */
    /* we use 'return' here to signify we can do nothing more
       at this point in the code. */
    sres = dwarf_srclines_b(cu_die, &lineversion,
        &table_count, &line_context, error);
    if (sres != DW_DLV_OK) {
        /* Handle the DW_DLV_NO_ENTRY or DW_DLV_ERROR
           No memory was allocated so there nothing
           to dealloc here. */
        return sres;
    }
    if (table_count == 0) {
        /* A line table with no actual lines. */
        /*...do something, see dwarf_srclines_files_count()
           etc below. */

        dwarf_srclines_dealloc_b(line_context);
        /* All the memory is released, the line_context
           and linebuf zeroed now
           as a reminder they are stale. */
        linebuf = 0;
        line_context = 0;
    } else if (table_count == 1) {
        Dwarf_Signed i = 0;
        Dwarf_Signed baseindex = 0;
        Dwarf_Signed file_count = 0;
        Dwarf_Signed endindex = 0;
        /* Standard dwarf 2,3,4, or 5 line table */
        /* Do something. */

        /* First let us index through all the files listed
           in the line table header. */
        sres = dwarf_srclines_files_indexes(line_context,
            &baseindex, &file_count, &endindex, error);
        if (sres != DW_DLV_OK) {
            /* Something badly wrong! */
            return sres;
        }
        /* Works for DWARF2,3,4 (one-based index)
           and DWARF5 (zero-based index) */
        for (i = baseindex; i < endindex; i++) {
            Dwarf_Unsigned dirindex = 0;

```

```

Dwarf_Unsigned modtime = 0;
Dwarf_Unsigned flength = 0;
Dwarf_Form_Data16 *md5data = 0;
int vres = 0;
const char *name = 0;

vres = dwarf_srclines_files_data_b(line_context, i,
    &name, &dirindex, &modtime, &flength,
    &md5data, error);
if (vres != DW_DLV_OK) {
    /* something very wrong. */
    return vres;
}
/* do something */
}

/* For this case where we have a line table we will likely
   wish to get the line details: */
sres = dwarf_srclines_from_linecontext(line_context,
    &linebuf, &linecount,
    error);
if (sres != DW_DLV_OK) {
    /* Error. Clean up the context information. */
    dwarf_srclines_dealloc_b(line_context);
    return sres;
}
/* The lines are normal line table lines. */
for (i = 0; i < linecount; ++i) {
    /* use linebuf[i] */
}
dwarf_srclines_dealloc_b(line_context);
/* All the memory is released, the line_context
   and linebuf zeroed now as a reminder they are stale */
linebuf = 0;
line_context = 0;
linecount = 0;
} else {
    Dwarf_Signed i = 0;
    /* ASSERT: table_count == 2,
       Experimental two-level line table. Version 0xf006
       We do not define the meaning of this non-standard
       set of tables here. */

    /* For 'something C' (two-level line tables)
       one codes something like this
       Note that we do not define the meaning or
       use of two-level line
       tables as these are experimental, not standard DWARF. */
    sres = dwarf_srclines_two_level_from_linecontext(line_context,
        &linebuf, &linecount,
        &linebuf_actuals, &linecount_actuals,
        error);
    if (sres == DW_DLV_OK) {
        for (i = 0; i < linecount; ++i) {
            /* use linebuf[i], these are the 'logicals'
               entries. */
        }
        for (i = 0; i < linecount_actuals; ++i) {
            /* use linebuf_actuals[i], these are the
               actuals entries */
        }
        dwarf_srclines_dealloc_b(line_context);
        line_context = 0;
        linebuf = 0;
        linecount = 0;
        linebuf_actuals = 0;
        linecount_actuals = 0;
    } else if (sres == DW_DLV_NO_ENTRY) {
        /* This should be impossible, but do something. */
        /* Then Free the line_context */
        dwarf_srclines_dealloc_b(line_context);
        line_context = 0;
        linebuf = 0;
        linecount = 0;
        linebuf_actuals = 0;
        linecount_actuals = 0;
    } else {
        /* ERROR, show the error or something.
           Free the line_context. */
        dwarf_srclines_dealloc_b(line_context);
        line_context = 0;
        linebuf = 0;
        linecount = 0;
        linebuf_actuals = 0;
        linecount_actuals = 0;
    }
}
}

```

```

    return DW_DLV_OK;
}

```

9.61 Using dwarf_srclines_b() and linecontext

Example two using [dwarf_srclines_b\(\)](#), [dwarf_linesrc\(\)](#).

Example two using [dwarf_srclines_b\(\)](#), [dwarf_linesrc\(\)](#).

See also

[dwarf_srclines_b](#)

[dwarf_srclines_from_linecontext](#)

[dwarf_srclines_dealloc_b](#)

```

*/
int exempld(Dwarf_Debug dbg, Dwarf_Die somedie, Dwarf_Error *error)
{
    Dwarf_Signed count = 0;
    Dwarf_Line_Context context = 0;
    Dwarf_Line *linebuf = 0;
    Dwarf_Signed i = 0;
    Dwarf_Line *line;
    Dwarf_Small table_count = 0;
    Dwarf_Unsigned version = 0;
    int sres = 0;

    sres = dwarf_srclines_b(somedie,
        &version, &table_count, &context, error);
    if (sres != DW_DLV_OK) {
        return sres;
    }
    sres = dwarf_srclines_from_linecontext(context,
        &linebuf, &count, error);
    if (sres != DW_DLV_OK) {
        dwarf_srclines_dealloc_b(context);
        return sres;
    }
    line = linebuf;
    for (i = 0; i < count; ++line, ++i) {
        char * filename = 0;
        int lres = 0;
        Dwarf_Line dline = linebuf[i];

        lres = dwarf_linesrc(dline, &filename, error);
        if (lres != DW_DLV_OK) {
            dwarf_srclines_dealloc_b(context);
            return lres;
        }
        /* use filename */
        dwarf_dealloc(dbg, filename, DW_DLA_STRING);
    }
    dwarf_srclines_dealloc_b(context);
    return DW_DLV_OK;
}

```

9.62 Using dwarf_srcfiles()

Example getting source file names given a DIE.

Example getting source file names given a DIE.

```

*/
int examplee(Dwarf_Debug dbg, Dwarf_Die somedie, Dwarf_Error *error)
{
    /* It is an annoying historical mistake in libdwarf
       that the count is a signed value. */
    Dwarf_Signed count = 0;
    char **srcfiles = 0;
    Dwarf_Signed i = 0;

```

```

int                res = 0;
Dwarf_Line_Context line_context = 0;
Dwarf_Small        table_count = 0;
Dwarf_Unsigned     lineversion = 0;

res = dwarf_srclines_b(somedie,&lineversion,
    &table_count,&line_context,error);
if (res != DW_DLV_OK) {
    /* dwarf_finish() will dealloc srcfiles, not doing
       that here. */
    return res;
}
res = dwarf_srcfiles(somedie, &srcfiles,&count,error);
if (res != DW_DLV_OK) {
    dwarf_srclines_dealloc_b(line_context);
    return res;
}

for (i = 0; i < count; ++i) {
    Dwarf_Signed propernumber = 0;

    /* Use srcfiles[i] If you wish to print 'i'
       mostusefully
       you should reflect the numbering that
       a DW_AT_decl_file attribute would report in
       this CU. */
    if (lineversion == 5) {
        propernumber = i;
    } else {
        propernumber = i+1;
    }
    printf("File %4ld %s\n", (unsigned long)propernumber,
        srcfiles[i]);
    dwarf_dealloc(dbg, srcfiles[i], DW_DLA_STRING);
    srcfiles[i] = 0;
}
/* We could leave all dealloc to dwarf_finish() to
   handle, but this tidies up sooner. */
dwarf_dealloc(dbg, srcfiles, DW_DLA_LIST);
dwarf_srclines_dealloc_b(line_context);
return DW_DLV_OK;
}

```

9.63 Using dwarf_get_globals()

Example using global symbol names.

Example using global symbol names.

For 0.4.2 and earlier this returned .debug_pubnames content. As of version 0.5.0 (October 2022) this returns .debug_pubnames (if it exists) and the relevant portion of .debug_names (if .debug_names exists) data.

```

*/
int examplef(Dwarf_Debug dbg,Dwarf_Error *error)
{
    Dwarf_Signed count = 0;
    Dwarf_Global *globs = 0;
    Dwarf_Signed i = 0;
    int res = 0;

    res = dwarf_get_globals(dbg, &globs,&count, error);
    if (res != DW_DLV_OK) {
        return res;
    }
    for (i = 0; i < count; ++i) {
        /* use globs[i] */
        char *name = 0;
        res = dwarf_globname(globs[i],&name,error);
        if (res != DW_DLV_OK) {
            dwarf_globals_dealloc(dbg,globs,count);
            return res;
        }
    }
    dwarf_globals_dealloc(dbg, globs, count);
    return DW_DLV_OK;
}

```


9.64 Using dwarf_globals_by_type()

Example reading .debug_pubtypes.

Example reading .debug_pubtypes.

The .debug_pubtypes section was in DWARF4, it could appear as an extension in other DWARF versions.. In libdwarf 0.5.0 and earlier the function `dwarf_get_pubtypes()` was used instead.

```
*/
int exampleg(Dwarf_Debug dbg, Dwarf_Error *error)
{
    Dwarf_Signed count = 0;
    Dwarf_Global *types = 0;
    Dwarf_Signed i = 0;
    int res = 0;

    res = dwarf_globals_by_type(dbg, DW_GL_PUBTYPES,
                              &types, &count, error);
    /* Alternatively the 0.5.0 and earlier call:
       res=dwarf_get_pubtypes(dbg, &types, &count, error); */
    if (res != DW_DLV_OK) {
        return res;
    }
    for (i = 0; i < count; ++i) {
        /* use types[i] */
    }
    dwarf_globals_dealloc(dbg, types, count);
    return DW_DLV_OK;
}
```

9.65 Reading .debug_weaknames (nonstandard)

Example. weaknames was IRIX/MIPS only.

Example. weaknames was IRIX/MIPS only.

This section is an SGI/MIPS extension, not created by modern compilers.

```
*/
int exampleh(Dwarf_Debug dbg, Dwarf_Error *error)
{
    Dwarf_Signed count = 0;
    Dwarf_Global *weaknames = 0;
    Dwarf_Signed i = 0;
    int res = 0;

    res = dwarf_globals_by_type(dbg, DW_GL_WEAKS,
                              &weaknames, &count, error);
    if (res != DW_DLV_OK) {
        return res;
    }
    for (i = 0; i < count; ++i) {
        /* use weaknames[i] */
    }
    dwarf_globals_dealloc(dbg, weaknames, count);
    return DW_DLV_OK;
}
```

9.66 Reading .debug_funcnames (nonstandard)

Example. funcnames was IRIX/MIPS only.

Example. funcnames was IRIX/MIPS only.

This section is an SGI/MIPS extension, not created by modern compilers.

```
*/
int examplej(Dwarf_Debug dbg, Dwarf_Error *error)
{
    Dwarf_Signed count = 0;
    Dwarf_Global *funcnames = 0;
    Dwarf_Signed i = 0;
    int res = 0;

    res = dwarf_globals_by_type(dbg, DW_GL_FUNCNAMES,
                              &funcnames, &count, error);
    if (res != DW_DLV_OK) {
        return res;
    }
    for (i = 0; i < count; ++i) {
        /* use funcnames[i] */
    }
    dwarf_globals_dealloc(dbg, funcnames, count);
    return DW_DLV_OK;
}
```

```

Dwarf_Signed count = 0;
Dwarf_Global *funcs = 0;
Dwarf_Signed i = 0;
int fres = 0;

fres = dwarf_globals_by_type(dbg, DW_GL_FUNCS,
    &funcs, &count, error);
if (fres != DW_DLV_OK) {
    return fres;
}
for (i = 0; i < count; ++i) {
    /* use funcs[i] */
}
dwarf_globals_dealloc(dbg, funcs, count);
return DW_DLV_OK;
}

```

9.67 Reading .debug_types (nonstandard)

Example .debug_types was IRIX/MIPS only.

Example .debug_types was IRIX/MIPS only.

This section is an SGI/MIPS extension, not created by modern compilers.

```

*/
int example1(Dwarf_Debug dbg, Dwarf_Error *error)
{
    Dwarf_Signed count = 0;
    Dwarf_Global *types = 0;
    Dwarf_Signed i = 0;
    int res = 0;

    res = dwarf_globals_by_type(dbg, DW_GL_TYPES,
        &types, &count, error);
    if (res != DW_DLV_OK) {
        return res;
    }
    for (i = 0; i < count; ++i) {
        /* use types[i] */
    }
    dwarf_globals_dealloc(dbg, types, count);
    return DW_DLV_OK;
}

```

9.68 Reading .debug_varnames data (nonstandard)

Example .debug_varnames was IRIX/MIPS only.

Example .debug_varnames was IRIX/MIPS only.

This section is an SGI/MIPS extension, not created by modern compilers.

```

*/
int example2(Dwarf_Debug dbg, Dwarf_Error *error)
{
    Dwarf_Signed count = 0;
    Dwarf_Global *vars = 0;
    Dwarf_Signed i = 0;
    int res = 0;

    res = dwarf_globals_by_type(dbg, DW_GL_VARS,
        &vars, &count, error);
    if (res != DW_DLV_OK) {
        return res;
    }
    for (i = 0; i < count; ++i) {
        /* use vars[i] */
    }
    dwarf_globals_dealloc(dbg, vars, count);
    return DW_DLV_OK;
}

```

9.69 Reading .debug_names data

Example access to .debug_names.

Example access to .debug_names.

This is accessing DWARF5 .debug_names, a section intended to provide fast access to DIEs.

It bears a strong resemblance to what libdwarf does in dwarf_global.c.

Making this a single (long) function here, though that is not how libdwarf or dwarfdump are written.

That is just one possible sort of access. There are many, and we would love to hear suggestions for specific new API functions in the library.

There is a wealth of information in .debug_names and the following is all taken care of for you by [dwarf_get_globals\(\)](#).

```

*/
#define MAXPAIRS 8 /* The standard defines 5.*/
int exampledebugnames(Dwarf_Debug dbg,
    Dwarf_Unsigned *dnentrycount,
    Dwarf_Error *error)
{
    int res = DW_DLV_OK;
    Dwarf_Unsigned offset = 0;
    Dwarf_Dnames_Head dn = 0;
    Dwarf_Unsigned new_offset = 0;

    for ( ; res == DW_DLV_OK; offset = new_offset) {
        Dwarf_Unsigned comp_unit_count = 0;
        Dwarf_Unsigned local_type_unit_count = 0;
        Dwarf_Unsigned foreign_type_unit_count = 0;
        Dwarf_Unsigned bucket_count = 0;
        Dwarf_Unsigned name_count = 0;
        Dwarf_Unsigned abbrev_table_size = 0;
        Dwarf_Unsigned entry_pool_size = 0;
        Dwarf_Unsigned augmentation_string_size = 0;
        char *aug_string = 0;
        Dwarf_Unsigned section_size = 0;
        Dwarf_Half table_version = 0;
        Dwarf_Half offset_size = 0;
        Dwarf_Unsigned i = 0;

        res = dwarf_dnames_header(dbg, offset, &dn,
            &new_offset, error);
        if (res == DW_DLV_ERROR) {
            /* Something wrong. */
            return res;
        }
        if (res == DW_DLV_NO_ENTRY) {
            /* Done. Normal end of the .debug_names section. */
            break;
        }
        *dnentrycount += 1;

        res = dwarf_dnames_sizes(dn, &comp_unit_count,
            &local_type_unit_count,
            &foreign_type_unit_count,
            &bucket_count,
            &name_count, &abbrev_table_size,
            &entry_pool_size, &augmentation_string_size,
            &aug_string,
            &section_size, &table_version,
            &offset_size,
            error);
        if (res != DW_DLV_OK) {
            /* Something wrong. */
            return res;
        }
        /* name indexes start with one */
        for (i = 1 ; i <= name_count; ++i) {
            Dwarf_Unsigned j = 0;
            /* dnames_name data */
            Dwarf_Unsigned bucketnum = 0;
            Dwarf_Unsigned hashvalunsign = 0;
            Dwarf_Unsigned offset_to_debug_str = 0;
            char *ptrtostr = 0;
            Dwarf_Unsigned offset_in_entrypool = 0;

```

```

Dwarf_Unsigned abbrev_code = 0;
Dwarf_Half abbrev_tag = 0;
Dwarf_Half nt_idxattr_array[MAXPAIRS];
Dwarf_Half nt_form_array[MAXPAIRS];
Dwarf_Unsigned attr_count = 0;

/* dnames_entrypool data */
Dwarf_Half tag = 0;
Dwarf_Bool single_cu_case = 0;
Dwarf_Unsigned single_cu_offset = 0;
Dwarf_Unsigned value_count = 0;
Dwarf_Unsigned index_of_abbrev = 0;
Dwarf_Unsigned offset_of_initial_value = 0;
Dwarf_Unsigned offset_next_entry_pool = 0;
Dwarf_Half idx_array[MAXPAIRS];
Dwarf_Half form_array[MAXPAIRS];
Dwarf_Unsigned offsets_array[MAXPAIRS];
Dwarf_Sig8 signatures_array[MAXPAIRS];

Dwarf_Unsigned cu_table_index = 0;
Dwarf_Unsigned tu_table_index = 0;
Dwarf_Unsigned local_die_offset = 0;
Dwarf_Unsigned parent_index = 0;
Dwarf_Sig8 parenthash;

(void)parent_index; /* avoids warning */
(void)local_die_offset; /* avoids warning */
(void)tu_table_index; /* avoids warning */
(void)cu_table_index; /* avoids warning */

memset(&parenthash, 0, sizeof(parenthash));
/* This gets us the entry pool offset we need.
   we provide idxattr and nt_form arrays (need
   not be initialized) and on return
   attr_count of those arrays are filled in.
   if attr_count < array_size then array_size
   is too small and things will not go well!
   See the count of DW_IDX entries in dwarf.h
   and make the arrays (say) 2 or more larger
   ensuring against future new DW_IDX index
   attributes..

   ptrtostring is the name in the Names Table. */
res = dwarf_dnames_name(dn, i,
&bucketnum, &hashvalunsign,
&offset_to_debug_str, &ptrtostr,
&offset_in_entrypool, &abbrev_code,
&abbrev_tag,
MAXPAIRS,
nt_idxattr_array, nt_form_array,
&attr_count, error);
if (res == DW_DLV_NO_ENTRY) {
    /* past end. Normal. */
    break;
}
if (res == DW_DLV_ERROR) {
    dwarf_dealloc_dnames(dn);
    return res;
}

/* Check attr_count < MAXPAIRS ! */
/* Now check the value of TAG to ensure it
   is something of interest as data or function.
   Plausible choices: */
switch (abbrev_tag) {
case DW_TAG_subprogram:
case DW_TAG_variable:
case DW_TAG_label:
case DW_TAG_member:
case DW_TAG_common_block:
case DW_TAG_enumerator:
case DW_TAG_namelist:
case DW_TAG_module:
    break;
default:
    /* Not data or variable DIE involved.
       Loop on the next i */
    continue;
}

/* We need the number of values for this name
   from this call. tag will match abbrev_tag. */
res = dwarf_dnames_entrypool(dn,
offset_in_entrypool,
&abbrev_code, &tag, &value_count, &index_of_abbrev,
&offset_of_initial_value,
error);

```

```

    if (res != DW_DLV_OK) {
        dwarf_dealloc_dnames(dn);
        return res;
    }

    /* This gets us an actual array of values
       as the library combines abbreviations,
       IDX attributes and values. We use
       the idx_array and form_array data
       created above. */

    res = dwarf_dnames_entrypool_values(dn,
        index_of_abbrev,
        offset_of_initial_value,
        value_count,
        idx_array,
        form_array,
        offsets_array,
        signatures_array,
        &single_cu_case, &single_cu_offset,
        &offset_next_entry_pool,
        error);
    if (res != DW_DLV_OK) {
        dwarf_dealloc_dnames(dn);
        return res;
    }
    for (j = 0; j < value_count; ++j) {
        Dwarf_Half idx = idx_array[j];

        switch(idx) {
            case DW_IDX_compile_unit:
                cu_table_index = offsets_array[j];
                break;
            case DW_IDX_die_offset:
                local_die_offset = offsets_array[j];
                break;
            /* The following are not meaningful when
               reading globals. */
            case DW_IDX_type_unit:
                tu_table_index = offsets_array[j];
                break;
            case DW_IDX_parent:
                parent_index = offsets_array[j];
                break;
            case DW_IDX_type_hash:
                parenthash = signatures_array[j];
                break;
            default:
                /* Not handled DW_IDX_GNU... */
                break;
        }
    }
    /* Now do something with the data aggregated */

}
dwarf_dealloc_dnames(dn);
}
return DW_DLV_OK;
}

```

9.70 Reading .debug_macro data (DWARF5)

Example reading DWARF5 macro data.

Example reading DWARF5 macro data.

This builds an list or some other data structure (not defined) to give an import somewhere to list the import offset and then later to enquire if the list has unexamined offsets. The code compiles but is not yet tested.

This example does not actually do the import at the correct time as this is just checking import offsets, not creating a proper full list (in the proper order) of the macros with the imports inserted. Here we find the macro context for a DIE, report those macro entries, noting any macro_import in a list loop extracting unchecked macro offsets from the list note any import in a list Of course some functions are not implemented here...

```

*/
int has_unchecked_import_in_list(void)
{

```

```

    /* Do something */
    return DW_DLV_OK;
}
Dwarf_Unsigned get_next_import_from_list(void)
{
    /* Do something */
    return 22;
}
void mark_this_offset_as_examined(
    Dwarf_Unsigned macro_unit_offset)
{
    /* do something */
    /* avoid compiler warnings. */
    (void)macro_unit_offset;
}
void add_offset_to_list(Dwarf_Unsigned offset)
{
    /* do something */
    /* avoid compiler warnings. */
    (void)offset;;
}
int examplep5(Dwarf_Die cu_die, Dwarf_Error *error)
{
    int lres = 0;
    Dwarf_Unsigned k = 0;
    Dwarf_Unsigned version = 0;
    Dwarf_Macro_Context macro_context = 0;
    Dwarf_Unsigned macro_unit_offset = 0;
    Dwarf_Unsigned number_of_ops = 0;
    Dwarf_Unsigned ops_total_byte_len = 0;
    Dwarf_Bool is_primary = TRUE;

    /* Just call once each way to test both.
       Really the second is just for imported units.*/
    for ( ; ; ) {
        if (is_primary) {
            lres = dwarf_get_macro_context(cu_die,
                &version, &macro_context,
                &macro_unit_offset,
                &number_of_ops,
                &ops_total_byte_len,
                error);
            is_primary = FALSE;
        } else {
            if (has_unchecked_import_in_list()) {
                macro_unit_offset = get_next_import_from_list();
            } else {
                /* We are done */
                break;
            }
            lres = dwarf_get_macro_context_by_offset(cu_die,
                macro_unit_offset,
                &version,
                &macro_context,
                &number_of_ops,
                &ops_total_byte_len,
                error);
            mark_this_offset_as_examined(macro_unit_offset);
        }

        if (lres == DW_DLV_ERROR) {
            /* Something is wrong. */
            return lres;
        }
        if (lres == DW_DLV_NO_ENTRY) {
            /* We are done. */
            break;
        }
        /* lres == DW_DLV_OK */
        for (k = 0; k < number_of_ops; ++k) {
            Dwarf_Unsigned section_offset = 0;
            Dwarf_Half macro_operator = 0;
            Dwarf_Half forms_count = 0;
            const Dwarf_Small *formcode_array = 0;
            Dwarf_Unsigned line_number = 0;
            Dwarf_Unsigned index = 0;
            Dwarf_Unsigned offset = 0;
            const char * macro_string = 0;
            int lres2 = 0;

            lres2 = dwarf_get_macro_op(macro_context,
                k, &section_offset, &macro_operator,
                &forms_count, &formcode_array, error);
            if (lres2 != DW_DLV_OK) {
                /* Some error. Deal with it */
                dwarf_dealloc_macro_context(macro_context);
                return lres2;
            }
        }
    }
}

```

```

    }
    switch(macro_operator) {
    case 0:
        /* Nothing to do. */
        break;
    case DW_MACRO_end_file:
        /* Do something */
        break;
    case DW_MACRO_define:
    case DW_MACRO_undef:
    case DW_MACRO_define_strp:
    case DW_MACRO_undef_strp:
    case DW_MACRO_define_strx:
    case DW_MACRO_undef_strx:
    case DW_MACRO_define_sup:
    case DW_MACRO_undef_sup: {
        lres2 = dwarf_get_macro_defundef(macro_context,
            k,
            &line_number,
            &index,
            &offset,
            &forms_count,
            &macro_string,
            error);
        if (lres2 != DW_DLV_OK) {
            /* Some error. Deal with it */
            dwarf_dealloc_macro_context(macro_context);
            return lres2;
        }
        /* do something */
    }
    break;
    case DW_MACRO_start_file: {
        lres2 = dwarf_get_macro_startend_file(macro_context,
            k,&line_number,
            &index,
            &macro_string,error);
        if (lres2 != DW_DLV_OK) {
            /* Some error. Deal with it */
            dwarf_dealloc_macro_context(macro_context);
            return lres2;
        }
        /* do something */
    }
    break;
    case DW_MACRO_import: {
        lres2 = dwarf_get_macro_import(macro_context,
            k,&offset,error);
        if (lres2 != DW_DLV_OK) {
            /* Some error. Deal with it */
            dwarf_dealloc_macro_context(macro_context);
            return lres2;
        }
        add_offset_to_list(offset);
    }
    break;
    case DW_MACRO_import_sup: {
        lres2 = dwarf_get_macro_import(macro_context,
            k,&offset,error);
        if (lres2 != DW_DLV_OK) {
            /* Some error. Deal with it */
            dwarf_dealloc_macro_context(macro_context);
            return lres2;
        }
        /* do something */
    }
    break;
    default:
        /* This is an error or an omission
           in the code here. We do not
           know what to do.
           Do something appropriate, print something?. */
        break;
    }
}
dwarf_dealloc_macro_context(macro_context);
macro_context = 0;
}
return DW_DLV_OK;
}
/*

```

9.71 Reading .debug_macinfo (DWARF2-4)

Example reading .debug_macinfo, DWARF2-4.

Example reading .debug_macinfo, DWARF2-4.

```

*/

void functionusingsigned(Dwarf_Signed s) {
    /* Do something */
    /* Avoid compiler warnings. */
    (void)s;
}

int examplep2(Dwarf_Debug dbg, Dwarf_Off cur_off,
Dwarf_Error*error)
{
    Dwarf_Signed count = 0;
    Dwarf_Macro_Details *maclist = 0;
    Dwarf_Signed i = 0;
    Dwarf_Unsigned max = 500000; /* sanity limit */
    int errv = 0;

    /* This is for DWARF2, DWARF3, and DWARF4
    .debug_macinfo section only.*/
    /* Given an offset from a compilation unit,
    start at that offset (from DW_AT_macroinfo)
    and get its macro details. */
    errv = dwarf_get_macro_details(dbg, cur_off, max,
    &count, &maclist, error);
    if (errv == DW_DLV_OK) {
        for (i = 0; i < count; ++i) {
            Dwarf_Macro_Details * mentry = maclist + i;
            /* example of use */
            Dwarf_Signed lineno = mentry->dmd_lineno;
            functionusingsigned(lineno);
        }
        dwarf_dealloc(dbg, maclist, DW_DLA_STRING);
    }
    /* Loop through all the compilation units macro info from zero.
    This is not guaranteed to work because DWARF does not
    guarantee every byte in the section is meaningful:
    there can be garbage between the macro info
    for CUs. But this loop will sometimes work.
    /
    cur_off = 0;
    while((errv = dwarf_get_macro_details(dbg, cur_off, max,
    &count, &maclist, error)) == DW_DLV_OK) {
        for (i = 0; i < count; ++i) {
            Dwarf_Macro_Details * mentry = maclist + i;
            /* example of use */
            Dwarf_Signed lineno = mentry->dmd_lineno;
            functionusingsigned(lineno);
        }
        cur_off = maclist[count-1].dmd_offset + 1;
        dwarf_dealloc(dbg, maclist, DW_DLA_STRING);
    }
    return DW_DLV_OK;
}

```

9.72 Extracting fde, cie lists.

Example Opening FDE and CIE lists.

Example Opening FDE and CIE lists.

```

*/
int exampleq(Dwarf_Debug dbg, Dwarf_Error *error)
{
    Dwarf_Cie *cie_data = 0;
    Dwarf_Signed cie_count = 0;
    Dwarf_Fde *fde_data = 0;
    Dwarf_Signed fde_count = 0;
    int fres = 0;

    fres = dwarf_get_fde_list(dbg, &cie_data, &cie_count,
    &fde_data, &fde_count, error);
    if (fres != DW_DLV_OK) {
        return fres;
    }
}

```



```

    }
    /* Do something with the lists*/
    dwarf_dealloc_fde_cie_list(dbg, cie_data, cie_count,
                              fde_data, fde_count);
    return fres;
}

```

9.73 Reading the .eh_frame section

Example access to .eh_frame.

Example access to .eh_frame.

```

/*
int exemplar(Dwarf_Debug dbg, Dwarf_Addr mypcval, Dwarf_Error *error)
{
    /* Given a pc value
       for a function find the FDE and CIE data for
       the function.
       Example shows basic access to FDE/CIE plus
       one way to access details given a PC value.
       dwarf_get_fde_n() allows accessing all FDE/CIE
       data so one could build up an application-specific
       table of information if that is more useful. */
    Dwarf_Cie *cie_data = 0;
    Dwarf_Signed cie_count = 0;
    Dwarf_Fde *fde_data = 0;
    Dwarf_Signed fde_count = 0;
    int fres = 0;

    fres = dwarf_get_fde_list_eh(dbg, &cie_data, &cie_count,
                                &fde_data, &fde_count, error);
    if (fres == DW_DLV_OK) {
        Dwarf_Fde myfde = 0;
        Dwarf_Addr low_pc = 0;
        Dwarf_Addr high_pc = 0;

        fres = dwarf_get_fde_at_pc(fde_data, mypcval,
                                   &myfde, &low_pc, &high_pc,
                                   error);
        if (fres == DW_DLV_OK) {
            Dwarf_Cie mycie = 0;
            fres = dwarf_get_cie_of_fde(myfde, &mycie, error);
            if (fres == DW_DLV_ERROR) {
                return fres;
            }
            if (fres == DW_DLV_OK) {
                /* Now we can access a range of information
                   about the fde and cie applicable. */
            }
        }
        dwarf_dealloc_fde_cie_list(dbg, cie_data, cie_count,
                                   fde_data, fde_count);
        return fres;
    }
}

```

9.74 Using dwarf_expand_frame_instructions

Example using dwarf_expand_frame_instructions.

Example using dwarf_expand_frame_instructions.

```

/*
int examples(Dwarf_Cie cie,
             Dwarf_Ptr instruction, Dwarf_Unsigned len,
             Dwarf_Error *error)
{
    Dwarf_Frame_Instr_Head head = 0;
    Dwarf_Unsigned count = 0;
    int res = 0;
    Dwarf_Unsigned i = 0;

    res = dwarf_expand_frame_instructions(cie, instruction, len,

```

```

        &head,&count, error);
    if (res != DW_DLV_OK) {
        return res;
    }

    for (i = 0; i < count; ++i) {
        Dwarf_Unsigned instr_offset_in_instrs = 0;
        Dwarf_Small cfa_operation = 0;
        const char *fields_description = 0;
        Dwarf_Unsigned u0 = 0;
        Dwarf_Unsigned u1 = 0;
        Dwarf_Signed s0 = 0;
        Dwarf_Signed s1 = 0;
        Dwarf_Unsigned code_alignment_factor = 0;
        Dwarf_Signed data_alignment_factor = 0;
        Dwarf_Block expression_block;
        const char * op_name = 0;

        memset(&expression_block,0,sizeof(expression_block));
        res = dwarf_get_frame_instruction(head,i,
            &instr_offset_in_instrs,&cfa_operation,
            &fields_description,&u0,&u1,
            &s0,&s1,
            &code_alignment_factor,
            &data_alignment_factor,
            &expression_block,error);
        if (res == DW_DLV_ERROR) {
            dwarf_dealloc_frame_instr_head(head);
            return res;
        }
        if (res == DW_DLV_OK) {
            res = dwarf_get_CFA_name(cfa_operation,
                &op_name);
            if (res != DW_DLV_OK) {
                op_name = "unknown op";
            }
            printf("Instr %2lu %-22s %s\n",
                (unsigned long)i,
                op_name,
                fields_description);
            /* Do something with the various data
               as guided by the fields_description. */
        }
    }
    dwarf_dealloc_frame_instr_head(head);
    return DW_DLV_OK;
}

```

9.75 Reading string offsets section data

Example accessing the string offsets section.

Example accessing the string offsets section.

An example accessing the string offsets section

Parameters

<i>dbg</i>	The Dwarf_Debug of interest.
<i>dw_error</i>	On error <i>dw_error</i> is set to point to the error details.

Returns

DW_DLV_OK etc.

```

*/
int examplestrngoffsets(Dwarf_Debug dbg,Dwarf_Error *error)
{
    int res = 0;
    Dwarf_Str_Offsets_Table sot = 0;
    Dwarf_Unsigned wasted_byte_count = 0;
    Dwarf_Unsigned table_count = 0;

```

```

Dwarf_Error          closeerror = 0;

res = dwarf_open_str_offsets_table_access(dbg, &sot,error);
if (res == DW_DLV_NO_ENTRY) {
    /* No such table */
    return res;
}
if (res == DW_DLV_ERROR) {
    /* Something is very wrong. Print the error? */
    return res;
}
for (;;) {
    Dwarf_Unsigned unit_length = 0;
    Dwarf_Unsigned unit_length_offset = 0;
    Dwarf_Unsigned table_start_offset = 0;
    Dwarf_Half     entry_size = 0;
    Dwarf_Half     version = 0;
    Dwarf_Half     padding = 0;
    Dwarf_Unsigned table_value_count = 0;
    Dwarf_Unsigned i = 0;
    Dwarf_Unsigned table_entry_value = 0;

    res = dwarf_next_str_offsets_table(sot,
        &unit_length, &unit_length_offset,
        &table_start_offset,
        &entry_size, &version, &padding,
        &table_value_count, error);
    if (res == DW_DLV_NO_ENTRY) {
        /* We have dealt with all tables */

        break;
    }
    if (res == DW_DLV_ERROR) {
        /* Something badly wrong. Do something. */
        dwarf_close_str_offsets_table_access(sot, &closeerror);
        dwarf_dealloc_error(dbg, closeerror);
        return res;
    }
    /* One could call dwarf_str_offsets_statistics to
       get the wasted bytes so far, but we do not do that
       in this example. */
    /* Possibly print the various table-related values
       returned just above. */
    for (i=0; i < table_value_count; ++i) {
        res = dwarf_str_offsets_value_by_index(sot, i,
            &table_entry_value, error);
        if (res != DW_DLV_OK) {
            /* Something is badly wrong. Do something. */
            dwarf_close_str_offsets_table_access(sot, &closeerror);
            dwarf_dealloc_error(dbg, closeerror);
            return res;
        }
        /* Do something with the table_entry_value
           at this index. Maybe just print it.
           It is an offset in .debug_str. */
    }
}
res = dwarf_str_offsets_statistics(sot, &wasted_byte_count,
    &table_count, error);
if (res != DW_DLV_OK) {
    dwarf_close_str_offsets_table_access(sot, &closeerror);
    dwarf_dealloc_error(dbg, closeerror);
    return res;
}
res = dwarf_close_str_offsets_table_access(sot, error);
/* little can be done about any error. */
sot = 0;
return res;
}
/*

```

9.76 Reading an aranges section

Example reading .debug_aranges.

Example reading .debug_aranges.

An example accessing the .debug_aranges section. Looking all the aranges entries. This example is not searching for anything.

Parameters

<i>dbg</i>	The Dwarf_Debug of interest.
<i>dw_error</i>	On error dw_error is set to point to the error details.

Returns

DW_DLV_OK etc.

```

*/
static void cleanupbadarange(Dwarf_Debug dbg,
    Dwarf_Arange *arange, Dwarf_Signed i, Dwarf_Signed count)
{
    Dwarf_Signed k = i;

    for ( ; k < count; ++k) {
        dwarf_dealloc(dbg, arange[k], DW_DLA_ARANGE);
        arange[k] = 0;
    }
}

int exampleu(Dwarf_Debug dbg, Dwarf_Error *error)
{
    /* It is a historical accident that the count is signed.
       No negative count is possible. */
    Dwarf_Signed count = 0;
    Dwarf_Arange *arange = 0;
    int res = 0;

    res = dwarf_get_aranges(dbg, &arange, &count, error);
    if (res == DW_DLV_OK) {
        Dwarf_Signed i = 0;

        for (i = 0; i < count; ++i) {
            Dwarf_Arange ara = arange[i];
            Dwarf_Unsigned segment = 0;
            Dwarf_Unsigned segment_entry_size = 0;
            Dwarf_Addr start = 0;
            Dwarf_Unsigned length = 0;
            Dwarf_Off cu_die_offset = 0;

            res = dwarf_get_arange_info_b(ara,
                &segment, &segment_entry_size,
                &start, &length,
                &cu_die_offset, error);
            if (res != DW_DLV_OK) {
                cleanupbadarange(dbg, arange, i, count);
                dwarf_dealloc(dbg, arange, DW_DLA_LIST);
                return res;
            }
            /* Do something with ara */
            dwarf_dealloc(dbg, ara, DW_DLA_ARANGE);
            arange[i] = 0;
        }
        dwarf_dealloc(dbg, arange, DW_DLA_LIST);
    }
    return res;
}

```

9.77 Example getting .debug_ranges data

Example accessing ranges data.

Example accessing ranges data.

If have_base_addr is false there is no die (as in reading the raw .debug_ranges section) or there is some serious data corruption somewhere.

```

*/
static
void functionusingrange(Dwarf_Signed i, Dwarf_Ranges *r,
    Dwarf_Bool *have_base_addr,
    Dwarf_Unsigned *baseaddr)
{

```

```

Dwarf_Unsigned base = *baseaddr;

printf("[%4ld] ", (signed long)i);
switch(r->dwr_type) {
case DW_RANGES_ENTRY:
    printf(
        "DW_RANGES_ENTRY: raw      addr1 " PRX
        " addr2 " PRX,
        r->dwr_addr1, r->dwr_addr2);
    if (r->dwr_addr1 == r->dwr_addr2) {
        printf(" (empty range)");
    }
    printf("\n");
    if (*have_base_addr) {
        printf("      "
            "DW_RANGES_ENTRY: cooked addr1 0x%08llx"
            " addr2 " PRX "\n",
            r->dwr_addr1+base, r->dwr_addr2+base);
    }
    break;
case DW_RANGES_ADDRESS_SELECTION:
    printf(
        "Base Address      : " PRX "\n",
        r->dwr_addr2);
    *have_base_addr = TRUE;
    *baseaddr = r->dwr_addr2;
    break;
case DW_RANGES_END:
    printf(
        "DW_RANGES_END : 0,0\n");
    *have_base_addr = FALSE;
    *baseaddr = 0;
    break;
default:
    printf(
        "ERROR              : incorrect dwr_type is 0x%lx\n",
        (unsigned long)r->dwr_type);
}
}

/* On call the rangesoffset is a default zero. */
int examplev(Dwarf_Debug dbg, Dwarf_Off rangesoffset_in,
    Dwarf_Die die, Dwarf_Error*error)
{
    Dwarf_Signed    count = 0;
    Dwarf_Off       realoffset = 0;
    Dwarf_Ranges    *rangesbuf = 0;
    Dwarf_Unsigned  bytewidth = 0;
    int             res = 0;
    Dwarf_Unsigned  base_address = 0;
    Dwarf_Bool      have_base_addr = FALSE;
    Dwarf_Bool      have_rangesoffset = FALSE;
    Dwarf_Unsigned  rangesoffset = (Dwarf_Unsigned)rangesoffset_in;

    (void)have_rangesoffset;
    if (die) {
        /* Find the ranges for a specific DIE */
        res = dwarf_get_ranges_baseaddress(dbg, die, &have_base_addr,
            &base_address, &have_rangesoffset, &rangesoffset, error);
        if (res == DW_DLV_ERROR) {
            /* Just pretend not an error. */
            dwarf_dealloc_error(dbg, *error);
            *error = 0;
        }
    }
    else {
        /* To test getting all ranges and no knowledge
           of the base address (so cooked values
           cannot be definitely known unless
           the base is in the .debug_ranges entries
           themselves */
    }
    res = dwarf_get_ranges_b(dbg, rangesoffset, die,
        &realoffset,
        &rangesbuf, &count, &bytewidth, error);
    if (res != DW_DLV_OK) {
        if (res == DW_DLV_ERROR) {
            printf("ERROR dwarf_get_ranges_b %s\n",
                dwarf_errmsg(*error));
        }
        else {
            printf("NO_ENTRY dwarf_get_ranges_b\n");
        }
        return res;
    }
    {
        Dwarf_Signed i = 0;
        printf("Range group base address: " PRX

```

```

        ", offset in .debug_ranges:"
        " 0x%08llx\n",
        base_address, rangesoffset);
    for ( i = 0; i < count; ++i ) {
        Dwarf_Ranges *cur = rangesbuf+i;
        /* Use cur. */
        functionusingrange(i, cur, &have_base_addr, &base_address);
    }
    dwarf_dealloc_ranges(dbg, rangesbuf, count);
}
return DW_DLV_OK;
}

```

9.78 Reading gdbindex data

Example accessing gdbindex section data.

Example accessing gdbindex section data.

```

/*
int examplew(Dwarf_Debug dbg, Dwarf_Error *error)
{
    Dwarf_Gdbindex gindexptr = 0;
    Dwarf_Unsigned version = 0;
    Dwarf_Unsigned cu_list_offset = 0;
    Dwarf_Unsigned types_cu_list_offset = 0;
    Dwarf_Unsigned address_area_offset = 0;
    Dwarf_Unsigned symbol_table_offset = 0;
    Dwarf_Unsigned constant_pool_offset = 0;
    Dwarf_Unsigned section_size = 0;
    const char * section_name = 0;
    int res = 0;

    res = dwarf_gdbindex_header(dbg, &gindexptr,
        &version, &cu_list_offset, &types_cu_list_offset,
        &address_area_offset, &symbol_table_offset,
        &constant_pool_offset, &section_size,
        &section_name, error);
    if (res != DW_DLV_OK) {
        return res;
    }
    {
        /* do something with the data */
        Dwarf_Unsigned length = 0;
        Dwarf_Unsigned typeslength = 0;
        Dwarf_Unsigned i = 0;
        res = dwarf_gdbindex_culist_array(gindexptr,
            &length, error);
        /* Example actions. */
        if (res != DW_DLV_OK) {
            dwarf_dealloc_gdbindex(gindexptr);
            return res;
        }
        for (i = 0; i < length; ++i) {
            Dwarf_Unsigned cuoffset = 0;
            Dwarf_Unsigned culength = 0;
            res = dwarf_gdbindex_culist_entry(gindexptr,
                i, &cuoffset, &culength, error);
            if (res != DW_DLV_OK) {
                return res;
            }
            /* Do something with cuoffset, culength */
        }
        res = dwarf_gdbindex_types_culist_array(gindexptr,
            &typeslength, error);
        if (res != DW_DLV_OK) {
            dwarf_dealloc_gdbindex(gindexptr);
            return res;
        }
        for (i = 0; i < typeslength; ++i) {
            Dwarf_Unsigned cuoffset = 0;
            Dwarf_Unsigned tuoffset = 0;
            Dwarf_Unsigned type_signature = 0;
            res = dwarf_gdbindex_types_culist_entry(gindexptr,
                i, &cuoffset, &tuoffset, &type_signature, error);
            if (res != DW_DLV_OK) {
                dwarf_dealloc_gdbindex(gindexptr);
                return res;
            }
            /* Do something with cuoffset etc. */
        }
    }
}

```

```

        dwarf_dealloc_gdbindex(gindexptr);
    }
    return DW_DLV_OK;
}

```

9.79 Reading gdbindex addressarea

Example accessing gdbindex addressarea data.

Example accessing gdbindex addressarea data.

```

/*
int examplew gdbindex(Dwarf_Gdbindex gdbindex,
    Dwarf_Error *error)
{
    Dwarf_Unsigned list_len = 0;
    Dwarf_Unsigned i = 0;
    int res = 0;

    res = dwarf_gdbindex_addressarea(gdbindex, &list_len, error);
    if (res != DW_DLV_OK) {
        /* Something wrong, ignore the addressarea */
        return res;
    }
    /* Iterate through the address area. */
    for (i = 0; i < list_len; i++) {
        Dwarf_Unsigned lowpc = 0;
        Dwarf_Unsigned highpc = 0;
        Dwarf_Unsigned cu_index = 0;

        res = dwarf_gdbindex_addressarea_entry(gdbindex, i,
            &lowpc, &highpc,
            &cu_index,
            error);
        if (res != DW_DLV_OK) {
            /* Something wrong, ignore the addressarea */
            return res;
        }
        /* We have a valid address area entry, do something
           with it. */
    }
    return DW_DLV_OK;
}

```

9.80 Reading the gdbindex symbol table

Example accessing gdbindex symbol table data.

Example accessing gdbindex symbol table data.

```

/*
int examplex(Dwarf_Gdbindex gdbindex, Dwarf_Error *error)
{
    Dwarf_Unsigned symtab_list_length = 0;
    Dwarf_Unsigned i = 0;
    int res = 0;

    res = dwarf_gdbindex_symboltable_array(gdbindex,
        &symtab_list_length, error);
    if (res != DW_DLV_OK) {
        return res;
    }
    for (i = 0; i < symtab_list_length; i++) {
        Dwarf_Unsigned symnameoffset = 0;
        Dwarf_Unsigned cuvecoffset = 0;
        Dwarf_Unsigned cuvec_len = 0;
        Dwarf_Unsigned ii = 0;
        const char *name = 0;
        int res1 = 0;

        res1 = dwarf_gdbindex_symboltable_entry(gdbindex, i,
            &symnameoffset, &cuvecoffset,
            error);
        if (res1 != DW_DLV_OK) {
            return res1;
        }
    }
}

```

```

    }
    res1 = dwarf_gdbindex_string_by_offset(gdbindex,
        symnameoffset, &name, error);
    if (res1 != DW_DLV_OK) {
        return res1;
    }
    res1 = dwarf_gdbindex_cuvector_length(gdbindex,
        cuvecoffset, &cuvec_len, error);
    if (res1 != DW_DLV_OK) {
        return res1;
    }
    for (ii = 0; ii < cuvec_len; ++ii) {
        Dwarf_Unsigned attributes = 0;
        Dwarf_Unsigned cu_index = 0;
        Dwarf_Unsigned symbol_kind = 0;
        Dwarf_Unsigned is_static = 0;
        int res2 = 0;

        res2 = dwarf_gdbindex_cuvector_inner_attributes(
            gdbindex, cuvecoffset, ii,
            &attributes, error);
        if (res2 != DW_DLV_OK) {
            return res2;
        }
        /* 'attributes' is a value with various internal
           fields so we expand the fields. */
        res2 = dwarf_gdbindex_cuvector_instance_expand_value(
            gdbindex, attributes, &cu_index,
            &symbol_kind, &is_static,
            error);
        if (res2 != DW_DLV_OK) {
            return res2;
        }
        /* Do something with the attributes. */
    }
}
return DW_DLV_OK;
}

```

9.81 Reading cu and tu Debug Fission data

Example using `dwarf_get_xu_index_header`.

Example using `dwarf_get_xu_index_header`.

Debug Fission is an older name for Split Dwarf.

```

/*
int example(Dwarf_Debug dbg, const char *type,
    Dwarf_Error *error)
{
    /* type is "tu" or "cu" */
    int res = 0;
    Dwarf_Xu_Index_Header xuhdr = 0;
    Dwarf_Unsigned version_number = 0;
    Dwarf_Unsigned offsets_count = 0; /*L*/
    Dwarf_Unsigned units_count = 0; /*M*/
    Dwarf_Unsigned hash_slots_count = 0; /*N*/
    const char *section_name = 0;

    res = dwarf_get_xu_index_header(dbg,
        type,
        &xuhdr,
        &version_number,
        &offsets_count,
        &units_count,
        &hash_slots_count,
        &section_name,
        error);
    if (res != DW_DLV_OK) {
        return res;
    }
    /* Do something with the xuhdr here . */
    dwarf_dealloc_xu_header(xuhdr);
    return DW_DLV_OK;
}

```


9.82 Reading Split Dwarf (Debug Fission) hash slots

Example using `dwarf_get_xu_hash_entry()`

Example using `dwarf_get_xu_hash_entry()`

```

/*
int examplez( Dwarf_Xu_Index_Header xuhdr,
    Dwarf_Unsigned hash_slots_count,
    Dwarf_Error *error)
{
    /* hash_slots_count returned by
       dwarf_get_xu_index_header() */
    static Dwarf_Sig8 zerohashval;
    Dwarf_Unsigned h = 0;

    for (h = 0; h < hash_slots_count; h++) {
        Dwarf_Sig8 hashval;
        Dwarf_Unsigned index = 0;
        int res = 0;

        res = dwarf_get_xu_hash_entry(xuhdr,h,
            &hashval,&index,error);
        if (res != DW_DLV_OK) {
            return res;
        }
        if (!memcmp(&hashval,&zerohashval,
            sizeof(Dwarf_Sig8)) && index == 0 ) {
            /* An unused hash slot */
            continue;
        }
        /* Here, hashval and index (a row index into
           offsets and lengths) are valid. Do
           something with them */
    }
    return DW_DLV_OK;
}

```

9.83 Reading high pc from a DIE.

Example get high-pc from a DIE.

Example get high-pc from a DIE.

```

/*
int examplehighpc(Dwarf_Die die,
    Dwarf_Addr *highpc,
    Dwarf_Error *error)
{
    int res = 0;
    Dwarf_Addr localhighpc = 0;
    Dwarf_Half form = 0;
    enum Dwarf_Form_Class formclass = DW_FORM_CLASS_UNKNOWN;

    res = dwarf_highpc_b(die,&localhighpc,
        &form,&formclass, error);
    if (res != DW_DLV_OK) {
        return res;
    }
    if (form != DW_FORM_addr &&
        !dwarf_addr_form_is_indexed(form)) {
        Dwarf_Addr low_pc = 0;

        /* The localhighpc is an offset from
           DW_AT_low_pc. */
        res = dwarf_lowpc(die,&low_pc,error);
        if (res != DW_DLV_OK) {
            return res;
        } else {
            localhighpc += low_pc;
        }
    }
    *highpc = localhighpc;
    return DW_DLV_OK;
}

```

9.84 Reading Split Dwarf (Debug Fission) data

Example getting cu/tu name, offset.

Example getting cu/tu name, offset.

```

/*
int exampleza(Dwarf_Xu_Index_Header xuhdr,
Dwarf_Unsigned offsets_count,
Dwarf_Unsigned index,
Dwarf_Error *error)
{
    Dwarf_Unsigned col = 0;

    /* We use 'offsets_count' returned by
    a dwarf_get_xu_index_header() call.
    We use 'index' returned by a
    dwarf_get_xu_hash_entry() call. */
    for (col = 0; col < offsets_count; col++) {
        Dwarf_Unsigned off = 0;
        Dwarf_Unsigned len = 0;
        const char *name = 0;
        Dwarf_Unsigned num = 0;
        int res = 0;

        res = dwarf_get_xu_section_names(xuhdr,
            col, &num, &name, error);
        if (res == DW_DLV_ERROR) {
            return res;
        }
        if (res == DW_DLV_NO_ENTRY) {
            break;
        }
        res = dwarf_get_xu_section_offset(xuhdr,
            index, col, &off, &len, error);
        if (res == DW_DLV_ERROR) {
            return res;
        }
        if (res == DW_DLV_NO_ENTRY) {
            break;
        }
        /* Here we have the DW_SECT_ name and number
        and the base offset and length of the
        section data applicable to the hash
        that got us here.
        Use the values.*/
    }
    return DW_DLV_OK;
}

```

9.85 Retrieving tag,attribute,etc names

Example getting tag,attribute,etc names as strings.

Example getting tag,attribute,etc names as strings.

```

/*
void examplezb(void)
{
    const char * out = "unknown something";
    int res = 0;

    /* The following is wrong, do not do it!
    Confusing TAG with ACCESS! */
    res = dwarf_get_ACCESS_name(DW_TAG_entry_point, &out);
    /* Nothing one does here with 'res' or 'out'
    is meaningful. */

    out = "<unknown TAG>"; /* Not a malloc'd string! */
    /* The following is meaningful.*/
    res = dwarf_get_TAG_name(DW_TAG_entry_point, &out);
    (void)res; /* avoids unused var compiler warning */
    /* If res == DW_DLV_ERROR or DW_DLV_NO_ENTRY
    out will be the locally assigned static string.
    If res == DW_DLV_OK it will be a usable
    TAG name string.
    In no case should a returned string be free()d. */
}

```

9.86 Using GNU debuglink data

Example showing dwarf_add_debuglink_global_path.

Example showing dwarf_gnu_debuglink_global_path.

An example using both dwarf_add_debuglink_global_path and dwarf_gnu_debuglink .

```

/*
int exampledebuglink(Dwarf_Debug dbg, Dwarf_Error* error)
{
    int      res = 0;
    char      *debuglink_path = 0;
    unsigned char *crc = 0;
    char      *debuglink_fullpath = 0;
    unsigned debuglink_fullpath_strlen = 0;
    unsigned buildid_type = 0;
    char *    buildidowner_name = 0;
    unsigned char *buildid_itself = 0;
    unsigned buildid_length = 0;
    char **   paths = 0;
    unsigned paths_count = 0;
    unsigned i = 0;

    /* This is just an example if one knows
       of another place full-DWARF objects
       may be. "/usr/lib/debug" is automatically
       set. */
    res = dwarf_add_debuglink_global_path(dbg,
        "/some/path/debug", error);
    if (res != DW_DLV_OK) {
        /* Something is wrong*/
        return res;
    }
    res = dwarf_gnu_debuglink(dbg,
        &debuglink_path,
        &crc,
        &debuglink_fullpath,
        &debuglink_fullpath_strlen,
        &buildid_type,
        &buildidowner_name,
        &buildid_itself,
        &buildid_length,
        &paths,
        &paths_count,
        error);
    if (res == DW_DLV_ERROR) {
        return res;
    }
    if (res == DW_DLV_NO_ENTRY) {
        /* No such sections as .note.gnu.build-id
           or .gnu_debuglink */
        return res;
    }
    if (debuglink_fullpath_strlen) {
        printf("debuglink      path: %s\n", debuglink_path);
        printf("crc length      : %u  crc: ", 4);
        for (i = 0; i < 4; ++i) {
            printf("%02x", crc[i]);
        }
        printf("\n");
        printf("debuglink fullpath: %s\n", debuglink_fullpath);
    }
    if (buildid_length) {
        printf("buildid type      : %u\n", buildid_type);
        printf("Buildid owner     : %s\n", buildidowner_name);
        printf("buildid byte count: %u\n", buildid_length);
        printf(" ");
        /* buildid_length should be 20. */
        for (i = 0; i < buildid_length; ++i) {
            printf("%02x", buildid_itself[i]);
        }
        printf("\n");
    }
    printf("Possible paths count %u\n", paths_count);
    for (i = 0; i < paths_count; ++i) {
        printf("%2u: %s\n", i, paths[i]);
    }
    free(debuglink_fullpath);
    free(paths);
    return DW_DLV_OK;
}

```

9.87 Accessing accessing raw rnglist

Example showing access to rnglist.

Example showing access to rnglist.

This is accessing DWARF5 .debug_rnglists.

```

*/
int example_raw_rnglist(Dwarf_Debug dbg,Dwarf_Error *error)
{
    Dwarf_Unsigned count = 0;
    int res = 0;
    Dwarf_Unsigned i = 0;

    res = dwarf_load_rnglists(dbg,&count,error);
    if (res != DW_DLV_OK) {
        return res;
    }
    for (i=0 ; i < count ; ++i) {
        Dwarf_Unsigned header_offset = 0;
        Dwarf_Small offset_size = 0;
        Dwarf_Small extension_size = 0;
        unsigned version = 0; /* 5 */
        Dwarf_Small address_size = 0;
        Dwarf_Small segment_selector_size = 0;
        Dwarf_Unsigned offset_entry_count = 0;
        Dwarf_Unsigned offset_of_offset_array = 0;
        Dwarf_Unsigned offset_of_first_rangeentry = 0;
        Dwarf_Unsigned offset_past_last_rangeentry = 0;

        res = dwarf_get_rnglist_context_basics(dbg,i,
            &header_offset,&offset_size,&extension_size,
            &version,&address_size,&segment_selector_size,
            &offset_entry_count,&offset_of_offset_array,
            &offset_of_first_rangeentry,
            &offset_past_last_rangeentry,error);
        if (res != DW_DLV_OK) {
            return res;
        }
        {
            Dwarf_Unsigned e = 0;
            unsigned colmax = 4;
            unsigned col = 0;
            Dwarf_Unsigned global_offset_of_value = 0;

            for ( ; e < offset_entry_count; ++e) {
                Dwarf_Unsigned value = 0;
                int resc = 0;

                resc = dwarf_get_rnglist_offset_index_value(dbg,
                    i,e,&value,
                    &global_offset_of_value,error);
                if (resc != DW_DLV_OK) {
                    return resc;
                }
                /* Do something */
                col++;
                if (col == colmax) {
                    col = 0;
                }
            }
        }
        {
            Dwarf_Unsigned curoffset = offset_of_first_rangeentry;
            Dwarf_Unsigned endoffset = offset_past_last_rangeentry;
            int rese = 0;
            Dwarf_Unsigned ct = 0;

            for ( ; curoffset < endoffset; ++ct ) {
                unsigned entrylen = 0;
                unsigned code = 0;
                Dwarf_Unsigned v1 = 0;
                Dwarf_Unsigned v2 = 0;
                rese = dwarf_get_rnglist_rle(dbg,i,
                    curoffset,endoffset,
                    &entrylen,
                    &code,&v1,&v2,error);
                if (rese != DW_DLV_OK) {
                    return rese;
                }
                /* Do something with the values */
                curoffset += entrylen;
                if (curoffset > endoffset) {

```

```

        return DW_DLV_ERROR;
    }
}
}
return DW_DLV_OK;
}

```

9.88 Accessing rnglists section

Example showing access to rnglists on an Attribute.

Example showing access to rnglists on an Attribute.

This is accessing DWARF5 .debug_rnglists. The section first appears in DWARF5.

```

/*
int example_rnglist_for_attribute(Dwarf_Attribute attr,
    Dwarf_Unsigned attrvalue, Dwarf_Error *error)
{
    /* attrvalue must be the DW_AT_ranges
       DW_FORM_rnglistx or DW_FORM_sec_offset value
       extracted from attr. */
    int res = 0;
    Dwarf_Half theform = 0;
    Dwarf_Unsigned entries_count;
    Dwarf_Unsigned global_offset_of_rle_set;
    Dwarf_Rnglists_Head rnglhead = 0;
    Dwarf_Unsigned i = 0;

    res = dwarf_rnglists_get_rle_head(attr,
        theform,
        attrvalue,
        &rnglhead,
        &entries_count,
        &global_offset_of_rle_set,
        error);
    if (res != DW_DLV_OK) {
        return res;
    }
    for (i = 0; i < entries_count; ++i) {
        unsigned entrylen = 0;
        unsigned code = 0;
        Dwarf_Unsigned rawlowpc = 0;
        Dwarf_Unsigned rawhighpc = 0;
        Dwarf_Bool debug_addr_unavailable = FALSE;
        Dwarf_Unsigned lowpc = 0;
        Dwarf_Unsigned highpc = 0;

        /* Actual addresses are most likely what one
           wants to know, not the lengths/offsets
           recorded in .debug_rnglists. */
        res = dwarf_get_rnglists_entry_fields_a(rnglhead,
            i, &entrylen, &code,
            &rawlowpc, &rawhighpc,
            &debug_addr_unavailable,
            &lowpc, &highpc, error);
        if (res != DW_DLV_OK) {
            dwarf_dealloc_rnglists_head(rnglhead);
            return res;
        }
        if (code == DW_RLE_end_of_list) {
            /* we are done */
            break;
        }
        if (code == DW_RLE_base_addressx ||
            code == DW_RLE_base_address) {
            /* We do not need to use these, they
               have been accounted for already. */
            continue;
        }
        if (debug_addr_unavailable) {
            /* lowpc and highpc are not real addresses */
            continue;
        }
        /* Here do something with lowpc and highpc, these
           are real addresses */
    }
    dwarf_dealloc_rnglists_head(rnglhead);
    return DW_DLV_OK;
}

```

9.89 Demonstrating reading DWARF without a file.

How to read DWARF2 and later from memory.

How to read DWARF2 and later from memory.

```

*/

#include <config.h>

#include <stddef.h> /* NULL */
#include <stdio.h> /* printf() */
#include <stdlib.h> /* exit() */
#include <string.h> /* strcmp() */

#include "dwarf.h"
#include "libdwarf.h"
#include "libdwarf_private.h"

/*
   This demonstrates processing DWARF
   from in_memory data. For simplicity
   in this example we are using static arrays.
   The C source is src/bin/dwarfexample/jitreader.c

   The motivation is from JIT compiling, where
   at runtime of some application, it generates
   code on the file and DWARF information for it too.

   This gives an example of enabling all of libdwarf's
   functions without actually having the DWARF information
   in a file. (If you have a file in some odd format
   you can use this approach to have libdwarf access
   the format for DWARF data and work normally without
   ever exposing the format to libdwarf.)

   None of the structures defined here in this source
   (or any source using this feature)
   are ever known to libdwarf. They are totally
   private to your code.
   The code you write (like this example) you compile
   separate from libdwarf. You never place your code
   into libdwarf, you just link your code into
   your application and link against libdwarf.
*/

/* Some valid DWARF2 data */
static Dwarf_Small abbrevbytes[] = {
0x01, 0x11, 0x01, 0x25, 0x0e, 0x13, 0x0b, 0x03, 0x08, 0x1b,
0x0e, 0x11, 0x01, 0x12, 0x01, 0x10, 0x06, 0x00, 0x00, 0x02,
0x2e, 0x01, 0x3f, 0x0c, 0x03, 0x08, 0x3a, 0x0b, 0x3b, 0x0b,
0x39, 0x0b, 0x27, 0x0c, 0x11, 0x01, 0x12, 0x01, 0x40, 0x06,
0x97, 0x42, 0x0c, 0x01, 0x13, 0x00, 0x00, 0x03, 0x34, 0x00,
0x03, 0x08, 0x3a, 0x0b, 0x3b, 0x0b, 0x39, 0x0b, 0x49, 0x13,
0x02, 0x0a, 0x00, 0x00, 0x04, 0x24, 0x00, 0x0b, 0x0b, 0x3e,
0x0b, 0x03, 0x08, 0x00, 0x00, 0x00, 0x00, };
static Dwarf_Small infobytes[] = {
0x60, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00,
0x08, 0x01, 0x00, 0x00, 0x00, 0x00, 0x0c, 0x74, 0x2e, 0x63,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0x01, 0x66, 0x00, 0x01,
0x02, 0x06, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x01, 0x5c, 0x00, 0x00, 0x00, 0x03, 0x69,
0x00, 0x01, 0x03, 0x08, 0x5c, 0x00, 0x00, 0x00, 0x02, 0x91,
0x6c, 0x00, 0x04, 0x04, 0x05, 0x69, 0x6e, 0x74, 0x00, 0x00, };
static Dwarf_Small strbytes[] = {
0x47, 0x4e, 0x55, 0x20, 0x43, 0x31, 0x37, 0x20, 0x39, 0x2e,
0x33, 0x2e, 0x30, 0x20, 0x2d, 0x6d, 0x74, 0x75, 0x6e, 0x65,
0x3d, 0x67, 0x65, 0x6e, 0x65, 0x72, 0x69, 0x63, 0x20, 0x2d,
0x6d, 0x61, 0x72, 0x63, 0x68, 0x3d, 0x78, 0x38, 0x36, 0x2d,
0x36, 0x34, 0x20, 0x2d, 0x67, 0x64, 0x77, 0x61, 0x72, 0x66,
0x2d, 0x32, 0x20, 0x2d, 0x4f, 0x30, 0x20, 0x2d, 0x66, 0x61,
0x73, 0x79, 0x6e, 0x63, 0x68, 0x72, 0x6f, 0x6e, 0x6f, 0x75,
0x73, 0x2d, 0x75, 0x6e, 0x77, 0x69, 0x6e, 0x64, 0x2d, 0x74,
0x61, 0x62, 0x6c, 0x65, 0x73, 0x20, 0x2d, 0x66, 0x73, 0x74,
0x61, 0x63, 0x6b, 0x2d, 0x70, 0x72, 0x6f, 0x74, 0x65, 0x63,
0x74, 0x6f, 0x72, 0x2d, 0x73, 0x74, 0x72, 0x6f, 0x6e, 0x67,
0x20, 0x2d, 0x66, 0x73, 0x74, 0x61, 0x63, 0x6b, 0x2d, 0x63,
0x6c, 0x61, 0x73, 0x68, 0x2d, 0x70, 0x72, 0x6f, 0x74, 0x65,
0x63, 0x74, 0x69, 0x6f, 0x6e, 0x20, 0x2d, 0x66, 0x63, 0x66,
0x2d, 0x70, 0x72, 0x6f, 0x74, 0x65, 0x63, 0x74, 0x69, 0x6f,
0x6e, 0x00, 0x2f, 0x76, 0x61, 0x72, 0x2f, 0x74, 0x6d, 0x70,
0x2f, 0x74, 0x69, 0x6e, 0x79, 0x64, 0x77, 0x61, 0x72, 0x66,

```

```

0x00, };

/* An internals_t , data used elsewhere but
   not directly visible elsewhere. One needs to have one
   of these but maybe the content here too little
   or useless, this is just an example of sorts. */
#define SECCOUNT 4
struct sectiondata_s {
    unsigned int    sd_addr;
    unsigned int    sd_objoffsetlen;
    unsigned int    sd_objpointersize;
    Dwarf_Unsigned  sd_sectionsize;
    const char      * sd_secname;
    Dwarf_Small     * sd_content;
};

/* The secname must not be 0 , pass "" if
   there is no name. */
static struct sectiondata_s sectiondata[SECCOUNT] = {
    {0,0,0,0,"",0},
    {0,32,32,sizeof(abbrevbytes),".debug_abbrev",abbrevbytes},
    {0,32,32,sizeof(infobytes),".debug_info",infobytes},
    {0,32,32,sizeof(strbytes),".debug_str",strbytes}
};

typedef struct special_filedata_s {
    int            f_is_64bit;
    Dwarf_Small    f_object_endian;
    unsigned       f_pointersize;
    unsigned       f_offsetsize;
    Dwarf_Unsigned f_filesize;
    Dwarf_Unsigned f_sectioncount;
    struct sectiondata_s * f_sectarray;
} special_filedata_internals_t;

/* Use DW_END_little.
   Libdwarf finally sets the file-format-specific
   f_object_endianness field to a DW_END_little or
   DW_END_big (see dwarf.h).
   Here we must do that ourselves. */
static special_filedata_internals_t base_internals =
{ FALSE,DW_END_little,32,32,200,SECCOUNT, sectiondata };

static
int gsinfo(void* obj,
    Dwarf_Unsigned section_index,
    Dwarf_Obj_Access_Section_a* return_section,
    int* error )
{
    special_filedata_internals_t *internals =
        (special_filedata_internals_t *) (obj);
    struct sectiondata_s *finfo = 0;

    *error = 0; /* No error. Avoids unused arg */
    if (section_index >= SECCOUNT) {
        return DW_DLV_NO_ENTRY;
    }
    finfo = internals->f_sectarray + section_index;
    return_section->as_name      = finfo->sd_secname;
    return_section->as_type      = 0;
    return_section->as_flags     = 0;
    return_section->as_addr      = finfo->sd_addr;
    return_section->as_offset    = 0;
    return_section->as_size      = finfo->sd_sectionsize;
    return_section->as_link      = 0;
    return_section->as_info      = 0;
    return_section->as_addralign = 0;
    return_section->as_entrysize = 1;
    return DW_DLV_OK;
}

static Dwarf_Small
gborder(void * obj)
{
    special_filedata_internals_t *internals =
        (special_filedata_internals_t *) (obj);
    return internals->f_object_endian;
}

static
Dwarf_Small glensize(void * obj)
{
    /* offset size */
    special_filedata_internals_t *internals =
        (special_filedata_internals_t *) (obj);
    return internals->f_offsetsize/8;
}

static
Dwarf_Small gptrsize(void * obj)

```

```

{
    special_filedata_internals_t *internals =
        (special_filedata_internals_t *) (obj);
    return internals->f_pointersize/8;
}
static
Dwarf_Unsigned gfilesize(void * obj)
{
    special_filedata_internals_t *internals =
        (special_filedata_internals_t *) (obj);
    return internals->f_filesize;
}
static
Dwarf_Unsigned gseccount(void* obj)
{
    special_filedata_internals_t *internals =
        (special_filedata_internals_t *) (obj);
    return internals->f_sectioncount;
}
static
int gloadsec(void * obj,
    Dwarf_Unsigned secindex,
    Dwarf_Small**rdata,
    int *error)
{
    special_filedata_internals_t *internals =
        (special_filedata_internals_t *) (obj);
    struct sectiondata_s *secp = 0;

    *error = 0; /* No Error, avoids compiler warning */
    if (secindex >= internals->f_sectioncount) {
        return DW_DLV_NO_ENTRY;
    }
    secp = secindex + internals->f_sectarray;
    *rdata = secp->sd_content;
    return DW_DLV_OK;
}

const Dwarf_Obj_Access_Methods_a methods = {
    gsinfo,
    gborder,
    glensize,
    gpysize,
    gfilesize,
    gseccount,
    gloadsec,
    0 /* no relocating anything */,
    0 /* no file with DWARF here, so mmap impossible */,
    0 /* no destructor appropriate */
};

struct Dwarf_Obj_Access_Interface_a_s dw_interface =
{ &base_internals,&methods };

static const Dwarf_Sig8 zerosignature;
static int
isformstring(Dwarf_Half form)
{
    /* Not handling every form string, just the
       ones used in simple cases. */
    switch(form) {
    case DW_FORM_string:      return TRUE;
    case DW_FORM_GNU_strp_alt: return TRUE;
    case DW_FORM_GNU_str_index: return TRUE;
    case DW_FORM_strx:       return TRUE;
    case DW_FORM_strx1:      return TRUE;
    case DW_FORM_strx2:      return TRUE;
    case DW_FORM_strx3:      return TRUE;
    case DW_FORM_strx4:      return TRUE;
    case DW_FORM_strp:       return TRUE;
    default: break;
    };
    return FALSE;
}

static int
print_attr(Dwarf_Attribute atr,
    Dwarf_Signed anumber, Dwarf_Error *error)
{
    int res = 0;
    char *str = 0;
    const char *attrname = 0;
    const char *formname = 0;
    Dwarf_Half form = 0;
    Dwarf_Half attrnum = 0;
    res = dwarf_whatform(atr,&form,error);
    if (res != DW_DLV_OK) {

```



```

        printf("dwarf_whatform failed! res %d\n",res);
        return res;
    }
    res = dwarf_whatattr(atr,&attrnum,error);
    if (res != DW_DLV_OK) {
        printf("dwarf_whatattr failed! res %d\n",res);
        return res;
    }
    res = dwarf_get_AT_name(attrnum,&attrname);
    if (res == DW_DLV_NO_ENTRY) {
        printf("Bogus attrnum 0x%x\n",attrnum);
        attrname = "<internal error ?>";
    }
    res = dwarf_get_FORM_name(form,&formname);
    if (res == DW_DLV_NO_ENTRY) {
        printf("Bogus form 0x%x\n",attrnum);
        attrname = "<internal error ?>";
    }
    if (!isformstring(form)) {
        printf(" [%2d] Attr: %-15s Form: %-15s\n",
            (int)anumber,attrname,formname);
        return DW_DLV_OK;
    }
    res = dwarf_formstring(atr,&str,error);
    if (res != DW_DLV_OK) {
        printf("dwarf_formstring failed! res %d\n",res);
        return res;
    }
    printf(" [%2d] Attr: %-15s Form: %-15s %s\n",
        (int)anumber,attrname,formname,str);
    return DW_DLV_OK;
}

static void
dealloc_list(Dwarf_Debug dbg,
    Dwarf_Attribute *attrbuf,
    Dwarf_Signed attrcount,
    Dwarf_Signed i)
{
    for ( ; i < attrcount; ++i) {
        dwarf_dealloc_attribute(attrbuf[i]);
    }
    dwarf_dealloc(dbg,attrbuf,DW_DLA_LIST);
}

static int
print_one_die(Dwarf_Debug dbg,Dwarf_Die in_die,int level,
    Dwarf_Error *error)
{
    Dwarf_Attribute *attrbuf = 0;
    Dwarf_Signed attrcount = 0;
    Dwarf_Half tag = 0;
    const char * tagname = 0;
    int res = 0;
    Dwarf_Signed i = 0;

    res = dwarf_tag(in_die,&tag,error);
    if (res != DW_DLV_OK) {
        printf("dwarf_tag failed! res %d\n",res);
        return res;
    }
    res = dwarf_get_TAG_name(tag,&tagname);
    if (res != DW_DLV_OK) {
        tagname = "<bogus tag>";
    }
    printf("%3d: Die: %s\n",level,tagname);
    res = dwarf_attrlist(in_die,&attrbuf,&attrcount,error);
    if (res != DW_DLV_OK) {
        printf("dwarf_attrlist failed! res %d\n",res);
        return res;
    }
    for (i = 0; i < attrcount; ++i) {
        res = print_attr(attrbuf[i],i,error);
        if (res != DW_DLV_OK) {
            dealloc_list(dbg,attrbuf,attrcount,0);
            printf("dwarf_attr print failed! res %d\n",res);
            return res;
        }
    }
    dealloc_list(dbg,attrbuf,attrcount,0);
    return DW_DLV_OK;
}

static int
print_object_info(Dwarf_Debug dbg,Dwarf_Error *error)
{
    Dwarf_Bool is_info = TRUE; /* our data is not DWARF4

```

```

        .debug_types. */
Dwarf_Unsigned cu_header_length = 0;
Dwarf_Half     version_stamp = 0;
Dwarf_Off      abbrev_offset = 0;
Dwarf_Half     address_size  = 0;
Dwarf_Half     length_size   = 0;
Dwarf_Half     extension_size = 0;
Dwarf_Sig8     type_signature;
Dwarf_Unsigned typeoffset    = 0;
Dwarf_Unsigned next_cu_header_offset = 0;
Dwarf_Half     header_cu_type = 0;

int res = 0;
Dwarf_Die cu_die = 0;
int level = 0;

type_signature = zerosignature;
res = dwarf_next_cu_header_d(dbg,
    is_info,
    &cu_header_length,
    &version_stamp,
    &abbrev_offset,
    &address_size,
    &length_size,
    &extension_size,
    &type_signature,
    &typeoffset,
    &next_cu_header_offset,
    &header_cu_type,
    error);
if (res != DW_DLV_OK) {
    printf("Next cu header result %d. "
        "Something is wrong FAIL, line %d\n", res, __LINE__);
    if (res == DW_DLV_ERROR) {
        printf("Error is: %s\n", dwarf_errmsg(*error));
    }
    exit(EXIT_FAILURE);
}
printf("CU header length.....0x%lx\n",
    (unsigned long)cu_header_length);
printf("Version stamp.....%d\n", version_stamp);
printf("Address size .....%d\n", address_size);
printf("Offset size.....%d\n", length_size);
printf("Next cu header offset.....0x%lx\n",
    (unsigned long)next_cu_header_offset);

res = dwarf_siblingof_b(dbg, NULL, is_info, &cu_die, error);
if (res != DW_DLV_OK) {
    /* There is no CU die, which should be impossible. */
    if (res == DW_DLV_ERROR) {
        printf("ERROR: dwarf_siblingof_b failed, no CU die\n");
        printf("Error is: %s\n", dwarf_errmsg(*error));
        return res;
    } else {
        printf("ERROR: dwarf_siblingof_b got NO_ENTRY, "
            "no CU die\n");
        return res;
    }
}
res = print_one_die(dbg, cu_die, level, error);
if (res != DW_DLV_OK) {
    dwarf_dealloc_die(cu_die);
    printf("print_one_die failed! %d\n", res);
    return res;
}
dwarf_dealloc_die(cu_die);
return DW_DLV_OK;
}

/* testing interfaces useful for embedding
libdwarf inside another program or library. */
int main(int argc, char **argv)
{
    int res = 0;
    Dwarf_Debug dbg = 0;
    Dwarf_Error error = 0;
    int fail = FALSE;
    int i = 1;

    if (i >= argc) {
        /* OK */
    } else {
        if (!strcmp(argv[i], "--suppress-de-alloc-tree")) {
            /* Do nothing, ignore the argument */
            ++i;
        }
    }
}

/* Fill in interface before this call.

```

```

    We are using a static area, see above. */
    res = dwarf_object_init_b(&dw_interface,
        0,0,DW_GROUPNUMBER_ANY,&dbg,
        &error);
    if (res == DW_DLV_NO_ENTRY) {
        printf("FAIL Cannot dwarf_object_init_b() NO ENTRY. \n");
        exit(EXIT_FAILURE);
    } else if (res == DW_DLV_ERROR) {
        printf("FAIL Cannot dwarf_object_init_b(). \n");
        printf("msg: %s\n",dwarf_errmsg(error));
        dwarf_dealloc_error(dbg,error);
        exit(EXIT_FAILURE);
    }
    res = print_object_info(dbg,&error);
    if (res != DW_DLV_OK) {
        if (res == DW_DLV_ERROR) {
            dwarf_dealloc_error(dbg,error);
        }
        printf("FAIL printing, res %d line %d\n",res,__LINE__);
        exit(EXIT_FAILURE);
    }
    dwarf_object_finish(dbg);
    if (fail) {
        printf("FAIL objectaccess.c\n");
        exit(EXIT_FAILURE);
    }
    return 0;
}

```

9.90 A simple report on section groups.

Section groups are for Split DWARF.

Section groups are for Split DWARF.

The C source is src/bin/dwarfexample/showsectiongroups.c

```

*/
#include <config.h>

#include <stdio.h> /* printf() */
#include <stdlib.h> /* calloc() exit() free() */
#include <string.h> /* strcmp() */

#include "dwarf.h"
#include "libdwarf.h"
#define FALSE 0

char trueoutpath[2000];

static int
one_file_show_groups(char *path_in,
    char *shortpath,
    int chosengroup)
{
    int res = 0;
    Dwarf_Debug dbg = 0;
    Dwarf_Error error = 0;
    char *path = 0;
    Dwarf_Unsigned section_count = 0;
    Dwarf_Unsigned group_count = 0;
    Dwarf_Unsigned selected_group = 0;
    Dwarf_Unsigned map_entry_count = 0;
    Dwarf_Unsigned * group_numbers_array = 0;
    Dwarf_Unsigned * sec_numbers_array = 0;
    const char ** sec_names_array = 0;
    Dwarf_Unsigned i = 0;
    const char *grpname = 0;

    switch(chosengroup) {
    case DW_GROUPNUMBER_ANY:
        grpname="DW_GROUPNUMBER_ANY";
        break;
    case DW_GROUPNUMBER_BASE:
        grpname="DW_GROUPNUMBER_BASE";
        break;
    case DW_GROUPNUMBER_DWO:
        grpname="DW_GROUPNUMBER_DWO";
        break;
    default:
        grpname = "";
    }
}

```

```

}
path = path_in;
res = dwarf_init_path(path,
    0,0,
    chosengroup,
    0,0, &dbg, &error);
if (res == DW_DLV_ERROR) {
    printf("Error from libdwarf opening \"%s\": %s\n",
        shortpath, dwarf_errmsg(error));
    dwarf_dealloc_error(dbg,error);
    error = 0;
    return res;
}
if (res == DW_DLV_NO_ENTRY) {
    printf("There is no such file as \"%s\" "
        "or the selected group %d (%s) does "
        "not appear in the file\n",
        shortpath,chosengroup,grpname);
    return DW_DLV_NO_ENTRY;
}

res = dwarf_sec_group_sizes(dbg, &section_count,
    &group_count, &selected_group, &map_entry_count,
    &error);
if (res == DW_DLV_ERROR) {
    printf("Error from libdwarf getting group "
        "sizes \"%s\": %s\n",
        shortpath, dwarf_errmsg(error));
    dwarf_dealloc_error(dbg,error);
    error = 0;
    dwarf_finish(dbg);
    return res;
}
if (res == DW_DLV_NO_ENTRY) {
    printf("Impossible. libdwarf claims no groups from %s\n",
        shortpath);
    dwarf_finish(dbg);
    return res;
}
printf("Group Map data sizes\n");
printf(" requested group : %4lu\n",
    (unsigned long)chosengroup);
printf(" section count   : %4lu\n",
    (unsigned long)section_count);
printf(" group count      : %4lu\n",
    (unsigned long)group_count);
printf(" selected group   : %4lu\n",
    (unsigned long)selected_group);
printf(" map entry count  : %4lu\n",
    (unsigned long)map_entry_count);

group_numbers_array = (Dwarf_Unsigned *)calloc(map_entry_count,
    sizeof(Dwarf_Unsigned));
if (!group_numbers_array) {
    printf("Error calloc fail, group count %lu\n",
        (unsigned long)group_count);
    dwarf_finish(dbg);
    return DW_DLV_ERROR;
}
sec_numbers_array = (Dwarf_Unsigned *)calloc(map_entry_count,
    sizeof(Dwarf_Unsigned));
if (!sec_numbers_array) {
    free(group_numbers_array);
    printf("Error calloc fail sec numbers, section count %lu\n",
        (unsigned long)section_count);
    dwarf_finish(dbg);
    return DW_DLV_ERROR;
}
sec_names_array = (const char **)calloc(map_entry_count,
    sizeof(const char *));
if (!sec_names_array) {
    free(sec_numbers_array);
    free(group_numbers_array);
    printf("Error calloc fail on names, section count %lu\n",
        (unsigned long)section_count);
    dwarf_finish(dbg);
    return DW_DLV_ERROR;
}
res = dwarf_sec_group_map(dbg,map_entry_count,
    group_numbers_array,
    sec_numbers_array, sec_names_array,&error);
if (res == DW_DLV_ERROR) {
    free(sec_names_array);
    free(sec_numbers_array);
    free(group_numbers_array);
    printf("Error from libdwarf getting group details "
        "sizes \"%s\": %s\n",

```

```

        shortpath, dwarf_errmsg(error));
dwarf_dealloc_error(dbg,error);
error = 0;
dwarf_finish(dbg);
return res;
}
if (res == DW_DLV_NO_ENTRY) {
    free(sec_names_array);
    free(sec_numbers_array);
    free(group_numbers_array);
    printf("Impossible. libdwarf claims details from %s\n",
        shortpath);
    dwarf_finish(dbg);
    return res;
}
printf(" [index] group section \n");
for (i = 0; i < map_entry_count; ++i) {
    printf(" [%5lu] %4lu %4lu %s\n",
        (unsigned long)i,
        (unsigned long)group_numbers_array[i],
        (unsigned long)sec_numbers_array[i],
        sec_names_array[i]);
}
free(sec_names_array);
free(sec_numbers_array);
free(group_numbers_array);
dwarf_finish(dbg);
return DW_DLV_OK;
}

/* Does not return */
static void
usage(void)
{
    printf("Usage: showsectiongroups [-group <n>] "
        "<objectfile> ...\n");
    printf("Usage: group defaults to zero (DW_GROUPNUMBER ANY)\n");
    exit(EXIT_FAILURE);
}

/* This trimming of the file path makes libdwarf regression
testing easier by arranging baseline output
not show the full path. */
static void
trimpathprefix(char *out,unsigned int outlen, char *in)
{
    char *cpo = out;
    char *cpi = in;
    char *suffix = 0;
    unsigned int lencopied = 0;
    for ( ; *cpi ; ++cpi) {
        if (*cpi == '/') {
            suffix= cpi+1;
        }
    }
    if (suffix) {
        cpi = suffix;
    }
    lencopied = 0;
    for ( ; lencopied < outlen; ++cpo,++cpi)
    {
        *cpo = *cpi;
        if (! *cpi) {
            return;
        }
        ++lencopied;
    }
    printf("FAIL copy file name: not terminated \n");
    exit(EXIT_FAILURE);
}

int
main(int argc, char **argv)
{
    int res = 0;
    int i = 1;
    int chosengroup = DW_GROUPNUMBER_ANY;
    static char reportingpath[16000];

    if (argc < 2) {
        usage();
        return 0;
    }
    for ( ; i < argc; ++i) {
        char *arg = argv[i];
        if (!strcmp(arg,"-group")) {
            i++;

```

```
    if (i >= argc) {
        usage();
    }
    arg = argv[i];
    chosengroup = atoi(arg);
    /* We are ignoring errors to simplify
       this source. Use strtol, carefully,
       in real code. */
    continue;
}
if (!strcmp(argv[i], "--suppress-de-alloc-tree")) {
    /* Do nothing, ignore the argument */
    continue;
}
trimpathprefix(reportingpath, sizeof(reportingpath), arg);
res = one_file_show_groups(arg,
    reportingpath, chosengroup);
printf("====done with %s, status %s\n", reportingpath,
    (res == DW_DLV_OK) ? "DW_DLV_OK":
    (res == DW_DLV_ERROR) ? "DW_DLV_ERROR":
    "DW_DLV_NO_ENTRY");
printf("\n");
}
return 0;
}
```

Chapter 10

Class Documentation

10.1 Dwarf_Block_s Struct Reference

Public Attributes

- [Dwarf_Unsigned](#) **bl_len**
- [Dwarf_Ptr](#) **bl_data**
- [Dwarf_Small](#) **bl_from_loclist**
- [Dwarf_Unsigned](#) **bl_section_offset**

The documentation for this struct was generated from the following file:

- </home/davea/dwarf/code/src/lib/libdwarf/libdwarf.h>

10.2 Dwarf_Cmdline_Options_s Struct Reference

```
#include <libdwarf.h>
```

Public Attributes

- [Dwarf_Bool](#) **check_verbose_mode**

10.2.1 Detailed Description

`check_verbose_mode` defaults to `FALSE`. If a `libdwarf`-calling program sets this `TRUE` it means some errors in Line Table headers get a much more detailed description of the error which is reported the caller via `printf()`↵`_callback()` function (the caller can do something with the message). Or the `libdwarf` calling code can call [dwarf_record_cmdline_options\(\)](#) to set the new value.

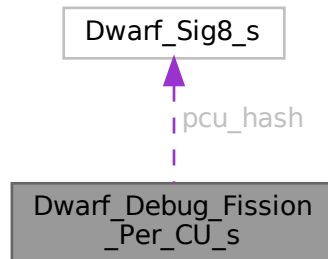
For convenience the type name for the struct is `Dwarf_Cmdline_Options`.

The documentation for this struct was generated from the following file:

- </home/davea/dwarf/code/src/lib/libdwarf/libdwarf.h>

10.3 Dwarf_Debug_Fission_Per_CU_s Struct Reference

Collaboration diagram for Dwarf_Debug_Fission_Per_CU_s:



Public Attributes

- const char * **pcu_type**
- Dwarf_Unsigned **pcu_index**
- Dwarf_Sig8 **pcu_hash**
- Dwarf_Unsigned **pcu_offset** [DW_FFISSION_SECT_COUNT]
- Dwarf_Unsigned **pcu_size** [DW_FFISSION_SECT_COUNT]
- Dwarf_Unsigned **unused1**
- Dwarf_Unsigned **unused2**

The documentation for this struct was generated from the following file:

- </home/davea/dwarf/code/src/lib/libdwarf/libdwarf.h>

10.4 Dwarf_Form_Data16_s Struct Reference

Public Attributes

- unsigned char **fd_data** [16]

The documentation for this struct was generated from the following file:

- </home/davea/dwarf/code/src/lib/libdwarf/libdwarf.h>

10.5 Dwarf_Macro_Details_s Struct Reference

```
#include <libdwarf.h>
```


Public Attributes

- [Dwarf_Off](#) **dmd_offset**
- [Dwarf_Small](#) **dmd_type**
- [Dwarf_Signed](#) **dmd_lineno**
- [Dwarf_Signed](#) **dmd_fileindex**
- char * **dmd_macro**

10.5.1 Detailed Description

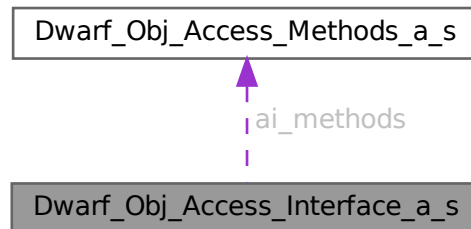
This applies to DWARF2, DWARF3, and DWARF4 compilation units. DWARF5 .debug_macro has its own function interface which does not use this struct.

The documentation for this struct was generated from the following file:

- [/home/davea/dwarf/code/src/lib/libdwarf/libdwarf.h](#)

10.6 Dwarf_Obj_Access_Interface_a_s Struct Reference

Collaboration diagram for Dwarf_Obj_Access_Interface_a_s:

**Public Attributes**

- void * **ai_object**
- const [Dwarf_Obj_Access_Methods_a](#) * **ai_methods**

The documentation for this struct was generated from the following file:

- [/home/davea/dwarf/code/src/lib/libdwarf/libdwarf.h](#)

10.7 Dwarf_Obj_Access_Methods_a_s Struct Reference

```
#include <libdwarf.h>
```

Public Attributes

- `int(* om_get_section_info)(void *obj, Dwarf_Unsigned section_index, Dwarf_Obj_Access_Section_a *return_section, int *error)`
- `Dwarf_Small(* om_get_byte_order)(void *obj)`
- `Dwarf_Small(* om_get_length_size)(void *obj)`
- `Dwarf_Small(* om_get_pointer_size)(void *obj)`
- `Dwarf_Unsigned(* om_get_filesize)(void *obj)`
- `Dwarf_Unsigned(* om_get_section_count)(void *obj)`
- `int(* om_load_section)(void *obj, Dwarf_Unsigned dw_section_index, Dwarf_Small **dw_return_data, int *dw_error)`
- `int(* om_relocate_a_section)(void *obj, Dwarf_Unsigned section_index, Dwarf_Debug dbg, int *error)`
- `int(* om_load_section_a)(void *obj, Dwarf_Unsigned dw_section_index, enum Dwarf_Sec_Alloc_Pref *dw_alloc_pref, Dwarf_Small **dw_return_data_ptr, Dwarf_Unsigned *dw_return_data_len, Dwarf_Small **dw_return_mmap_base_ptr, Dwarf_Unsigned *dw_return_mmap_offset, Dwarf_Unsigned *dw_return_mmap_len, int *dw_error)`
- `void(* om_finish)(void *obj)`

10.7.1 Detailed Description

The functions we need to access object data from libdwarf are declared here.

Unless you are reading object sections with your own code (as in [src/bin/dwarfexample/jitreader.c](#)) you will not need to fill in or use the struct.

`om_relocate_a_section` uses `malloc/read` to get section contents and returns a pointer to the malloc space through `dw_return_data`, which is recorded in the applicable section data.

`om_load_section_a` uses either `malloc/read` or `mmap` and consequently returns more data as needed for eventual `free()` or `munmap()`.

The documentation for this struct was generated from the following file:

- [/home/davea/dwarf/code/src/lib/libdwarf/libdwarf.h](#)

10.8 Dwarf_Obj_Access_Section_a_s Struct Reference

Public Attributes

- `const char * as_name`
- `Dwarf_Unsigned as_type`
- `Dwarf_Unsigned as_flags`
- `Dwarf_Addr as_addr`
- `Dwarf_Unsigned as_offset`
- `Dwarf_Unsigned as_size`
- `Dwarf_Unsigned as_link`
- `Dwarf_Unsigned as_info`
- `Dwarf_Unsigned as_addralign`
- `Dwarf_Unsigned as_entsize`

The documentation for this struct was generated from the following file:

- [/home/davea/dwarf/code/src/lib/libdwarf/libdwarf.h](#)

10.9 Dwarf_Printf_Callback_Info_s Struct Reference

```
#include <libdwarf.h>
```

Public Attributes

- void * **dp_user_pointer**
- [dwarf_printf_callback_function_type](#) **dp_fptr**
- char * **dp_buffer**
- unsigned int **dp_buffer_len**
- int **dp_buffer_user_provided**
- void * **dp_reserved**

10.9.1 Detailed Description

If one wishes to print detailed line table information one creates an instance of this struct and fills in the fields and passes the struct to the relevant init, for example, [dwarf_init_path\(\)](#).

The documentation for this struct was generated from the following file:

- [/home/davea/dwarf/code/src/lib/libdwarf/libdwarf.h](#)

10.10 Dwarf_Ranges_s Struct Reference

Public Attributes

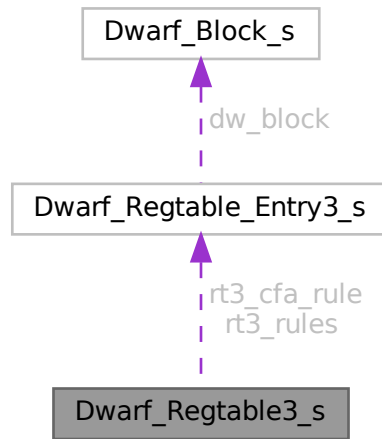
- [Dwarf_Addr](#) **dwr_addr1**
- [Dwarf_Addr](#) **dwr_addr2**
- enum [Dwarf_Ranges_Entry_Type](#) **dwr_type**

The documentation for this struct was generated from the following file:

- [/home/davea/dwarf/code/src/lib/libdwarf/libdwarf.h](#)

10.11 Dwarf_Regtable3_s Struct Reference

Collaboration diagram for Dwarf_Regtable3_s:



Public Attributes

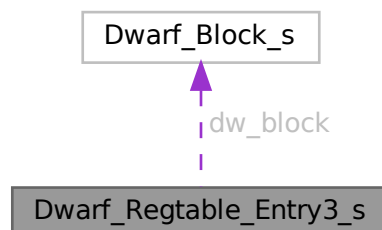
- struct [Dwarf_Regtable_Entry3_s](#) `rt3_cfa_rule`
- [Dwarf_Half](#) `rt3_reg_table_size`
- struct [Dwarf_Regtable_Entry3_s](#) * `rt3_rules`

The documentation for this struct was generated from the following file:

- [/home/davea/dwarf/code/src/lib/libdwarf/libdwarf.h](#)

10.12 Dwarf_Regtable_Entry3_s Struct Reference

Collaboration diagram for Dwarf_Regtable_Entry3_s:



Public Attributes

- [Dwarf_Small](#) **dw_offset_relevant**
- [Dwarf_Small](#) **dw_value_type**
- [Dwarf_Half](#) **dw_regnum**
- [Dwarf_Unsigned](#) **dw_offset**
- [Dwarf_Unsigned](#) **dw_args_size**
- [Dwarf_Block](#) **dw_block**

The documentation for this struct was generated from the following file:

- [/home/davea/dwarf/code/src/lib/libdwarf/libdwarf.h](#)

10.13 Dwarf_Sig8_s Struct Reference

Public Attributes

- char **signature** [8]

The documentation for this struct was generated from the following file:

- [/home/davea/dwarf/code/src/lib/libdwarf/libdwarf.h](#)

Chapter 11

File Documentation

Chapter 12

checkexamples.c

[checkexamples.c](#) contains what user code should be. Hence the code typed in [checkexamples.c](#) is PUBLIC DOMAIN and may be copied, used, and altered without any restrictions.

[checkexamples.c](#) need not be compiled routinely nor should it ever be executed.

To verify syntactic correctness compile in the libdwarf-code/doc directory with:

```
cc -c -Wall -O0 -Wpointer-arith \
-Wdeclaration-after-statement \
-Wextra -Wcomment -Wformat -Wpedantic -Wuninitialized \
-Wno-long-long -Wshadow -Wbad-function-cast \
-Wmissing-parameter-type -Wnested-externs \
-I../src/lib/libdwarf checkexamples.c
```

12.1 /home/davea/dwarf/code/src/bin/dwarfexample/jitreader.c File Reference

12.2 /home/davea/dwarf/code/src/bin/dwarfexample/showsectiongroups.c File Reference

Chapter 13

dwarf.h

[dwarf.h](#) contains all the identifiers such as DW_TAG_compile_unit etc from the various versions of the DWARF Standard beginning with DWARF2 and containing all later Dwarf Standard identifiers.

In addition, it contains all user-defined identifiers that we have been able to find.

All identifiers here are C defines with the prefix "DW_" .

13.1 dwarf.h

[Go to the documentation of this file.](#)

```
00001 /*
00002 Copyright (C) 2000-2006 Silicon Graphics, Inc. All Rights Reserved.
00003 Portions Copyright 2002-2010 Sun Microsystems, Inc. All rights reserved.
00004 Portions Copyright 2007-2023 David Anderson. All rights reserved.
00005
00006 This program is free software; you can redistribute it
00007 and/or modify it under the terms of version 2.1 of the
00008 GNU Lesser General Public License as published by the Free
00009 Software Foundation.
00010
00011 This program is distributed in the hope that it would be
00012 useful, but WITHOUT ANY WARRANTY; without even the implied
00013 warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
00014 PURPOSE.
00015
00016 Further, this software is distributed without any warranty
00017 that it is free of the rightful claim of any third person
00018 regarding infringement or the like. Any license provided
00019 herein, whether implied or otherwise, applies only to this
00020 software file. Patent licenses, if any, provided herein
00021 do not apply to combinations of this program with other
00022 software, or any other product whatsoever.
00023
00024 You should have received a copy of the GNU Lesser General
00025 Public License along with this program; if not, write the
00026 Free Software Foundation, Inc., 51 Franklin Street - Fifth
00027 Floor, Boston MA 02110-1301, USA.
00028 */
00029
00044 #ifndef __DWARF_H
00045 #define __DWARF_H
00046 #ifdef __cplusplus
00047 extern "C" {
00048 #endif
00049
00050 /*
00051 dwarf.h DWARF debugging information values
00052 $Revision: 1.41 $ $Date: 2006/04/17 00:09:56 $
00053
00054 The comment "DWARF3" appears where there are
00055 new entries from DWARF3 as of 2004, "DWARF3f"
00056 where there are new entries as of the November 2005
```

```

00057     public review document and other comments apply
00058     where extension entries appear.
00059
00060     Extensions part of DWARF4 are marked DWARF4.
00061
00062     A few extension names have omitted the 'vendor id'
00063     (See chapter 7, "Vendor Extensibility"). Please
00064     always use a 'vendor id' string in extension names.
00065
00066     Vendors should use a vendor string in names and
00067     wherever possible avoid duplicating values used by
00068     other vendor extensions
00069
00070     The DWARF1 comments indicate values unused in
00071     DWARF2 and later but used or reserved in DWARF1.
00072 */
00073
00074 #define DW_TAG_array_type          0x01
00075 #define DW_TAG_class_type         0x02
00076 #define DW_TAG_entry_point        0x03
00077 #define DW_TAG_enumeration_type   0x04
00078 #define DW_TAG_formal_parameter   0x05
00079 /* TAG_global_subroutine         0x06 DWARF1 only */
00080 /* TAG_global_variable           0x07 DWARF1 only */
00081 #define DW_TAG_imported_declaration 0x08
00082 /* reserved by DWARF1           0x09 DWARF1 only */
00083 #define DW_TAG_label              0x0a
00084 #define DW_TAG_lexical_block      0x0b
00085 /* TAG_local_variable            0x0c DWARF1 only. */
00086 #define DW_TAG_member             0x0d
00087 /* reserved by DWARF1           0x0e DWARF1 only */
00088 #define DW_TAG_pointer_type       0x0f
00089 #define DW_TAG_reference_type     0x10
00090 #define DW_TAG_compile_unit      0x11
00091 #define DW_TAG_string_type        0x12
00092 #define DW_TAG_structure_type     0x13
00093 /* TAG_subroutine                0x14 DWARF1 only */
00094 #define DW_TAG_subroutine_type    0x15
00095 #define DW_TAG_typedef            0x16
00096 #define DW_TAG_union_type         0x17
00097 #define DW_TAG_unspecified_parameters 0x18
00098 #define DW_TAG_variant            0x19
00099 #define DW_TAG_common_block       0x1a
00100 #define DW_TAG_common_inclusion     0x1b
00101 #define DW_TAG_inheritance        0x1c
00102 #define DW_TAG_inlined_subroutine 0x1d
00103 #define DW_TAG_module             0x1e
00104 #define DW_TAG_ptr_to_member_type 0x1f
00105 #define DW_TAG_set_type           0x20
00106 #define DW_TAG_subrange_type      0x21
00107 #define DW_TAG_with_stmt          0x22
00108 #define DW_TAG_access_declaration 0x23
00109 #define DW_TAG_base_type          0x24
00110 #define DW_TAG_catch_block        0x25
00111 #define DW_TAG_const_type         0x26
00112 #define DW_TAG_constant           0x27
00113 #define DW_TAG_enumerator         0x28
00114 #define DW_TAG_file_type          0x29
00115 #define DW_TAG_friend             0x2a
00116 #define DW_TAG_namelist           0x2b
00117 /* Early releases of this header had the following
00118     misspelled with a trailing 's' */
00119 #define DW_TAG_namelist_item      0x2c /* DWARF2 spelling*/
00120 #define DW_TAG_namelist_items     0x2c /* SGI misspelling/typo*/
00121 #define DW_TAG_packed_type        0x2d
00122 #define DW_TAG_subprogram         0x2e
00123 /* The DWARF2 document had two spellings of the following
00124     two TAGs, DWARF3 specifies the longer spelling. */
00125 #define DW_TAG_template_type_parameter 0x2f /* DWARF3-5 spelling*/
00126 #define DW_TAG_template_type_param    0x2f /* DWARF2 inconsistent*/
00127 #define DW_TAG_template_value_parameter 0x30 /* DWARF all versions*/
00128 #define DW_TAG_template_value_param    0x30 /* SGI misspelling/typo*/
00129 #define DW_TAG_thrown_type            0x31
00130 #define DW_TAG_try_block               0x32
00131 #define DW_TAG_variant_part            0x33
00132 #define DW_TAG_variable                0x34
00133 #define DW_TAG_volatile_type           0x35
00134 #define DW_TAG_dwarf_procedure         0x36 /* DWARF3 */
00135 #define DW_TAG_restrict_type           0x37 /* DWARF3 */
00136 #define DW_TAG_interface_type          0x38 /* DWARF3 */
00137 #define DW_TAG_namespace               0x39 /* DWARF3 */
00138 #define DW_TAG_imported_module         0x3a /* DWARF3 */
00139 #define DW_TAG_unspecified_type        0x3b /* DWARF3 */
00140 #define DW_TAG_partial_unit            0x3c /* DWARF3 */
00141 #define DW_TAG_imported_unit           0x3d /* DWARF3 */
00142 /* Do not use DW_TAG_mutable_type */
00143 #define DW_TAG_mutable_type            0x3e /*Withdrawn from DWARF3 by DWARF3f*/

```

```

00144 #define DW_TAG_condition 0x3f /* DWARF3f */
00145 #define DW_TAG_shared_type 0x40 /* DWARF3f */
00146 #define DW_TAG_type_unit 0x41 /* DWARF4 */
00147 #define DW_TAG_rvalue_reference_type 0x42 /* DWARF4 */
00148 #define DW_TAG_template_alias 0x43 /* DWARF4 */
00149 #define DW_TAG_coarray_type 0x44 /* DWARF5 */
00150 #define DW_TAG_generic_subrange 0x45 /* DWARF5 */
00151 #define DW_TAG_dynamic_type 0x46 /* DWARF5 */
00152 #define DW_TAG_atomic_type 0x47 /* DWARF5 */
00153 #define DW_TAG_call_site 0x48 /* DWARF5 */
00154 #define DW_TAG_call_site_parameter 0x49 /* DWARF5 */
00155 #define DW_TAG_skeleton_unit 0x4a /* DWARF5 */
00156 #define DW_TAG_immutable_type 0x4b /* DWARF5 */
00157
00158 /* TI = Texas Instruments, for DWARF in COFF */
00159 /* https://www.ti.com/lit/an/spraab5/spraab5.pdf?ts=1705994928599 */
00160
00161 #define DW_TAG_TI_far_type 0x4080 /* TI */
00162 #define DW_TAG_lo_user 0x4080 /* TI */
00163 #define DW_TAG_MIPS_loop 0x4081
00164 #define DW_TAG_TI_near_type 0x4081 /* TI */
00165 #define DW_TAG_TI_assign_register 0x4082 /* TI */
00166 #define DW_TAG_TI_ioport_type 0x4083 /* TI */
00167 #define DW_TAG_TI_restrict_type 0x4084 /* TI */
00168 #define DW_TAG_TI_onchip_type 0x4085 /* TI */
00169
00170 /* HP extensions: ftp://ftp.hp.com/pub/lang/tools/\
00171 WDB/wdb-4.0.tar.gz */
00172 #define DW_TAG_HP_array_descriptor 0x4090 /* HP */
00173
00174 /* GNU extensions. The first 3 missing the GNU_ */
00175 #define DW_TAG_format_label 0x4101 /* GNU. Fortran. */
00176 #define DW_TAG_GNU_function_template 0x4102 /* GNU. For C++ */
00177 #define DW_TAG_GNU_class_template 0x4103 /* GNU. For C++ */
00178 #define DW_TAG_GNU_BINCL 0x4104 /* GNU */
00179 #define DW_TAG_GNU_EINCL 0x4105 /* GNU */
00180
00181 /* GNU extension. http://gcc.gnu.org/wiki/TemplateParmsDwarf */
00182 #define DW_TAG_GNU_template_template_parameter 0x4106 /* GNU */
00183 #define DW_TAG_GNU_template_template_param 0x4106 /* GNU */
00184 #define DW_TAG_GNU_template_parameter_pack 0x4107 /* GNU */
00185 #define DW_TAG_GNU_formal_parameter_pack 0x4108 /* GNU */
00186
00187 #define DW_TAG_GNU_call_site 0x4109 /* GNU */
00188 #define DW_TAG_GNU_call_site_parameter 0x410a /* GNU */
00189
00190 /* The following are SUN extensions */
00191 #define DW_TAG_SUN_function_template 0x4201 /* SUN */
00192 #define DW_TAG_SUN_class_template 0x4202 /* SUN */
00193 #define DW_TAG_SUN_struct_template 0x4203 /* SUN */
00194 #define DW_TAG_SUN_union_template 0x4204 /* SUN */
00195 #define DW_TAG_SUN_indirect_inheritance 0x4205 /* SUN */
00196 #define DW_TAG_SUN_codeflags 0x4206 /* SUN */
00197 #define DW_TAG_SUN_memop_info 0x4207 /* SUN */
00198 #define DW_TAG_SUN_omp_child_func 0x4208 /* SUN */
00199 #define DW_TAG_SUN_rtti_descriptor 0x4209 /* SUN */
00200 #define DW_TAG_SUN_dtor_info 0x420a /* SUN */
00201 #define DW_TAG_SUN_dtor 0x420b /* SUN */
00202 #define DW_TAG_SUN_f90_interface 0x420c /* SUN */
00203 #define DW_TAG_SUN_fortran_vax_structure 0x420d /* SUN */
00204 #define DW_TAG_SUN_hi 0x42ff /* SUN */
00205
00206 /* ALTIUM extensions */
00207 /* DSP-C/Starcore __circ qualifier */
00208 #define DW_TAG_ALTIUM_circ_type 0x5101 /* ALTIUM */
00209 /* Starcore __mwa_circ qualifier */
00210 #define DW_TAG_ALTIUM_mwa_circ_type 0x5102 /* ALTIUM */
00211 /* Starcore __rev_carry qualifier */
00212 #define DW_TAG_ALTIUM_rev_carry_type 0x5103 /* ALTIUM */
00213 /* M16 __rom qualifier */
00214 #define DW_TAG_ALTIUM_rom 0x5111 /* ALTIUM */
00215
00216 #define DW_TAG_LLVM_annotation 0x6000 /* September 2021*/
00217
00218 /* GHS C */
00219 #define DW_TAG_ghs_namespace 0x8004
00220 #define DW_TAG_ghs_using_namespace 0x8005
00221 #define DW_TAG_ghs_using_declaration 0x8006
00222 #define DW_TAG_ghs_template_template_param 0x8007
00223
00224 /* The following 3 are extensions to support UPC */
00225 #define DW_TAG_upc_shared_type 0x8765 /* UPC */
00226 #define DW_TAG_upc_strict_type 0x8766 /* UPC */
00227 #define DW_TAG_upc_relaxed_type 0x8767 /* UPC */
00228
00229 /* PGI (STMicroelectronics) extensions. */
00230 #define DW_TAG_PGI_kanji_type 0xa000 /* PGI */

```

```

00231 #define DW_TAG_PGI_interface_block      0xa020 /* PGI */
00232
00233 #define DW_TAG_BORLAND_property          0xb000
00234 #define DW_TAG_BORLAND_Delphi_string     0xb001
00235 #define DW_TAG_BORLAND_Delphi_dynamic_array 0xb002
00236 #define DW_TAG_BORLAND_Delphi_set       0xb003
00237 #define DW_TAG_BORLAND_Delphi_variant   0xb004
00238
00239 #define DW_TAG_hi_user                    0xffff
00240
00241 /* The following two are non-standard. Use DW_CHILDREN_yes
00242    and DW_CHILDREN_no instead. These could
00243    probably be deleted, but someone might be using them,
00244    so they remain. */
00245 #define DW_children_no                    0
00246 #define DW_children_yes                  1
00247
00248 #define DW_FORM_addr                     0x01
00249 /* FORM_REF                             0x02 DWARF1 only */
00250 #define DW_FORM_block2                   0x03
00251 #define DW_FORM_block4                   0x04
00252 #define DW_FORM_data2                     0x05
00253 #define DW_FORM_data4                     0x06
00254 #define DW_FORM_data8                     0x07
00255 #define DW_FORM_string                   0x08
00256 #define DW_FORM_block                     0x09
00257 #define DW_FORM_block1                     0x0a
00258 #define DW_FORM_data1                     0x0b
00259 #define DW_FORM_flag                       0x0c
00260 #define DW_FORM_sdata                     0x0d
00261 #define DW_FORM_strp                       0x0e
00262 #define DW_FORM_uda                       0x0f
00263 #define DW_FORM_ref_addr                 0x10
00264 #define DW_FORM_ref1                      0x11
00265 #define DW_FORM_ref2                      0x12
00266 #define DW_FORM_ref4                      0x13
00267 #define DW_FORM_ref8                      0x14
00268 #define DW_FORM_ref_uda                   0x15
00269 #define DW_FORM_indirect                  0x16
00270 #define DW_FORM_sec_offset                0x17 /* DWARF4 */
00271 #define DW_FORM_exprloc                   0x18 /* DWARF4 */
00272 #define DW_FORM_flag_present              0x19 /* DWARF4 */
00273 #define DW_FORM_strx                      0x1a /* DWARF5 */
00274 #define DW_FORM_addrx                     0x1b /* DWARF5 */
00275 #define DW_FORM_ref_sup4                  0x1c /* DWARF5 */
00276 #define DW_FORM_strp_sup                  0x1d /* DWARF5 */
00277 #define DW_FORM_data16                    0x1e /* DWARF5 */
00278 #define DW_FORM_line_strp                 0x1f /* DWARF5 */
00279 #define DW_FORM_ref_sig8                  0x20 /* DWARF4 */
00280 #define DW_FORM_implicit_const            0x21 /* DWARF5 */
00281 #define DW_FORM_loclistx                  0x22 /* DWARF5 */
00282 #define DW_FORM_rnglistx                  0x23 /* DWARF5 */
00283 #define DW_FORM_ref_sup8                  0x24 /* DWARF5 */
00284 #define DW_FORM_strx1                     0x25 /* DWARF5 */
00285 #define DW_FORM_strx2                     0x26 /* DWARF5 */
00286 #define DW_FORM_strx3                     0x27 /* DWARF5 */
00287 #define DW_FORM_strx4                     0x28 /* DWARF5 */
00288 #define DW_FORM_addrx1                    0x29 /* DWARF5 */
00289 #define DW_FORM_addrx2                    0x2a /* DWARF5 */
00290 #define DW_FORM_addrx3                    0x2b /* DWARF5 */
00291 #define DW_FORM_addrx4                    0x2c /* DWARF5 */
00292
00293 /* Extensions http://gcc.gnu.org/wiki/DebugFission. */
00294 #define DW_FORM_GNU_addr_index 0x1f01 /* GNU, debug_info.dwo.*/
00295
00296 /* GNU, somewhat like DW_FORM_strp */
00297 #define DW_FORM_GNU_str_index 0x1f02
00298
00299 #define DW_FORM_GNU_ref_alt 0x1f20 /* GNU, Offset in .debug_info. */
00300
00301 /* GNU extension. Offset in .debug_str of another object file. */
00302 #define DW_FORM_GNU_strp_alt 0x1f21
00303
00304 #define DW_FORM_LLVM_addrx_offset          0x2001
00305
00306 #define DW_AT_sibling                      0x01
00307 #define DW_AT_location                     0x02
00308 #define DW_AT_name                         0x03
00309 /* reserved DWARF1                        0x04, DWARF1 only */
00310 /* AT_fund_type                           0x05, DWARF1 only */
00311 /* AT_mod_fund_type                       0x06, DWARF1 only */
00312 /* AT_user_def_type                       0x07, DWARF1 only */
00313 /* AT_mod_u_d_type                        0x08, DWARF1 only */
00314 #define DW_AT_ordering                     0x09
00315 #define DW_AT_subscr_data                  0x0a
00316 #define DW_AT_byte_size                    0x0b
00317 #define DW_AT_bit_offset                   0x0c

```

```

00318 #define DW_AT_bit_size                0x0d
00319 /* reserved DWARF1                    0x0d, DWARF1 only */
00320 #define DW_AT_element_list             0x0f
00321 #define DW_AT_stmt_list                0x10
00322 #define DW_AT_low_pc                  0x11
00323 #define DW_AT_high_pc                 0x12
00324 #define DW_AT_language                0x13
00325 #define DW_AT_member                  0x14
00326 #define DW_AT_discr                   0x15
00327 #define DW_AT_discr_value             0x16
00328 #define DW_AT_visibility               0x17
00329 #define DW_AT_import                  0x18
00330 #define DW_AT_string_length           0x19
00331 #define DW_AT_common_reference        0x1a
00332 #define DW_AT_comp_dir                0x1b
00333 #define DW_AT_const_value             0x1c
00334 #define DW_AT_containing_type         0x1d
00335 #define DW_AT_default_value           0x1e
00336 /* reserved                           0x1f */
00337 #define DW_AT_inline                   0x20
00338 #define DW_AT_is_optional              0x21
00339 #define DW_AT_lower_bound              0x22
00340 /* reserved                           0x23 */
00341 /* reserved                           0x24 */
00342 #define DW_AT_producer                 0x25
00343 /* reserved                           0x26 */
00344 #define DW_AT_prototyped               0x27
00345 /* reserved                           0x28 */
00346 /* reserved                           0x29 */
00347 #define DW_AT_return_addr              0x2a
00348 /* reserved                           0x2b */
00349 #define DW_AT_start_scope              0x2c
00350 /* reserved                           0x2d */
00351 #define DW_AT_bit_stride               0x2e /* DWARF3 name */
00352 #define DW_AT_stride_size              0x2e /* DWARF2 name */
00353 #define DW_AT_upper_bound              0x2f
00354 /* AT_virtual                         0x30, DWARF1 only */
00355 #define DW_AT_abstract_origin          0x31
00356 #define DW_AT_accessibility            0x32
00357 #define DW_AT_address_class            0x33
00358 #define DW_AT_artificial               0x34
00359 #define DW_AT_base_types               0x35
00360 #define DW_AT_calling_convention       0x36
00361 #define DW_AT_count                   0x37
00362 #define DW_AT_data_member_location     0x38
00363 #define DW_AT_decl_column              0x39
00364 #define DW_AT_decl_file                0x3a
00365 #define DW_AT_decl_line                0x3b
00366 #define DW_AT_declaration              0x3c
00367 #define DW_AT_discr_list               0x3d /* DWARF2 */
00368 #define DW_AT_encoding                 0x3e
00369 #define DW_AT_external                 0x3f
00370 #define DW_AT_frame_base               0x40
00371 #define DW_AT_friend                   0x41
00372 #define DW_AT_identifier_case          0x42
00373 #define DW_AT_macro_info               0x43 /* DWARF(234) not DWARF5 */
00374 #define DW_AT_namelist_item            0x44
00375 #define DW_AT_priority                 0x45
00376 #define DW_AT_segment                 0x46
00377 #define DW_AT_specification            0x47
00378 #define DW_AT_static_link              0x48
00379 #define DW_AT_type                     0x49
00380 #define DW_AT_use_location              0x4a
00381 #define DW_AT_variable_parameter       0x4b
00382 #define DW_AT_virtuality               0x4c
00383 #define DW_AT_vtable_elem_location     0x4d
00384 #define DW_AT_allocated                0x4e /* DWARF3 */
00385 #define DW_AT_associated                0x4f /* DWARF3 */
00386 #define DW_AT_data_location            0x50 /* DWARF3 */
00387 #define DW_AT_byte_stride               0x51 /* DWARF3f */
00388 #define DW_AT_stride                   0x51 /* DWARF3 (do not use) */
00389 #define DW_AT_entry_pc                 0x52 /* DWARF3 */
00390 #define DW_AT_use_UTF8                 0x53 /* DWARF3 */
00391 #define DW_AT_extension                 0x54 /* DWARF3 */
00392 #define DW_AT_ranges                   0x55 /* DWARF3 */
00393 #define DW_AT_trampoline               0x56 /* DWARF3 */
00394 #define DW_AT_call_column              0x57 /* DWARF3 */
00395 #define DW_AT_call_file                0x58 /* DWARF3 */
00396 #define DW_AT_call_line                0x59 /* DWARF3 */
00397 #define DW_AT_description               0x5a /* DWARF3 */
00398 #define DW_AT_binary_scale              0x5b /* DWARF3f */
00399 #define DW_AT_decimal_scale            0x5c /* DWARF3f */
00400 #define DW_AT_small                    0x5d /* DWARF3f */
00401 #define DW_AT_decimal_sign              0x5e /* DWARF3f */
00402 #define DW_AT_digit_count              0x5f /* DWARF3f */
00403 #define DW_AT_picture_string            0x60 /* DWARF3f */
00404 #define DW_AT_mutable                  0x61 /* DWARF3f */

```

```

00405 #define DW_AT_threads_scaled 0x62 /* DWARF3f */
00406 #define DW_AT_explicit 0x63 /* DWARF3f */
00407 #define DW_AT_object_pointer 0x64 /* DWARF3f */
00408 #define DW_AT_endianity 0x65 /* DWARF3f */
00409 #define DW_AT_elemental 0x66 /* DWARF3f */
00410 #define DW_AT_pure 0x67 /* DWARF3f */
00411 #define DW_AT_recursive 0x68 /* DWARF3f */
00412 #define DW_AT_signature 0x69 /* DWARF4 */
00413 #define DW_AT_main_subprogram 0x6a /* DWARF4 */
00414 #define DW_AT_data_bit_offset 0x6b /* DWARF4 */
00415 #define DW_AT_const_expr 0x6c /* DWARF4 */
00416 #define DW_AT_enum_class 0x6d /* DWARF4 */
00417 #define DW_AT_linkage_name 0x6e /* DWARF4 */
00418 #define DW_AT_string_length_bit_size 0x6f /* DWARF5 */
00419 #define DW_AT_string_length_byte_size 0x70 /* DWARF5 */
00420 #define DW_AT_rank 0x71 /* DWARF5 */
00421 #define DW_AT_str_offsets_base 0x72 /* DWARF5 */
00422 #define DW_AT_addr_base 0x73 /* DWARF5 */
00423 /* Use DW_AT_rnglists_base, DW_AT_ranges_base is obsolete as */
00424 /* it was only used in some DWARF5 drafts, not the final DWARF5. */
00425 #define DW_AT_rnglists_base 0x74 /* DWARF5 */
00426 /* DW_AT_dwo_id, an experiment in some DWARF4+. Not DWARF5! */
00427 #define DW_AT_dwo_id 0x75 /* DWARF4! */
00428 #define DW_AT_dwo_name 0x76 /* DWARF5 */
00429 #define DW_AT_reference 0x77 /* DWARF5 */
00430 #define DW_AT_rvalue_reference 0x78 /* DWARF5 */
00431 #define DW_AT_macros 0x79 /* DWARF5 */
00432 #define DW_AT_call_all_calls 0x7a /* DWARF5 */
00433 #define DW_AT_call_all_source_calls 0x7b /* DWARF5 */
00434 #define DW_AT_call_all_tail_calls 0x7c /* DWARF5 */
00435 #define DW_AT_call_return_pc 0x7d /* DWARF5 */
00436 #define DW_AT_call_value 0x7e /* DWARF5 */
00437 #define DW_AT_call_origin 0x7f /* DWARF5 */
00438 #define DW_AT_call_parameter 0x80 /* DWARF5 */
00439 #define DW_AT_call_pc 0x81 /* DWARF5 */
00440 #define DW_AT_call_tail_call 0x82 /* DWARF5 */
00441 #define DW_AT_call_target 0x83 /* DWARF5 */
00442 #define DW_AT_call_target_clobbered 0x84 /* DWARF5 */
00443 #define DW_AT_call_data_location 0x85 /* DWARF5 */
00444 #define DW_AT_call_data_value 0x86 /* DWARF5 */
00445 #define DW_AT_noreturn 0x87 /* DWARF5 */
00446 #define DW_AT_alignment 0x88 /* DWARF5 */
00447 #define DW_AT_export_symbols 0x89 /* DWARF5 */
00448 #define DW_AT_deleted 0x8a /* DWARF5 */
00449 #define DW_AT_defaulted 0x8b /* DWARF5 */
00450 #define DW_AT_loclists_base 0x8c /* DWARF5 */
00451 /* As of 6 January 2025 the DWARF committee promises
00452 not to change the name or the assigned number of
00453 the following two attributes. So
00454 compilers are free to use these now with DWARF 5
00455 or earlier. The applicable FORMs of are
00456 of form class constant (See DWARF5 Section 7.5.5 Classes
00457 and Forms). */
00458 #define DW_AT_language_name 0x90 /* DWARF6 */
00459 #define DW_AT_language_version 0x91 /* DWARF6 */
00460
00461 /* GreenHills, ghs.com GHS C */
00462 #define DW_AT_ghs_namespace_alias 0x806
00463 #define DW_AT_ghs_using_namespace 0x807
00464 #define DW_AT_ghs_using_declaration 0x808
00465
00466 /* In extensions, we attempt to include the vendor extension
00467 in the name even when the vendor leaves it out. */
00468 #define DW_AT_HP_block_index 0x2000 /* HP */
00469 /* 0x2000 follows extension so dwarfdump prints the
00470 most-likely-useful name. */
00471 #define DW_AT_lo_user 0x2000
00472
00473 #define DW_AT_TI_veneer 0x2000 /* TI */
00474
00475 #define DW_AT_MIPS_fde 0x2001 /* MIPS/SGI */
00476 #define DW_AT_TI_symbol_name 0x2001 /* TI */
00477 #define DW_AT_MIPS_loop_begin 0x2002 /* MIPS/SGI */
00478 #define DW_AT_MIPS_tail_loop_begin 0x2003 /* MIPS/SGI */
00479 #define DW_AT_MIPS_epilog_begin 0x2004 /* MIPS/SGI */
00480 #define DW_AT_MIPS_loop_unroll_factor 0x2005 /* MIPS/SGI */
00481 #define DW_AT_MIPS_software_pipeline_depth 0x2006 /* MIPS/SGI */
00482 #define DW_AT_MIPS_linkage_name 0x2007 /* MIPS/SGI, GNU, and others. */
00483 #define DW_AT_MIPS_stride 0x2008 /* MIPS/SGI */
00484 #define DW_AT_MIPS_abstract_name 0x2009 /* MIPS/SGI */
00485 #define DW_AT_MIPS_clone_origin 0x200a /* MIPS/SGI */
00486 #define DW_AT_MIPS_has_inlines 0x200b /* MIPS/SGI */
00487 #define DW_AT_TI_version 0x200b /* TI */
00488 #define DW_AT_MIPS_stride_byte 0x200c /* MIPS/SGI */
00489 #define DW_AT_TI_asm 0x200c /* TI */
00490 #define DW_AT_MIPS_stride_elem 0x200d /* MIPS/SGI */
00491 #define DW_AT_MIPS_ptr_dopetype 0x200e /* MIPS/SGI */

```



```

00492 #define DW_AT_TI_skeletal 0x200e /* TI */
00493 #define DW_AT_MIPS_allocatable_dopetype 0x200f /* MIPS/SGI */
00494 #define DW_AT_MIPS_assumed_shape_dopetype 0x2010 /* MIPS/SGI */
00495 #define DW_AT_MIPS_assumed_size 0x2011 /* MIPS/SGI */
00496 #define DW_AT_TT_interrupt 0x2011 /* TI */
00497
00498 /* HP extensions. */
00499 #define DW_AT_HP_unmodifiable 0x2001 /* conflict: MIPS */
00500 #define DW_AT_HP_prologue 0x2005 /* conflict: MIPS */
00501 #define DW_AT_HP_epilogue 0x2008 /* conflict: MIPS */
00502 #define DW_AT_HP_actuals_stmt_list 0x2010 /* conflict: MIPS */
00503 #define DW_AT_HP_proc_per_section 0x2011 /* conflict: MIPS */
00504 #define DW_AT_HP_raw_data_ptr 0x2012 /* HP */
00505 #define DW_AT_HP_pass_by_reference 0x2013 /* HP */
00506 #define DW_AT_HP_opt_level 0x2014 /* HP */
00507 #define DW_AT_HP_prof_version_id 0x2015 /* HP */
00508 #define DW_AT_HP_opt_flags 0x2016 /* HP */
00509 #define DW_AT_HP_cold_region_low_pc 0x2017 /* HP */
00510 #define DW_AT_HP_cold_region_high_pc 0x2018 /* HP */
00511 #define DW_AT_HP_all_variables_modifiable 0x2019 /* HP */
00512 #define DW_AT_HP_linkage_name 0x201a /* HP */
00513 #define DW_AT_HP_prof_flags 0x201b /* HP */
00514 #define DW_AT_HP_unit_name 0x201f /* HP */
00515 #define DW_AT_HP_unit_size 0x2020 /* HP */
00516 #define DW_AT_HP_widened_byte_size 0x2021 /* HP */
00517 #define DW_AT_HP_definition_points 0x2022 /* HP */
00518 #define DW_AT_HP_default_location 0x2023 /* HP */
00519 #define DW_AT_HP_is_result_param 0x2029 /* HP */
00520
00521 #define DW_AT_CPQ_discontig_ranges 0x2001 /* COMPAQ/HP */
00522 #define DW_AT_CPQ_semantic_events 0x2002 /* COMPAQ/HP */
00523 #define DW_AT_CPQ_split_lifetimes_var 0x2003 /* COMPAQ/HP */
00524 #define DW_AT_CPQ_split_lifetimes_rtn 0x2004 /* COMPAQ/HP */
00525 #define DW_AT_CPQ_prologue_length 0x2005 /* COMPAQ/HP */
00526
00527 /* From GHS C GreenHills ghs.com */
00528 #define DW_AT_ghs_mangled 0x2007 /* conflict MIPS */
00529 #define DW_AT_ghs_rsm 0x2083
00530 #define DW_AT_ghs_frsm 0x2085
00531 #define DW_AT_ghs_frames 0x2086
00532 #define DW_AT_ghs_rso 0x2087
00533 #define DW_AT_ghs_subcpu 0x2092
00534 #define DW_AT_ghs_lbrace_line 0x2093
00535
00536 #define DW_AT_INTEL_other_endian 0x2026 /* Intel, 1 if byte swapped.*/
00537
00538 /* GNU extensions. */
00539 #define DW_AT_sf_names 0x2101 /* GNU */
00540 #define DW_AT_src_info 0x2102 /* GNU */
00541 #define DW_AT_mac_info 0x2103 /* GNU */
00542 #define DW_AT_src_coords 0x2104 /* GNU */
00543 #define DW_AT_body_begin 0x2105 /* GNU */
00544 #define DW_AT_body_end 0x2106 /* GNU */
00545 #define DW_AT_GNU_vector 0x2107 /* GNU */
00546
00547 /* Thread safety, see
00548 http://gcc.gnu.org/wiki/ThreadSafetyAnnotation . */
00549 /* The values here are from gcc-4.6.2 include/dwarf2.h. The
00550 values are not given on the web page at all, nor on web pages
00551 it refers to. */
00552 #define DW_AT_GNU_guarded_by 0x2108 /* GNU */
00553 #define DW_AT_GNU_pt_guarded_by 0x2109 /* GNU */
00554 #define DW_AT_GNU_guarded 0x210a /* GNU */
00555 #define DW_AT_GNU_pt_guarded 0x210b /* GNU */
00556 #define DW_AT_GNU_locks_excluded 0x210c /* GNU */
00557 #define DW_AT_GNU_exclusive_locks_required 0x210d /* GNU */
00558 #define DW_AT_GNU_shared_locks_required 0x210e /* GNU */
00559
00560 /* See http://gcc.gnu.org/wiki/DwarfSeparateTypeInfo */
00561 #define DW_AT_GNU_odr_signature 0x210f /* GNU */
00562
00563 /* See http://gcc.gnu.org/wiki/TemplateParmsDwarf */
00564 /* The value here is from gcc-4.6.2 include/dwarf2.h. The value is
00565 not consistent with the web page as of December 2011. */
00566 #define DW_AT_GNU_template_name 0x2110 /* GNU */
00567 /* The GNU call site extension.
00568 See http://www.dwarfstd.org/ShowIssue.php?issue=100909.2&type=open . */
00569 #define DW_AT_GNU_call_site_value 0x2111 /* GNU */
00570 #define DW_AT_GNU_call_site_data_value 0x2112 /* GNU */
00571 #define DW_AT_GNU_call_site_target 0x2113 /* GNU */
00572 #define DW_AT_GNU_call_site_target_clobbered 0x2114 /* GNU */
00573 #define DW_AT_GNU_tail_call 0x2115 /* GNU */
00574 #define DW_AT_GNU_all_tail_call_sites 0x2116 /* GNU */
00575 #define DW_AT_GNU_all_call_sites 0x2117 /* GNU */
00576 #define DW_AT_GNU_all_source_call_sites 0x2118 /* GNU */
00577
00578 /* Section offset to .debug_macro section. */

```

```

00579 #define DW_AT_GNU_macros 0x2119 /* GNU */
00580 #define DW_AT_GNU_deleted 0x211a /* GNU */
00581 /* The GNU DebugFission project:
00582 http://gcc.gnu.org/wiki/DebugFission */
00583 #define DW_AT_GNU_dwo_name 0x2130 /* GNU */
00584 #define DW_AT_GNU_dwo_id 0x2131 /* GNU */
00585
00586 #define DW_AT_GNU_ranges_base 0x2132 /* GNU */
00587 #define DW_AT_GNU_addr_base 0x2133 /* GNU */
00588 #define DW_AT_GNU_pubnames 0x2134 /* GNU */
00589 #define DW_AT_GNU_pubtypes 0x2135 /* GNU */
00590
00591 /* To distinguish distinct basic blocks in a single source line. */
00592 #define DW_AT_GNU_discriminator 0x2136 /* GNU */
00593 #define DW_AT_GNU_locviews 0x2137 /* GNU */
00594 #define DW_AT_GNU_entry_view 0x2138 /* GNU */
00595
00596 /* Sun extensions */
00597 #define DW_AT_SUN_template 0x2201 /* SUN */
00598 #define DW_AT_VMS_rtnbeg_pd_address 0x2201 /* VMS */
00599 #define DW_AT_SUN_alignment 0x2202 /* SUN */
00600 #define DW_AT_SUN_vtable 0x2203 /* SUN */
00601 #define DW_AT_SUN_count_guarantee 0x2204 /* SUN */
00602 #define DW_AT_SUN_command_line 0x2205 /* SUN */
00603 #define DW_AT_SUN_vbase 0x2206 /* SUN */
00604 #define DW_AT_SUN_compile_options 0x2207 /* SUN */
00605 #define DW_AT_SUN_language 0x2208 /* SUN */
00606 #define DW_AT_SUN_browser_file 0x2209 /* SUN */
00607 #define DW_AT_SUN_vtable_abi 0x2210 /* SUN */
00608 #define DW_AT_SUN_func_offsets 0x2211 /* SUN */
00609 #define DW_AT_SUN_cf_kind 0x2212 /* SUN */
00610 #define DW_AT_SUN_vtable_index 0x2213 /* SUN */
00611 #define DW_AT_SUN_omp_tpriv_addr 0x2214 /* SUN */
00612 #define DW_AT_SUN_omp_child_func 0x2215 /* SUN */
00613 #define DW_AT_SUN_func_offset 0x2216 /* SUN */
00614 #define DW_AT_SUN_memop_type_ref 0x2217 /* SUN */
00615 #define DW_AT_SUN_profile_id 0x2218 /* SUN */
00616 #define DW_AT_SUN_memop_signature 0x2219 /* SUN */
00617 #define DW_AT_SUN_obj_dir 0x2220 /* SUN */
00618 #define DW_AT_SUN_obj_file 0x2221 /* SUN */
00619 #define DW_AT_SUN_original_name 0x2222 /* SUN */
00620 #define DW_AT_SUN_hwcprof_signature 0x2223 /* SUN */
00621 #define DW_AT_SUN_amd64_parmdump 0x2224 /* SUN */
00622 #define DW_AT_SUN_part_link_name 0x2225 /* SUN */
00623 #define DW_AT_SUN_link_name 0x2226 /* SUN */
00624 #define DW_AT_SUN_pass_with_const 0x2227 /* SUN */
00625 #define DW_AT_SUN_return_with_const 0x2228 /* SUN */
00626 #define DW_AT_SUN_import_by_name 0x2229 /* SUN */
00627 #define DW_AT_SUN_f90_pointer 0x222a /* SUN */
00628 #define DW_AT_SUN_pass_by_ref 0x222b /* SUN */
00629 #define DW_AT_SUN_f90_allocatable 0x222c /* SUN */
00630 #define DW_AT_SUN_f90_assumed_shape_array 0x222d /* SUN */
00631 #define DW_AT_SUN_c_vla 0x222e /* SUN */
00632 #define DW_AT_SUN_return_value_ptr 0x2230 /* SUN */
00633 #define DW_AT_SUN_dtor_start 0x2231 /* SUN */
00634 #define DW_AT_SUN_dtor_length 0x2232 /* SUN */
00635 #define DW_AT_SUN_dtor_state_initial 0x2233 /* SUN */
00636 #define DW_AT_SUN_dtor_state_final 0x2234 /* SUN */
00637 #define DW_AT_SUN_dtor_state_deltas 0x2235 /* SUN */
00638 #define DW_AT_SUN_import_by_lname 0x2236 /* SUN */
00639 #define DW_AT_SUN_f90_use_only 0x2237 /* SUN */
00640 #define DW_AT_SUN_namelist_spec 0x2238 /* SUN */
00641 #define DW_AT_SUN_is_omp_child_func 0x2239 /* SUN */
00642 #define DW_AT_SUN_fortran_main_alias 0x223a /* SUN */
00643 #define DW_AT_SUN_fortran_based 0x223b /* SUN */
00644
00645 /* ALTIUM extension: ALTIUM Compliant location lists (flag) */
00646 #define DW_AT_ALTIUM_loclist 0x2300 /* ALTIUM */
00647 /* Ada GNAT gcc attributes. constant integer forms. */
00648 /* See http://gcc.gnu.org/wiki/DW_AT_GNAT_descriptive_type . */
00649 #define DW_AT_use_GNAT_descriptive_type 0x2301
00650 #define DW_AT_GNAT_descriptive_type 0x2302
00651 #define DW_AT_GNU_numerator 0x2303 /* GNU */
00652 #define DW_AT_GNU_denominator 0x2304 /* GNU */
00653 /* See https://gcc.gnu.org/wiki/DW_AT_GNU_bias */
00654 #define DW_AT_GNU_bias 0x2305 /* GNU */
00655
00656 /* Go-specific type attributes
00657 Naming as lower-case go instead of GO is the choice
00658 the Go language folks chose, it seems. This is the
00659 common spelling for these. */
00660 #define DW_AT_go_kind 0x2900
00661 #define DW_AT_go_key 0x2901
00662 #define DW_AT_go_elem 0x2902
00663 /* Attribute for DW_TAG_member of a struct type.
00664 Nonzero value indicates the struct field is an embedded field.*/
00665 #define DW_AT_go_embedded_field 0x2903

```

```

00666 #define DW_AT_go_runtime_type 0x2904
00667 #define DW_AT_go_package_name 0x2905
00668 #define DW_AT_go_dict_index 0x2906
00669 #define DW_AT_go_closure_offset 0x2907
00670
00671 /* UPC extension. */
00672 #define DW_AT_upc_threads_scaled 0x3210 /* UPC */
00673
00674 #define DW_AT_IBM_wsa_addr 0x393e
00675 #define DW_AT_IBM_home_location 0x393f
00676 #define DW_AT_IBM_alt_srcview 0x3940
00677
00678 /* PGI (STMicroelectronics) extensions. */
00679 /* PGI. Block, constant, reference. This attribute is an ASTPLAB
00680 extension used to describe the array local base. */
00681 #define DW_AT_PGI_lbase 0x3a00
00682
00683 /* PGI. Block, constant, reference. ASTPLAB adds this attribute
00684 to describe the section offset, or the offset to the
00685 first element in the dimension. */
00686 #define DW_AT_PGI_soffset 0x3a01
00687
00688 /* PGI. Block, constant, reference. ASTPLAB adds this
00689 attribute to describe the linear stride or the distance
00690 between elements in the dimension. */
00691 #define DW_AT_PGI_lstride 0x3a02
00692
00693 #define DW_AT_BORLAND_property_read 0x3b11
00694 #define DW_AT_BORLAND_property_write 0x3b12
00695 #define DW_AT_BORLAND_property_implements 0x3b13
00696 #define DW_AT_BORLAND_property_index 0x3b14
00697 #define DW_AT_BORLAND_property_default 0x3b15
00698 #define DW_AT_BORLAND_Delphi_unit 0x3b20
00699 #define DW_AT_BORLAND_Delphi_class 0x3b21
00700 #define DW_AT_BORLAND_Delphi_record 0x3b22
00701 #define DW_AT_BORLAND_Delphi_metaclass 0x3b23
00702 #define DW_AT_BORLAND_Delphi_constructor 0x3b24
00703 #define DW_AT_BORLAND_Delphi_destructor 0x3b25
00704 #define DW_AT_BORLAND_Delphi_anonymous_method 0x3b26
00705 #define DW_AT_BORLAND_Delphi_interface 0x3b27
00706 #define DW_AT_BORLAND_Delphi_ABI 0x3b28
00707 #define DW_AT_BORLAND_Delphi_frameptr 0x3b30
00708 #define DW_AT_BORLAND_closure 0x3b31
00709
00710 #define DW_AT_LLVM_include_path 0x3e00
00711 #define DW_AT_LLVM_config_macros 0x3e01
00712 #define DW_AT_LLVM_sysroot 0x3e02
00713 #define DW_AT_LLVM_tag_offset 0x3e03
00714 /* LLVM intends to use 0x3e04 - 0x3e06 */
00715 #define DW_AT_LLVM_apinotes 0x3e07
00716 /* Next 6 are for Heterogeneous debugging */
00717 #define DW_AT_LLVM_active_lane 0x3e08
00718 #define DW_AT_LLVM_augmentation 0x3e09
00719 #define DW_AT_LLVM_lanes 0x3e0a
00720 #define DW_AT_LLVM_lane_pc 0x3e0b
00721 #define DW_AT_LLVM_vector_size 0x3e0c
00722
00723 #define DW_AT_APPLE_optimized 0x3fe1
00724 #define DW_AT_APPLE_flags 0x3fe2
00725 #define DW_AT_APPLE_isa 0x3fe3
00726 /* 0x3fe4 Also known as DW_AT_APPLE_closure, block preferred. */
00727 #define DW_AT_APPLE_block 0x3fe4
00728 /* The rest of APPLE here are in support of Objective C */
00729 #define DW_AT_APPLE_major_runtime_vers 0x3fe5
00730 #define DW_AT_APPLE_runtime_class 0x3fe6
00731 #define DW_AT_APPLE_omit_frame_ptr 0x3fe7
00732 #define DW_AT_APPLE_property_name 0x3fe8
00733 #define DW_AT_APPLE_property_getter 0x3fe9
00734 #define DW_AT_APPLE_property_setter 0x3fea
00735 #define DW_AT_APPLE_property_attribute 0x3feb
00736 #define DW_AT_APPLE_objc_complete_type 0x3fec
00737 #define DW_AT_APPLE_property 0x3fed
00738 #define DW_AT_APPLE_objc_direct 0x3fee
00739 #define DW_AT_APPLE_sdk 0x3fef
00740 #define DW_AT_APPLE_origin 0x3ff0
00741
00742 #define DW_AT_hi_user 0x3fff
00743
00744 /* OP values 0x01,0x02,0x04,0x05,0x07 are DWARF1 only */
00745 #define DW_OP_addr 0x03
00746 #define DW_OP_deref 0x06
00747 #define DW_OP_const1u 0x08
00748 #define DW_OP_const1s 0x09
00749 #define DW_OP_const2u 0x0a
00750 #define DW_OP_const2s 0x0b
00751 #define DW_OP_const4u 0x0c
00752 #define DW_OP_const4s 0x0d

```

```
00753 #define DW_OP_const8u      0x0e
00754 #define DW_OP_const8s      0x0f
00755 #define DW_OP_constu       0x10
00756 #define DW_OP_consts       0x11
00757 #define DW_OP_dup          0x12
00758 #define DW_OP_drop         0x13
00759 #define DW_OP_over         0x14
00760 #define DW_OP_pick         0x15
00761 #define DW_OP_swap         0x16
00762 #define DW_OP_rot          0x17
00763 #define DW_OP_xderef       0x18
00764 #define DW_OP_abs          0x19
00765 #define DW_OP_and          0x1a
00766 #define DW_OP_div          0x1b
00767 #define DW_OP_minus        0x1c
00768 #define DW_OP_mod          0x1d
00769 #define DW_OP_mul          0x1e
00770 #define DW_OP_neg          0x1f
00771 #define DW_OP_not          0x20
00772 #define DW_OP_or           0x21
00773 #define DW_OP_plus         0x22
00774 #define DW_OP_plus_uconst  0x23
00775 #define DW_OP_shl          0x24
00776 #define DW_OP_shr          0x25
00777 #define DW_OP_shra         0x26
00778 #define DW_OP_xor          0x27
00779 #define DW_OP_bra          0x28
00780 #define DW_OP_eq           0x29
00781 #define DW_OP_ge           0x2a
00782 #define DW_OP_gt           0x2b
00783 #define DW_OP_le           0x2c
00784 #define DW_OP_lt           0x2d
00785 #define DW_OP_ne           0x2e
00786 #define DW_OP_skip         0x2f
00787 #define DW_OP_lit0         0x30
00788 #define DW_OP_lit1         0x31
00789 #define DW_OP_lit2         0x32
00790 #define DW_OP_lit3         0x33
00791 #define DW_OP_lit4         0x34
00792 #define DW_OP_lit5         0x35
00793 #define DW_OP_lit6         0x36
00794 #define DW_OP_lit7         0x37
00795 #define DW_OP_lit8         0x38
00796 #define DW_OP_lit9         0x39
00797 #define DW_OP_lit10        0x3a
00798 #define DW_OP_lit11        0x3b
00799 #define DW_OP_lit12        0x3c
00800 #define DW_OP_lit13        0x3d
00801 #define DW_OP_lit14        0x3e
00802 #define DW_OP_lit15        0x3f
00803 #define DW_OP_lit16        0x40
00804 #define DW_OP_lit17        0x41
00805 #define DW_OP_lit18        0x42
00806 #define DW_OP_lit19        0x43
00807 #define DW_OP_lit20        0x44
00808 #define DW_OP_lit21        0x45
00809 #define DW_OP_lit22        0x46
00810 #define DW_OP_lit23        0x47
00811 #define DW_OP_lit24        0x48
00812 #define DW_OP_lit25        0x49
00813 #define DW_OP_lit26        0x4a
00814 #define DW_OP_lit27        0x4b
00815 #define DW_OP_lit28        0x4c
00816 #define DW_OP_lit29        0x4d
00817 #define DW_OP_lit30        0x4e
00818 #define DW_OP_lit31        0x4f
00819 #define DW_OP_reg0         0x50
00820 #define DW_OP_reg1         0x51
00821 #define DW_OP_reg2         0x52
00822 #define DW_OP_reg3         0x53
00823 #define DW_OP_reg4         0x54
00824 #define DW_OP_reg5         0x55
00825 #define DW_OP_reg6         0x56
00826 #define DW_OP_reg7         0x57
00827 #define DW_OP_reg8         0x58
00828 #define DW_OP_reg9         0x59
00829 #define DW_OP_reg10        0x5a
00830 #define DW_OP_reg11        0x5b
00831 #define DW_OP_reg12        0x5c
00832 #define DW_OP_reg13        0x5d
00833 #define DW_OP_reg14        0x5e
00834 #define DW_OP_reg15        0x5f
00835 #define DW_OP_reg16        0x60
00836 #define DW_OP_reg17        0x61
00837 #define DW_OP_reg18        0x62
00838 #define DW_OP_reg19        0x63
00839 #define DW_OP_reg20        0x64
```

```

00840 #define DW_OP_reg21      0x65
00841 #define DW_OP_reg22      0x66
00842 #define DW_OP_reg23      0x67
00843 #define DW_OP_reg24      0x68
00844 #define DW_OP_reg25      0x69
00845 #define DW_OP_reg26      0x6a
00846 #define DW_OP_reg27      0x6b
00847 #define DW_OP_reg28      0x6c
00848 #define DW_OP_reg29      0x6d
00849 #define DW_OP_reg30      0x6e
00850 #define DW_OP_reg31      0x6f
00851 #define DW_OP_breg0      0x70
00852 #define DW_OP_breg1      0x71
00853 #define DW_OP_breg2      0x72
00854 #define DW_OP_breg3      0x73
00855 #define DW_OP_breg4      0x74
00856 #define DW_OP_breg5      0x75
00857 #define DW_OP_breg6      0x76
00858 #define DW_OP_breg7      0x77
00859 #define DW_OP_breg8      0x78
00860 #define DW_OP_breg9      0x79
00861 #define DW_OP_breg10     0x7a
00862 #define DW_OP_breg11     0x7b
00863 #define DW_OP_breg12     0x7c
00864 #define DW_OP_breg13     0x7d
00865 #define DW_OP_breg14     0x7e
00866 #define DW_OP_breg15     0x7f
00867 #define DW_OP_breg16     0x80
00868 #define DW_OP_breg17     0x81
00869 #define DW_OP_breg18     0x82
00870 #define DW_OP_breg19     0x83
00871 #define DW_OP_breg20     0x84
00872 #define DW_OP_breg21     0x85
00873 #define DW_OP_breg22     0x86
00874 #define DW_OP_breg23     0x87
00875 #define DW_OP_breg24     0x88
00876 #define DW_OP_breg25     0x89
00877 #define DW_OP_breg26     0x8a
00878 #define DW_OP_breg27     0x8b
00879 #define DW_OP_breg28     0x8c
00880 #define DW_OP_breg29     0x8d
00881 #define DW_OP_breg30     0x8e
00882 #define DW_OP_breg31     0x8f
00883 #define DW_OP_regx       0x90
00884 #define DW_OP_fbreg      0x91
00885 #define DW_OP_bregx      0x92
00886 #define DW_OP_piece      0x93
00887 #define DW_OP_deref_size 0x94
00888 #define DW_OP_xderef_size 0x95
00889 #define DW_OP_nop        0x96
00890 #define DW_OP_push_object_address 0x97 /* DWARF3 */
00891 #define DW_OP_call2      0x98 /* DWARF3 */
00892 #define DW_OP_call4      0x99 /* DWARF3 */
00893 #define DW_OP_call_ref   0x9a /* DWARF3 */
00894 #define DW_OP_form_tls_address 0x9b /* DWARF3f */
00895 #define DW_OP_call_frame_cfa 0x9c /* DWARF3f */
00896 #define DW_OP_bit_piece  0x9d /* DWARF3f */
00897 #define DW_OP_implicit_value 0x9e /* DWARF4 */
00898 #define DW_OP_stack_value 0x9f /* DWARF4 */
00899 #define DW_OP_implicit_pointer 0xa0 /* DWARF5 */
00900 #define DW_OP_addrx      0xa1 /* DWARF5 */
00901 #define DW_OP_constx     0xa2 /* DWARF5 */
00902 #define DW_OP_entry_value 0xa3 /* DWARF5 */
00903 #define DW_OP_const_type  0xa4 /* DWARF5 */
00904 #define DW_OP_regval_type 0xa5 /* DWARF5 */
00905 #define DW_OP_deref_type  0xa6 /* DWARF5 */
00906 #define DW_OP_xderef_type 0xa7 /* DWARF5 */
00907 #define DW_OP_convert     0xa8 /* DWARF5 */
00908 #define DW_OP_reinterpret 0xa9 /* DWARF5 */
00909
00910 #define DW_OP_GNU_push_tls_address 0xe0 /* GNU */
00911 #define DW_OP_WASM_location 0xed
00912 #define DW_OP_WASM_location_int 0xee
00913
00914 /* Follows extension so dwarfdump prints the
00915 most-likely-useful name. */
00916 #define DW_OP_lo_user      0xe0
00917
00918 /* LLVM extensions. */
00919 #define DW_OP_LLVM_form_aspace_address 0xe1
00920 #define DW_OP_LLVM_push_lane 0xe2
00921 #define DW_OP_LLVM_offset 0xe3
00922 #define DW_OP_LLVM_offset_uconst 0xe4
00923 #define DW_OP_LLVM_bit_offset 0xe5
00924 #define DW_OP_LLVM_call_frame_entry_reg 0xe6
00925 #define DW_OP_LLVM_undefined 0xe7
00926 #define DW_OP_LLVM_aspace_bregx 0xe8

```

```

00927 #define DW_OP_LLVM_aspace_implicit_pointer 0xe9
00928 #define DW_OP_LLVM_piece_end 0xea
00929 #define DW_OP_LLVM_extend 0xeb
00930 #define DW_OP_LLVM_select_bit_piece 0xec
00931 /* HP extensions. */
00932 #define DW_OP_HP_unknown 0xe0 /* HP conflict: GNU */
00933 #define DW_OP_HP_is_value 0xe1 /* HP */
00934 #define DW_OP_HPfltconst4 0xe2 /* HP */
00935 #define DW_OP_HPfltconst8 0xe3 /* HP */
00936 #define DW_OP_HP_mod_range 0xe4 /* HP */
00937 #define DW_OP_HP_unmod_range 0xe5 /* HP */
00938 #define DW_OP_HP_tls 0xe6 /* HP */
00939
00940 /* Intel: made obsolete by DW_OP_bit_piece above. */
00941 #define DW_OP_INTEL_bit_piece 0xe8
00942
00943 /* Apple extension. */
00944 #define DW_OP_GNU_uninit 0xf0 /* GNU */
00945 #define DW_OP_APPLE_uninit 0xf0 /* Apple */
00946 #define DW_OP_GNU_encoded_addr 0xf1 /* GNU */
00947 #define DW_OP_GNU_implicit_pointer 0xf2 /* GNU */
00948 #define DW_OP_GNU_entry_value 0xf3 /* GNU */
00949 #define DW_OP_GNU_const_type 0xf4 /* GNU */
00950 #define DW_OP_GNU_regval_type 0xf5 /* GNU */
00951 #define DW_OP_GNU_deref_type 0xf6 /* GNU */
00952 #define DW_OP_GNU_convert 0xf7 /* GNU */
00953 #define DW_OP_GNU_reinterpret 0xf9 /* GNU */
00954 #define DW_OP_GNU_parameter_ref 0xfa /* GNU */
00955 #define DW_OP_GNU_addr_index 0xfb /* GNU Fission */
00956 #define DW_OP_GNU_const_index 0xfc /* GNU Fission */
00957 #define DW_OP_GNU_variable_value 0xfd /* GNU 2017 */
00958 #define DW_OP_PGI_omp_thread_num 0xf8 /* PGI (STMicroelectronics) */
00959
00960 #define DW_OP_hi_user 0xff
00961
00962 #define DW_ATE_address 0x01
00963 #define DW_ATE_boolean 0x02
00964 #define DW_ATE_complex_float 0x03
00965 #define DW_ATE_float 0x04
00966 #define DW_ATE_signed 0x05
00967 #define DW_ATE_signed_char 0x06
00968 #define DW_ATE_unsigned 0x07
00969 #define DW_ATE_unsigned_char 0x08
00970 #define DW_ATE_imaginary_float 0x09 /* DWARF3 */
00971 #define DW_ATE_packed_decimal 0x0a /* DWARF3f */
00972 #define DW_ATE_numeric_string 0x0b /* DWARF3f */
00973 #define DW_ATE_edited 0x0c /* DWARF3f */
00974 #define DW_ATE_signed_fixed 0x0d /* DWARF3f */
00975 #define DW_ATE_unsigned_fixed 0x0e /* DWARF3f */
00976 #define DW_ATE_decimal_float 0x0f /* DWARF3f */
00977 #define DW_ATE_UTF 0x10 /* DWARF4 */
00978 #define DW_ATE_UCS 0x11 /* DWARF5 */
00979 #define DW_ATE_ASCII 0x12 /* DWARF5 */
00980
00981 /* ALTIUM extensions. x80, x81 */
00982 #define DW_ATE_ALTIUM_fract 0x80 /* ALTIUM __fract type */
00983
00984 /* Follows extension so dwarfdump prints
00985    the most-likely-useful name. */
00986 #define DW_ATE_lo_user 0x80
00987
00988 /* Shown here to help dwarfdump build script. */
00989 #define DW_ATE_ALTIUM_accum 0x81 /* ALTIUM __accum type */
00990
00991 /* HP extensions. */
00992 #define DW_ATE_HP_float80 0x80 /* (80 bit). HP */
00993 #define DW_ATE_HP_complex_float80 0x81 /* Complex (80 bit). HP */
00994 #define DW_ATE_HP_float128 0x82 /* (128 bit). HP */
00995 #define DW_ATE_HP_complex_float128 0x83 /* Complex (128 bit). HP */
00996 #define DW_ATE_HP_floathpintel 0x84 /* (82 bit IA64). HP */
00997 #define DW_ATE_HP_imaginary_float80 0x85 /* HP */
00998 #define DW_ATE_HP_imaginary_float128 0x86 /* HP */
00999 #define DW_ATE_HP_VAX_float 0x88 /* F or G floating. */
01000 #define DW_ATE_HP_VAX_float_d 0x89 /* D floating. */
01001 #define DW_ATE_HP_packed_decimal 0x8a /* Cobol. */
01002 #define DW_ATE_HP_zoned_decimal 0x8b /* Cobol. */
01003 #define DW_ATE_HP_edited 0x8c /* Cobol. */
01004 #define DW_ATE_HP_signed_fixed 0x8d /* Cobol. */
01005 #define DW_ATE_HP_unsigned_fixed 0x8e /* Cobol. */
01006 #define DW_ATE_HP_VAX_complex_float 0x8f /* ForG floating complex. */
01007 #define DW_ATE_HP_VAX_complex_float_d 0x90 /* D floating complex. */
01008
01009 /* Sun extensions */
01010 #define DW_ATE_SUN_interval_float 0x91
01011
01012 /* Obsolete: See DW_ATE_imaginary_float */
01013 #define DW_ATE_SUN_imaginary_float 0x92 /* Really SUN 0x86 ? */

```



```

01014
01015 #define DW_ATE_hi_user          0xff
01016
01017 /* DWARF5 Defaulted Member Encodings. */
01018 #define DW_DEFAULTED_no          0x0    /* DWARF5 */
01019 #define DW_DEFAULTED_in_class    0x1    /* DWARF5 */
01020 #define DW_DEFAULTED_out_of_class 0x2    /* DWARF5 */
01021
01022 #define DW_IDX_compile_unit      0x1    /* DWARF5 */
01023 #define DW_IDX_type_unit         0x2    /* DWARF5 */
01024 #define DW_IDX_die_offset        0x3    /* DWARF5 */
01025 #define DW_IDX_parent            0x4    /* DWARF5 */
01026 #define DW_IDX_type_hash         0x5    /* DWARF5 */
01027 #define DW_IDX_GNU_internal      0x2000
01028 #define DW_IDX_lo_user           0x2000 /* DWARF5 */
01029 #define DW_IDX_GNU_external      0x2001
01030 #define DW_IDX_GNU_main          0x2002
01031 #define DW_IDX_GNU_language      0x2003
01032 #define DW_IDX_GNU_linkage_name  0x2004
01033 #define DW_IDX_hi_user           0x3fff /* DWARF5 */
01034
01035 /* These with not-quite-the-same-names were used in DWARF4
01036    We call then DW_LLEX.
01037    Never official and should not be used by anyone.*/
01038 #define DW_LLEX_end_of_list_entry 0x0
01039 #define DW_LLEX_base_address_selection_entry 0x01
01040 #define DW_LLEX_start_end_entry 0x02
01041 #define DW_LLEX_start_length_entry 0x03
01042 #define DW_LLEX_offset_pair_entry 0x04
01043
01044 /* DWARF5 Location List Entries in Split Objects */
01045 #define DW_LLE_end_of_list        0x0    /* DWARF5 */
01046 #define DW_LLE_base_addressx      0x01    /* DWARF5 */
01047 #define DW_LLE_startx_endx        0x02    /* DWARF5 */
01048 #define DW_LLE_startx_length      0x03    /* DWARF5 */
01049 #define DW_LLE_offset_pair        0x04    /* DWARF5 */
01050 #define DW_LLE_default_location   0x05    /* DWARF5 */
01051 #define DW_LLE_base_address       0x06    /* DWARF5 */
01052 #define DW_LLE_start_end          0x07    /* DWARF5 */
01053 #define DW_LLE_start_length       0x08    /* DWARF5 */
01054
01055 /* DWARF5 Range List Entries */
01056 #define DW_RLE_end_of_list        0x00    /* DWARF5 */
01057 #define DW_RLE_base_addressx      0x01    /* DWARF5 */
01058 #define DW_RLE_startx_endx        0x02    /* DWARF5 */
01059 #define DW_RLE_startx_length      0x03    /* DWARF5 */
01060 #define DW_RLE_offset_pair        0x04    /* DWARF5 */
01061 #define DW_RLE_base_address       0x05    /* DWARF5 */
01062 #define DW_RLE_start_end          0x06    /* DWARF5 */
01063 #define DW_RLE_start_length       0x07    /* DWARF5 */
01064
01065 /* GNUIndex encodings non-standard. New in 2020,
01066    used in .debug_gnu_pubnames .debug_gnu_pubtypes
01067    but no spellings provided in documentation. */
01068 #define DW_GNUIVIS_global 0
01069 #define DW_GNUIVIS_static 1
01070
01071 /* GNUIndex encodings non-standard. New in 2020,
01072    used in .debug_gnu_pubnames .debug_gnu_pubtypes
01073    but no spellings provided in documentation. */
01074 #define DW_GNUIKIND_none 0
01075 #define DW_GNUIKIND_type 1
01076 #define DW_GNUIKIND_variable 2
01077 #define DW_GNUIKIND_function 3
01078 #define DW_GNUIKIND_other 4
01079
01080 /* DWARF5 Unit header unit type encodings */
01081 #define DW_UT_compile 0x01 /* DWARF5 */
01082 #define DW_UT_type 0x02 /* DWARF5 */
01083 #define DW_UT_partial 0x03 /* DWARF5 */
01084 #define DW_UT_skeleton 0x04 /* DWARF5 */
01085 #define DW_UT_split_compile 0x05 /* DWARF5 */
01086 #define DW_UT_split_type 0x06 /* DWARF5 */
01087 #define DW_UT_lo_user 0x80 /* DWARF5 */
01088 #define DW_UT_hi_user 0xff /* DWARF5 */
01089
01090 /* DWARF5 DebugFission object section id values
01091    for .dwp object section offsets hash table.
01092    0 is reserved, not used.
01093    2 is actually reserved, not used in DWARF5.
01094    But 2 may be seen in some DWARF4 objects.
01095 */
01096 #define DW_SECT_INFO 1 /* .debug_info.dwo DWARF5 */
01097 #define DW_SECT_TYPES 2 /* .debug_types.dwo pre-DWARF5 */
01098 #define DW_SECT_ABBREV 3 /* .debug_abbrev.dwo DWARF5 */
01099 #define DW_SECT_LINE 4 /* .debug_line.dwo DWARF5 */
01100 #define DW_SECT_LOCLISTS 5 /* .debug_loclists.dwo DWARF5 */

```

```

01101 #define DW_SECT_STR_OFFSETS 6 /* .debug_str_offsets.dwo DWARF5 */
01102 #define DW_SECT_MACRO 7 /* .debug_macro.dwo DWARF5 */
01103 #define DW_SECT_RNGLISTS 8 /* .debug_rnglists.dwo DWARF5 */
01104
01105 /* Decimal Sign codes. */
01106 #define DW_DS_unsigned 0x01 /* DWARF3f */
01107 #define DW_DS_leading_overpunch 0x02 /* DWARF3f */
01108 #define DW_DS_trailing_overpunch 0x03 /* DWARF3f */
01109 #define DW_DS_leading_separate 0x04 /* DWARF3f */
01110 #define DW_DS_trailing_separate 0x05 /* DWARF3f */
01111
01112 /* Endian code name. */
01113 #define DW_END_default 0x00 /* DWARF3f */
01114 #define DW_END_big 0x01 /* DWARF3f */
01115 #define DW_END_little 0x02 /* DWARF3f */
01116
01117 #define DW_END_lo_user 0x40 /* DWARF3f */
01118 #define DW_END_hi_user 0xff /* DWARF3f */
01119
01120 /* For use with DW_TAG_SUN_codeflags
01121 If DW_TAG_SUN_codeflags is accepted as a dwarf standard, then
01122 standard dwarf ATCF entries start at 0x01 */
01123 #define DW_ATCF_lo_user 0x40 /* SUN */
01124 #define DW_ATCF_SUN_mop_bitfield 0x41 /* SUN */
01125 #define DW_ATCF_SUN_mop_spill 0x42 /* SUN */
01126 #define DW_ATCF_SUN_mop_scoppy 0x43 /* SUN */
01127 #define DW_ATCF_SUN_func_start 0x44 /* SUN */
01128 #define DW_ATCF_SUN_end_ctors 0x45 /* SUN */
01129 #define DW_ATCF_SUN_branch_target 0x46 /* SUN */
01130 #define DW_ATCF_SUN_mop_stack_probe 0x47 /* SUN */
01131 #define DW_ATCF_SUN_func_epilog 0x48 /* SUN */
01132 #define DW_ATCF_hi_user 0xff /* SUN */
01133
01134 /* Accessibility code name. */
01135 #define DW_ACCESS_public 0x01
01136 #define DW_ACCESS_protected 0x02
01137 #define DW_ACCESS_private 0x03
01138
01139 /* Visibility code name. */
01140 #define DW_VIS_local 0x01
01141 #define DW_VIS_exported 0x02
01142 #define DW_VIS_qualified 0x03
01143
01144 /* Virtuality code name. */
01145 #define DW_VIRTUALITY_none 0x00
01146 #define DW_VIRTUALITY_virtual 0x01
01147 #define DW_VIRTUALITY_pure_virtual 0x02
01148
01149 #define DW_LANG_C89 0x0001
01150 #define DW_LANG_C 0x0002
01151 #define DW_LANG_Ada83 0x0003
01152 #define DW_LANG_C_plus_plus 0x0004
01153 #define DW_LANG_Cobol74 0x0005
01154 #define DW_LANG_Cobol85 0x0006
01155 #define DW_LANG_Fortran77 0x0007
01156 #define DW_LANG_Fortran90 0x0008
01157 #define DW_LANG_Pascal83 0x0009
01158 #define DW_LANG_Modula2 0x000a
01159 #define DW_LANG_Java 0x000b /* DWARF3 */
01160 #define DW_LANG_C99 0x000c /* DWARF3 */
01161 #define DW_LANG_Ada95 0x000d /* DWARF3 */
01162 #define DW_LANG_Fortran95 0x000e /* DWARF3 */
01163 #define DW_LANG_PLI 0x000f /* DWARF3 */
01164 #define DW_LANG_ObjC 0x0010 /* DWARF3f */
01165 #define DW_LANG_ObjC_plus_plus 0x0011 /* DWARF3f */
01166 #define DW_LANG_UPC 0x0012 /* DWARF3f */
01167 #define DW_LANG_D 0x0013 /* DWARF3f */
01168 #define DW_LANG_Python 0x0014 /* DWARF4 */
01169 #define DW_LANG_OpenCL 0x0015 /* DWARF5 */
01170 #define DW_LANG_Go 0x0016 /* DWARF5 */
01171 #define DW_LANG_Modula3 0x0017 /* DWARF5 */
01172 #define DW_LANG_Haskell 0x0018 /* DWARF5 */
01173 #define DW_LANG_C_plus_plus_03 0x0019 /* DWARF5 */
01174 #define DW_LANG_C_plus_plus_11 0x001a /* DWARF5 */
01175 #define DW_LANG_OCaml 0x001b /* DWARF5 */
01176 #define DW_LANG_Rust 0x001c /* DWARF5 */
01177 #define DW_LANG_C11 0x001d /* DWARF5 */
01178 #define DW_LANG_Swift 0x001e /* DWARF5 */
01179 #define DW_LANG_Julia 0x001f /* DWARF5 */
01180 #define DW_LANG_Dylan 0x0020 /* DWARF5 */
01181 #define DW_LANG_C_plus_plus_14 0x0021 /* DWARF5 */
01182 #define DW_LANG_Fortran03 0x0022 /* DWARF5 */
01183 #define DW_LANG_Fortran08 0x0023 /* DWARF5 */
01184 #define DW_LANG_RenderScript 0x0024 /* DWARF5 */
01185 #define DW_LANG_BLISS 0x0025 /* DWARF5 */
01186 /* The committee has, in
01187 https://dwarfstd.org/languages-v6.html

```



```

01188     specified that these language code, may be
01189     used by compilers now, and promises these
01190     will not change. */
01191 #define DW_LANG_Kotlin                0x0026 /* DWARF6 */
01192 #define DW_LANG_Zig                   0x0027 /* DWARF6 */
01193 #define DW_LANG_Crystal                0x0028 /* DWARF6 */
01194 /* 0x0029 has not been assigned to a language. */
01195 #define DW_LANG_C_plus_plus_17        0x002a /* DWARF6 */
01196 #define DW_LANG_C_plus_plus_20        0x002b /* DWARF6 */
01197 #define DW_LANG_C17                    0x002c /* DWARF6 */
01198 #define DW_LANG_Fortran18              0x002d /* DWARF6 */
01199 #define DW_LANG_Ada2005                0x002e /* DWARF6 */
01200 #define DW_LANG_Ada2012                0x002f /* DWARF6 */
01201 #define DW_LANG_HIIP                   0x0030 /* DWARF6 */
01202 #define DW_LANG_Assembly               0x0031 /* DWARF6 */
01203 #define DW_LANG_C_sharp                0x0032 /* DWARF6 */
01204 #define DW_LANG_Mojo                   0x0033 /* DWARF6 */
01205 #define DW_LANG_GLSL                  0x0034 /* DWARF6 */
01206 #define DW_LANG_GLSL_ES                0x0035 /* DWARF6 */
01207 #define DW_LANG_HLSL                   0x0036 /* DWARF6 */
01208 #define DW_LANG_OpenCL_CPP             0x0037 /* DWARF6 */
01209 #define DW_LANG_CPP_for_OpenCL         0x0038 /* DWARF6 */
01210 #define DW_LANG_SYCL                   0x0039 /* DWARF6 */
01211 #define DW_LANG_C_plus_plus_23        0x003a /* DWARF6 */
01212 #define DW_LANG_Odin                   0x003b /* DWARF6 */
01213 #define DW_LANG_P4                     0x003c /* DWARF6 */
01214 #define DW_LANG_Metal                  0x003d /* DWARF6 */
01215 #define DW_LANG_C23                    0x003e /* DWARF6 */
01216 #define DW_LANG_Fortran23              0x003f /* DWARF6 */
01217 #define DW_LANG_Ruby                   0x0040 /* DWARF6 */
01218 #define DW_LANG_Move                   0x0041 /* DWARF6 */
01219 #define DW_LANG_Hylo                   0x0042 /* DWARF6 */
01220 #define DW_LANG_V                       0x0043 /* DWARF6 */
01221 #define DW_LANG_Algo168                0x0044 /* DWARF6 */
01222 #define DW_LANG_NIM                    0x0045 /* DWARF6 */
01223 #define DW_LANG_Erlang                  0x0046 /* DWARF6 */
01224 #define DW_LANG_Elixir                  0x0047 /* DWARF6 */
01225 #define DW_LANG_Gleam                   0x0048 /* DWARF6 */
01226
01227 #define DW_LANG_lo_user                 0x8000
01228 #define DW_LANG_Mips_Assembler          0x8001 /* MIPS */
01229 #define DW_LANG_Upc                     0x8765 /* UPC, use
01230     DW_LANG_UPC instead. */
01231 #define DW_LANG_GOOGLE_RenderScript    0x8e57
01232 #define DW_LANG_ALTIUM_Assembler        0x9101
01233 #define DW_LANG_BORLAND_Delphi          0xb000
01234
01235 /* Sun extensions */
01236 #define DW_LANG_SUN_Assembler            0x9001 /* SUN */
01237
01238 #define DW_LANG_hi_user                 0xffff
01239
01240 /* The committee has, in
01241     https://dwarfstd.org/languages-v6.html
01242     specified that these language code, may be
01243     used by compilers now, and promises these
01244     will not change. */
01245 #define DW_LNAME_Ada                   0x0001 /* DWARF6 */
01246 #define DW_LNAME_BLISS                  0x0002 /* DWARF6 */
01247 #define DW_LNAME_C                      0x0003 /* DWARF6 */
01248 #define DW_LNAME_C_plus_plus           0x0004 /* DWARF6 */
01249 #define DW_LNAME_Cobol                  0x0005 /* DWARF6 */
01250 #define DW_LNAME_Crystal                 0x0006 /* DWARF6 */
01251 #define DW_LNAME_D                      0x0007 /* DWARF6 */
01252 #define DW_LNAME_Dylan                  0x0008 /* DWARF6 */
01253 #define DW_LNAME_Fortran                 0x0009 /* DWARF6 */
01254 #define DW_LNAME_Go                     0x000a /* DWARF6 */
01255 #define DW_LNAME_Haskell                 0x000b /* DWARF6 */
01256 #define DW_LNAME_Java                    0x000c /* DWARF6 */
01257 #define DW_LNAME_Julia                   0x000d /* DWARF6 */
01258 #define DW_LNAME_Kotlin                  0x000e /* DWARF6 */
01259 #define DW_LNAME_Modula2                 0x000f /* DWARF6 */
01260 #define DW_LNAME_Modula3                 0x0010 /* DWARF6 */
01261 #define DW_LNAME_ObjC                    0x0011 /* DWARF6 */
01262 #define DW_LNAME_ObjC_plus_plus         0x0012 /* DWARF6 */
01263 #define DW_LNAME_OCaml                   0x0013 /* DWARF6 */
01264 #define DW_LNAME_OpenCL_C               0x0014 /* DWARF6 */
01265 #define DW_LNAME_Pascal                  0x0015 /* DWARF6 */
01266 #define DW_LNAME_PLI                     0x0016 /* DWARF6 */
01267 #define DW_LNAME_Python                  0x0017 /* DWARF6 */
01268 #define DW_LNAME_RenderScript            0x0018 /* DWARF6 */
01269 #define DW_LNAME_Rust                    0x0019 /* DWARF6 */
01270 #define DW_LNAME_Swift                   0x001a /* DWARF6 */
01271 #define DW_LNAME_UPC                     0x001b /* DWARF6 */
01272 #define DW_LNAME_Zig                     0x001c /* DWARF6 */
01273 #define DW_LNAME_Assembly                0x001d /* DWARF6 */
01274 #define DW_LNAME_C_sharp                 0x001e /* DWARF6 */

```

```

01275 #define DW_LNAME_Mojo 0x001f /* DWARF6 */
01276 #define DW_LNAME_GLSL 0x0020 /* DWARF6 */
01277 #define DW_LNAME_GLSL_ES 0x0021 /* DWARF6 */
01278 #define DW_LNAME_HLSL 0x0022 /* DWARF6 */
01279 #define DW_LNAME_OpenCL_CPP 0x0023 /* DWARF6 */
01280 #define DW_LNAME_CPP_for_OpenCL 0x0024 /* DWARF6 */
01281 #define DW_LNAME_SYCL 0x0025 /* DWARF6 */
01282 #define DW_LNAME_Ruby 0x0026 /* DWARF6 */
01283 #define DW_LNAME_Move 0x0027 /* DWARF6 */
01284 #define DW_LNAME_Hylo 0x0028 /* DWARF6 */
01285 #define DW_LNAME_HIP 0x0029 /* DWARF6 */
01286 #define DW_LNAME_Odin 0x002a /* DWARF6 */
01287 #define DW_LNAME_P4 0x002b /* DWARF6 */
01288 #define DW_LNAME_Metal 0x002c /* DWARF6 */
01289 #define DW_LNAME_V 0x002d /* DWARF6 */
01290 #define DW_LNAME_Algo168 0x002e /* DWARF6 */
01291 #define DW_LNAME_Nim 0x002f /* DWARF6 */
01292 #define DW_LNAME_Erlang 0x0030 /* DWARF6 */
01293 #define DW_LNAME_Elixir 0x0031 /* DWARF6 */
01294 #define DW_LNAME_Gleam 0x0032 /* DWARF6 */
01295
01296 /* Identifier case name. */
01297 #define DW_ID_case_sensitive 0x00
01298 #define DW_ID_up_case 0x01
01299 #define DW_ID_down_case 0x02
01300 #define DW_ID_case_insensitive 0x03
01301
01302 /* Calling Convention Name. */
01303 #define DW_CC_normal 0x01
01304 #define DW_CC_program 0x02
01305 #define DW_CC_nocall 0x03
01306 #define DW_CC_pass_by_reference 0x04 /* DWARF5 */
01307 #define DW_CC_pass_by_value 0x05 /* DWARF5 */
01308
01309 #define DW_CC_GNU_renesas_sh 0x40 /* GNU */
01310 #define DW_CC_lo_user 0x40
01311 #define DW_CC_GNU_borland_fastcall_i386 0x41 /* GNU */
01312
01313 /* ALTIUM extensions. */
01314 /* Function is an interrupt handler,
01315    return address on system stack. */
01316 #define DW_CC_ALTIUM_interrupt 0x65 /* ALTIUM */
01317
01318 /* Near function model, return address on system stack. */
01319 #define DW_CC_ALTIUM_near_system_stack 0x66 /* ALTIUM */
01320
01321 /* Near function model, return address on user stack. */
01322 #define DW_CC_ALTIUM_near_user_stack 0x67 /* ALTIUM */
01323
01324 /* Huge function model, return address on user stack. */
01325 #define DW_CC_ALTIUM_huge_user_stack 0x68 /* ALTIUM */
01326
01327 #define DW_CC_GNU_BORLAND_safecall 0xb0
01328 #define DW_CC_GNU_BORLAND_stdcall 0xb1
01329 #define DW_CC_GNU_BORLAND_pascal 0xb2
01330 #define DW_CC_GNU_BORLAND_msfastcall 0xb3
01331 #define DW_CC_GNU_BORLAND_msreturn 0xb4
01332 #define DW_CC_GNU_BORLAND_thiscall 0xb5
01333 #define DW_CC_GNU_BORLAND_fastcall 0xb6
01334
01335 #define DW_CC_LLVM_vectorcall 0xc0
01336 #define DW_CC_LLVM_Win64 0xc1
01337 #define DW_CC_LLVM_X86_64SysV 0xc2
01338 #define DW_CC_LLVM_AAPCS 0xc3
01339 #define DW_CC_LLVM_AAPCS_VFP 0xc4
01340 #define DW_CC_LLVM_IntelOclBicc 0xc5
01341 #define DW_CC_LLVM_SpirFunction 0xc6
01342 #define DW_CC_LLVM_OpenCLKernel 0xc7
01343 #define DW_CC_LLVM_Swift 0xc8
01344 #define DW_CC_LLVM_PreserveMost 0xc9
01345 #define DW_CC_LLVM_PreserveAll 0xca
01346 #define DW_CC_LLVM_X86RegCall 0xcb
01347 #define DW_CC_GDB_IBM_OpenCL 0xff
01348
01349 #define DW_CC_hi_user 0xff
01350
01351 /* Inline Code Name. */
01352 #define DW_INL_not_inlined 0x00
01353 #define DW_INL_inlined 0x01
01354 #define DW_INL_declared_not_inlined 0x02
01355 #define DW_INL_declared_inlined 0x03
01356
01357 /* Ordering Name. */
01358 #define DW_ORD_row_major 0x00
01359 #define DW_ORD_col_major 0x01
01360
01361 /* Discriminant Descriptor Name. */

```

```

01362 #define DW_DSC_label 0x00
01363 #define DW_DSC_range 0x01
01364
01365 /* Line number header entry format encodings. DWARF5 */
01366 #define DW_LNCT_path 0x1 /* DWARF5 */
01367 #define DW_LNCT_directory_index 0x2 /* DWARF5 */
01368 #define DW_LNCT_timestamp 0x3 /* DWARF5 */
01369 #define DW_LNCT_size 0x4 /* DWARF5 */
01370 #define DW_LNCT_MD5 0x5 /* DWARF5 */
01371 /* Experimental two-level line tables. Non standard */
01372 #define DW_LNCT_GNU_subprogram_name 0x6
01373 #define DW_LNCT_GNU_decl_file 0x7
01374 #define DW_LNCT_GNU_decl_line 0x8
01375 #define DW_LNCT_lo_user 0x2000 /* DWARF5 */
01376 #define DW_LNCT_LLVM_source 0x2001
01377 #define DW_LNCT_LLVM_is_MD5 0x2002
01378 #define DW_LNCT_hi_user 0x3fff /* DWARF5 */
01379
01380 /* Line number standard opcode name. */
01381 #define DW_LNS_copy 0x01
01382 #define DW_LNS_advance_pc 0x02
01383 #define DW_LNS_advance_line 0x03
01384 #define DW_LNS_set_file 0x04
01385 #define DW_LNS_set_column 0x05
01386 #define DW_LNS_negate_stmt 0x06
01387 #define DW_LNS_set_basic_block 0x07
01388 #define DW_LNS_const_add_pc 0x08
01389 #define DW_LNS_fixed_advance_pc 0x09
01390 #define DW_LNS_set_prologue_end 0x0a /* DWARF3 */
01391 #define DW_LNS_set_epilogue_begin 0x0b /* DWARF3 */
01392 #define DW_LNS_set_isa 0x0c /* DWARF3 */
01393
01394 /* Experimental two-level line tables. NOT STD DWARF5 */
01395 /* Not saying GNU or anything. There are no
01396 DW_LNS_lo_user or DW_LNS_hi_user values though.
01397 DW_LNS_set_address_from_logical and
01398 DW_LNS_set_subprogram being both 0xd
01399 to avoid using up more space in the special opcode table.
01400 EXPERIMENTAL DW_LNS follow.
01401 */
01402 #define DW_LNS_set_address_from_logical 0x0d /* Actuals table only */
01403 #define DW_LNS_set_subprogram 0x0d /* Logicals table only */
01404 #define DW_LNS_inlined_call 0x0e /* Logicals table only */
01405 #define DW_LNS_pop_context 0x0f /* Logicals table only */
01406
01407 /* Line number extended opcode name. */
01408 #define DW_LNE_end_sequence 0x01
01409 #define DW_LNE_set_address 0x02
01410 #define DW_LNE_define_file 0x03 /* DWARF4 and earlier only */
01411 #define DW_LNE_set_discriminator 0x04 /* DWARF4 */
01412
01413 /* HP extensions. */
01414 #define DW_LNE_HP_negate_is_UV_update 0x11 /* 17 HP */
01415 #define DW_LNE_HP_push_context 0x12 /* 18 HP */
01416 #define DW_LNE_HP_pop_context 0x13 /* 19 HP */
01417 #define DW_LNE_HP_set_file_line_column 0x14 /* 20 HP */
01418 #define DW_LNE_HP_set_routine_name 0x15 /* 21 HP */
01419 #define DW_LNE_HP_set_sequence 0x16 /* 22 HP */
01420 #define DW_LNE_HP_negate_post_semantics 0x17 /* 23 HP */
01421 #define DW_LNE_HP_negate_function_exit 0x18 /* 24 HP */
01422 #define DW_LNE_HP_negate_front_end_logical 0x19 /* 25 HP */
01423 #define DW_LNE_HP_define_proc 0x20 /* 32 HP */
01424
01425 #define DW_LNE_HP_source_file_correlation 0x80 /* HP */
01426 #define DW_LNE_lo_user 0x80 /* DWARF3 */
01427 #define DW_LNE_hi_user 0xff /* DWARF3 */
01428
01429 /* These are known values for DW_LNS_set_isa. */
01430 /* These identifiers are not defined by any DWARF standard. */
01431 #define DW_ISA_UNKNOWN 0
01432 /* The following two are ARM specific. */
01433 #define DW_ISA_ARM_thumb 1 /* ARM ISA */
01434 #define DW_ISA_ARM_arm 2 /* ARM ISA */
01435
01436 /* Macro information, DWARF5 */
01437 #define DW_MACRO_define 0x01 /* DWARF5 */
01438 #define DW_MACRO_undef 0x02 /* DWARF5 */
01439 #define DW_MACRO_start_file 0x03 /* DWARF5 */
01440 #define DW_MACRO_end_file 0x04 /* DWARF5 */
01441 #define DW_MACRO_define_strp 0x05 /* DWARF5 */
01442 #define DW_MACRO_undef_strp 0x06 /* DWARF5 */
01443 #define DW_MACRO_import 0x07 /* DWARF5 */
01444 #define DW_MACRO_define_sup 0x08 /* DWARF5 */
01445 #define DW_MACRO_undef_sup 0x09 /* DWARF5 */
01446 #define DW_MACRO_import_sup 0x0a /* DWARF5 */
01447 #define DW_MACRO_define_strx 0x0b /* DWARF5 */
01448 #define DW_MACRO_undef_strx 0x0c /* DWARF5 */

```

```

01449 #define DW_MACRO_lo_user          0xe0
01450 #define DW_MACRO_hi_user          0xff
01451
01452 /* Macro information, DWARF2-DWARF4. */
01453 #define DW_MACRO_define            0x01
01454 #define DW_MACRO_undef            0x02
01455 #define DW_MACRO_start_file       0x03
01456 #define DW_MACRO_end_file         0x04
01457 #define DW_MACRO_vendor_ext       0xff
01458
01459 /* CFA operator compaction (a space saving measure, see
01460    the DWARF standard) means DW_CFA_extended and DW_CFA_nop
01461    have the same value here. */
01462 #define DW_CFA_advance_loc        0x40
01463 #define DW_CFA_offset             0x80
01464 #define DW_CFA_restore            0xc0
01465 #define DW_CFA_nop                0x00
01466 #define DW_CFA_extended           0
01467 #define DW_CFA_set_loc            0x01
01468 #define DW_CFA_advance_loc1       0x02
01469 #define DW_CFA_advance_loc2       0x03
01470 #define DW_CFA_advance_loc4       0x04
01471 #define DW_CFA_offset_extended    0x05
01472 #define DW_CFA_restore_extended   0x06
01473 #define DW_CFA_undefined          0x07
01474 #define DW_CFA_same_value         0x08
01475 #define DW_CFA_register           0x09
01476 #define DW_CFA_remember_state     0x0a
01477 #define DW_CFA_restore_state      0x0b
01478 #define DW_CFA_def_cfa            0x0c
01479 #define DW_CFA_def_cfa_register   0x0d
01480 #define DW_CFA_def_cfa_offset     0x0e
01481 #define DW_CFA_def_cfa_expression 0x0f /* DWARF3 */
01482 #define DW_CFA_expression         0x10 /* DWARF3 */
01483 #define DW_CFA_offset_extended_sf 0x11 /* DWARF3 */
01484 #define DW_CFA_def_cfa_sf         0x12 /* DWARF3 */
01485 #define DW_CFA_def_cfa_offset_sf  0x13 /* DWARF3 */
01486 #define DW_CFA_val_offset         0x14 /* DWARF3f */
01487 #define DW_CFA_val_offset_sf      0x15 /* DWARF3f */
01488 #define DW_CFA_val_expression     0x16 /* DWARF3f */
01489 #define DW_CFA_TF_offset_extended 0x1c /* TI */
01490 #define DW_CFA_lo_user            0x1c
01491 #define DW_CFA_low_user           0x1c /* Incorrect spelling, do not use. */
01492
01493 /* SGI/MIPS extension. */
01494 #define DW_CFA_MIPS_advance_loc8   0x1d /* MIPS */
01495 #define DW_CFA_TF_def_cfa_offset   0x1d /* TI */
01496
01497 /* GNU extensions. */
01498 #define DW_CFA_GNU_window_save     0x2d /* GNU */
01499 #define DW_CFA_AARCH64_negate_ra_state 0x2d
01500 #define DW_CFA_GNU_args_size       0x2e /* GNU */
01501 #define DW_CFA_GNU_negative_offset_extended 0x2f /* GNU */
01502 #define DW_CFA_LLVM_def_aspace_cfa 0x30
01503 #define DW_CFA_LLVM_def_aspace_cfa_sf 0x31
01504
01505 /* Metaware if HC is augmentation, apparently meaning High C
01506    and the op has a single uleb operand.
01507    See http://sourceforge.net/p/elftoolchain/tickets/397/ */
01508 #define DW_CFA_METAWARE_info       0x34
01509
01510 #define DW_CFA_hi_user             0x3f
01511 #define DW_CFA_high_user           0x3f /* Misspelled. Do not use. */
01512
01513 /* GNU exception header encoding. See the Generic
01514    Elf Specification of the Linux Standard Base (LSB).
01515    http://refspecs.freestdards.org/LSB\_3.0.0/
01516    LSB-Core-generic/LSB-Core-generic/dwarfext.html
01517    The upper 4 bits indicate how the value is to be applied.
01518    The lower 4 bits indicate the format of the data.
01519    These identifiers are not defined by any DWARF standard.
01520 */
01521 #define DW_EH_PE_absptr            0x00 /* GNU */
01522 #define DW_EH_PE_uleb128           0x01 /* GNU */
01523 #define DW_EH_PE_udata2            0x02 /* GNU */
01524 #define DW_EH_PE_udata4            0x03 /* GNU */
01525 #define DW_EH_PE_udata8            0x04 /* GNU */
01526 #define DW_EH_PE_sleb128           0x09 /* GNU */
01527 #define DW_EH_PE_sdata2            0x0a /* GNU */
01528 #define DW_EH_PE_sdata4            0x0b /* GNU */
01529 #define DW_EH_PE_sdata8            0x0c /* GNU */
01530
01531 #define DW_EH_PE_pcrel             0x10 /* GNU */
01532 #define DW_EH_PE_textrel           0x20 /* GNU */
01533 #define DW_EH_PE_datarel           0x30 /* GNU */
01534 #define DW_EH_PE_funcrel           0x40 /* GNU */
01535 #define DW_EH_PE_aligned           0x50 /* GNU */

```

```
01536
01537 #define DW_EH_PE_omit      0xff /* GNU. Means no value present. */
01538
01539 /* Mapping from machine registers and pseudo-regs into the
01540    .debug_frame table. DW_FRAME entries are machine specific.
01541    These describe MIPS/SGI R3000, R4K, R4400 and all later
01542    MIPS/SGI IRIX machines. They describe a mapping from
01543    hardware register number to the number used in the table
01544    to identify that register.
01545
01546    The CFA (Canonical Frame Address) described in DWARF is
01547    called the Virtual Frame Pointer on MIPS/SGI machines.
01548
01549    The DW_FRAME* names here are MIPS/SGI specific.
01550    Libdwarf interfaces defined in 2008 make the
01551    frame definitions here (and the fixed table sizes
01552    they imply) obsolete. They are left here for compatibility.
01553 */
01554 /* These identifiers are not defined by any DWARF standard. */
01555
01556 #define DW_FRAME_REG1      1 /* integer reg 1 */
01557 #define DW_FRAME_REG2      2 /* integer reg 2 */
01558 #define DW_FRAME_REG3      3 /* integer reg 3 */
01559 #define DW_FRAME_REG4      4 /* integer reg 4 */
01560 #define DW_FRAME_REG5      5 /* integer reg 5 */
01561 #define DW_FRAME_REG6      6 /* integer reg 6 */
01562 #define DW_FRAME_REG7      7 /* integer reg 7 */
01563 #define DW_FRAME_REG8      8 /* integer reg 8 */
01564 #define DW_FRAME_REG9      9 /* integer reg 9 */
01565 #define DW_FRAME_REG10     10 /* integer reg 10 */
01566 #define DW_FRAME_REG11     11 /* integer reg 11 */
01567 #define DW_FRAME_REG12     12 /* integer reg 12 */
01568 #define DW_FRAME_REG13     13 /* integer reg 13 */
01569 #define DW_FRAME_REG14     14 /* integer reg 14 */
01570 #define DW_FRAME_REG15     15 /* integer reg 15 */
01571 #define DW_FRAME_REG16     16 /* integer reg 16 */
01572 #define DW_FRAME_REG17     17 /* integer reg 17 */
01573 #define DW_FRAME_REG18     18 /* integer reg 18 */
01574 #define DW_FRAME_REG19     19 /* integer reg 19 */
01575 #define DW_FRAME_REG20     20 /* integer reg 20 */
01576 #define DW_FRAME_REG21     21 /* integer reg 21 */
01577 #define DW_FRAME_REG22     22 /* integer reg 22 */
01578 #define DW_FRAME_REG23     23 /* integer reg 23 */
01579 #define DW_FRAME_REG24     24 /* integer reg 24 */
01580 #define DW_FRAME_REG25     25 /* integer reg 25 */
01581 #define DW_FRAME_REG26     26 /* integer reg 26 */
01582 #define DW_FRAME_REG27     27 /* integer reg 27 */
01583 #define DW_FRAME_REG28     28 /* integer reg 28 */
01584 #define DW_FRAME_REG29     29 /* integer reg 29 */
01585 #define DW_FRAME_REG30     30 /* integer reg 30 */
01586 #define DW_FRAME_REG31     31 /* integer reg 31, aka ra */
01587
01588 /* MIPS1,2 have only some of these 64-bit registers.
01589 ** MIPS1 save/restore takes 2 instructions per 64-bit reg, and
01590 ** in that case, the register is considered stored after
01591 ** the second swc1. */
01592 #define DW_FRAME_FREG0     32 /* 64-bit floating point reg 0 */
01593 #define DW_FRAME_FREG1     33 /* 64-bit floating point reg 1 */
01594 #define DW_FRAME_FREG2     34 /* 64-bit floating point reg 2 */
01595 #define DW_FRAME_FREG3     35 /* 64-bit floating point reg 3 */
01596 #define DW_FRAME_FREG4     36 /* 64-bit floating point reg 4 */
01597 #define DW_FRAME_FREG5     37 /* 64-bit floating point reg 5 */
01598 #define DW_FRAME_FREG6     38 /* 64-bit floating point reg 6 */
01599 #define DW_FRAME_FREG7     39 /* 64-bit floating point reg 7 */
01600 #define DW_FRAME_FREG8     40 /* 64-bit floating point reg 8 */
01601 #define DW_FRAME_FREG9     41 /* 64-bit floating point reg 9 */
01602 #define DW_FRAME_FREG10    42 /* 64-bit floating point reg 10 */
01603 #define DW_FRAME_FREG11    43 /* 64-bit floating point reg 11 */
01604 #define DW_FRAME_FREG12    44 /* 64-bit floating point reg 12 */
01605 #define DW_FRAME_FREG13    45 /* 64-bit floating point reg 13 */
01606 #define DW_FRAME_FREG14    46 /* 64-bit floating point reg 14 */
01607 #define DW_FRAME_FREG15    47 /* 64-bit floating point reg 15 */
01608 #define DW_FRAME_FREG16    48 /* 64-bit floating point reg 16 */
01609 #define DW_FRAME_FREG17    49 /* 64-bit floating point reg 17 */
01610 #define DW_FRAME_FREG18    50 /* 64-bit floating point reg 18 */
01611 #define DW_FRAME_FREG19    51 /* 64-bit floating point reg 19 */
01612 #define DW_FRAME_FREG20    52 /* 64-bit floating point reg 20 */
01613 #define DW_FRAME_FREG21    53 /* 64-bit floating point reg 21 */
01614 #define DW_FRAME_FREG22    54 /* 64-bit floating point reg 22 */
01615 #define DW_FRAME_FREG23    55 /* 64-bit floating point reg 23 */
01616 #define DW_FRAME_FREG24    56 /* 64-bit floating point reg 24 */
01617 #define DW_FRAME_FREG25    57 /* 64-bit floating point reg 25 */
01618 #define DW_FRAME_FREG26    58 /* 64-bit floating point reg 26 */
01619 #define DW_FRAME_FREG27    59 /* 64-bit floating point reg 27 */
01620 #define DW_FRAME_FREG28    60 /* 64-bit floating point reg 28 */
01621 #define DW_FRAME_FREG29    61 /* 64-bit floating point reg 29 */
01622 #define DW_FRAME_FREG30    62 /* 64-bit floating point reg 30 */
```

```

01623 #define DW_FRAME_FREG31 63 /* 64-bit floating point reg 31 */
01624
01625 #define DW_FRAME_FREG32 64 /* 64-bit floating point reg 32 */
01626 #define DW_FRAME_FREG33 65 /* 64-bit floating point reg 33 */
01627 #define DW_FRAME_FREG34 66 /* 64-bit floating point reg 34 */
01628 #define DW_FRAME_FREG35 67 /* 64-bit floating point reg 35 */
01629 #define DW_FRAME_FREG36 68 /* 64-bit floating point reg 36 */
01630 #define DW_FRAME_FREG37 69 /* 64-bit floating point reg 37 */
01631 #define DW_FRAME_FREG38 70 /* 64-bit floating point reg 38 */
01632 #define DW_FRAME_FREG39 71 /* 64-bit floating point reg 39 */
01633 #define DW_FRAME_FREG40 72 /* 64-bit floating point reg 40 */
01634 #define DW_FRAME_FREG41 73 /* 64-bit floating point reg 41 */
01635 #define DW_FRAME_FREG42 74 /* 64-bit floating point reg 42 */
01636 #define DW_FRAME_FREG43 75 /* 64-bit floating point reg 43 */
01637 #define DW_FRAME_FREG44 76 /* 64-bit floating point reg 44 */
01638 #define DW_FRAME_FREG45 77 /* 64-bit floating point reg 45 */
01639 #define DW_FRAME_FREG46 78 /* 64-bit floating point reg 46 */
01640 #define DW_FRAME_FREG47 79 /* 64-bit floating point reg 47 */
01641 #define DW_FRAME_FREG48 80 /* 64-bit floating point reg 48 */
01642 #define DW_FRAME_FREG49 81 /* 64-bit floating point reg 49 */
01643 #define DW_FRAME_FREG50 82 /* 64-bit floating point reg 50 */
01644 #define DW_FRAME_FREG51 83 /* 64-bit floating point reg 51 */
01645 #define DW_FRAME_FREG52 84 /* 64-bit floating point reg 52 */
01646 #define DW_FRAME_FREG53 85 /* 64-bit floating point reg 53 */
01647 #define DW_FRAME_FREG54 86 /* 64-bit floating point reg 54 */
01648 #define DW_FRAME_FREG55 87 /* 64-bit floating point reg 55 */
01649 #define DW_FRAME_FREG56 88 /* 64-bit floating point reg 56 */
01650 #define DW_FRAME_FREG57 89 /* 64-bit floating point reg 57 */
01651 #define DW_FRAME_FREG58 90 /* 64-bit floating point reg 58 */
01652 #define DW_FRAME_FREG59 91 /* 64-bit floating point reg 59 */
01653 #define DW_FRAME_FREG60 92 /* 64-bit floating point reg 60 */
01654 #define DW_FRAME_FREG61 93 /* 64-bit floating point reg 61 */
01655 #define DW_FRAME_FREG62 94 /* 64-bit floating point reg 62 */
01656 #define DW_FRAME_FREG63 95 /* 64-bit floating point reg 63 */
01657 #define DW_FRAME_FREG64 96 /* 64-bit floating point reg 64 */
01658 #define DW_FRAME_FREG65 97 /* 64-bit floating point reg 65 */
01659 #define DW_FRAME_FREG66 98 /* 64-bit floating point reg 66 */
01660 #define DW_FRAME_FREG67 99 /* 64-bit floating point reg 67 */
01661 #define DW_FRAME_FREG68 100 /* 64-bit floating point reg 68 */
01662 #define DW_FRAME_FREG69 101 /* 64-bit floating point reg 69 */
01663 #define DW_FRAME_FREG70 102 /* 64-bit floating point reg 70 */
01664 #define DW_FRAME_FREG71 103 /* 64-bit floating point reg 71 */
01665 #define DW_FRAME_FREG72 104 /* 64-bit floating point reg 72 */
01666 #define DW_FRAME_FREG73 105 /* 64-bit floating point reg 73 */
01667 #define DW_FRAME_FREG74 106 /* 64-bit floating point reg 74 */
01668 #define DW_FRAME_FREG75 107 /* 64-bit floating point reg 75 */
01669 #define DW_FRAME_FREG76 108 /* 64-bit floating point reg 76 */
01670
01671 /* Having DW_FRAME_HIGHEST_NORMAL_REGISTER be higher than
01672    is strictly needed ... is safe.
01673    These values can be changed at runtime by libdwarf.
01674 */
01675 #ifndef DW_FRAME_HIGHEST_NORMAL_REGISTER
01676 #define DW_FRAME_HIGHEST_NORMAL_REGISTER 188
01677 #endif
01678 /* This is the number of columns in the Frame Table.
01679 */
01680 #ifndef DW_FRAME_LAST_REG_NUM
01681 #define DW_FRAME_LAST_REG_NUM (DW_FRAME_HIGHEST_NORMAL_REGISTER + 1)
01682 #endif
01683
01684 #define DW_CHILDREN_no 0x00
01685 #define DW_CHILDREN_yes 0x01
01686
01687 #define DW_ADDR_none 0
01688 #define DW_ADDR_TI_PTR8 0x0008 /* TI */
01689 #define DW_ADDR_TI_PTR16 0x0010 /* TI */
01690 #define DW_ADDR_TI_PTR22 0x0016 /* TI */
01691 #define DW_ADDR_TI_PTR23 0x0017 /* TI */
01692 #define DW_ADDR_TI_PTR24 0x0018 /* TI */
01693 #define DW_ADDR_TI_PTR32 0x0020 /* TI */
01694
01695 #ifdef __cplusplus
01696 }
01697 #endif
01698 #endif /* __DWARF_H */

```


Chapter 14

libdwarf.h

[libdwarf.h](#) contains all the type declarations and function declarations needed to use the library. It is essential that coders include [dwarf.h](#) before including [libdwarf.h](#).

All identifiers here in the public namespace begin with DW_ or Dwarf_ or dwarf_ . All function argument names declared here begin with dw_ .

14.1 libdwarf.h

[Go to the documentation of this file.](#)

```
00001 /*
00002  Copyright (C) 2000-2010 Silicon Graphics, Inc. All Rights Reserved.
00003  Portions Copyright 2007-2010 Sun Microsystems, Inc. All rights reserved.
00004  Portions Copyright 2008-2024 David Anderson. All rights reserved.
00005  Portions Copyright 2008-2010 Arxan Technologies, Inc. All rights reserved.
00006  Portions Copyright 2010-2012 SN Systems Ltd. All rights reserved.
00007
00008  This program is free software; you can redistribute it
00009  and/or modify it under the terms of version 2.1 of the
00010  GNU Lesser General Public License as published by the Free
00011  Software Foundation.
00012
00013  This program is distributed in the hope that it would be
00014  useful, but WITHOUT ANY WARRANTY; without even the implied
00015  warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
00016  PURPOSE.
00017
00018  Further, this software is distributed without any warranty
00019  that it is free of the rightful claim of any third person
00020  regarding infringement or the like. Any license provided
00021  herein, whether implied or otherwise, applies only to this
00022  software file. Patent licenses, if any, provided herein
00023  do not apply to combinations of this program with other
00024  software, or any other product whatsoever.
00025
00026  You should have received a copy of the GNU Lesser General
00027  Public License along with this program; if not, write the
00028  Free Software Foundation, Inc., 51 Franklin Street - Fifth
00029  Floor, Boston MA 02110-1301, USA.
00030
00031 */
00046 #ifndef _LIBDWARF_H
00047 #define _LIBDWARF_H
00048
00049 #ifdef DW_API
00050 #undef DW_API
00051 #endif /* DW_API */
00052
00053 #ifndef LIBDWARF_STATIC
00054 # if defined(_WIN32) || defined(__CYGWIN__)
00055 #  ifdef LIBDWARF_BUILD
00056 #   define DW_API __declspec(dllexport)
00057 #  else /* !LIBDWARF_BUILD */
00058 #   define DW_API __declspec(dllimport)
```

```

00059 # endif /* LIBDWARF_BUILD */
00060 # elif (defined(__SUNPRO_C) || defined(__SUNPRO_CC))
00061 # if defined(PIC) || defined(__PIC__)
00062 # define DW_API __global
00063 # endif /* __PIC__ */
00064 # elif (defined(__GNUC__) && __GNUC__ >= 4) || \
00065 defined(__INTEL_COMPILER)
00066 # if defined(PIC) || defined(__PIC__)
00067 # define DW_API __attribute__((visibility("default")))
00068 # endif /* PIC */
00069 # endif /* WIN32 SUNPRO GNUC */
00070 #endif /* !LIBDWARF_STATIC */
00071
00072 #ifndef DW_API
00073 #define DW_API
00074 #endif /* DW_API */
00075
00076 #ifdef __cplusplus
00077 extern "C" {
00078 #endif /* __cplusplus */
00079
00080 /*
00081 libdwarf.h
00082 Revision: #9 Date: 2008/01/17
00083
00084 For libdwarf consumers (reading DWARF2 and later)
00085
00086 The interface is defined as having 8-byte signed and unsigned
00087 values so it can handle 64-or-32bit target on 64-or-32bit host.
00088 Dwarf_Ptr is the native size: it represents pointers on
00089 the host machine (not the target!).
00090
00091 This contains declarations for types and all producer
00092 and consumer functions.
00093
00094 Function declarations are written on a single line each here
00095 so one can use grep to each declaration in its entirety.
00096 The declarations are a little harder to read this way, but...
00097 */
00101 /* Semantic Version identity for this libdwarf.h */
00102 #define DW_LIBDWARF_VERSION "2.3.0"
00103 #define DW_LIBDWARF_VERSION_MAJOR 2
00104 #define DW_LIBDWARF_VERSION_MINOR 3
00105 #define DW_LIBDWARF_VERSION_MICRO 0
00106
00107 #define DW_PATHSOURCE_unspecified 0
00108 #define DW_PATHSOURCE_basic 1
00109 #define DW_PATHSOURCE_dsym 2 /* MacOS dSYM */
00110 #define DW_PATHSOURCE_debuglink 3 /* GNU debuglink */
00111
00112 #ifndef DW_FTYPE_UNKNOWN
00113 #define DW_FTYPE_UNKNOWN 0
00114 #define DW_FTYPE_ELF 1 /* Unix/Linux/etc */
00115 #define DW_FTYPE_MACH_O 2 /* MacOS. */
00116 #define DW_FTYPE_PE 3 /* Windows */
00117 #define DW_FTYPE_ARCHIVE 4 /* unix archive */
00118 #define DW_FTYPE_APPLEUNIVERSAL 5
00119 #endif /* DW_FTYPE_UNKNOWN */
00120 /* standard return values for functions */
00121 #define DW_DLV_NO_ENTRY -1
00122 #define DW_DLV_OK 0
00123 #define DW_DLV_ERROR 1
00124 /* These support opening DWARF5 split dwarf objects and
00125 Elf SHT_GROUP blocks of DWARF sections. */
00126 #define DW_GROUPNUMBER_ANY 0
00127 #define DW_GROUPNUMBER_BASE 1
00128 #define DW_GROUPNUMBER_DWO 2
00129
00130 /* FRAME special values */
00131 /* The following 3 are assigned numbers, but
00132 are only present at run time.
00133 Must not conflict with DW_FRAME values in dwarf.h */
00134 /* Taken as meaning 'undefined value', this is not
00135 a column or register number. */
00136 #ifndef DW_FRAME_UNDEFINED_VAL
00137 #define DW_FRAME_UNDEFINED_VAL 12288
00138 #endif
00139 /* Taken as meaning 'same value' as caller had,
00140 not a column or register number */
00141 #ifndef DW_FRAME_SAME_VAL
00142 #define DW_FRAME_SAME_VAL 12289
00143 #endif
00144 /* DW_FRAME_CFA_COL is assigned a virtual table position
00145 but is accessed via CFA specific calls. */
00146 #ifndef DW_FRAME_CFA_COL
00147 #define DW_FRAME_CFA_COL 12290
00148 #endif

```



```

00149 #define DW_FRAME_CFA_COL3 DW_FRAME_CFA_COL /*compatibility name*/
00150 /* END FRAME special values */
00151
00152 /* dwarf_pcline function, slide arguments
00153 */
00154 #define DW_DLS_BACKWARD -1 /* slide backward to find line */
00155 #define DW_DLS_NOSLIDE 0 /* match exactly without sliding */
00156 #define DW_DLS_FORWARD 1 /* slide forward to find line */
00157
00158 /* Defined larger than necessary.
00159 struct Dwarf_Debug_Fission_Per_CU_s,
00160 being visible, will be difficult to change:
00161 binary compatibility. The count is for arrays
00162 inside the struct, the struct itself is
00163 a single struct. */
00164 #define DW_FFISSION_SECT_COUNT 12
00165
00166 typedef unsigned long long Dwarf_Unsigned;
00167 typedef signed long long Dwarf_Signed;
00168 typedef unsigned long long Dwarf_Off;
00169 typedef unsigned long long Dwarf_Addr;
00170 /* Dwarf_Bool as int is wasteful, but for compatibility
00171 it must stay as int, not unsigned char. */
00172 typedef int Dwarf_Bool; /* boolean type */
00173 typedef unsigned short Dwarf_Half; /* 2 byte unsigned value */
00174 typedef unsigned char Dwarf_Small; /* 1 byte unsigned value */
00175 /* If sizeof(Dwarf_Half) is greater than 2
00176 we believe libdwarf still works properly. */
00177
00178 typedef void* Dwarf_Ptr; /* host machine pointer */
00179 enum Dwarf_Ranges_Entry_Type { DW_RANGES_ENTRY,
00180 DW_RANGES_ADDRESS_SELECTION,
00181 DW_RANGES_END
00182 };
00183
00184 enum Dwarf_Form_Class {
00185 DW_FORM_CLASS_UNKNOWN = 0,
00186 DW_FORM_CLASS_ADDRESS = 1,
00187 DW_FORM_CLASS_BLOCK = 2,
00188 DW_FORM_CLASS_CONSTANT = 3,
00189 DW_FORM_CLASS_EXPRLOC = 4,
00190 DW_FORM_CLASS_FLAG = 5,
00191 DW_FORM_CLASS_LINEPTR = 6,
00192 DW_FORM_CLASS_LOCLISTPTR = 7, /* DWARF2,3,4 only */
00193 DW_FORM_CLASS_MACPTR = 8, /* DWARF2,3,4 only */
00194 DW_FORM_CLASS_RANGELISTPTR = 9, /* DWARF2,3,4 only */
00195 DW_FORM_CLASS_REFERENCE = 10,
00196 DW_FORM_CLASS_STRING = 11,
00197 DW_FORM_CLASS_FRAMEPTR = 12, /* MIPS/IRIX DWARF2 only */
00198 DW_FORM_CLASS_MACROPTR = 13, /* DWARF5 */
00199 DW_FORM_CLASS_ADDRPTR = 14, /* DWARF5 */
00200 DW_FORM_CLASS_LOCLIST = 15, /* DWARF5 */
00201 DW_FORM_CLASS_LOCLISTSPTR = 16, /* DWARF5 */
00202 DW_FORM_CLASS_RNGLIST = 17, /* DWARF5 */
00203 DW_FORM_CLASS_RNGLISTSPTR = 18, /* DWARF5 */
00204 DW_FORM_CLASS_STROFFSETSPTR = 19 /* DWARF5 */
00205 };
00206
00207 typedef struct Dwarf_Form_Data16_s {
00208 unsigned char fd_data[16];
00209 } Dwarf_Form_Data16;
00210
00211 typedef struct Dwarf_Sig8_s {
00212 char signature[8];
00213 } Dwarf_Sig8;
00214
00215 typedef struct Dwarf_Block_s {
00216 Dwarf_Unsigned bl_len;
00217 Dwarf_Ptr bl_data;
00218 Dwarf_Small bl_from_loclist;
00219 Dwarf_Unsigned bl_section_offset;
00220 } Dwarf_Block;
00221
00222 typedef struct Dwarf_Locdesc_c_s * Dwarf_Locdesc_c;
00223 typedef struct Dwarf_Loc_Head_c_s * Dwarf_Loc_Head_c;
00224
00225 typedef struct Dwarf_Gnu_Index_Head_s * Dwarf_Gnu_Index_Head;
00226
00227 typedef struct Dwarf_Dsc_Head_s * Dwarf_Dsc_Head;
00228
00229 typedef struct Dwarf_Frame_Instr_Head_s * Dwarf_Frame_Instr_Head;
00230
00231 typedef void (* dwarf_printf_callback_function_type)
00232 (void * dw_user_pointer, const char * dw_linecontent);
00233
00234 struct Dwarf_Printf_Callback_Info_s {
00235 void * dp_user_pointer;
00236 dwarf_printf_callback_function_type dp_fptr;

```

```

00416     char *                dp_buffer;
00417     unsigned int           dp_buffer_len;
00418     int                   dp_buffer_user_provided;
00419     void *                dp_reserved;
00420 };
00421
00441 struct Dwarf_Cmdline_Options_s {
00442     Dwarf_Bool check_verbose_mode;
00443 };
00447 typedef struct Dwarf_Cmdline_Options_s Dwarf_Cmdline_Options;
00448
00455 typedef struct Dwarf_Str_Offsets_Table_s * Dwarf_Str_Offsets_Table;
00456
00469 typedef struct Dwarf_Ranges_s {
00470     Dwarf_Addr dwr_addr1;
00471     Dwarf_Addr dwr_addr2;
00472     enum Dwarf_Ranges_Entry_Type dwr_type;
00473 } Dwarf_Ranges;
00474
00561 typedef struct Dwarf_Regtable_Entry3_s {
00562     Dwarf_Small dw_offset_relevant;
00563     Dwarf_Small dw_value_type;
00564     Dwarf_Half dw_regnum;
00565     Dwarf_Unsigned dw_offset; /* Should be Dwarf_Signed */
00566     Dwarf_Unsigned dw_args_size; /* Always zero. */
00567     Dwarf_Block dw_block;
00568 } Dwarf_Regtable_Entry3;
00569
00589 typedef struct Dwarf_Regtable3_s {
00590     struct Dwarf_Regtable_Entry3_s rt3_cfa_rule;
00591     /* Required Condition:
00592      rt3_rules points to array rt3_reg_table_size
00593      of struct Dwarf_Regtable_Entry3_s and the
00594      array entries should be all zero bits
00595      on calling dwarf_get_fde_info_for_all_regs3_b().
00596      */
00597     Dwarf_Half rt3_reg_table_size;
00598     struct Dwarf_Regtable_Entry3_s * rt3_rules;
00599 } Dwarf_Regtable3;
00600
00601 /* Opaque types for Consumer Library. */
00611 typedef struct Dwarf_Error_s * Dwarf_Error;
00612
00617 typedef struct Dwarf_Debug_s * Dwarf_Debug;
00622 typedef struct Dwarf_Section_s * Dwarf_Section;
00623
00627 typedef struct Dwarf_Die_s * Dwarf_Die;
00628
00632 typedef struct Dwarf_Debug_Addr_Table_s * Dwarf_Debug_Addr_Table;
00633
00638 typedef struct Dwarf_Line_s * Dwarf_Line;
00639
00644 typedef struct Dwarf_Global_s * Dwarf_Global;
00645
00653 typedef struct Dwarf_Type_s * Dwarf_Type;
00654
00660 typedef struct Dwarf_Func_s * Dwarf_Func;
00666 typedef struct Dwarf_Var_s * Dwarf_Var;
00672 typedef struct Dwarf_Weak_s * Dwarf_Weak;
00673
00677 typedef struct Dwarf_Attribute_s * Dwarf_Attribute;
00678
00684 typedef struct Dwarf_Abbrev_s * Dwarf_Abbrev;
00685
00690 typedef struct Dwarf_Fde_s * Dwarf_Fde;
00695 typedef struct Dwarf_Cie_s * Dwarf_Cie;
00696
00701 typedef struct Dwarf_Arange_s * Dwarf_Arange;
00706 typedef struct Dwarf_Gdbindex_s * Dwarf_Gdbindex;
00712 typedef struct Dwarf_Xu_Index_Header_s * Dwarf_Xu_Index_Header;
00716 typedef struct Dwarf_Line_Context_s * Dwarf_Line_Context;
00717
00721 typedef struct Dwarf_Macro_Context_s * Dwarf_Macro_Context;
00722
00728 typedef struct Dwarf_Dnames_Head_s * Dwarf_Dnames_Head;
00729
00737 typedef void (*Dwarf_Handler)(Dwarf_Error dw_error,
00738     Dwarf_Ptr dw_errarg);
00739
00747 struct Dwarf_Macro_Details_s {
00748     Dwarf_Off dmd_offset; /* offset, in the section,
00749      of this macro info */
00750     Dwarf_Small dmd_type; /* the type, DW_MACINFO_define etc*/
00751     Dwarf_Signed dmd_lineno; /* the source line number where
00752      applicable and vend_def number if
00753      vendor_extension op */
00754     Dwarf_Signed dmd_fileindex; /* the source file index */

```

```

00755     char *          dmd_macro; /* macro name string */
00756 };
00761 typedef struct Dwarf_Macro_Details_s Dwarf_Macro_Details;
00762
00767 typedef struct Dwarf_Debug_Fission_Per_CU_s
00768     Dwarf_Debug_Fission_Per_CU;
00769
00770 /* ===== BEGIN Obj_Access data ===== */
00776 typedef struct Dwarf_Obj_Access_Interface_a_s
00777     Dwarf_Obj_Access_Interface_a;
00778
00784 typedef struct Dwarf_Obj_Access_Methods_a_s
00785     Dwarf_Obj_Access_Methods_a;
00786
00795 typedef struct Dwarf_Obj_Access_Section_a_s
00796     Dwarf_Obj_Access_Section_a;
00797 struct Dwarf_Obj_Access_Section_a_s {
00798     const char*      as_name;
00799     Dwarf_Unsigned   as_type;
00800     Dwarf_Unsigned   as_flags;
00801     Dwarf_Addr       as_addr;
00802     Dwarf_Unsigned   as_offset;
00803     Dwarf_Unsigned   as_size;
00804     Dwarf_Unsigned   as_link;
00805     Dwarf_Unsigned   as_info;
00806     Dwarf_Unsigned   as_addralign;
00807     Dwarf_Unsigned   as_entsize;
00808 };
00809
00822 enum Dwarf_Sec_Alloc_Pref {
00823     /* No dynamic allocation */
00824     Dwarf_Alloc_None=0,
00825     /* alternative allocations */
00826     Dwarf_Alloc_Malloc=1,
00827     Dwarf_Alloc_Mmap=2};
00828
00850 struct Dwarf_Obj_Access_Methods_a_s {
00851     int      (*om_get_section_info)(void* obj,
00852         Dwarf_Unsigned      section_index,
00853         Dwarf_Obj_Access_Section_a* return_section,
00854         int                  * error);
00855     Dwarf_Small      (*om_get_byte_order)(void* obj);
00856     Dwarf_Small      (*om_get_length_size)(void* obj);
00857     Dwarf_Small      (*om_get_pointer_size)(void* obj);
00858     Dwarf_Unsigned   (*om_get_filesizes)(void* obj);
00859     Dwarf_Unsigned   (*om_get_section_count)(void* obj);
00860     /* Always uses malloc/read */
00861     int      (*om_load_section)(void* obj,
00862         Dwarf_Unsigned dw_section_index,
00863         Dwarf_Small **dw_return_data,
00864         int *dw_error);
00865     int      (*om_relocate_a_section)(void* obj,
00866         Dwarf_Unsigned section_index,
00867         Dwarf_Debug dbg,
00868         int * error);
00869     /* Added in 0.12.0 to allow mmap in section loading.
00870     If you are just using malloc for section loading
00871     and referring to this struct in your code
00872     you should leave this function pointer NULL (zero). */
00873     int      (*om_load_section_a)(void* obj,
00874         Dwarf_Unsigned      dw_section_index,
00875         /* dw_alloc_pref is input preference and also
00876         output with the actual allocated type */
00877         enum Dwarf_Sec_Alloc_Pref *dw_alloc_pref,
00878         Dwarf_Small      **dw_return_data_ptr,
00879         Dwarf_Unsigned   *dw_return_data_len,
00880         Dwarf_Small      **dw_return_mmap_base_ptr,
00881         Dwarf_Unsigned   *dw_return_mmap_offset,
00882         Dwarf_Unsigned   *dw_return_mmap_len,
00883         int *dw_error);
00884     void      (*om_finish)(void * obj);
00885 };
00886 struct Dwarf_Obj_Access_Interface_a_s {
00887     void*      ai_object;
00888     const Dwarf_Obj_Access_Methods_a *ai_methods;
00889 };
00890 /* ===== END Obj_Access data ===== */
00891
00892 /* User code must allocate this struct, zero it,
00893 and pass a pointer to it
00894 into dwarf_get_debugfission_for_cu . */
00895 struct Dwarf_Debug_Fission_Per_CU_s {
00896     /* Do not free the string. It contains "cu" or "tu". */
00897     /* If this is not set (ie, not a CU/TU in DWP Package File)
00898     then pcu_type will be NULL. */
00899     const char * pcu_type;
00900     /* pcu_index is the index (range 1 to N )

```

```

00901         into the tu/cu table of offsets and the table
00902         of sizes. 1 to N as the zero index is reserved
00903         for special purposes. Not a value one
00904         actually needs. */
00905         Dwarf_Unsigned pcu_index;
00906         Dwarf_Sig8      pcu_hash; /* 8 byte */
00907         /* [0] has offset and size 0.
00908            [1]-[8] are DW_SECT_* indexes and the
00909            values are the offset and size
00910            of the respective section contribution
00911            of a single .dwo object. When pcu_size[n] is
00912            zero the corresponding section is not present. */
00913         Dwarf_Unsigned pcu_offset[DW_FISSION_SECT_COUNT];
00914         Dwarf_Unsigned pcu_size[DW_FISSION_SECT_COUNT];
00915         Dwarf_Unsigned unused1;
00916         Dwarf_Unsigned unused2;
00917     };
00918
00923     typedef struct Dwarf_Rnglists_Head_s * Dwarf_Rnglists_Head;
00924
00930     /* Special values for offset_into_exception_table field
00931        of dwarf fde's
00932        The following value indicates that there is no
00933        Exception table offset
00934        associated with a dwarf frame.
00935     */
00936     #define DW_DLX_NO_EH_OFFSET          (-1LL)
00937     /* The following value indicates that the producer
00938        was unable to analyze the
00939        source file to generate Exception tables for this function.
00940     */
00941     #define DW_DLX_EH_OFFSET_UNAVAILABLE (-2LL)
00942
00943     /* The augmenter string for CIE */
00944     #define DW_CIE_AUGMENTER_STRING_V0  "z"
00945
00946     /* ***IMPORTANT NOTE, TARGET DEPENDENCY ***
00947        DW_REG_TABLE_SIZE must be at least as large as
00948        the number of registers
00949        DW_FRAME_LAST_REG_NUM as defined in dwarf.h
00950     */
00951     #ifndef DW_REG_TABLE_SIZE
00952     #define DW_REG_TABLE_SIZE  DW_FRAME_LAST_REG_NUM
00953     #endif
00954
00955     /* For MIPS, DW_FRAME_SAME_VAL is the correct default value
00956        for a frame register value. For other CPUS another value
00957        may be better, such as DW_FRAME_UNDEFINED_VAL.
00958        See dwarf_set_frame_rule_table_size
00959     */
00960     #ifndef DW_FRAME_REG_INITIAL_VALUE
00961     #define DW_FRAME_REG_INITIAL_VALUE DW_FRAME_SAME_VAL
00962     #endif
00963
00964     /* The following are all needed to evaluate DWARF3 register rules.
00965        These have nothing to do simply printing
00966        frame instructions.
00967     */
00968     #define DW_EXPR_OFFSET          0 /* offset is from CFA reg */
00969     #define DW_EXPR_VAL_OFFSET      1
00970     #define DW_EXPR_EXPRESSION      2
00971     #define DW_EXPR_VAL_EXPRESSION  3
00982     #define DW_DLA_STRING           0x01 /* char* */
00983     #define DW_DLA_LOC              0x02 /* Dwarf_Loc */
00984     #define DW_DLA_LOCDDESC         0x03 /* Dwarf_Locdesc */
00985     #define DW_DLA_ELLIST           0x04 /* Dwarf_Ellist (not used) */
00986     #define DW_DLA_BOUNDS           0x05 /* Dwarf_Bounds (not used) */
00987     #define DW_DLA_BLOCK            0x06 /* Dwarf_Block */
00988     #define DW_DLA_DEBUG            0x07 /* Dwarf_Debug */
00989     #define DW_DLA_DIE              0x08 /* Dwarf_Die */
00990     #define DW_DLA_LINE             0x09 /* Dwarf_Line */
00991     #define DW_DLA_ATTR             0x0a /* Dwarf_Attribute */
00992     #define DW_DLA_TYPE             0x0b /* Dwarf_Type (not used) */
00993     #define DW_DLA_SUBSCR           0x0c /* Dwarf_Subscr (not used) */
00994     #define DW_DLA_GLOBAL           0x0d /* Dwarf_Global */
00995     #define DW_DLA_ERROR            0x0e /* Dwarf_Error */
00996     #define DW_DLA_LIST            0x0f /* a list */
00997     #define DW_DLA_LINEBUF          0x10 /* Dwarf_Line* (not used) */
00998     #define DW_DLA_ARANGE           0x11 /* Dwarf_Arange */
00999     #define DW_DLA_ABBREV           0x12 /* Dwarf_Abbrev */
01000     #define DW_DLA_FRAME_INSTR_HEAD 0x13 /* Dwarf_Frame_Instr_Head */
01001     #define DW_DLA_CIE             0x14 /* Dwarf_Cie */
01002     #define DW_DLA_FDE             0x15 /* Dwarf_Fde */
01003     #define DW_DLA_LOC_BLOCK       0x16 /* Dwarf_Loc */
01004
01005     #define DW_DLA_FRAME_OP         0x17 /* Dwarf_Frame_Op (not used) */
01006     #define DW_DLA_FUNC            0x18 /* Dwarf_Func */

```

```
01007 #define DW_DLA_UARRAY          0x19 /* Array of Dwarf_Off:Jan2023 */
01008 #define DW_DLA_VAR                0x1a /* Dwarf_Var */
01009 #define DW_DLA_WEAK               0x1b /* Dwarf_Weak */
01010 #define DW_DLA_ADDR               0x1c /* Dwarf_Addr sized entries */
01011 #define DW_DLA_RANGES             0x1d /* Dwarf_Ranges */
01012 /* 0x1e (30) to 0x34 (52) reserved for internal to libdwarf types. */
01013 /* .debug_gnu_typenames/pubnames, 2020 */
01014 #define DW_DLA_GNU_INDEX_HEAD    0x35
01015
01016 #define DW_DLA_RNGLISTS_HEAD      0x36 /* .debug_rnglists DW5 */
01017 #define DW_DLA_GDBINDEX           0x37 /* Dwarf_Gdbindex */
01018 #define DW_DLA_XU_INDEX           0x38 /* Dwarf_Xu_Index_Header */
01019 #define DW_DLA_LOC_BLOCK_C        0x39 /* Dwarf_Loc_c */
01020 #define DW_DLA_LOCDDESC_C         0x3a /* Dwarf_Locdesc_c */
01021 #define DW_DLA_LOC_HEAD_C         0x3b /* Dwarf_Loc_Head_c */
01022 #define DW_DLA_MACRO_CONTEXT      0x3c /* Dwarf_Macro_Context */
01023 /* 0x3d (61) is for libdwarf internal use. */
01024 #define DW_DLA_DSC_HEAD           0x3e /* Dwarf_Dsc_Head */
01025 #define DW_DLA_DNAMES_HEAD        0x3f /* Dwarf_Dnames_Head */
01026
01027 /* struct Dwarf_Str_Offsets_Table_s */
01028 #define DW_DLA_STR_OFFSETS        0x40
01029 /* struct Dwarf_Debug_Addr_Table_s */
01030 #define DW_DLA_DEBUG_ADDR         0x41
01042 /* libdwarf error numbers */
01043 #define DW_DLE_NE                 0 /* no error */
01044 #define DW_DLE_VMM                1 /* dwarf format/library version mismatch */
01045 #define DW_DLE_MAP                2 /* memory map failure */
01046 #define DW_DLE_LEE                3 /* libelf error */
01047 #define DW_DLE_NDS                4 /* no debug section */
01048 #define DW_DLE_NLS                5 /* no line section */
01049 #define DW_DLE_ID                 6 /* invalid descriptor for query */
01050 #define DW_DLE_IOF                7 /* I/O failure */
01051 #define DW_DLE_MAF                8 /* memory allocation failure */
01052 #define DW_DLE_IA                 9 /* invalid argument */
01053 #define DW_DLE_MDE               10 /* mangled debugging entry */
01054 #define DW_DLE_MLE               11 /* mangled line number entry */
01055 #define DW_DLE_FNO               12 /* file not open */
01056 #define DW_DLE_FNR               13 /* file not a regular file */
01057 #define DW_DLE_FWA               14 /* file open with wrong access */
01058 #define DW_DLE_NOB               15 /* not an object file */
01059 #define DW_DLE_MOF               16 /* mangled object file header */
01060 #define DW_DLE_EOLL              17 /* end of location list entries */
01061 #define DW_DLE_NOLL              18 /* no location list section */
01062 #define DW_DLE_BADOFF            19 /* Invalid offset */
01063 #define DW_DLE_EOS               20 /* end of section */
01064 #define DW_DLE_ATRUNC            21 /* abbreviations section appears truncated */
01065 #define DW_DLE_BADBITC           22 /* Address size passed to dwarf bad */
01066 /* It is not an allowed size (64 or 32) */
01067 /* Error codes defined by the current Libdwarf Implementation. */
01068 #define DW_DLE_DBG_ALLOC          23
01069 #define DW_DLE_FSTAT_ERROR        24
01070 #define DW_DLE_FSTAT_MODE_ERROR   25
01071 #define DW_DLE_INIT_ACCESS_WRONG  26
01072 #define DW_DLE_ELF_BEGIN_ERROR    27
01073 #define DW_DLE_ELF_GETEHDR_ERROR  28
01074 #define DW_DLE_ELF_GETSHDR_ERROR  29
01075 #define DW_DLE_ELF_STRPTR_ERROR   30
01076 #define DW_DLE_DEBUG_INFO_DUPLICATE 31
01077 #define DW_DLE_DEBUG_INFO_NULL    32
01078 #define DW_DLE_DEBUG_ABBREV_DUPLICATE 33
01079 #define DW_DLE_DEBUG_ABBREV_NULL   34
01080 #define DW_DLE_DEBUG_ARANGES_DUPLICATE 35
01081 #define DW_DLE_DEBUG_ARANGES_NULL  36
01082 #define DW_DLE_DEBUG_LINE_DUPLICATE 37
01083 #define DW_DLE_DEBUG_LINE_NULL     38
01084 #define DW_DLE_DEBUG_LOC_DUPLICATE 39
01085 #define DW_DLE_DEBUG_LOC_NULL      40
01086 #define DW_DLE_DEBUG_MACINFO_DUPLICATE 41
01087 #define DW_DLE_DEBUG_MACINFO_NULL  42
01088 #define DW_DLE_DEBUG_PUBNAMES_DUPLICATE 43
01089 #define DW_DLE_DEBUG_PUBNAMES_NULL  44
01090 #define DW_DLE_DEBUG_STR_DUPLICATE  45
01091 #define DW_DLE_DEBUG_STR_NULL       46
01092 #define DW_DLE_CU_LENGTH_ERROR      47
01093 #define DW_DLE_VERSION_STAMP_ERROR  48
01094 #define DW_DLE_ABBREV_OFFSET_ERROR  49
01095 #define DW_DLE_ADDRESS_SIZE_ERROR    50
01096 #define DW_DLE_DEBUG_INFO_PTR_NULL  51
01097 #define DW_DLE_DIE_NULL             52
01098 #define DW_DLE_STRING_OFFSET_BAD     53
01099 #define DW_DLE_DEBUG_LINE_LENGTH_BAD 54
01100 #define DW_DLE_LINE_PROLOG_LENGTH_BAD 55
01101 #define DW_DLE_LINE_NUM_OPERANDS_BAD 56
01102 #define DW_DLE_LINE_SET_ADDR_ERROR   57
01103 #define DW_DLE_LINE_EXT_OPCODE_BAD   58
01104 #define DW_DLE_DWARF_LINE_NULL       59
```

```
01105 #define DW_DLE_INCL_DIR_NUM_BAD 60
01106 #define DW_DLE_LINE_FILE_NUM_BAD 61
01107 #define DW_DLE_ALLOC_FAIL 62
01108 #define DW_DLE_NO_CALLBACK_FUNC 63
01109 #define DW_DLE_SECT_ALLOC 64
01110 #define DW_DLE_FILE_ENTRY_ALLOC 65
01111 #define DW_DLE_LINE_ALLOC 66
01112 #define DW_DLE_FPGM_ALLOC 67
01113 #define DW_DLE_INCDIR_ALLOC 68
01114 #define DW_DLE_STRING_ALLOC 69
01115 #define DW_DLE_CHUNK_ALLOC 70
01116 #define DW_DLE_BYTEOFF_ERR 71
01117 #define DW_DLE_CIE_ALLOC 72
01118 #define DW_DLE_FDE_ALLOC 73
01119 #define DW_DLE_REGNO_OVFL 74
01120 #define DW_DLE_CIE_OFFS_ALLOC 75
01121 #define DW_DLE_WRONG_ADDRESS 76
01122 #define DW_DLE_EXTRA_NEIGHBORS 77
01123 #define DW_DLE_WRONG_TAG 78
01124 #define DW_DLE_DIE_ALLOC 79
01125 #define DW_DLE_PARENT_EXISTS 80
01126 #define DW_DLE_DBG_NULL 81
01127 #define DW_DLE_DEBUGLINE_ERROR 82
01128 #define DW_DLE_DEBUGFRAME_ERROR 83
01129 #define DW_DLE_DEBUGINFO_ERROR 84
01130 #define DW_DLE_ATTR_ALLOC 85
01131 #define DW_DLE_ABBREV_ALLOC 86
01132 #define DW_DLE_OFFSET_UFLW 87
01133 #define DW_DLE_ELF_SECT_ERR 88
01134 #define DW_DLE_DEBUG_FRAME_LENGTH_BAD 89
01135 #define DW_DLE_FRAME_VERSION_BAD 90
01136 #define DW_DLE_CIE_RET_ADDR_REG_ERROR 91
01137 #define DW_DLE_FDE_NULL 92
01138 #define DW_DLE_FDE_DBG_NULL 93
01139 #define DW_DLE_CIE_NULL 94
01140 #define DW_DLE_CIE_DBG_NULL 95
01141 #define DW_DLE_FRAME_TABLE_COL_BAD 96
01142 #define DW_DLE_PC_NOT_IN_FDE_RANGE 97
01143 #define DW_DLE_CIE_INSTR_EXEC_ERROR 98
01144 #define DW_DLE_FRAME_INSTR_EXEC_ERROR 99
01145 #define DW_DLE_FDE_PTR_NULL 100
01146 #define DW_DLE_RET_OP_LIST_NULL 101
01147 #define DW_DLE_LINE_CONTEXT_NULL 102
01148 #define DW_DLE_DBG_NO_CU_CONTEXT 103
01149 #define DW_DLE_DIE_NO_CU_CONTEXT 104
01150 #define DW_DLE_FIRST_DIE_NOT_CU 105
01151 #define DW_DLE_NEXT_DIE_PTR_NULL 106
01152 #define DW_DLE_DEBUG_FRAME_DUPLICATE 107
01153 #define DW_DLE_DEBUG_FRAME_NULL 108
01154 #define DW_DLE_ABBREV_DECODE_ERROR 109
01155 #define DW_DLE_DWARF_ABBREV_NULL 110
01156 #define DW_DLE_ATTR_NULL 111
01157 #define DW_DLE_DIE_BAD 112
01158 #define DW_DLE_DIE_ABBREV_BAD 113
01159 #define DW_DLE_ATTR_FORM_BAD 114
01160 #define DW_DLE_ATTR_NO_CU_CONTEXT 115
01161 #define DW_DLE_ATTR_FORM_SIZE_BAD 116
01162 #define DW_DLE_ATTR_DBG_NULL 117
01163 #define DW_DLE_BAD_REF_FORM 118
01164 #define DW_DLE_ATTR_FORM_OFFSET_BAD 119
01165 #define DW_DLE_LINE_OFFSET_BAD 120
01166 #define DW_DLE_DEBUG_STR_OFFSET_BAD 121
01167 #define DW_DLE_STRING_PTR_NULL 122
01168 #define DW_DLE_PUBNAMES_VERSION_ERROR 123
01169 #define DW_DLE_PUBNAMES_LENGTH_BAD 124
01170 #define DW_DLE_GLOBAL_NULL 125
01171 #define DW_DLE_GLOBAL_CONTEXT_NULL 126
01172 #define DW_DLE_DIR_INDEX_BAD 127
01173 #define DW_DLE_LOC_EXPR_BAD 128
01174 #define DW_DLE_DIE_LOC_EXPR_BAD 129
01175 #define DW_DLE_ADDR_ALLOC 130
01176 #define DW_DLE_OFFSET_BAD 131
01177 #define DW_DLE_MAKE_CU_CONTEXT_FAIL 132
01178 #define DW_DLE_REL_ALLOC 133
01179 #define DW_DLE_ARANGE_OFFSET_BAD 134
01180 #define DW_DLE_SEGMENT_SIZE_BAD 135
01181 #define DW_DLE_ARANGE_LENGTH_BAD 136
01182 #define DW_DLE_ARANGE_DECODE_ERROR 137
01183 #define DW_DLE_ARANGES_NULL 138
01184 #define DW_DLE_ARANGE_NULL 139
01185 #define DW_DLE_NO_FILE_NAME 140
01186 #define DW_DLE_NO_COMP_DIR 141
01187 #define DW_DLE_CU_ADDRESS_SIZE_BAD 142
01188 #define DW_DLE_INPUT_ATTR_BAD 143
01189 #define DW_DLE_EXPR_NULL 144
01190 #define DW_DLE_BAD_EXPR_OPCODE 145
01191 #define DW_DLE_EXPR_LENGTH_BAD 146
```

```

01192 #define DW_DLE_MULTIPLE_RELOC_IN_EXPR 147
01193 #define DW_DLE_ELF_GETIDENT_ERROR 148
01194 #define DW_DLE_NO_AT_MIPS_FDE 149
01195 #define DW_DLE_NO_CIE_FOR_FDE 150
01196 #define DW_DLE_DIE_ABBREV_LIST_NULL 151
01197 #define DW_DLE_DEBUG_FUNCNAMES_DUPLICATE 152
01198 #define DW_DLE_DEBUG_FUNCNAMES_NULL 153
01199 #define DW_DLE_DEBUG_FUNCNAMES_VERSION_ERROR 154
01200 #define DW_DLE_DEBUG_FUNCNAMES_LENGTH_BAD 155
01201 #define DW_DLE_FUNC_NULL 156
01202 #define DW_DLE_FUNC_CONTEXT_NULL 157
01203 #define DW_DLE_DEBUG_TYPENAMES_DUPLICATE 158
01204 #define DW_DLE_DEBUG_TYPENAMES_NULL 159
01205 #define DW_DLE_DEBUG_TYPENAMES_VERSION_ERROR 160
01206 #define DW_DLE_DEBUG_TYPENAMES_LENGTH_BAD 161
01207 #define DW_DLE_TYPE_NULL 162
01208 #define DW_DLE_TYPE_CONTEXT_NULL 163
01209 #define DW_DLE_DEBUG_VARNAMES_DUPLICATE 164
01210 #define DW_DLE_DEBUG_VARNAMES_NULL 165
01211 #define DW_DLE_DEBUG_VARNAMES_VERSION_ERROR 166
01212 #define DW_DLE_DEBUG_VARNAMES_LENGTH_BAD 167
01213 #define DW_DLE_VAR_NULL 168
01214 #define DW_DLE_VAR_CONTEXT_NULL 169
01215 #define DW_DLE_DEBUG_WEAKNAMES_DUPLICATE 170
01216 #define DW_DLE_DEBUG_WEAKNAMES_NULL 171
01217 #define DW_DLE_DEBUG_WEAKNAMES_VERSION_ERROR 172
01218 #define DW_DLE_DEBUG_WEAKNAMES_LENGTH_BAD 173
01219 #define DW_DLE_WEAK_NULL 174
01220 #define DW_DLE_WEAK_CONTEXT_NULL 175
01221 #define DW_DLE_LODESC_COUNT_WRONG 176
01222 #define DW_DLE_MACINFO_STRING_NULL 177
01223 #define DW_DLE_MACINFO_STRING_EMPTY 178
01224 #define DW_DLE_MACINFO_INTERNAL_ERROR_SPACE 179
01225 #define DW_DLE_MACINFO_MALLOC_FAIL 180
01226 #define DW_DLE_DEBUGMACINFO_ERROR 181
01227 #define DW_DLE_DEBUG_MACRO_LENGTH_BAD 182
01228 #define DW_DLE_DEBUG_MACRO_MAX_BAD 183
01229 #define DW_DLE_DEBUG_MACRO_INTERNAL_ERR 184
01230 #define DW_DLE_DEBUG_MACRO_MALLOC_SPACE 185
01231 #define DW_DLE_DEBUG_MACRO_INCONSISTENT 186
01232 #define DW_DLE_DF_NO_CIE_AUGMENTATION 187
01233 #define DW_DLE_DF_REG_NUM_TOO_HIGH 188
01234 #define DW_DLE_DF_MAKE_INSTR_NO_INIT 189
01235 #define DW_DLE_DF_NEW_LOC_LESS_OLD_LOC 190
01236 #define DW_DLE_DF_POP_EMPTY_STACK 191
01237 #define DW_DLE_DF_ALLOC_FAIL 192
01238 #define DW_DLE_DF_FRAME_DECODING_ERROR 193
01239 #define DW_DLE_DEBUG_LOC_SECTION_SHORT 194
01240 #define DW_DLE_FRAME_AUGMENTATION_UNKNOWN 195
01241 #define DW_DLE_PUBTYPE_CONTEXT 196 /* Unused. */
01242 #define DW_DLE_DEBUG_PUBTYPES_LENGTH_BAD 197
01243 #define DW_DLE_DEBUG_PUBTYPES_VERSION_ERROR 198
01244 #define DW_DLE_DEBUG_PUBTYPES_DUPLICATE 199
01245 #define DW_DLE_FRAME_CIE_DECODE_ERROR 200
01246 #define DW_DLE_FRAME_REGISTER_UNREPRESENTABLE 201
01247 #define DW_DLE_FRAME_REGISTER_COUNT_MISMATCH 202
01248 #define DW_DLE_LINK_LOOP 203
01249 #define DW_DLE_STRP_OFFSET_BAD 204
01250 #define DW_DLE_DEBUG_RANGES_DUPLICATE 205
01251 #define DW_DLE_DEBUG_RANGES_OFFSET_BAD 206
01252 #define DW_DLE_DEBUG_RANGES_MISSING_END 207
01253 #define DW_DLE_DEBUG_RANGES_OUT_OF_MEM 208
01254 #define DW_DLE_DEBUG_SYMTAB_ERR 209
01255 #define DW_DLE_DEBUG_STRTAB_ERR 210
01256 #define DW_DLE_RELOC_MISMATCH_INDEX 211
01257 #define DW_DLE_RELOC_MISMATCH_RELOC_INDEX 212
01258 #define DW_DLE_RELOC_MISMATCH_STRTAB_INDEX 213
01259 #define DW_DLE_RELOC_SECTION_MISMATCH 214
01260 #define DW_DLE_RELOC_SECTION_MISSING_INDEX 215
01261 #define DW_DLE_RELOC_SECTION_LENGTH_ODD 216
01262 #define DW_DLE_RELOC_SECTION_PTR_NULL 217
01263 #define DW_DLE_RELOC_SECTION_MALLOC_FAIL 218
01264 #define DW_DLE_NO_ELF64_SUPPORT 219
01265 #define DW_DLE_MISSING_ELF64_SUPPORT 220
01266 #define DW_DLE_ORPHAN_FDE 221
01267 #define DW_DLE_DUPLICATE_INST_BLOCK 222
01268 #define DW_DLE_BAD_REF_SIG8_FORM 223
01269 #define DW_DLE_ATTR_EXPRLOC_FORM_BAD 224
01270 #define DW_DLE_FORM_SEC_OFFSET_LENGTH_BAD 225
01271 #define DW_DLE_NOT_REF_FORM 226
01272 #define DW_DLE_DEBUG_FRAME_LENGTH_NOT_MULTIPLE 227
01273 #define DW_DLE_REF_SIG8_NOT_HANDLED 228
01274 #define DW_DLE_DEBUG_FRAME_POSSIBLE_ADDRESS_BOTCH 229
01275 #define DW_DLE_LOC_BAD_TERMINATION 230
01276 #define DW_DLE_SYMTAB_SECTION_LENGTH_ODD 231
01277 #define DW_DLE_RELOC_SECTION_SYMBOL_INDEX_BAD 232
01278 #define DW_DLE_RELOC_SECTION_RELOC_TARGET_SIZE_UNKNOWN 233

```



```

01279 #define DW_DLE_SYMTAB_SECTION_ENTRYSIZE_ZERO 234
01280 #define DW_DLE_LINE_NUMBER_HEADER_ERROR 235
01281 #define DW_DLE_DEBUG_TYPES_NULL 236
01282 #define DW_DLE_DEBUG_TYPES_DUPLICATE 237
01283 #define DW_DLE_DEBUG_TYPES_ONLY_DWARF4 238
01284 #define DW_DLE_DEBUG_TYPEOFFSET_BAD 239
01285 #define DW_DLE_GNU_OPCODE_ERROR 240
01286 #define DW_DLE_DEBUGPUBTYPES_ERROR 241
01287 #define DW_DLE_AT_FIXUP_NULL 242
01288 #define DW_DLE_AT_FIXUP_DUP 243
01289 #define DW_DLE_BAD_ABINAME 244
01290 #define DW_DLE_TOO_MANY_DEBUG 245
01291 #define DW_DLE_DEBUG_STR_OFFSETS_DUPLICATE 246
01292 #define DW_DLE_SECTION_DUPLICATION 247
01293 #define DW_DLE_SECTION_ERROR 248
01294 #define DW_DLE_DEBUG_ADDR_DUPLICATE 249
01295 #define DW_DLE_DEBUG_CU_UNAVAILABLE_FOR_FORM 250
01296 #define DW_DLE_DEBUG_FORM_HANDLING_INCOMPLETE 251
01297 #define DW_DLE_NEXT_DIE_PAST_END 252
01298 #define DW_DLE_NEXT_DIE_WRONG_FORM 253
01299 #define DW_DLE_NEXT_DIE_NO_ABBREV_LIST 254
01300 #define DW_DLE_NESTED_FORM_INDIRECT_ERROR 255
01301 #define DW_DLE_CU_DIE_NO_ABBREV_LIST 256
01302 #define DW_DLE_MISSING_NEEDED_DEBUG_ADDR_SECTION 257
01303 #define DW_DLE_ATTR_FORM_NOT_ADDR_INDEX 258
01304 #define DW_DLE_ATTR_FORM_NOT_STR_INDEX 259
01305 #define DW_DLE_DUPLICATE_GDB_INDEX 260
01306 #define DW_DLE_ERRONEOUS_GDB_INDEX_SECTION 261
01307 #define DW_DLE_GDB_INDEX_COUNT_ERROR 262
01308 #define DW_DLE_GDB_INDEX_COUNT_ADDR_ERROR 263
01309 #define DW_DLE_GDB_INDEX_INDEX_ERROR 264
01310 #define DW_DLE_GDB_INDEX_CUVEC_ERROR 265
01311 #define DW_DLE_DUPLICATE_CU_INDEX 266
01312 #define DW_DLE_DUPLICATE_TU_INDEX 267
01313 #define DW_DLE_XU_TYPE_ARG_ERROR 268
01314 #define DW_DLE_XU_IMPOSSIBLE_ERROR 269
01315 #define DW_DLE_XU_NAME_COL_ERROR 270
01316 #define DW_DLE_XU_HASH_ROW_ERROR 271
01317 #define DW_DLE_XU_HASH_INDEX_ERROR 272
01318 /* ..._FAILSAFE_ERRVAL is an aid when out of memory. */
01319 #define DW_DLE_FAILSAFE_ERRVAL 273
01320 #define DW_DLE_ARANGE_ERROR 274
01321 #define DW_DLE_PUBNAMES_ERROR 275
01322 #define DW_DLE_FUNCNAMES_ERROR 276
01323 #define DW_DLE_TYPENAMES_ERROR 277
01324 #define DW_DLE_VARNAMES_ERROR 278
01325 #define DW_DLE_WEAKNAMES_ERROR 279
01326 #define DW_DLE_RELOCS_ERROR 280
01327 #define DW_DLE_ATTR_OUTSIDE_SECTION 281
01328 #define DW_DLE_FFISSION_INDEX_WRONG 282
01329 #define DW_DLE_FFISSION_VERSION_ERROR 283
01330 #define DW_DLE_NEXT_DIE_LOW_ERROR 284
01331 #define DW_DLE_CU_UT_TYPE_ERROR 285
01332 #define DW_DLE_NO_SUCH_SIGNATURE_FOUND 286
01333 #define DW_DLE_SIGNATURE_SECTION_NUMBER_WRONG 287
01334 #define DW_DLE_ATTR_FORM_NOT_DATA8 288
01335 #define DW_DLE_SIG_TYPE_WRONG_STRING 289
01336 #define DW_DLE_MISSING_REQUIRED_TU_OFFSET_HASH 290
01337 #define DW_DLE_MISSING_REQUIRED_CU_OFFSET_HASH 291
01338 #define DW_DLE_DWP_MISSING_DWO_ID 292
01339 #define DW_DLE_DWP_SIBLING_ERROR 293
01340 #define DW_DLE_DEBUG_FFISSION_INCOMPLETE 294
01341 #define DW_DLE_FFISSION_SECNUM_ERR 295
01342 #define DW_DLE_DEBUG_MACRO_DUPLICATE 296
01343 #define DW_DLE_DEBUG_NAMES_DUPLICATE 297
01344 #define DW_DLE_DEBUG_LINE_STR_DUPLICATE 298
01345 #define DW_DLE_DEBUG_SUP_DUPLICATE 299
01346 #define DW_DLE_NO_SIGNATURE_TO_LOOKUP 300
01347 #define DW_DLE_NO_TIED_ADDR_AVAILABLE 301
01348 #define DW_DLE_NO_TIED_SIG_AVAILABLE 302
01349 #define DW_DLE_STRING_NOT_TERMINATED 303
01350 #define DW_DLE_BAD_LINE_TABLE_OPERATION 304
01351 #define DW_DLE_LINE_CONTEXT_BOTCH 305
01352 #define DW_DLE_LINE_CONTEXT_INDEX_WRONG 306
01353 #define DW_DLE_NO_TIED_STRING_AVAILABLE 307
01354 #define DW_DLE_NO_TIED_FILE_AVAILABLE 308
01355 #define DW_DLE_CU_TYPE_MISSING 309
01356 #define DW_DLE_LLE_CODE_UNKNOWN 310
01357 #define DW_DLE_LOCLIST_INTERFACE_ERROR 311
01358 #define DW_DLE_LOCLIST_INDEX_ERROR 312
01359 #define DW_DLE_INTERFACE_NOT_SUPPORTED 313
01360 #define DW_DLE_ZDEBUG_REQUIRES_ZLIB 314
01361 #define DW_DLE_ZDEBUG_INPUT_FORMAT_ODD 315
01362 #define DW_DLE_ZLIB_BUF_ERROR 316
01363 #define DW_DLE_ZLIB_DATA_ERROR 317
01364 #define DW_DLE_MACRO_OFFSET_BAD 318
01365 #define DW_DLE_MACRO_OPCODE_BAD 319

```


01366	#define	DW_DLE_MACRO_OPCODE_FORM_BAD	320
01367	#define	DW_DLE_UNKNOWN_FORM	321
01368	#define	DW_DLE_BAD_MACRO_HEADER_POINTER	322
01369	#define	DW_DLE_BAD_MACRO_INDEX	323
01370	#define	DW_DLE_MACRO_OP_UNHANDLED	324
01371	#define	DW_DLE_MACRO_PAST_END	325
01372	#define	DW_DLE_LINE_STRP_OFFSET_BAD	326
01373	#define	DW_DLE_STRING_FORM_IMPROPER	327
01374	#define	DW_DLE_ELF_FLAGS_NOT_AVAILABLE	328
01375	#define	DW_DLE_LEB_IMPROPER	329
01376	#define	DW_DLE_DEBUG_LINE_RANGE_ZERO	330
01377	#define	DW_DLE_READ_LITTLEENDIAN_ERROR	331
01378	#define	DW_DLE_READ_BIGENDIAN_ERROR	332
01379	#define	DW_DLE_RELOC_INVALID	333
01380	#define	DW_DLE_INFO_HEADER_ERROR	334
01381	#define	DW_DLE_ARANGES_HEADER_ERROR	335
01382	#define	DW_DLE_LINE_OFFSET_WRONG_FORM	336
01383	#define	DW_DLE_FORM_BLOCK_LENGTH_ERROR	337
01384	#define	DW_DLE_ZLIB_SECTION_SHORT	338
01385	#define	DW_DLE_CIE_INSTR_PTR_ERROR	339
01386	#define	DW_DLE_FDE_INSTR_PTR_ERROR	340
01387	#define	DW_DLE_FISSION_ADDITION_ERROR	341
01388	#define	DW_DLE_HEADER_LEN_BIGGER_THAN_SECSIZE	342
01389	#define	DW_DLE_LOCEXPRESS_OFF_SECTION_END	343
01390	#define	DW_DLE_POINTER_SECTION_UNKNOWN	344
01391	#define	DW_DLE_ERRONEOUS_XU_INDEX_SECTION	345
01392	#define	DW_DLE_DIRECTORY_FORMAT_COUNT_VS_DIRECTORIES_MISMATCH	346
01393	#define	DW_DLE_COMPRESSED_EMPTY_SECTION	347
01394	#define	DW_DLE_SIZE_WRAPAROUND	348
01395	#define	DW_DLE_ILLOGICAL_TSEARCH	349
01396	#define	DW_DLE_BAD_STRING_FORM	350
01397	#define	DW_DLE_DEBUGSTR_ERROR	351
01398	#define	DW_DLE_DEBUGSTR_UNEXPECTED_REL	352
01399	#define	DW_DLE_DISCR_ARRAY_ERROR	353
01400	#define	DW_DLE_LEB_OUT_ERROR	354
01401	#define	DW_DLE_SIBLING_LIST_IMPROPER	355
01402	#define	DW_DLE_LOCLIST_OFFSET_BAD	356
01403	#define	DW_DLE_LINE_TABLE_BAD	357
01404	#define	DW_DLE_DEBUG_LOCLISTS_DUPLICATE	358
01405	#define	DW_DLE_DEBUG_RNGLISTS_DUPLICATE	359
01406	#define	DW_DLE_ABBREV_OFF_END	360
01407	#define	DW_DLE_FORM_STRING_BAD_STRING	361
01408	#define	DW_DLE_AUGMENTATION_STRING_OFF_END	362
01409	#define	DW_DLE_STRING_OFF_END_PUBNAMES_LIKE	363
01410	#define	DW_DLE_LINE_STRING_BAD	364
01411	#define	DW_DLE_DEFINE_FILE_STRING_BAD	365
01412	#define	DW_DLE_MACRO_STRING_BAD	366
01413	#define	DW_DLE_MACINFO_STRING_BAD	367
01414	#define	DW_DLE_ZLIB_UNCOMPRESS_ERROR	368
01415	#define	DW_DLE_IMPROPER_DWO_ID	369
01416	#define	DW_DLE_GROUPNUMBER_ERROR	370
01417	#define	DW_DLE_ADDRESS_SIZE_ZERO	371
01418	#define	DW_DLE_DEBUG_NAMES_HEADER_ERROR	372
01419	#define	DW_DLE_DEBUG_NAMES_AUG_STRING_ERROR	373
01420	#define	DW_DLE_DEBUG_NAMES_PAD_NON_ZERO	374
01421	#define	DW_DLE_DEBUG_NAMES_OFF_END	375
01422	#define	DW_DLE_DEBUG_NAMES_ABBREV_OVERFLOW	376
01423	#define	DW_DLE_DEBUG_NAMES_ABBREV_CORRUPTION	377
01424	#define	DW_DLE_DEBUG_NAMES_NULL_POINTER	378
01425	#define	DW_DLE_DEBUG_NAMES_BAD_INDEX_ARG	379
01426	#define	DW_DLE_DEBUG_NAMES_ENTRYPOOL_OFFSET	380
01427	#define	DW_DLE_DEBUG_NAMES_UNHANDLED_FORM	381
01428	#define	DW_DLE_LNCT_CODE_UNKNOWN	382
01429	#define	DW_DLE_LNCT_FORM_CODE_NOT_HANDLED	383
01430	#define	DW_DLE_LINE_HEADER_LENGTH_BOTCH	384
01431	#define	DW_DLE_STRING_HASHTAB_IDENTITY_ERROR	385
01432	#define	DW_DLE_UNIT_TYPE_NOT_HANDLED	386
01433	#define	DW_DLE_GROUP_MAP_ALLOC	387
01434	#define	DW_DLE_GROUP_MAP_DUPLICATE	388
01435	#define	DW_DLE_GROUP_COUNT_ERROR	389
01436	#define	DW_DLE_GROUP_INTERNAL_ERROR	390
01437	#define	DW_DLE_GROUP_LOAD_ERROR	391
01438	#define	DW_DLE_GROUP_LOAD_READ_ERROR	392
01439	#define	DW_DLE_AUG_DATA_LENGTH_BAD	393
01440	#define	DW_DLE_ABBREV_MISSING	394
01441	#define	DW_DLE_NO_TAG_FOR_DIE	395
01442	#define	DW_DLE_LOWPC_WRONG_CLASS	396
01443	#define	DW_DLE_HIGHPC_WRONG_FORM	397
01444	#define	DW_DLE_STR_OFFSETS_BASE_WRONG_FORM	398
01445	#define	DW_DLE_DATA16_OUTSIDE_SECTION	399
01446	#define	DW_DLE_LNCT_MD5_WRONG_FORM	400
01447	#define	DW_DLE_LINE_HEADER_CORRUPT	401
01448	#define	DW_DLE_STR_OFFSETS_NULLARGUMENT	402
01449	#define	DW_DLE_STR_OFFSETS_NULL_DBG	403
01450	#define	DW_DLE_STR_OFFSETS_NO_MAGIC	404
01451	#define	DW_DLE_STR_OFFSETS_ARRAY_SIZE	405
01452	#define	DW_DLE_STR_OFFSETS_VERSION_WRONG	406

```

01453 #define DW_DLE_STR_OFFSETS_ARRAY_INDEX_WRONG 407
01454 #define DW_DLE_STR_OFFSETS_EXTRA_BYTES 408
01455 #define DW_DLE_DUP_ATTR_ON_DIE 409
01456 #define DW_DLE_SECTION_NAME_BIG 410
01457 #define DW_DLE_FILE_UNAVAILABLE 411
01458 #define DW_DLE_FILE_WRONG_TYPE 412
01459 #define DW_DLE_SIBLING_OFFSET_WRONG 413
01460 #define DW_DLE_OPEN_FAIL 414
01461 #define DW_DLE_OFFSET_SIZE 415
01462 #define DW_DLE_MACH_O_SEGOFFSET_BAD 416
01463 #define DW_DLE_FILE_OFFSET_BAD 417
01464 #define DW_DLE_SEEK_ERROR 418
01465 #define DW_DLE_READ_ERROR 419
01466 #define DW_DLE_ELF_CLASS_BAD 420
01467 #define DW_DLE_ELF_ENDIAN_BAD 421
01468 #define DW_DLE_ELF_VERSION_BAD 422
01469 #define DW_DLE_FILE_TOO_SMALL 423
01470 #define DW_DLE_PATH_SIZE_TOO_SMALL 424
01471 #define DW_DLE_BAD_TYPE_SIZE 425
01472 #define DW_DLE_PE_SIZE_SMALL 426
01473 #define DW_DLE_PE_OFFSET_BAD 427
01474 #define DW_DLE_PE_STRING_TOO_LONG 428
01475 #define DW_DLE_IMAGE_FILE_UNKNOWN_TYPE 429
01476 #define DW_DLE_LINE_TABLE_LINENO_ERROR 430
01477 #define DW_DLE_PRODUCER_CODE_NOT_AVAILABLE 431
01478 #define DW_DLE_NO_ELF_SUPPORT 432
01479 #define DW_DLE_NO_STREAM_RELOC_SUPPORT 433
01480 #define DW_DLE_RETURN_EMPTY_PUBNAMES_ERROR 434
01481 #define DW_DLE_SECTION_SIZE_ERROR 435
01482 #define DW_DLE_INTERNAL_NULL_POINTER 436
01483 #define DW_DLE_SECTION_STRING_OFFSET_BAD 437
01484 #define DW_DLE_SECTION_INDEX_BAD 438
01485 #define DW_DLE_INTEGER_TOO_SMALL 439
01486 #define DW_DLE_ELF_SECTION_LINK_ERROR 440
01487 #define DW_DLE_ELF_SECTION_GROUP_ERROR 441
01488 #define DW_DLE_ELF_SECTION_COUNT_MISMATCH 442
01489 #define DW_DLE_ELF_STRING_SECTION_MISSING 443
01490 #define DW_DLE_SEEK_OFF_END 444
01491 #define DW_DLE_READ_OFF_END 445
01492 #define DW_DLE_ELF_SECTION_ERROR 446
01493 #define DW_DLE_ELF_STRING_SECTION_ERROR 447
01494 #define DW_DLE_MIXING_SPLIT_DWARF_VERSIONS 448
01495 #define DW_DLE_TAG_CORRUPT 449
01496 #define DW_DLE_FORM_CORRUPT 450
01497 #define DW_DLE_ATTR_CORRUPT 451
01498 #define DW_DLE_ABBREV_ATTR_DUPLICATION 452
01499 #define DW_DLE_DWP_SIGNATURE_MISMATCH 453
01500 #define DW_DLE_CU_UT_TYPE_VALUE 454
01501 #define DW_DLE_DUPLICATE_GNU_DEBUGLINK 455
01502 #define DW_DLE_CORRUPT_GNU_DEBUGLINK 456
01503 #define DW_DLE_CORRUPT_NOTE_GNU_DEBUGID 457
01504 #define DW_DLE_CORRUPT_GNU_DEBUGID_SIZE 458
01505 #define DW_DLE_CORRUPT_GNU_DEBUGID_STRING 459
01506 #define DW_DLE_HEX_STRING_ERROR 460
01507 #define DW_DLE_DECIMAL_STRING_ERROR 461
01508 #define DW_DLE_PRO_INIT_EXTRAS_UNKNOWN 462
01509 #define DW_DLE_PRO_INIT_EXTRAS_ERR 463
01510 #define DW_DLE_NULL_ARGS_DWARF_ADD_PATH 464
01511 #define DW_DLE_DWARF_INIT_DBG_NULL 465
01512 #define DW_DLE_ELF_RELOC_SECTION_ERROR 466
01513 #define DW_DLE_USER_DECLARED_ERROR 467
01514 #define DW_DLE_RNGLISTS_ERROR 468
01515 #define DW_DLE_LOCLISTS_ERROR 469
01516 #define DW_DLE_SECTION_SIZE_OR_OFFSET_LARGE 470
01517 #define DW_DLE_GDBINDEX_STRING_ERROR 471
01518 #define DW_DLE_GNU_PUBNAMES_ERROR 472
01519 #define DW_DLE_GNU_PUBTYPES_ERROR 473
01520 #define DW_DLE_DUPLICATE_GNU_DEBUG_PUBNAMES 474
01521 #define DW_DLE_DUPLICATE_GNU_DEBUG_PUBTYPES 475
01522 #define DW_DLE_DEBUG_SUP_STRING_ERROR 476
01523 #define DW_DLE_DEBUG_SUP_ERROR 477
01524 #define DW_DLE_LOCATION_ERROR 478
01525 #define DW_DLE_DEBUGLINK_PATH_SHORT 479
01526 #define DW_DLE_SIGNATURE_MISMATCH 480
01527 #define DW_DLE_MACRO_VERSION_ERROR 481
01528 #define DW_DLE_NEGATIVE_SIZE 482
01529 #define DW_DLE_UDATA_VALUE_NEGATIVE 483
01530 #define DW_DLE_DEBUG_NAMES_ERROR 484
01531 #define DW_DLE_CFA_INSTRUCTION_ERROR 485
01532 #define DW_DLE_MACHO_CORRUPT_HEADER 486
01533 #define DW_DLE_MACHO_CORRUPT_COMMAND 487
01534 #define DW_DLE_MACHO_CORRUPT_SECTIONDETAILS 488
01535 #define DW_DLE_RELOCATION_SECTION_SIZE_ERROR 489
01536 #define DW_DLE_SYMBOL_SECTION_SIZE_ERROR 490
01537 #define DW_DLE_PE_SECTION_SIZE_ERROR 491
01538 #define DW_DLE_DEBUG_ADDR_ERROR 492
01539 #define DW_DLE_NO_SECT_STRINGS 493

```

```

01540 #define DW_DLE_TOO_FEW_SECTIONS 494
01541 #define DW_DLE_BUILD_ID_DESCRIPTION_SIZE 495
01542 #define DW_DLE_BAD_SECTION_FLAGS 496
01543 #define DW_DLE_IMPROPER_SECTION_ZERO 497
01544 #define DW_DLE_INVALID_NULL_ARGUMENT 498
01545 #define DW_DLE_LINE_INDEX_WRONG 499
01546 #define DW_DLE_LINE_COUNT_WRONG 500
01547 #define DW_DLE_ARITHMETIC_OVERFLOW 501
01548 #define DW_DLE_UNIVERSAL_BINARY_ERROR 502
01549 #define DW_DLE_UNIV_BIN_OFFSET_SIZE_ERROR 503
01550 #define DW_DLE_PE_SECTION_SIZE_HEURISTIC_FAIL 504
01551 #define DW_DLE_LLE_ERROR 505
01552 #define DW_DLE_RLE_ERROR 506
01553 #define DW_DLE_MACHO_SEGMENT_COUNT_HEURISTIC_FAIL 507
01554 #define DW_DLE_DUPLICATE_NOTE_GNU_BUILD_ID 508
01555 #define DW_DLE_SYSCONF_VALUE_UNUSABLE 509
01556 #define DW_DLE_FRAME_ITERATOR_ERR 510
01557 #define DW_DLE_FRAME_FDE_TABLE_ERR 511
01558
01560 #define DW_DLE_LAST 511
01561 #define DW_DLE_LO_USER 0x10000
01626 DW_API int dwarf_init_path(const char * dw_path,
01627     char * dw_true_path_out_buffer,
01628     unsigned int dw_true_path_bufferlen,
01629     unsigned int dw_groupnumber,
01630     Dwarf_Handler dw_errhand,
01631     Dwarf_Ptr dw_errarg,
01632     Dwarf_Debug* dw_dbg,
01633     Dwarf_Error* dw_error);
01634
01647 DW_API int dwarf_init_path_a(const char * dw_path,
01648     char * dw_true_path_out_buffer,
01649     unsigned int dw_true_path_bufferlen,
01650     unsigned int dw_groupnumber,
01651     unsigned int dw_universalnumber,
01652     Dwarf_Handler dw_errhand,
01653     Dwarf_Ptr dw_errarg,
01654     Dwarf_Debug* dw_dbg,
01655     Dwarf_Error* dw_error);
01656
01713 DW_API int dwarf_init_path_dl(const char * dw_path,
01714     char * dw_true_path_out_buffer,
01715     unsigned int dw_true_path_bufferlen,
01716     unsigned int dw_groupnumber,
01717     Dwarf_Handler dw_errhand,
01718     Dwarf_Ptr dw_errarg,
01719     Dwarf_Debug* dw_dbg,
01720     char ** dw_dl_path_array,
01721     unsigned int dw_dl_path_array_size,
01722     unsigned char * dw_dl_path_source,
01723     Dwarf_Error* dw_error);
01724
01741 DW_API int dwarf_init_path_dl_a(const char * dw_path,
01742     char * dw_true_path_out_buffer,
01743     unsigned int dw_true_path_bufferlen,
01744     unsigned int dw_groupnumber,
01745     unsigned int dw_universalnumber,
01746     Dwarf_Handler dw_errhand,
01747     Dwarf_Ptr dw_errarg,
01748     Dwarf_Debug* dw_dbg,
01749     char ** dw_dl_path_array,
01750     unsigned int dw_dl_path_array_size,
01751     unsigned char * dw_dl_path_source,
01752     Dwarf_Error* dw_error);
01753
01787 DW_API int dwarf_init_b(int dw_fd,
01788     unsigned int dw_groupnumber,
01789     Dwarf_Handler dw_errhand,
01790     Dwarf_Ptr dw_errarg,
01791     Dwarf_Debug* dw_dbg,
01792     Dwarf_Error* dw_error);
01793
01809 DW_API int dwarf_finish(Dwarf_Debug dw_dbg);
01810
01847 DW_API int dwarf_object_init_b(Dwarf_Obj_Access_Interface* dw_obj,
01848     Dwarf_Handler dw_errhand,
01849     Dwarf_Ptr dw_errarg,
01850     unsigned int dw_groupnumber,
01851     Dwarf_Debug* dw_dbg,
01852     Dwarf_Error* dw_error);
01853
01868 DW_API int dwarf_object_finish(Dwarf_Debug dw_dbg);
01869
01900 DW_API int dwarf_set_tied_dbg(Dwarf_Debug dw_split_dbg,
01901     Dwarf_Debug dw_tied_dbg,
01902     Dwarf_Error* dw_error);
01903

```

```
01937 DW_API int dwarf_get_tied_dbg(Dwarf_Debug dw_dbg,
01938     Dwarf_Debug * dw_tieddbg_out,
01939     Dwarf_Error * dw_error);
02015 DW_API int dwarf_next_cu_header_e(Dwarf_Debug dw_dbg,
02016     Dwarf_Bool dw_is_info,
02017     Dwarf_Die *dw_cu_die,
02018     Dwarf_Unsigned *dw_cu_header_length,
02019     Dwarf_Half *dw_version_stamp,
02020     Dwarf_Off *dw_abbrev_offset,
02021     Dwarf_Half *dw_address_size,
02022     Dwarf_Half *dw_length_size,
02023     Dwarf_Half *dw_extension_size,
02024     Dwarf_Sig8 *dw_type_signature,
02025     Dwarf_Unsigned *dw_typeoffset,
02026     Dwarf_Unsigned *dw_next_cu_header_offset,
02027     Dwarf_Half *dw_header_cu_type,
02028     Dwarf_Error *dw_error);
02029
02067 DW_API int dwarf_next_cu_header_d(Dwarf_Debug dw_dbg,
02068     Dwarf_Bool dw_is_info,
02069     Dwarf_Unsigned *dw_cu_header_length,
02070     Dwarf_Half *dw_version_stamp,
02071     Dwarf_Off *dw_abbrev_offset,
02072     Dwarf_Half *dw_address_size,
02073     Dwarf_Half *dw_length_size,
02074     Dwarf_Half *dw_extension_size,
02075     Dwarf_Sig8 *dw_type_signature,
02076     Dwarf_Unsigned *dw_typeoffset,
02077     Dwarf_Unsigned *dw_next_cu_header_offset,
02078     Dwarf_Half *dw_header_cu_type,
02079     Dwarf_Error *dw_error);
02080
02096 DW_API int dwarf_siblingof_c(Dwarf_Die dw_die,
02097     Dwarf_Die *dw_return_siblingdie,
02098     Dwarf_Error *dw_error);
02099
02134 DW_API int dwarf_siblingof_b(Dwarf_Debug dw_dbg,
02135     Dwarf_Die dw_die,
02136     Dwarf_Bool dw_is_info,
02137     Dwarf_Die *dw_return_siblingdie,
02138     Dwarf_Error *dw_error);
02139
02180 DW_API int dwarf_cu_header_basics(Dwarf_Die dw_die,
02181     Dwarf_Half *dw_version,
02182     Dwarf_Bool *dw_is_info,
02183     Dwarf_Bool *dw_is_dwo,
02184     Dwarf_Half *dw_offset_size,
02185     Dwarf_Half *dw_address_size,
02186     Dwarf_Half *dw_extension_size,
02187     Dwarf_Sig8 **dw_signature,
02188     Dwarf_Off *dw_offset_of_length,
02189     Dwarf_Unsigned *dw_total_byte_length,
02190     Dwarf_Error *dw_error);
02191
02211 DW_API int dwarf_child(Dwarf_Die dw_die,
02212     Dwarf_Die* dw_return_childdie,
02213     Dwarf_Error* dw_error);
02214
02222 DW_API void dwarf_dealloc_die( Dwarf_Die dw_die);
02223
02241 DW_API int dwarf_die_from_hash_signature(Dwarf_Debug dw_dbg,
02242     Dwarf_Sig8 * dw_hash_sig,
02243     const char * dw_sig_type,
02244     Dwarf_Die* dw_returned_CU_die,
02245     Dwarf_Error* dw_error);
02246
02277 DW_API int dwarf_offdie_b(Dwarf_Debug dw_dbg,
02278     Dwarf_Off dw_offset,
02279     Dwarf_Bool dw_is_info,
02280     Dwarf_Die* dw_return_die,
02281     Dwarf_Error* dw_error);
02282
02304 DW_API int dwarf_find_die_given_sig8(Dwarf_Debug dw_dbg,
02305     Dwarf_Sig8 *dw_ref,
02306     Dwarf_Die *dw_die_out,
02307     Dwarf_Bool *dw_is_info,
02308     Dwarf_Error *dw_error);
02309
02320 DW_API Dwarf_Bool dwarf_get_die_infotypes_flag(Dwarf_Die dw_die);
02350 DW_API int dwarf_die_abbrev_global_offset(Dwarf_Die dw_die,
02351     Dwarf_Off * dw_abbrev_offset,
02352     Dwarf_Unsigned * dw_abbrev_count,
02353     Dwarf_Error* dw_error);
02354
02365 DW_API int dwarf_tag(Dwarf_Die dw_die,
02366     Dwarf_Half* dw_return_tag,
02367     Dwarf_Error* dw_error);
```

```

02368
02381 DW_API int dwarf_dieoffset(Dwarf_Die dw_die,
02382     Dwarf_Off* dw_return_offset,
02383     Dwarf_Error* dw_error);
02384
02403 DW_API int dwarf_debug_addr_index_to_addr(Dwarf_Die dw_die,
02404     Dwarf_Unsigned dw_index,
02405     Dwarf_Addr * dw_return_addr,
02406     Dwarf_Error * dw_error);
02407
02416 DW_API Dwarf_Bool dwarf_addr_form_is_indexed(int dw_form);
02417
02441 DW_API int dwarf_CU_dieoffset_given_die(Dwarf_Die dw_die,
02442     Dwarf_Off* dw_return_offset,
02443     Dwarf_Error* dw_error);
02444
02465 DW_API int dwarf_get_cu_die_offset_given_cu_header_offset_b(
02466     Dwarf_Debug dw_dbg,
02467     Dwarf_Off dw_in_cu_header_offset,
02468     Dwarf_Bool dw_is_info,
02469     Dwarf_Off * dw_out_cu_die_offset,
02470     Dwarf_Error *dw_error);
02471
02489 DW_API int dwarf_die_CU_offset(Dwarf_Die dw_die,
02490     Dwarf_Off* dw_return_offset,
02491     Dwarf_Error* dw_error);
02492
02513 DW_API int dwarf_die_CU_offset_range(Dwarf_Die dw_die,
02514     Dwarf_Off* dw_return_CU_header_offset,
02515     Dwarf_Off* dw_return_CU_length_bytes,
02516     Dwarf_Error* dw_error);
02517
02535 DW_API int dwarf_attr(Dwarf_Die dw_die,
02536     Dwarf_Half dw_attrnum,
02537     Dwarf_Attribute * dw_returned_attr,
02538     Dwarf_Error* dw_error);
02539
02561 DW_API int dwarf_die_text(Dwarf_Die dw_die,
02562     Dwarf_Half dw_attrnum,
02563     char ** dw_ret_name,
02564     Dwarf_Error * dw_error);
02565
02585 DW_API int dwarf_diename(Dwarf_Die dw_die,
02586     char ** dw_diename,
02587     Dwarf_Error* dw_error);
02588
02606 DW_API Dwarf_Unsigned dwarf_die_abbrev_code(Dwarf_Die dw_die);
02607
02621 DW_API int dwarf_die_abbrev_children_flag(Dwarf_Die dw_die,
02622     Dwarf_Half * dw_ab_has_child);
02623
02647 DW_API int dwarf_validate_die_sibling(Dwarf_Die dw_sibling,
02648     Dwarf_Off* dw_offset);
02649
02650 /* convenience functions, alternative to using dwarf_attrlist */
02651
02670 DW_API int dwarf_hasattr(Dwarf_Die dw_die,
02671     Dwarf_Half dw_attrnum,
02672     Dwarf_Bool * dw_returned_bool,
02673     Dwarf_Error* dw_error);
02674
02710 DW_API int dwarf_offset_list(Dwarf_Debug dw_dbg,
02711     Dwarf_Off dw_offset,
02712     Dwarf_Bool dw_is_info,
02713     Dwarf_Off ** dw_offbuf,
02714     Dwarf_Unsigned * dw_offcount,
02715     Dwarf_Error * dw_error);
02716
02729 DW_API int dwarf_get_die_address_size(Dwarf_Die dw_die,
02730     Dwarf_Half * dw_addr_size,
02731     Dwarf_Error * dw_error);
02732
02733 /* Get both offsets (local and global) */
02753 DW_API int dwarf_die_offsets(Dwarf_Die dw_die,
02754     Dwarf_Off* dw_global_offset,
02755     Dwarf_Off* dw_local_offset,
02756     Dwarf_Error* dw_error);
02757
02780 DW_API int dwarf_get_version_of_die(Dwarf_Die dw_die,
02781     Dwarf_Half * dw_version,
02782     Dwarf_Half * dw_offset_size);
02783
02797 DW_API int dwarf_lowpc(Dwarf_Die dw_die,
02798     Dwarf_Addr * dw_returned_addr,
02799     Dwarf_Error* dw_error);
02800
02832 DW_API int dwarf_highpc_b(Dwarf_Die dw_die,

```

```

02833     Dwarf_Addr *      dw_return_addr,
02834     Dwarf_Half *      dw_return_form,
02835     enum Dwarf_Form_Class * dw_return_class,
02836     Dwarf_Error *      dw_error);
02837
02863 DW_API int dwarf_dietype_offset(Dwarf_Die dw_die,
02864     Dwarf_Off * dw_return_offset,
02865     Dwarf_Bool * dw_is_info,
02866     Dwarf_Error * dw_error);
02867
02879 DW_API int dwarf_bytesize(Dwarf_Die dw_die,
02880     Dwarf_Unsigned * dw_returned_size,
02881     Dwarf_Error*     dw_error);
02882
02894 DW_API int dwarf_bitsize(Dwarf_Die dw_die,
02895     Dwarf_Unsigned * dw_returned_size,
02896     Dwarf_Error*     dw_error);
02897
02918 DW_API int dwarf_bitoffset(Dwarf_Die dw_die,
02919     Dwarf_Half * dw_attrnum,
02920     Dwarf_Unsigned * dw_returned_offset,
02921     Dwarf_Error*     dw_error);
02922
02948 DW_API int dwarf_srclang(Dwarf_Die dw_die,
02949     Dwarf_Unsigned * dw_returned_lang,
02950     Dwarf_Error * dw_error);
02951
02976 DW_API int dwarf_srclanglname(Dwarf_Die dw_die,
02977     Dwarf_Unsigned *dw_returned_lname,
02978     Dwarf_Error *dw_error);
02979
03004 DW_API int dwarf_srclanglname_version(Dwarf_Die dw_die,
03005     const char *dw_returned_verstring,
03006     Dwarf_Error *dw_error);
03007
03036 DW_API int dwarf_language_version_data(
03037     Dwarf_Unsigned dw_lname_name,
03038     int *dw_default_lower_bound,
03039     const char **dw_version_string);
03040
03046 DW_API int dwarf_language_version_string(
03047     Dwarf_Unsigned dw_lname_name,
03048     int *dw_default_lower_bound,
03049     const char **dw_version_string);
03050
03082 DW_API int dwarf_lvn_name_direct(Dwarf_Unsigned dw_lv_lang,
03083     Dwarf_Unsigned dw_lv_ver,
03084     const char **dw_ret_version_name,
03085     const char **dw_ret_version_scheme);
03086
03119 DW_API int dwarf_lvn_name(Dwarf_Die dw_die,
03120     const char **dw_ret_version_name,
03121     const char **dw_ret_version_scheme);
03122
03160 DW_API int dwarf_lvn_table_entry(Dwarf_Unsigned dw_lvn_index,
03161     Dwarf_Unsigned *dw_lvn_language_name,
03162     Dwarf_Unsigned *dw_lvn_language_version,
03163     const char **dw_lvn_language_version_scheme,
03164     const char **dw_lvn_language_version_name);
03165
03178 DW_API int dwarf_arrayorder(Dwarf_Die dw_die,
03179     Dwarf_Unsigned * dw_returned_order,
03180     Dwarf_Error*     dw_error);
03210 DW_API int dwarf_attrlist(Dwarf_Die dw_die,
03211     Dwarf_Attribute** dw_attrbuf,
03212     Dwarf_Signed * dw_attrcount,
03213     Dwarf_Error*     dw_error);
03214
03233 DW_API int dwarf_hasform(Dwarf_Attribute dw_attr,
03234     Dwarf_Half dw_form,
03235     Dwarf_Bool * dw_returned_bool,
03236     Dwarf_Error* dw_error);
03237
03256 DW_API int dwarf_whatform(Dwarf_Attribute dw_attr,
03257     Dwarf_Half * dw_returned_final_form,
03258     Dwarf_Error* dw_error);
03259
03276 DW_API int dwarf_whatform_direct(Dwarf_Attribute dw_attr,
03277     Dwarf_Half * dw_returned_initial_form,
03278     Dwarf_Error* dw_error);
03279
03295 DW_API int dwarf_whatattr(Dwarf_Attribute dw_attr,
03296     Dwarf_Half * dw_returned_attrnum,
03297     Dwarf_Error* dw_error);
03298
03323 DW_API int dwarf_formref(Dwarf_Attribute dw_attr,
03324     Dwarf_Off*     dw_return_offset,

```

```

03325     Dwarf_Bool *dw_is_info,
03326     Dwarf_Error *dw_error);
03327
03360 DW_API int dwarf_global_formref_b(Dwarf_Attribute dw_attr,
03361     Dwarf_Off *dw_return_offset,
03362     Dwarf_Bool *dw_offset_is_info,
03363     Dwarf_Error *dw_error);
03364
03375 DW_API int dwarf_global_formref(Dwarf_Attribute dw_attr,
03376     Dwarf_Off* dw_return_offset,
03377     Dwarf_Error* dw_error);
03378
03397 DW_API int dwarf_formsig8(Dwarf_Attribute dw_attr,
03398     Dwarf_Sig8 * dw_returned_sig_bytes,
03399     Dwarf_Error* dw_error);
03400
03419 DW_API int dwarf_formsig8_const(Dwarf_Attribute dw_attr,
03420     Dwarf_Sig8 * dw_returned_sig_bytes,
03421     Dwarf_Error* dw_error);
03422
03442 DW_API int dwarf_formaddr(Dwarf_Attribute dw_attr,
03443     Dwarf_Addr * dw_returned_addr,
03444     Dwarf_Error* dw_error);
03445
03465 DW_API int dwarf_get_debug_addr_index(Dwarf_Attribute dw_attr,
03466     Dwarf_Unsigned * dw_return_index,
03467     Dwarf_Error * dw_error);
03468
03482 DW_API int dwarf_formflag(Dwarf_Attribute dw_attr,
03483     Dwarf_Bool * dw_returned_bool,
03484     Dwarf_Error* dw_error);
03485
03501 DW_API int dwarf_formudata(Dwarf_Attribute dw_attr,
03502     Dwarf_Unsigned * dw_returned_val,
03503     Dwarf_Error* dw_error);
03504
03519 DW_API int dwarf_formsdata(Dwarf_Attribute dw_attr,
03520     Dwarf_Signed * dw_returned_val,
03521     Dwarf_Error* dw_error);
03522
03540 DW_API int dwarf_formdata16(Dwarf_Attribute dw_attr,
03541     Dwarf_Form_Data16 * dw_returned_val,
03542     Dwarf_Error* dw_error);
03543
03562 DW_API int dwarf_formblock(Dwarf_Attribute dw_attr,
03563     Dwarf_Block ** dw_returned_block,
03564     Dwarf_Error* dw_error);
03565
03580 DW_API int dwarf_formstring(Dwarf_Attribute dw_attr,
03581     char ** dw_returned_string,
03582     Dwarf_Error* dw_error);
03583
03599 DW_API int dwarf_get_debug_str_index(Dwarf_Attribute dw_attr,
03600     Dwarf_Unsigned * dw_return_index,
03601     Dwarf_Error * dw_error);
03602
03621 DW_API int dwarf_formexprloc(Dwarf_Attribute dw_attr,
03622     Dwarf_Unsigned * dw_return_exprlen,
03623     Dwarf_Ptr * dw_block_ptr,
03624     Dwarf_Error * dw_error);
03625
03645 DW_API enum Dwarf_Form_Class dwarf_get_form_class(
03646     Dwarf_Half dw_version,
03647     Dwarf_Half dw_attrnum,
03648     Dwarf_Half dw_offset_size,
03649     Dwarf_Half dw_form);
03650
03666 DW_API int dwarf_attr_offset(Dwarf_Die dw_die,
03667     Dwarf_Attribute dw_attr,
03668     Dwarf_Off * dw_return_offset,
03669     Dwarf_Error * dw_error);
03670
03678 DW_API int dwarf_uncompress_integer_block_a(Dwarf_Debug dw_dbg,
03679     Dwarf_Unsigned dw_input_length_in_bytes,
03680     void * dw_input_block,
03681     Dwarf_Unsigned * dw_value_count,
03682     Dwarf_Signed ** dw_value_array,
03683     Dwarf_Error * dw_error);
03684
03692 DW_API void dwarf_dealloc_uncompressed_block(Dwarf_Debug dw_dbg,
03693     void *dw_value_array);
03694
03714 DW_API int dwarf_convert_to_global_offset(Dwarf_Attribute dw_attr,
03715     Dwarf_Off dw_offset,
03716     Dwarf_Off* dw_return_offset,
03717     Dwarf_Error* dw_error);
03718

```

```

03724 DW_API void dwarf_dealloc_attribute(Dwarf_Attribute dw_attr);
03725
03752 DW_API int dwarf_discr_list(Dwarf_Debug dw_dbg,
03753     Dwarf_Small * dw_blockpointer,
03754     Dwarf_Unsigned dw_blocklen,
03755     Dwarf_Dsc_Head * dw_dsc_head_out,
03756     Dwarf_Unsigned * dw_dsc_array_length_out,
03757     Dwarf_Error * dw_error);
03758
03784 DW_API int dwarf_discr_entry_u(Dwarf_Dsc_Head dw_dsc,
03785     Dwarf_Unsigned dw_entrynum,
03786     Dwarf_Half * dw_out_type,
03787     Dwarf_Unsigned * dw_out_discr_low,
03788     Dwarf_Unsigned * dw_out_discr_high,
03789     Dwarf_Error * dw_error);
03790
03796 DW_API int dwarf_discr_entry_s(Dwarf_Dsc_Head dw_dsc,
03797     Dwarf_Unsigned dw_entrynum,
03798     Dwarf_Half * dw_out_type,
03799     Dwarf_Signed * dw_out_discr_low,
03800     Dwarf_Signed * dw_out_discr_high,
03801     Dwarf_Error * dw_error);
03802
03880 DW_API int dwarf_srcfiles(Dwarf_Die dw_cu_die,
03881     char *** dw_srcfiles,
03882     Dwarf_Signed * dw_filecount,
03883     Dwarf_Error * dw_error);
03884
03911 DW_API int dwarf_srclines_b(Dwarf_Die dw_cudie,
03912     Dwarf_Unsigned * dw_version_out,
03913     Dwarf_Small * dw_table_count,
03914     Dwarf_Line_Context * dw_linecontext,
03915     Dwarf_Error * dw_error);
03916
03937 DW_API int dwarf_srclines_from_linecontext(
03938     Dwarf_Line_Context dw_linecontext,
03939     Dwarf_Line ** dw_linebuf,
03940     Dwarf_Signed * dw_linecount,
03941     Dwarf_Error * dw_error);
03942
03959 DW_API int dwarf_srclines_two_level_from_linecontext(
03960     Dwarf_Line_Context dw_context,
03961     Dwarf_Line ** dw_linebuf,
03962     Dwarf_Signed * dw_linecount,
03963     Dwarf_Line ** dw_linebuf_actuals,
03964     Dwarf_Signed * dw_linecount_actuals,
03965     Dwarf_Error * dw_error);
03966
03976 DW_API void dwarf_srclines_dealloc_b(Dwarf_Line_Context dw_context);
03977
03993 DW_API int dwarf_srclines_table_offset(Dwarf_Line_Context dw_context,
03994     Dwarf_Unsigned * dw_offset,
03995     Dwarf_Error * dw_error);
03996
04012 DW_API int dwarf_srclines_comp_dir(Dwarf_Line_Context dw_context,
04013     const char ** dw_compilation_directory,
04014     Dwarf_Error * dw_error);
04015
04031 DW_API int dwarf_srclines_subprog_count(Dwarf_Line_Context dw_context,
04032     Dwarf_Signed * dw_count,
04033     Dwarf_Error * dw_error);
04034
04053 DW_API int dwarf_srclines_subprog_data(Dwarf_Line_Context dw_context,
04054     Dwarf_Signed dw_index,
04055     const char ** dw_name,
04056     Dwarf_Unsigned * dw_decl_file,
04057     Dwarf_Unsigned * dw_decl_line,
04058     Dwarf_Error * dw_error);
04059
04084 DW_API int dwarf_srclines_files_indexes(
04085     Dwarf_Line_Context dw_context,
04086     Dwarf_Signed * dw_baseindex,
04087     Dwarf_Signed * dw_count,
04088     Dwarf_Signed * dw_endindex,
04089     Dwarf_Error * dw_error);
04090
04142 DW_API int dwarf_srclines_files_data_b(
04143     Dwarf_Line_Context dw_context,
04144     Dwarf_Signed dw_index_in,
04145     const char ** dw_name,
04146     Dwarf_Unsigned * dw_directory_index,
04147     Dwarf_Unsigned * dw_last_mod_time,
04148     Dwarf_Unsigned * dw_file_length,
04149     Dwarf_Form_Data16 ** dw_md5ptr,
04150     Dwarf_Error * dw_error);
04151
04166 DW_API int dwarf_srclines_include_dir_count(

```



```

04167     Dwarf_Line_Context dw_line_context,
04168     Dwarf_Signed * dw_count,
04169     Dwarf_Error * dw_error);
04170
04193 DW_API int dwarf_srclines_include_dir_data(
04194     Dwarf_Line_Context dw_line_context,
04195     Dwarf_Signed dw_index,
04196     const char ** dw_name,
04197     Dwarf_Error * dw_error);
04198
04227 DW_API int dwarf_srclines_version(Dwarf_Line_Context dw_line_context,
04228     Dwarf_Unsigned * dw_version,
04229     Dwarf_Small * dw_table_count,
04230     Dwarf_Error * dw_error);
04231
04247 DW_API int dwarf_linebeginstatement(Dwarf_Line dw_line,
04248     Dwarf_Bool * dw_returned_bool,
04249     Dwarf_Error * dw_error);
04250
04266 DW_API int dwarf_lineendsequence(Dwarf_Line dw_line,
04267     Dwarf_Bool * dw_returned_bool,
04268     Dwarf_Error * dw_error);
04269
04284 DW_API int dwarf_lineno(Dwarf_Line dw_line,
04285     Dwarf_Unsigned * dw_returned_lineno,
04286     Dwarf_Error * dw_error);
04287
04302 DW_API int dwarf_line_srcfileno(Dwarf_Line dw_line,
04303     Dwarf_Unsigned * dw_returned_filenum,
04304     Dwarf_Error * dw_error);
04305
04319 DW_API int dwarf_line_is_addr_set(Dwarf_Line dw_line,
04320     Dwarf_Bool * dw_is_addr_set,
04321     Dwarf_Error * dw_error);
04322
04337 DW_API int dwarf_lineaddr(Dwarf_Line dw_line,
04338     Dwarf_Addr * dw_returned_addr,
04339     Dwarf_Error* dw_error);
04340
04355 DW_API int dwarf_lineoff_b(Dwarf_Line dw_line,
04356     Dwarf_Unsigned * dw_returned_lineoffset,
04357     Dwarf_Error* dw_error);
04358
04383 DW_API int dwarf_linesrc(Dwarf_Line dw_line,
04384     char ** dw_returned_name,
04385     Dwarf_Error* dw_error);
04386
04401 DW_API int dwarf_lineblock(Dwarf_Line dw_line,
04402     Dwarf_Bool * dw_returned_bool,
04403     Dwarf_Error* dw_error);
04404
04405 /* We gather these into one call as it's likely one
04406    will want all or none of them. */
04430 DW_API int dwarf_prologue_end_etc(Dwarf_Line dw_line,
04431     Dwarf_Bool * dw_prologue_end,
04432     Dwarf_Bool * dw_epilogue_begin,
04433     Dwarf_Unsigned * dw_isa,
04434     Dwarf_Unsigned * dw_discriminator,
04435     Dwarf_Error * dw_error);
04436 /* End line table operations */
04437
04443 DW_API int dwarf_linelogical(Dwarf_Line dw_line,
04444     Dwarf_Unsigned * dw_returned_logical,
04445     Dwarf_Error* dw_error);
04446
04453 DW_API int dwarf_linecontext(Dwarf_Line dw_line,
04454     Dwarf_Unsigned * dw_returned_context,
04455     Dwarf_Error* dw_error);
04456
04465 DW_API int dwarf_line_subprogn(Dwarf_Line /*line*/,
04466     Dwarf_Unsigned * /*ret_subprogn*/,
04467     Dwarf_Error * /*error*/);
04468
04475 DW_API int dwarf_line_subprog(Dwarf_Line /*line*/,
04476     char ** /*returned_subprog_name*/,
04477     char ** /*returned_filename*/,
04478     Dwarf_Unsigned * /*returned_lineno*/,
04479     Dwarf_Error * /*error*/);
04480
04501 DW_API int dwarf_check_lineheader_b(Dwarf_Die dw_cu_die,
04502     int * dw_errcount_out,
04503     Dwarf_Error * dw_error);
04504
04534 DW_API int dwarf_print_lines(Dwarf_Die dw_cu_die,
04535     Dwarf_Error * dw_error,
04536     int * dw_errcount_out);
04537

```

```

04558 DW_API struct Dwarf_Printf_Callback_Info_s
04559 dwarf_register_printf_callback(Dwarf_Debug dw_dbg,
04560 struct Dwarf_Printf_Callback_Info_s * dw_callbackinfo);
04561
04621 DW_API int dwarf_get_ranges_b(Dwarf_Debug dw_dbg,
04622 Dwarf_Off dw_rangesoffset,
04623 Dwarf_Die dw_die,
04624 Dwarf_Off * dw_return_realoffset,
04625 Dwarf_Ranges ** dw_rangesbuf,
04626 Dwarf_Signed * dw_rangecount,
04627 Dwarf_Unsigned * dw_bytecount,
04628 Dwarf_Error * dw_error);
04629
04639 DW_API void dwarf_dealloc_ranges(Dwarf_Debug dw_dbg,
04640 Dwarf_Ranges * dw_rangesbuf,
04641 Dwarf_Signed dw_rangecount);
04642
04683 DW_API int dwarf_get_ranges_baseaddress(Dwarf_Debug dw_dbg,
04684 Dwarf_Die dw_die,
04685 Dwarf_Bool *dw_known_base,
04686 Dwarf_Unsigned *dw_baseaddress,
04687 Dwarf_Bool *dw_at_ranges_offset_present,
04688 Dwarf_Unsigned *dw_at_ranges_offset,
04689 Dwarf_Error *dw_error);
04690
04736 DW_API int dwarf_rnglists_get_rle_head(Dwarf_Attribute dw_attr,
04737 Dwarf_Half dw_theform,
04738 Dwarf_Unsigned dw_index_or_offset_value,
04739 Dwarf_Rnglists_Head * dw_head_out,
04740 Dwarf_Unsigned * dw_count_of_entries_in_head,
04741 Dwarf_Unsigned * dw_global_offset_of_rle_set,
04742 Dwarf_Error * dw_error);
04743
04786 DW_API int dwarf_get_rnglists_entry_fields_a(
04787 Dwarf_Rnglists_Head dw_head,
04788 Dwarf_Unsigned dw_entrynum,
04789 unsigned int * dw_entrylen,
04790 unsigned int * dw_rle_value_out,
04791 Dwarf_Unsigned * dw_raw1,
04792 Dwarf_Unsigned * dw_raw2,
04793 Dwarf_Bool * dw_debug_addr_unavailable,
04794 Dwarf_Unsigned * dw_cooked1,
04795 Dwarf_Unsigned * dw_cooked2,
04796 Dwarf_Error * dw_error);
04797
04805 DW_API void dwarf_dealloc_rnglists_head(Dwarf_Rnglists_Head dw_head);
04806
04837 DW_API int dwarf_load_rnglists(Dwarf_Debug dw_dbg,
04838 Dwarf_Unsigned * dw_rnglists_count,
04839 Dwarf_Error * dw_error);
04840
04867 DW_API int dwarf_get_rnglist_offset_index_value(Dwarf_Debug dw_dbg,
04868 Dwarf_Unsigned dw_context_index,
04869 Dwarf_Unsigned dw_offsetentry_index,
04870 Dwarf_Unsigned * dw_offset_value_out,
04871 Dwarf_Unsigned * dw_global_offset_value_out,
04872 Dwarf_Error * dw_error);
04873
04880 DW_API int dwarf_get_rnglist_head_basics(Dwarf_Rnglists_Head dw_head,
04881 Dwarf_Unsigned * dw_rle_count,
04882 Dwarf_Unsigned * dw_rnglists_version,
04883 Dwarf_Unsigned * dw_rnglists_index_returned,
04884 Dwarf_Unsigned * dw_bytes_total_in_rle,
04885 Dwarf_Half * dw_offset_size,
04886 Dwarf_Half * dw_address_size,
04887 Dwarf_Half * dw_segment_selector_size,
04888 Dwarf_Unsigned * dw_overall_offset_of_this_context,
04889 Dwarf_Unsigned * dw_total_length_of_this_context,
04890 Dwarf_Unsigned * dw_offset_table_offset,
04891 Dwarf_Unsigned * dw_offset_table_entrycount,
04892 Dwarf_Bool * dw_rnglists_base_present,
04893 Dwarf_Unsigned * dw_rnglists_base,
04894 Dwarf_Bool * dw_rnglists_base_address_present,
04895 Dwarf_Unsigned * dw_rnglists_base_address,
04896 Dwarf_Bool * dw_rnglists_debug_addr_base_present,
04897 Dwarf_Unsigned * dw_rnglists_debug_addr_base,
04898 Dwarf_Error * dw_error);
04899
04915 DW_API int dwarf_get_rnglist_context_basics(Dwarf_Debug dw_dbg,
04916 Dwarf_Unsigned dw_index,
04917 Dwarf_Unsigned * dw_header_offset,
04918 Dwarf_Small * dw_offset_size,
04919 Dwarf_Small * dw_extension_size,
04920 unsigned int * dw_version,
04921 Dwarf_Small * dw_address_size,
04922 Dwarf_Small * dw_segment_selector_size,
04923 Dwarf_Unsigned * dw_offset_entry_count,

```

```

04924     Dwarf_Unsigned * dw_offset_of_offset_array,
04925     Dwarf_Unsigned * dw_offset_of_first_rangeentry,
04926     Dwarf_Unsigned * dw_offset_past_last_rangeentry,
04927     Dwarf_Error *   dw_error);
04928
04938 DW_API int dwarf_get_rnglist_rle(Dwarf_Debug dw_dbg,
04939     Dwarf_Unsigned dw_contextnumber,
04940     Dwarf_Unsigned dw_entry_offset,
04941     Dwarf_Unsigned dw_endoffset,
04942     unsigned int   * dw_entrylen,
04943     unsigned int   * dw_entry_kind,
04944     Dwarf_Unsigned * dw_entry_operand1,
04945     Dwarf_Unsigned * dw_entry_operand2,
04946     Dwarf_Error *   dw_error);
04973 DW_API int dwarf_get_loclist_c(Dwarf_Attribute dw_attr,
04974     Dwarf_Loc_Head_c * dw_loclist_head,
04975     Dwarf_Unsigned * dw_locentry_count,
04976     Dwarf_Error *   dw_error);
04977
04978 #define DW_LKIND_expression 0 /* DWARF2,3,4,5 */
04979 #define DW_LKIND_loclist 1 /* DWARF 2,3,4 */
04980 #define DW_LKIND_GNU_exp_list 2 /* GNU DWARF4 .dwo extension */
04981 #define DW_LKIND_loclists 5 /* DWARF5 loclists */
04982 #define DW_LKIND_unknown 99
04983
04996 DW_API int dwarf_get_loclist_head_kind(
04997     Dwarf_Loc_Head_c dw_loclist_head,
04998     unsigned int * dw_lkind,
04999     Dwarf_Error * dw_error);
05000
05055 DW_API int dwarf_get_locdesc_entry_d(Dwarf_Loc_Head_c dw_loclist_head,
05056     Dwarf_Unsigned dw_index,
05057     Dwarf_Small * dw_lle_value_out,
05058     Dwarf_Unsigned * dw_rawlowpc,
05059     Dwarf_Unsigned * dw_rawhipc,
05060     Dwarf_Bool * dw_debug_addr_unavailable,
05061     Dwarf_Addr * dw_lowpc_cooked,
05062     Dwarf_Addr * dw_hipc_cooked,
05063     Dwarf_Unsigned * dw_locexpr_op_count_out,
05064     Dwarf_Locdesc_c * dw_locentry_out,
05065     Dwarf_Small * dw_loclist_source_out,
05066     Dwarf_Unsigned * dw_expression_offset_out,
05067     Dwarf_Unsigned * dw_locdesc_offset_out,
05068     Dwarf_Error * dw_error);
05069
05089 DW_API int dwarf_get_locdesc_entry_e(Dwarf_Loc_Head_c dw_loclist_head,
05090     Dwarf_Unsigned dw_index,
05091     Dwarf_Small * dw_lle_value_out,
05092     Dwarf_Unsigned * dw_rawlowpc,
05093     Dwarf_Unsigned * dw_rawhipc,
05094     Dwarf_Bool * dw_debug_addr_unavailable,
05095     Dwarf_Addr * dw_lowpc_cooked,
05096     Dwarf_Addr * dw_hipc_cooked,
05097     Dwarf_Unsigned * dw_locexpr_op_count_out,
05098     Dwarf_Unsigned * dw_lle_bytecount,
05099     Dwarf_Locdesc_c * dw_locentry_out,
05100     Dwarf_Small * dw_loclist_source_out,
05101     Dwarf_Unsigned * dw_expression_offset_out,
05102     Dwarf_Unsigned * dw_locdesc_offset_out,
05103     Dwarf_Error * dw_error);
05104
05130 DW_API int dwarf_get_location_op_value_c(Dwarf_Locdesc_c dw_locdesc,
05131     Dwarf_Unsigned dw_index,
05132     Dwarf_Small * dw_operator_out,
05133     Dwarf_Unsigned * dw_operand1,
05134     Dwarf_Unsigned * dw_operand2,
05135     Dwarf_Unsigned * dw_operand3,
05136     Dwarf_Unsigned * dw_offset_for_branch,
05137     Dwarf_Error * dw_error);
05169 DW_API int dwarf_loclist_from_expr_c(Dwarf_Debug dw_dbg,
05170     Dwarf_Ptr dw_expression_in,
05171     Dwarf_Unsigned dw_expression_length,
05172     Dwarf_Half dw_address_size,
05173     Dwarf_Half dw_offset_size,
05174     Dwarf_Half dw_dwarf_version,
05175     Dwarf_Loc_Head_c * dw_loc_head,
05176     Dwarf_Unsigned * dw_listlen,
05177     Dwarf_Error * dw_error);
05178
05186 DW_API void dwarf_dealloc_loc_head_c(Dwarf_Loc_Head_c dw_head);
05187
05188 /* These interfaces allow reading the .debug_loclists
05189 section. Independently of DIEs.
05190 Normal use of .debug_loclists uses
05191 dwarf_get_loclist_c() to open access to any kind of location
05192 or loclist and uses dwarf_loc_head_c_dealloc() to
05193 deallocate that memory once one is finished with

```

```

05194     that data. So for most purposes you do not need
05195     to use these functions
05196     See dwarf_get_loclist_c() to open a Dwarf_Loc_Head_c
05197     on any type of location list or expression. */
05198
05199 /* Loads all the loclists headers and
05200    returns DW_DLV_NO_ENTRY if the section
05201    is missing or empty.
05202    Intended to be done quite early and
05203    it is automatically
05204    done if .debug_info is loaded.
05205    Doing it more than once is never necessary
05206    or harmful. There is no deallocation call
05207    made visible, deallocation happens
05208    when dwarf_finish() is called.
05209    With DW_DLV_OK it returns the number of
05210    loclists headers in the section through
05211    loclists_count. */
05241 DW_API int dwarf_load_loclists(Dwarf_Debug dw_dbg,
05242     Dwarf_Unsigned * dw_loclists_count,
05243     Dwarf_Error * dw_error);
05244
05270 DW_API int dwarf_get_loclist_offset_index_value(Dwarf_Debug dw_dbg,
05271     Dwarf_Unsigned dw_context_index,
05272     Dwarf_Unsigned dw_offsetentry_index,
05273     Dwarf_Unsigned * dw_offset_value_out,
05274     Dwarf_Unsigned * dw_global_offset_value_out,
05275     Dwarf_Error * dw_error);
05276
05291 DW_API int dwarf_get_loclist_head_basics(Dwarf_Loc_Head_c dw_head,
05292     Dwarf_Small * dw_lkind,
05293     Dwarf_Unsigned * dw_lle_count,
05294     Dwarf_Unsigned * dw_loclists_version,
05295     Dwarf_Unsigned * dw_loclists_index_returned,
05296     Dwarf_Unsigned * dw_bytes_total_in_rle,
05297     Dwarf_Half * dw_offset_size,
05298     Dwarf_Half * dw_address_size,
05299     Dwarf_Half * dw_segment_selector_size,
05300     Dwarf_Unsigned * dw_overall_offset_of_this_context,
05301     Dwarf_Unsigned * dw_total_length_of_this_context,
05302     Dwarf_Unsigned * dw_offset_table_offset,
05303     Dwarf_Unsigned * dw_offset_table_entrycount,
05304     Dwarf_Bool * dw_loclists_base_present,
05305     Dwarf_Unsigned * dw_loclists_base,
05306     Dwarf_Bool * dw_loclists_base_address_present,
05307     Dwarf_Unsigned * dw_loclists_base_address,
05308     Dwarf_Bool * dw_loclists_debug_addr_base_present,
05309     Dwarf_Unsigned * dw_loclists_debug_addr_base,
05310     Dwarf_Unsigned * dw_offset_this_lle_area,
05311     Dwarf_Error * dw_error);
05312
05321 DW_API int dwarf_get_loclist_context_basics(Dwarf_Debug dw_dbg,
05322     Dwarf_Unsigned dw_index,
05323     Dwarf_Unsigned * dw_header_offset,
05324     Dwarf_Small * dw_offset_size,
05325     Dwarf_Small * dw_extension_size,
05326     unsigned int * dw_version,
05327     Dwarf_Small * dw_address_size,
05328     Dwarf_Small * dw_segment_selector_size,
05329     Dwarf_Unsigned * dw_offset_entry_count,
05330     Dwarf_Unsigned * dw_offset_of_offset_array,
05331     Dwarf_Unsigned * dw_offset_of_first_locentry,
05332     Dwarf_Unsigned * dw_offset_past_last_locentry,
05333     Dwarf_Error * dw_error);
05334
05339 DW_API int dwarf_get_loclist_lle(Dwarf_Debug dw_dbg,
05340     Dwarf_Unsigned dw_contextnumber,
05341     Dwarf_Unsigned dw_entry_offset,
05342     Dwarf_Unsigned dw_endoffset,
05343     unsigned int * dw_entrylen,
05344     unsigned int * dw_entry_kind,
05345     Dwarf_Unsigned * dw_entry_operand1,
05346     Dwarf_Unsigned * dw_entry_operand2,
05347     Dwarf_Unsigned * dw_expr_ops_blocksize,
05348     Dwarf_Unsigned * dw_expr_ops_offset,
05349     Dwarf_Small ** dw_expr_opsdata,
05350     Dwarf_Error * dw_error);
05430 DW_API int dwarf_debug_addr_table(Dwarf_Debug dw_dbg,
05431     Dwarf_Unsigned dw_section_offset,
05432     Dwarf_Debug_Addr_Table * dw_table_header,
05433     Dwarf_Unsigned * dw_length,
05434     Dwarf_Half * dw_version,
05435     Dwarf_Small * dw_address_size,
05436     Dwarf_Unsigned * dw_at_addr_base,
05437     Dwarf_Unsigned * dw_entry_count,
05438     Dwarf_Unsigned * dw_next_table_offset,
05439     Dwarf_Error * dw_error);

```

```

05440
05463 DW_API int dwarf_debug_addr_by_index(Dwarf_Debug_Addr_Table dw_dat,
05464     Dwarf_Unsigned dw_entry_index,
05465     Dwarf_Unsigned *dw_address,
05466     Dwarf_Error *dw_error);
05467
05475 DW_API void dwarf_dealloc_debug_addr_table(
05476     Dwarf_Debug_Addr_Table dw_dat);
05477
05514 DW_API int dwarf_get_macro_context(Dwarf_Die dw_die,
05515     Dwarf_Unsigned * dw_version_out,
05516     Dwarf_Macro_Context * dw_macro_context,
05517     Dwarf_Unsigned * dw_macro_unit_offset_out,
05518     Dwarf_Unsigned * dw_macro_ops_count_out,
05519     Dwarf_Unsigned * dw_macro_ops_data_length_out,
05520     Dwarf_Error * dw_error);
05521
05549 DW_API int dwarf_get_macro_context_by_offset(Dwarf_Die dw_die,
05550     Dwarf_Unsigned dw_offset,
05551     Dwarf_Unsigned * dw_version_out,
05552     Dwarf_Macro_Context * dw_macro_context,
05553     Dwarf_Unsigned * dw_macro_ops_count_out,
05554     Dwarf_Unsigned * dw_macro_ops_data_length,
05555     Dwarf_Error * dw_error);
05556
05557 /* New December 2020. libdwarf 0.1.0
05558     Sometimes its necessary to know
05559     a context total length including macro 5 header */
05572 DW_API int dwarf_macro_context_total_length(
05573     Dwarf_Macro_Context dw_context,
05574     Dwarf_Unsigned * dw_mac_total_len,
05575     Dwarf_Error * dw_error);
05576
05584 DW_API void dwarf_dealloc_macro_context(Dwarf_Macro_Context dw_mc);
05585
05591 DW_API int dwarf_macro_context_head(Dwarf_Macro_Context dw_mc,
05592     Dwarf_Half * dw_version,
05593     Dwarf_Unsigned * dw_mac_offset,
05594     Dwarf_Unsigned * dw_mac_len,
05595     Dwarf_Unsigned * dw_mac_header_len,
05596     unsigned int * dw_flags,
05597     Dwarf_Bool * dw_has_line_offset,
05598     Dwarf_Unsigned * dw_line_offset,
05599     Dwarf_Bool * dw_has_offset_size_64,
05600     Dwarf_Bool * dw_has_operands_table,
05601     Dwarf_Half * dw_opcode_count,
05602     Dwarf_Error * dw_error);
05603
05626 DW_API int dwarf_macro_operands_table(Dwarf_Macro_Context dw_mc,
05627     Dwarf_Half dw_index, /* 0 to opcode_count -1 */
05628     Dwarf_Half * dw_opcode_number,
05629     Dwarf_Half * dw_operand_count,
05630     const Dwarf_Small ** dw_operand_array,
05631     Dwarf_Error * dw_error);
05632
05657 DW_API int dwarf_get_macro_op(Dwarf_Macro_Context dw_macro_context,
05658     Dwarf_Unsigned dw_op_number,
05659     Dwarf_Unsigned * dw_op_start_section_offset,
05660     Dwarf_Half * dw_macro_operator,
05661     Dwarf_Half * dw_forms_count,
05662     const Dwarf_Small ** dw_formcode_array,
05663     Dwarf_Error * dw_error);
05664
05702 DW_API int dwarf_get_macro_defundef(
05703     Dwarf_Macro_Context dw_macro_context,
05704     Dwarf_Unsigned dw_op_number,
05705     Dwarf_Unsigned * dw_line_number,
05706     Dwarf_Unsigned * dw_index,
05707     Dwarf_Unsigned * dw_offset,
05708     Dwarf_Half * dw_forms_count,
05709     const char ** dw_macro_string,
05710     Dwarf_Error * dw_error);
05711
05739 DW_API int dwarf_get_macro_startend_file(
05740     Dwarf_Macro_Context dw_macro_context,
05741     Dwarf_Unsigned dw_op_number,
05742     Dwarf_Unsigned * dw_line_number,
05743     Dwarf_Unsigned * dw_name_index_to_line_tab,
05744     const char ** dw_src_file_name,
05745     Dwarf_Error * dw_error);
05746
05762 DW_API int dwarf_get_macro_import(
05763     Dwarf_Macro_Context dw_macro_context,
05764     Dwarf_Unsigned dw_op_number,
05765     Dwarf_Unsigned * dw_target_offset,
05766     Dwarf_Error * dw_error);
05795 DW_API char* dwarf_find_macro_value_start(char * dw_macro_string);

```

```

05796
05822 DW_API int dwarf_get_macro_details(Dwarf_Debug dw_dbg,
05823     Dwarf_Off      dw_macro_offset,
05824     Dwarf_Unsigned  dw_maximum_count,
05825     Dwarf_Signed    * dw_entry_count,
05826     Dwarf_Macro_Details ** dw_details,
05827     Dwarf_Error *    dw_error);
05828
05872 DW_API int dwarf_get_fde_list(Dwarf_Debug dw_dbg,
05873     Dwarf_Cie** dw_cie_data,
05874     Dwarf_Signed* dw_cie_element_count,
05875     Dwarf_Fde** dw_fde_data,
05876     Dwarf_Signed* dw_fde_element_count,
05877     Dwarf_Error* dw_error);
05887 DW_API int dwarf_get_fde_list_eh(Dwarf_Debug dw_dbg,
05888     Dwarf_Cie** dw_cie_data,
05889     Dwarf_Signed* dw_cie_element_count,
05890     Dwarf_Fde** dw_fde_data,
05891     Dwarf_Signed* dw_fde_element_count,
05892     Dwarf_Error* dw_error);
05893
05913 DW_API void dwarf_dealloc_fde_cie_list(Dwarf_Debug dw_dbg,
05914     Dwarf_Cie * dw_cie_data,
05915     Dwarf_Signed dw_cie_element_count,
05916     Dwarf_Fde * dw_fde_data,
05917     Dwarf_Signed dw_fde_element_count);
05918
05946 DW_API int dwarf_get_fde_range(Dwarf_Fde dw_fde,
05947     Dwarf_Addr* dw_low_pc,
05948     Dwarf_Unsigned* dw_func_length,
05949     Dwarf_Small **dw_fde_bytes,
05950     Dwarf_Unsigned* dw_fde_byte_length,
05951     Dwarf_Off* dw_cie_offset,
05952     Dwarf_Signed* dw_cie_index,
05953     Dwarf_Off* dw_fde_offset,
05954     Dwarf_Error* dw_error);
05955
05961 DW_API int dwarf_get_fde_exception_info(Dwarf_Fde dw_fde,
05962     Dwarf_Signed* dw_offset_into_exception_tables,
05963     Dwarf_Error* dw_error);
05964
05976 DW_API int dwarf_get_cie_of_fde(Dwarf_Fde dw_fde,
05977     Dwarf_Cie * dw_cie_returned,
05978     Dwarf_Error* dw_error);
05979
06013 DW_API int dwarf_get_cie_info_b(Dwarf_Cie dw_cie,
06014     Dwarf_Unsigned * dw_bytes_in_cie,
06015     Dwarf_Small* dw_version,
06016     char ** dw_augmenter,
06017     Dwarf_Unsigned* dw_code_alignment_factor,
06018     Dwarf_Signed* dw_data_alignment_factor,
06019     Dwarf_Half* dw_return_address_register_rule,
06020     Dwarf_Small ** dw_initial_instructions,
06021     Dwarf_Unsigned* dw_initial_instructions_length,
06022     Dwarf_Half* dw_offset_size,
06023     Dwarf_Error* dw_error);
06024
06037 DW_API int dwarf_get_cie_index(Dwarf_Cie dw_cie,
06038     Dwarf_Signed* dw_index,
06039     Dwarf_Error * dw_error);
06040
06059 DW_API int dwarf_get_fde_instr_bytes(Dwarf_Fde dw_fde,
06060     Dwarf_Small ** dw_outinstrs,
06061     Dwarf_Unsigned * dw_outlen,
06062     Dwarf_Error * dw_error);
06063
06084 typedef int (*dwarf_iterate_fde_callback_function_type) (
06085     Dwarf_Regtable3* dw_reg_table,
06086     Dwarf_Addr dw_row_pc,
06087     Dwarf_Bool dw_has_more_rows,
06088     Dwarf_Addr dw_subsequent_pc,
06089     void * dw_user_data);
06090
06122 DW_API int dwarf_iterate_fde_all_regs3(
06123     Dwarf_Fde dw_fde,
06124     Dwarf_Regtable3 *dw_reg_table,
06125     dwarf_iterate_fde_callback_function_type dw_callback,
06126     void *dw_callback_user_data,
06127     Dwarf_Error *dw_error);
06128
06166 DW_API int dwarf_get_fde_info_for_all_regs3_b(
06167     Dwarf_Fde dw_fde,
06168     Dwarf_Addr dw_pc_requested,
06169     Dwarf_Regtable3* dw_reg_table,
06170     Dwarf_Addr* dw_row_pc,
06171     Dwarf_Bool* dw_has_more_rows,
06172     Dwarf_Addr* dw_subsequent_pc,

```

```

06173     Dwarf_Error*      dw_error);
06174
06187 DW_API int dwarf_get_fde_info_for_all_regs3(Dwarf_Fde dw_fde,
06188     Dwarf_Addr      dw_pc_requested,
06189     Dwarf_Regtable3* dw_reg_table,
06190     Dwarf_Addr*      dw_row_pc,
06191     Dwarf_Error*      dw_error);
06192
06193 /* See discussion of dw_value_type, libdwarf.h. */
06264 DW_API int dwarf_get_fde_info_for_reg3_c(Dwarf_Fde dw_fde,
06265     Dwarf_Half      dw_table_column,
06266     Dwarf_Addr      dw_pc_requested,
06267     Dwarf_Small     * dw_value_type,
06268     Dwarf_Unsigned * dw_offset_relevant,
06269     Dwarf_Unsigned * dw_register,
06270     Dwarf_Signed   * dw_offset,
06271     Dwarf_Block    * dw_block_content,
06272     Dwarf_Addr     * dw_row_pc_out,
06273     Dwarf_Bool     * dw_has_more_rows,
06274     Dwarf_Addr     * dw_subsequent_pc,
06275     Dwarf_Error    * dw_error);
06276
06286 DW_API int dwarf_get_fde_info_for_reg3_b(Dwarf_Fde dw_fde,
06287     Dwarf_Half      dw_table_column,
06288     Dwarf_Addr      dw_pc_requested,
06289     Dwarf_Small     * dw_value_type,
06290     Dwarf_Unsigned * dw_offset_relevant,
06291     Dwarf_Unsigned * dw_register,
06292     Dwarf_Unsigned * dw_offset,
06293     Dwarf_Block    * dw_block_content,
06294     Dwarf_Addr     * dw_row_pc_out,
06295     Dwarf_Bool     * dw_has_more_rows,
06296     Dwarf_Addr     * dw_subsequent_pc,
06297     Dwarf_Error    * dw_error);
06298
06322 DW_API int dwarf_get_fde_info_for_cfa_reg3_c(Dwarf_Fde dw_fde,
06323     Dwarf_Addr      dw_pc_requested,
06324     Dwarf_Small     * dw_value_type,
06325     Dwarf_Unsigned* dw_offset_relevant,
06326     Dwarf_Unsigned* dw_register,
06327     Dwarf_Signed   * dw_offset,
06328     Dwarf_Block    * dw_block,
06329     Dwarf_Addr     * dw_row_pc_out,
06330     Dwarf_Bool     * dw_has_more_rows,
06331     Dwarf_Addr     * dw_subsequent_pc,
06332     Dwarf_Error    * dw_error);
06342 DW_API int dwarf_get_fde_info_for_cfa_reg3_b(Dwarf_Fde dw_fde,
06343     Dwarf_Addr      dw_pc_requested,
06344     Dwarf_Small     * dw_value_type,
06345     Dwarf_Unsigned* dw_offset_relevant,
06346     Dwarf_Unsigned* dw_register,
06347     Dwarf_Unsigned* dw_offset,
06348     Dwarf_Block    * dw_block,
06349     Dwarf_Addr     * dw_row_pc_out,
06350     Dwarf_Bool     * dw_has_more_rows,
06351     Dwarf_Addr     * dw_subsequent_pc,
06352     Dwarf_Error    * dw_error);
06353
06362 DW_API int dwarf_get_fde_for_die(Dwarf_Debug dw_dbg,
06363     Dwarf_Die      dw_subr_die,
06364     Dwarf_Fde      * dw_returned_fde,
06365     Dwarf_Error*    dw_error);
06366
06374 DW_API int dwarf_get_fde_n(Dwarf_Fde* dw_fde_data,
06375     Dwarf_Unsigned dw_fde_index,
06376     Dwarf_Fde      * dw_returned_fde,
06377     Dwarf_Error    * dw_error);
06378
06409 DW_API int dwarf_get_fde_at_pc(Dwarf_Fde* dw_fde_data,
06410     Dwarf_Addr      dw_pc_of_interest,
06411     Dwarf_Fde      * dw_returned_fde,
06412     Dwarf_Addr     * dw_lopc,
06413     Dwarf_Addr     * dw_hipc,
06414     Dwarf_Error*    dw_error);
06415
06435 DW_API int dwarf_get_cie_augmentation_data(Dwarf_Cie dw_cie,
06436     Dwarf_Small ** dw_augdata,
06437     Dwarf_Unsigned * dw_augdata_len,
06438     Dwarf_Error*    dw_error);
06439
06459 DW_API int dwarf_get_fde_augmentation_data(Dwarf_Fde dw_fde,
06460     Dwarf_Small ** dw_augdata,
06461     Dwarf_Unsigned * dw_augdata_len,
06462     Dwarf_Error*    dw_error);
06463
06497 DW_API int dwarf_expand_frame_instructions(Dwarf_Cie dw_cie,
06498     Dwarf_Small * dw_instructionspointer,

```

```

06499     Dwarf_Unsigned    dw_length_in_bytes,
06500     Dwarf_Frame_Instr_Head * dw_head,
06501     Dwarf_Unsigned    * dw_instr_count,
06502     Dwarf_Error        * dw_error);
06503
06576 DW_API int dwarf_get_frame_instruction(
06577     Dwarf_Frame_Instr_Head dw_head,
06578     Dwarf_Unsigned    dw_instr_index,
06579     Dwarf_Unsigned    * dw_instr_offset_in_instrs,
06580     Dwarf_Small       * dw_cfa_operation,
06581     const char        ** dw_fields_description,
06582     Dwarf_Unsigned    * dw_u0,
06583     Dwarf_Unsigned    * dw_u1,
06584     Dwarf_Signed      * dw_s0,
06585     Dwarf_Signed      * dw_s1,
06586     Dwarf_Unsigned    * dw_code_alignment_factor,
06587     Dwarf_Signed      * dw_data_alignment_factor,
06588     Dwarf_Block       * dw_expression_block,
06589     Dwarf_Error        * dw_error);
06590
06612 DW_API int dwarf_get_frame_instruction_a(
06613     Dwarf_Frame_Instr_Head dw_/* head*/,
06614     Dwarf_Unsigned    dw_instr_index,
06615     Dwarf_Unsigned    * dw_instr_offset_in_instrs,
06616     Dwarf_Small       * dw_cfa_operation,
06617     const char        ** dw_fields_description,
06618     Dwarf_Unsigned    * dw_u0,
06619     Dwarf_Unsigned    * dw_u1,
06620     Dwarf_Unsigned    * dw_u2,
06621     Dwarf_Signed      * dw_s0,
06622     Dwarf_Signed      * dw_s1,
06623     Dwarf_Unsigned    * dw_code_alignment_factor,
06624     Dwarf_Signed      * dw_data_alignment_factor,
06625     Dwarf_Block       * dw_expression_block,
06626     Dwarf_Error        * dw_error);
06627
06636 DW_API void dwarf_dealloc_frame_instr_head(Dwarf_Frame_Instr_Head
06637     dw_head);
06638
06655 DW_API int dwarf_fde_section_offset(Dwarf_Debug dw_dbg,
06656     Dwarf_Fde        dw_in_fde,
06657     Dwarf_Off         * dw_fde_off,
06658     Dwarf_Off         * dw_cie_off,
06659     Dwarf_Error        * dw_error);
06660
06675 DW_API int dwarf_cie_section_offset(Dwarf_Debug dw_dbg,
06676     Dwarf_Cie        dw_in_cie,
06677     Dwarf_Off         * dw_cie_off,
06678     Dwarf_Error        * dw_error);
06679
06697 DW_API Dwarf_Half dwarf_set_frame_rule_table_size(
06698     Dwarf_Debug dw_dbg,
06699     Dwarf_Half dw_value);
06713 DW_API Dwarf_Half dwarf_set_frame_rule_initial_value(
06714     Dwarf_Debug dw_dbg,
06715     Dwarf_Half dw_value);
06727 DW_API Dwarf_Half dwarf_set_frame_cfa_value(
06728     Dwarf_Debug dw_dbg,
06729     Dwarf_Half dw_value);
06730
06742 DW_API Dwarf_Half dwarf_set_frame_same_value(
06743     Dwarf_Debug dw_dbg,
06744     Dwarf_Half dw_value);
06756 DW_API Dwarf_Half dwarf_set_frame_undefined_value(
06757     Dwarf_Debug dw_dbg,
06758     Dwarf_Half dw_value);
06812 DW_API int dwarf_get_abbrev(Dwarf_Debug dw_dbg,
06813     Dwarf_Unsigned    dw_offset,
06814     Dwarf_Abbrev      * dw_returned_abbrev,
06815     Dwarf_Unsigned* dw_length,
06816     Dwarf_Unsigned* dw_attr_count,
06817     Dwarf_Error* dw_error);
06818
06830 DW_API int dwarf_get_abbrev_tag(Dwarf_Abbrev dw_abbrev,
06831     Dwarf_Half* dw_return_tag_number,
06832     Dwarf_Error* dw_error);
06833
06847 DW_API int dwarf_get_abbrev_code(Dwarf_Abbrev dw_abbrev,
06848     Dwarf_Unsigned* dw_return_code_number,
06849     Dwarf_Error* dw_error);
06850
06864 DW_API int dwarf_get_abbrev_children_flag(Dwarf_Abbrev dw_abbrev,
06865     Dwarf_Signed* dw_return_flag,
06866     Dwarf_Error* dw_error);
06867
06901 DW_API int dwarf_get_abbrev_entry_b(Dwarf_Abbrev dw_abbrev,
06902     Dwarf_Unsigned    dw_indx,

```



```

06903     Dwarf_Bool      dw_filter_outliers,
06904     Dwarf_Unsigned * dw_returned_attr_num,
06905     Dwarf_Unsigned * dw_returned_form,
06906     Dwarf_Signed   * dw_returned_implicit_const,
06907     Dwarf_Off      * dw_offset,
06908     Dwarf_Error    * dw_error);
06909
06943 DW_API int dwarf_get_str(Dwarf_Debug dw_dbg,
06944     Dwarf_Off      dw_offset,
06945     char**         dw_string,
06946     Dwarf_Signed * dw_strlen_of_string,
06947     Dwarf_Error*   dw_error);
06948
06959 /* Allows applications to print the .debug_str_offsets
06960    section.
06961    Beginning at starting_offset zero,
06962    returns data about the first table found.
06963    The value *next_table_offset is the value
06964    of the next table (if any), one byte past
06965    the end of the table whose data is returned..
06966    Returns DW_DLV_NO_ENTRY if the starting offset
06967    is past the end of valid data.
06968
06969    There is no guarantee that there are no non-0 nonsense
06970    bytes in the section outside of useful tables,
06971    so this can fail and return nonsense or
06972    DW_DLV_ERROR if such garbage exists.
06973 */
06974
06991 DW_API int dwarf_open_str_offsets_table_access(Dwarf_Debug dw_dbg,
06992     Dwarf_Str_Offsets_Table * dw_table_data,
06993     Dwarf_Error             * dw_error);
06994
07012 DW_API int dwarf_close_str_offsets_table_access(
07013     Dwarf_Str_Offsets_Table dw_table_data,
07014     Dwarf_Error             * dw_error);
07015
07049 DW_API int dwarf_next_str_offsets_table(
07050     Dwarf_Str_Offsets_Table dw_table_data,
07051     Dwarf_Unsigned * dw_unit_length,
07052     Dwarf_Unsigned * dw_unit_length_offset,
07053     Dwarf_Unsigned * dw_table_start_offset,
07054     Dwarf_Half     * dw_entry_size,
07055     Dwarf_Half     * dw_version,
07056     Dwarf_Half     * dw_padding,
07057     Dwarf_Unsigned * dw_table_value_count,
07058     Dwarf_Error    * dw_error);
07059
07079 DW_API int dwarf_str_offsets_value_by_index(
07080     Dwarf_Str_Offsets_Table dw_table_data,
07081     Dwarf_Unsigned          dw_index_to_entry,
07082     Dwarf_Unsigned * dw_entry_value,
07083     Dwarf_Error    * dw_error);
07084
07102 DW_API int dwarf_str_offsets_statistics(
07103     Dwarf_Str_Offsets_Table dw_table_data,
07104     Dwarf_Unsigned * dw_wasted_byte_count,
07105     Dwarf_Unsigned * dw_table_count,
07106     Dwarf_Error    * dw_error);
07107
07119 DW_API Dwarf_Unsigned dwarf_errno(Dwarf_Error dw_error);
07126 DW_API char* dwarf_errmsg(Dwarf_Error dw_error);
07134 DW_API char* dwarf_errmsg_by_number(Dwarf_Unsigned dw_errnum);
07135
07149 DW_API void dwarf_error_creation(Dwarf_Debug dw_dbg ,
07150     Dwarf_Error * dw_error, char * dw_errmsg);
07151
07160 DW_API void dwarf_dealloc_error(Dwarf_Debug dw_dbg,
07161     Dwarf_Error dw_error);
07203 DW_API void dwarf_dealloc(Dwarf_Debug dw_dbg,
07204     void* dw_space, Dwarf_Unsigned dw_type);
07224 DW_API int dwarf_get_debug_sup(Dwarf_Debug dw_dbg,
07225     Dwarf_Half * dw_version,
07226     Dwarf_Small * dw_is_supplementary,
07227     char ** dw_filename,
07228     Dwarf_Unsigned * dw_checksum_len,
07229     Dwarf_Small ** dw_checksum,
07230     Dwarf_Error * dw_error);
07266 DW_API int dwarf_dnames_header(Dwarf_Debug dw_dbg,
07267     Dwarf_Off      dw_starting_offset,
07268     Dwarf_Dnames_Head * dw_dn,
07269     Dwarf_Off      * dw_offset_of_next_table,
07270     Dwarf_Error * dw_error);
07271
07279 DW_API void dwarf_dealloc_dnames(Dwarf_Dnames_Head dw_dn);
07280
07325 DW_API int dwarf_dnames_abbrevtable(Dwarf_Dnames_Head dw_dn,

```

```

07326 Dwarf_Unsigned dw_index,
07327 Dwarf_Unsigned *dw_abbrev_offset,
07328 Dwarf_Unsigned *dw_abbrev_code,
07329 Dwarf_Unsigned *dw_abbrev_tag,
07330 Dwarf_Unsigned dw_array_size,
07331 Dwarf_Half *dw_idxattr_array,
07332 Dwarf_Half *dw_form_array,
07333 Dwarf_Unsigned *dw_idxattr_count);
07334
07352 DW_API int dwarf_dnames_sizes(Dwarf_Dnames_Head dw_dn,
07353 Dwarf_Unsigned * dw_comp_unit_count,
07354 Dwarf_Unsigned * dw_local_type_unit_count,
07355 Dwarf_Unsigned * dw_foreign_type_unit_count,
07356 Dwarf_Unsigned * dw_bucket_count,
07357 Dwarf_Unsigned * dw_name_count,
07358 /* The following are counted in bytes */
07359 Dwarf_Unsigned * dw_abbrev_table_size,
07360 Dwarf_Unsigned * dw_entry_pool_size,
07361 Dwarf_Unsigned * dw_augmentation_string_size,
07362 char ** dw_augmentation_string,
07363 Dwarf_Unsigned * dw_section_size,
07364 Dwarf_Half * dw_table_version,
07365 Dwarf_Half * dw_offset_size,
07366 Dwarf_Error * dw_error);
07367
07378 DW_API int dwarf_dnames_offsets(Dwarf_Dnames_Head dw_dn,
07379 Dwarf_Unsigned * dw_header_offset,
07380 Dwarf_Unsigned * dw_cu_table_offset,
07381 Dwarf_Unsigned * dw_tu_local_offset,
07382 Dwarf_Unsigned * dw_foreign_tu_offset,
07383 Dwarf_Unsigned * dw_bucket_offset,
07384 Dwarf_Unsigned * dw_hashes_offset,
07385 Dwarf_Unsigned * dw_stringoffsets_offset,
07386 Dwarf_Unsigned * dw_entryoffsets_offset,
07387 Dwarf_Unsigned * dw_abbrev_table_offset,
07388 Dwarf_Unsigned * dw_entry_pool_offset,
07389 Dwarf_Error * dw_error);
07390
07419 DW_API int dwarf_dnames_cu_table(Dwarf_Dnames_Head dw_dn,
07420 const char * dw_type,
07421 Dwarf_Unsigned dw_index_number,
07422 Dwarf_Unsigned * dw_offset,
07423 Dwarf_Sig8 * dw_sig,
07424 Dwarf_Error * dw_error);
07425
07447 DW_API int dwarf_dnames_bucket(Dwarf_Dnames_Head dw_dn,
07448 Dwarf_Unsigned dw_bucket_number,
07449 Dwarf_Unsigned * dw_index,
07450 Dwarf_Unsigned * dw_indexcount,
07451 Dwarf_Error * dw_error);
07452
07502 DW_API int dwarf_dnames_name(Dwarf_Dnames_Head dw_dn,
07503 Dwarf_Unsigned dw_name_index,
07504 Dwarf_Unsigned * dw_bucket_number,
07505 Dwarf_Unsigned * dw_hash_value,
07506 Dwarf_Unsigned * dw_offset_to_debug_str,
07507 char * dw_ptrtostr,
07508 Dwarf_Unsigned * dw_offset_in_entrypool,
07509 Dwarf_Unsigned * dw_abbrev_number,
07510 Dwarf_Half * dw_abbrev_tag,
07511 Dwarf_Unsigned dw_array_size,
07512 Dwarf_Half * dw_idxattr_array,
07513 Dwarf_Half * dw_form_array,
07514 Dwarf_Unsigned * dw_idxattr_count,
07515 Dwarf_Error * dw_error);
07516
07558 DW_API int dwarf_dnames_entrypool(Dwarf_Dnames_Head dw_dn,
07559 Dwarf_Unsigned dw_offset_in_entrypool,
07560 Dwarf_Unsigned * dw_abbrev_code,
07561 Dwarf_Half * dw_tag,
07562 Dwarf_Unsigned * dw_value_count,
07563 Dwarf_Unsigned * dw_index_of_abbrev,
07564 Dwarf_Unsigned * dw_offset_of_initial_value,
07565 Dwarf_Error * dw_error);
07566
07626 DW_API int dwarf_dnames_entrypool_values(Dwarf_Dnames_Head dw_dn,
07627 Dwarf_Unsigned dw_index_of_abbrev,
07628 Dwarf_Unsigned dw_offset_in_entrypool_of_values,
07629 Dwarf_Unsigned dw_arrays_length,
07630 Dwarf_Half *dw_array_idx_number,
07631 Dwarf_Half *dw_array_form,
07632 Dwarf_Unsigned *dw_array_of_offsets,
07633 Dwarf_Sig8 *dw_array_of_signatures,
07634 Dwarf_Bool *dw_single_cu,
07635 Dwarf_Unsigned *dw_cu_offset,
07636 Dwarf_Unsigned *dw_offset_of_next_entrypool,
07637 Dwarf_Error *dw_error);

```

```

07638
07665 DW_API int dwarf_get_aranges(Dwarf_Debug dw_dbg,
07666     Dwarf_Arange** dw_aranges,
07667     Dwarf_Signed * dw_arange_count,
07668     Dwarf_Error* dw_error);
07669
07689 DW_API int dwarf_get_arange(Dwarf_Arange* dw_aranges,
07690     Dwarf_Unsigned dw_arange_count,
07691     Dwarf_Addr dw_address,
07692     Dwarf_Arange * dw_returned_arange,
07693     Dwarf_Error* dw_error);
07694
07707 DW_API int dwarf_get_cu_die_offset(Dwarf_Arange dw_arange,
07708     Dwarf_Off * dw_return_offset,
07709     Dwarf_Error* dw_error);
07710
07723 DW_API int dwarf_get_arange_cu_header_offset(Dwarf_Arange dw_arange,
07724     Dwarf_Off * dw_return_cu_header_offset,
07725     Dwarf_Error* dw_error);
07726
07752 DW_API int dwarf_get_arange_info_b(Dwarf_Arange dw_arange,
07753     Dwarf_Unsigned* dw_segment,
07754     Dwarf_Unsigned* dw_segment_entry_size,
07755     Dwarf_Addr * dw_start,
07756     Dwarf_Unsigned* dw_length,
07757     Dwarf_Off * dw_cu_die_offset,
07758     Dwarf_Error * dw_error );
07807 DW_API int dwarf_get_globals(Dwarf_Debug dw_dbg,
07808     Dwarf_Global** dw_globals,
07809     Dwarf_Signed * dw_number_of_globals,
07810     Dwarf_Error * dw_error);
07811
07812 #define DW_GL_GLOBALS 0 /* .debug_pubnames and .debug_names */
07813 #define DW_GL_PUBTYPES 1 /* .debug_pubtypes */
07814 /* the following are IRIX ONLY */
07815 #define DW_GL_FUNCS 2 /* .debug_funcnames */
07816 #define DW_GL_TYPES 3 /* .debug_tynames */
07817 #define DW_GL_VARS 4 /* .debug_varnames */
07818 #define DW_GL_WEAKS 5 /* .debug_weaknames */
07841 DW_API int dwarf_get_pubtypes(Dwarf_Debug dw_dbg,
07842     Dwarf_Global** dw_pubtypes,
07843     Dwarf_Signed * dw_number_of_pubtypes,
07844     Dwarf_Error * dw_error);
07845
07871 DW_API int dwarf_globals_by_type(Dwarf_Debug dw_dbg,
07872     int dw_requested_section,
07873     Dwarf_Global **dw_contents,
07874     Dwarf_Signed *dw_count,
07875     Dwarf_Error *dw_error);
07876
07887 DW_API void dwarf_globals_dealloc(Dwarf_Debug dw_dbg,
07888     Dwarf_Global* dw_global_like,
07889     Dwarf_Signed dw_count);
07890
07903 DW_API int dwarf_globname(Dwarf_Global dw_global,
07904     char ** dw_returned_name,
07905     Dwarf_Error* dw_error);
07906
07919 DW_API int dwarf_global_die_offset(Dwarf_Global dw_global,
07920     Dwarf_Off * dw_die_offset,
07921     Dwarf_Error * dw_error);
07922
07937 DW_API int dwarf_global_cu_offset(Dwarf_Global dw_global,
07938     Dwarf_Off* dw_cu_header_offset,
07939     Dwarf_Error* dw_error);
07940
07959 DW_API int dwarf_global_name_offsets(Dwarf_Global dw_global,
07960     char ** dw_returned_name,
07961     Dwarf_Off* dw_die_offset,
07962     Dwarf_Off* dw_cu_die_offset,
07963     Dwarf_Error* dw_error);
07964
07977 DW_API Dwarf_Half dwarf_global_tag_number(Dwarf_Global dw_global);
07978
07989 DW_API int dwarf_get_globals_header(Dwarf_Global dw_global,
07990     int * dw_category, /* DW_GL_GLOBAL for example */
07991     Dwarf_Off * dw_offset_pub_header,
07992     Dwarf_Unsigned * dw_length_size,
07993     Dwarf_Unsigned * dw_length_pub,
07994     Dwarf_Unsigned * dw_version,
07995     Dwarf_Unsigned * dw_header_info_offset,
07996     Dwarf_Unsigned * dw_info_length,
07997     Dwarf_Error * dw_error);
07998
08021 DW_API int dwarf_return_empty_pubnames(Dwarf_Debug dw_dbg,
08022     int dw_flag);
08023

```

```

08057 DW_API int dwarf_get_gnu_index_head(Dwarf_Debug dw_dbg,
08058     Dwarf_Bool dw_which_section,
08059     Dwarf_Gnu_Index_Head *dw_head,
08060     Dwarf_Unsigned *dw_index_block_count_out,
08061     Dwarf_Error *dw_error);
08069 DW_API void dwarf_gnu_index_dealloc(Dwarf_Gnu_Index_Head dw_head);
08108 DW_API int dwarf_get_gnu_index_block(Dwarf_Gnu_Index_Head dw_head,
08109     Dwarf_Unsigned dw_number,
08110     Dwarf_Unsigned *dw_block_length,
08111     Dwarf_Half *dw_version,
08112     Dwarf_Unsigned *dw_offset_into_debug_info,
08113     Dwarf_Unsigned *dw_size_of_debug_info_area,
08114     Dwarf_Unsigned *dw_count_of_index_entries,
08115     Dwarf_Error *dw_error);
08116
08148 DW_API int dwarf_get_gnu_index_block_entry(
08149     Dwarf_Gnu_Index_Head dw_head,
08150     Dwarf_Unsigned dw_blocknumber,
08151     Dwarf_Unsigned dw_entrynumber,
08152     Dwarf_Unsigned *dw_offset_in_debug_info,
08153     const char **dw_name_string,
08154     unsigned char *dw_flagbyte,
08155     unsigned char *dw_staticorglobal,
08156     unsigned char *dw_typeofentry,
08157     Dwarf_Error *dw_error);
08158
08219 DW_API int dwarf_gdbindex_header(Dwarf_Debug dw_dbg,
08220     Dwarf_Gdbindex *dw_gdbindexptr,
08221     Dwarf_Unsigned *dw_version,
08222     Dwarf_Unsigned *dw_cu_list_offset,
08223     Dwarf_Unsigned *dw_types_cu_list_offset,
08224     Dwarf_Unsigned *dw_address_area_offset,
08225     Dwarf_Unsigned *dw_symbol_table_offset,
08226     Dwarf_Unsigned *dw_constant_pool_offset,
08227     Dwarf_Unsigned *dw_section_size,
08228     const char **dw_section_name,
08229     Dwarf_Error *dw_error);
08230
08238 DW_API void dwarf_dealloc_gdbindex(Dwarf_Gdbindex dw_gdbindexptr);
08239
08250 DW_API int dwarf_gdbindex_culist_array(
08251     Dwarf_Gdbindex dw_gdbindexptr,
08252     Dwarf_Unsigned *dw_list_length,
08253     Dwarf_Error *dw_error);
08254
08272 DW_API int dwarf_gdbindex_culist_entry(
08273     Dwarf_Gdbindex dw_gdbindexptr,
08274     Dwarf_Unsigned dw_entryindex,
08275     Dwarf_Unsigned *dw_cu_offset,
08276     Dwarf_Unsigned *dw_cu_length,
08277     Dwarf_Error *dw_error);
08278
08290 DW_API int dwarf_gdbindex_types_culist_array(
08291     Dwarf_Gdbindex dw_gdbindexptr,
08292     Dwarf_Unsigned *dw_types_list_length,
08293     Dwarf_Error *dw_error);
08294
08295 /* entryindex: 0 to types_list_length -1 */
08317 DW_API int dwarf_gdbindex_types_culist_entry(
08318     Dwarf_Gdbindex dw_gdbindexptr,
08319     Dwarf_Unsigned dw_types_entryindex,
08320     Dwarf_Unsigned *dw_cu_offset,
08321     Dwarf_Unsigned *dw_tu_offset,
08322     Dwarf_Unsigned *dw_type_signature,
08323     Dwarf_Error *dw_error);
08324
08339 DW_API int dwarf_gdbindex_addressarea(
08340     Dwarf_Gdbindex dw_gdbindexptr,
08341     Dwarf_Unsigned *dw_addressarea_list_length,
08342     Dwarf_Error *dw_error);
08343
08362 DW_API int dwarf_gdbindex_addressarea_entry(
08363     Dwarf_Gdbindex dw_gdbindexptr,
08364     Dwarf_Unsigned dw_entryindex,
08365     Dwarf_Unsigned *dw_low_address,
08366     Dwarf_Unsigned *dw_high_address,
08367     Dwarf_Unsigned *dw_cu_index,
08368     Dwarf_Error *dw_error);
08369
08382 DW_API int dwarf_gdbindex_symboltable_array(
08383     Dwarf_Gdbindex dw_gdbindexptr,
08384     Dwarf_Unsigned *dw_symtab_list_length,
08385     Dwarf_Error *dw_error);
08386
08406 DW_API int dwarf_gdbindex_symboltable_entry(
08407     Dwarf_Gdbindex dw_gdbindexptr,
08408     Dwarf_Unsigned dw_entryindex,

```

```

08409     Dwarf_Unsigned * dw_string_offset,
08410     Dwarf_Unsigned * dw_cu_vector_offset,
08411     Dwarf_Error * dw_error);
08412
08430 DW_API int dwarf_gdbindex_cuvector_length(
08431     Dwarf_Gdbindex dw_gdbindexptr,
08432     Dwarf_Unsigned dw_cuvector_offset,
08433     Dwarf_Unsigned * dw_innercount,
08434     Dwarf_Error * dw_error);
08435
08452 DW_API int dwarf_gdbindex_cuvector_inner_attributes(
08453     Dwarf_Gdbindex dw_gdbindexptr,
08454     Dwarf_Unsigned dw_cuvector_offset_in,
08455     Dwarf_Unsigned dw_innerindex,
08456     Dwarf_Unsigned * dw_field_value,
08457     Dwarf_Error * dw_error);
08458
08481 DW_API int dwarf_gdbindex_cuvector_instance_expand_value(
08482     Dwarf_Gdbindex dw_gdbindexptr,
08483     Dwarf_Unsigned dw_field_value,
08484     Dwarf_Unsigned * dw_cu_index,
08485     Dwarf_Unsigned * dw_symbol_kind,
08486     Dwarf_Unsigned * dw_is_static,
08487     Dwarf_Error * dw_error);
08488
08504 DW_API int dwarf_gdbindex_string_by_offset(
08505     Dwarf_Gdbindex dw_gdbindexptr,
08506     Dwarf_Unsigned dw_stringoffset,
08507     const char ** dw_string_ptr,
08508     Dwarf_Error * dw_error);
08549 DW_API int dwarf_get_xu_index_header(Dwarf_Debug dw_dbg,
08550     const char * dw_section_type, /* "tu" or "cu" */
08551     Dwarf_Xu_Index_Header * dw_xuhdr,
08552     Dwarf_Unsigned * dw_version_number,
08553     Dwarf_Unsigned * dw_section_count,
08554     Dwarf_Unsigned * dw_units_count,
08555     Dwarf_Unsigned * dw_hash_slots_count,
08556     const char ** dw_sect_name,
08557     Dwarf_Error * dw_error);
08558
08567 DW_API void dwarf_dealloc_xu_header(Dwarf_Xu_Index_Header dw_xuhdr);
08568
08583 DW_API int dwarf_get_xu_index_section_type(
08584     Dwarf_Xu_Index_Header dw_xuhdr,
08585     const char ** dw_typename,
08586     const char ** dw_sectionname,
08587     Dwarf_Error * dw_error);
08588
08620 DW_API int dwarf_get_xu_hash_entry(Dwarf_Xu_Index_Header dw_xuhdr,
08621     Dwarf_Unsigned dw_index,
08622     Dwarf_Sig8 * dw_hash_value,
08623     Dwarf_Unsigned * dw_index_to_sections,
08624     Dwarf_Error * dw_error);
08625
08626 /* Columns 0 to L-1, valid. */
08649 DW_API int dwarf_get_xu_section_names(Dwarf_Xu_Index_Header dw_xuhdr,
08650     Dwarf_Unsigned dw_column_index,
08651     Dwarf_Unsigned* dw_SECT_number,
08652     const char ** dw_SECT_name,
08653     Dwarf_Error * dw_error);
08654
08683 DW_API int dwarf_get_xu_section_offset(
08684     Dwarf_Xu_Index_Header dw_xuhdr,
08685     Dwarf_Unsigned dw_row_index,
08686     Dwarf_Unsigned dw_column_index,
08687     Dwarf_Unsigned* dw_sec_offset,
08688     Dwarf_Unsigned* dw_sec_size,
08689     Dwarf_Error * dw_error);
08690
08712 DW_API int dwarf_get_debugfission_for_die(Dwarf_Die dw_die,
08713     Dwarf_Debug_Fission_Per_CU * dw_percu_out,
08714     Dwarf_Error * dw_error);
08715
08733 DW_API int dwarf_get_debugfission_for_key(Dwarf_Debug dw_dbg,
08734     Dwarf_Sig8 * dw_hash_sig,
08735     const char * dw_cu_type,
08736     Dwarf_Debug_Fission_Per_CU * dw_percu_out,
08737     Dwarf_Error * dw_error);
08738
08739 /* END debugfission dwp .debug_cu_index
08740 and .debug_tu_index meaningful operations. */
08741
08835 DW_API int dwarf_gnu_debuglink(Dwarf_Debug dw_dbg,
08836     char ** dw_debuglink_path_returned,
08837     unsigned char ** dw_crc_returned,
08838     char ** dw_debuglink_fullpath_returned,
08839     unsigned int * dw_debuglink_path_length_returned,

```

```
08840     unsigned int    * dw_buildid_type_returned,
08841     char            ** dw_buildid_owner_name_returned,
08842     unsigned char ** dw_buildid_returned,
08843     unsigned int    * dw_buildid_length_returned,
08844     char            *** dw_paths_returned,
08845     unsigned int    * dw_paths_length_returned,
08846     Dwarf_Error*     dw_error);
08847
08880 DW_API int dwarf_suppress_debuglink_crc(int dw_suppress);
08881
08900 DW_API int dwarf_add_debuglink_global_path(Dwarf_Debug dw_dbg,
08901     const char * dw_pathname,
08902     Dwarf_Error* dw_error);
08903
08931 DW_API int dwarf_crc32(Dwarf_Debug dw_dbg,
08932     unsigned char * dw_crcbuf,
08933     Dwarf_Error * dw_error);
08934
08958 DW_API unsigned int dwarf_basic_crc32(const unsigned char * dw_buf,
08959     unsigned long dw_len,
08960     unsigned int dw_init);
08979 #define DW_HARMLESS_ERROR_CIRCULAR_LIST_DEFAULT_SIZE 4
08980
09023 DW_API int dwarf_get_harmless_error_list(Dwarf_Debug dw_dbg,
09024     unsigned int dw_count,
09025     const char ** dw_errmsg_ptrs_array,
09026     unsigned int * dw_newerr_count);
09027
09048 DW_API unsigned int dwarf_set_harmless_error_list_size(
09049     Dwarf_Debug dw_dbg,
09050     unsigned int dw_maxcount);
09051
09069 DW_API int dwarf_set_harmless_errors_enabled(Dwarf_Debug dw_dbg,
09070     int dw_v);
09071
09083 DW_API void dwarf_insert_harmless_error(Dwarf_Debug dw_dbg,
09084     char * dw_newerror);
09120 DW_API int dwarf_get_ACCESS_name(unsigned int dw_val_in,
09121     const char ** dw_s_out);
09124 DW_API int dwarf_get_ADDR_name(unsigned int dw_val_in,
09125     const char ** dw_s_out);
09128 DW_API int dwarf_get_AT_name(unsigned int dw_val_in,
09129     const char ** dw_s_out);
09132 DW_API int dwarf_get_ATCF_name(unsigned int dw_val_in,
09133     const char ** dw_s_out);
09136 DW_API int dwarf_get_ATE_name(unsigned int dw_val_in,
09137     const char ** dw_s_out);
09140 DW_API int dwarf_get_CC_name(unsigned int dw_val_in,
09141     const char ** dw_s_out);
09144 DW_API int dwarf_get_CFA_name(unsigned int dw_val_in,
09145     const char ** dw_s_out);
09148 DW_API int dwarf_get_children_name(unsigned int dw_val_in,
09149     const char ** dw_s_out);
09152 DW_API int dwarf_get_CHILDREN_name(unsigned int dw_val_in,
09153     const char ** dw_s_out);
09156 DW_API int dwarf_get_DEFAULTED_name(unsigned int dw_val_in,
09157     const char ** dw_s_out);
09160 DW_API int dwarf_get_DS_name(unsigned int dw_val_in,
09161     const char ** dw_s_out);
09164 DW_API int dwarf_get_DSC_name(unsigned int dw_val_in,
09165     const char ** dw_s_out);
09170 DW_API int dwarf_get_GNUKIND_name(unsigned int dw_val_in,
09171     const char ** dw_s_out);
09176 DW_API int dwarf_get_EH_name(unsigned int dw_val_in,
09177     const char ** dw_s_out);
09180 DW_API int dwarf_get_END_name(unsigned int dw_val_in,
09181     const char ** dw_s_out);
09184 DW_API int dwarf_get_FORM_name(unsigned int dw_val_in,
09185     const char ** dw_s_out);
09192 DW_API int dwarf_get_FRAME_name(unsigned int dw_val_in,
09193     const char ** dw_s_out);
09198 DW_API int dwarf_get_GNUIVIS_name(unsigned int dw_val_in,
09199     const char ** dw_s_out);
09200
09203 DW_API int dwarf_get_ID_name(unsigned int dw_val_in,
09204     const char ** dw_s_out);
09207 DW_API int dwarf_get_IDX_name(unsigned int dw_val_in,
09208     const char ** dw_s_out);
09211 DW_API int dwarf_get_INL_name(unsigned int dw_val_in,
09212     const char ** dw_s_out);
09215 DW_API int dwarf_get_ISA_name(unsigned int dw_val_in,
09216     const char ** dw_s_out);
09219 DW_API int dwarf_get_LANG_name(unsigned int dw_val_in,
09220     const char ** dw_s_out);
09223 DW_API int dwarf_get_LLE_name(unsigned int dw_val_in,
09224     const char ** dw_s_out);
09230 DW_API int dwarf_get_LLEX_name(unsigned int dw_val_in,
```

```

09231     const char ** dw_s_out );
09232
09235 DW_API int dwarf_get_LNAME_name(unsigned int dw_val_in,
09236     const char ** dw_s_out);
09239 DW_API int dwarf_get_LNCT_name(unsigned int dw_val_in,
09240     const char ** dw_s_out);
09243 DW_API int dwarf_get_LNE_name(unsigned int dw_val_in,
09244     const char ** dw_s_out);
09247 DW_API int dwarf_get_LNS_name(unsigned int dw_val_in,
09248     const char ** dw_s_out);
09253 DW_API int dwarf_get_MACINFO_name(unsigned int dw_val_in,
09254     const char ** dw_s_out);
09259 DW_API int dwarf_get_MACRO_name(unsigned int dw_val_in,
09260     const char ** dw_s_out);
09263 DW_API int dwarf_get_OP_name(unsigned int dw_val_in,
09264     const char ** dw_s_out);
09267 DW_API int dwarf_get_ORD_name(unsigned int dw_val_in,
09268     const char ** dw_s_out);
09271 DW_API int dwarf_get_RLE_name(unsigned int dw_val_in,
09272     const char ** dw_s_out);
09275 DW_API int dwarf_get_SECT_name(unsigned int dw_val_in,
09276     const char ** dw_s_out);
09279 DW_API int dwarf_get_TAG_name(unsigned int dw_val_in,
09280     const char ** dw_s_out);
09283 DW_API int dwarf_get_UT_name(unsigned int dw_val_in,
09284     const char ** dw_s_out);
09287 DW_API int dwarf_get_VIRTUALITY_name(unsigned int dw_val_in,
09288     const char ** dw_s_out);
09291 DW_API int dwarf_get_VIS_name(unsigned int dw_val_in,
09292     const char ** dw_s_out);
09293
09304 DW_API int dwarf_get_FORM_CLASS_name(enum Dwarf_Form_Class dw_fc,
09305     const char ** dw_s_out);
09359 DW_API int dwarf_get_die_section_name(Dwarf_Debug dw_dbg,
09360     Dwarf_Bool dw_is_info,
09361     const char **dw_sec_name,
09362     Dwarf_Error *dw_error);
09363
09370 DW_API int dwarf_get_die_section_name_b(Dwarf_Die dw_die,
09371     const char ** dw_sec_name,
09372     Dwarf_Error * dw_error);
09373
09376 DW_API int dwarf_get_macro_section_name(Dwarf_Debug dw_dbg,
09377     const char ** dw_sec_name_out,
09378     Dwarf_Error * dw_err);
09379
09422 DW_API int dwarf_get_real_section_name(Dwarf_Debug dw_dbg,
09423     const char * dw_std_section_name,
09424     const char ** dw_actual_sec_name_out,
09425     Dwarf_Small * dw_marked_zcompressed,
09426     Dwarf_Small * dw_marked_zlib_compressed,
09427     Dwarf_Small * dw_marked_shf_compressed,
09428     Dwarf_Unsigned * dw_compressed_length,
09429     Dwarf_Unsigned * dw_uncompressed_length,
09430     Dwarf_Error * dw_error);
09431
09436 DW_API int dwarf_get_frame_section_name(Dwarf_Debug dw_dbg,
09437     const char ** dw_section_name_out,
09438     Dwarf_Error * dw_error);
09439
09445 DW_API int dwarf_get_frame_section_name_gh_gnu(Dwarf_Debug dw_dbg,
09446     const char ** dw_section_name_out,
09447     Dwarf_Error * dw_error);
09448
09452 DW_API int dwarf_get_aranges_section_name(Dwarf_Debug dw_dbg,
09453     const char ** dw_section_name_out,
09454     Dwarf_Error * dw_error);
09455
09459 DW_API int dwarf_get_ranges_section_name(Dwarf_Debug dw_dbg,
09460     const char ** dw_section_name_out,
09461     Dwarf_Error * dw_error);
09462
09463 /* These two get the offset or address size as defined
09464    by the object format (not by DWARF). */
09470 DW_API int dwarf_get_offset_size(Dwarf_Debug dw_dbg,
09471     Dwarf_Half * dw_offset_size,
09472     Dwarf_Error * dw_error);
09473
09479 DW_API int dwarf_get_address_size(Dwarf_Debug dw_dbg,
09480     Dwarf_Half * dw_addr_size,
09481     Dwarf_Error * dw_error);
09482
09486 DW_API int dwarf_get_string_section_name(Dwarf_Debug dw_dbg,
09487     const char ** dw_section_name_out,
09488     Dwarf_Error * dw_error);
09489
09493 DW_API int dwarf_get_line_section_name(Dwarf_Debug dw_dbg,

```

```
09494     const char ** dw_section_name_out,
09495     Dwarf_Error * dw_error);
09496
09510 DW_API int dwarf_get_line_section_name_from_die(Dwarf_Die dw_die,
09511     const char ** dw_section_name_out,
09512     Dwarf_Error * dw_error);
09513
09560 DW_API int dwarf_get_section_info_by_name_a(Dwarf_Debug dw_dbg,
09561     const char * dw_section_name,
09562     Dwarf_Addr * dw_section_addr,
09563     Dwarf_Unsigned* dw_section_size,
09564     Dwarf_Unsigned* dw_section_flags,
09565     Dwarf_Unsigned* dw_section_offset,
09566     Dwarf_Error * dw_error);
09567
09580 DW_API int dwarf_get_section_info_by_name(Dwarf_Debug dw_dbg,
09581     const char * dw_section_name,
09582     Dwarf_Addr * dw_section_addr,
09583     Dwarf_Unsigned* dw_section_size,
09584     Dwarf_Error * dw_error);
09585
09631 DW_API int dwarf_get_section_info_by_index_a(Dwarf_Debug dw_dbg,
09632     int dw_section_index,
09633     const char ** dw_section_name,
09634     Dwarf_Addr* dw_section_addr,
09635     Dwarf_Unsigned* dw_section_size,
09636     Dwarf_Unsigned* dw_section_flags,
09637     Dwarf_Unsigned* dw_section_offset,
09638     Dwarf_Error* dw_error);
09639
09652 DW_API int dwarf_get_section_info_by_index(Dwarf_Debug dw_dbg,
09653     int dw_section_index,
09654     const char ** dw_section_name,
09655     Dwarf_Addr* dw_section_addr,
09656     Dwarf_Unsigned* dw_section_size,
09657     Dwarf_Error* dw_error);
09658
09745 DW_API int dwarf_machine_architecture_a(Dwarf_Debug dw_dbg,
09746     Dwarf_Small *dw_ftype,
09747     Dwarf_Small *dw_obj_pointersize,
09748     Dwarf_Bool *dw_obj_is_big_endian,
09749     Dwarf_Unsigned *dw_obj_machine, /*Elf e_machine */
09750     Dwarf_Unsigned *dw_obj_type, /* Elf e_type */
09751     Dwarf_Unsigned *dw_obj_flags,
09752     Dwarf_Small *dw_path_source,
09753     Dwarf_Unsigned *dw_ub_offset,
09754     Dwarf_Unsigned *dw_ub_count,
09755     Dwarf_Unsigned *dw_ub_index,
09756     Dwarf_Unsigned *dw_comdat_groupnumber);
09757
09765 DW_API int dwarf_machine_architecture(Dwarf_Debug dw_dbg,
09766     Dwarf_Small *dw_ftype,
09767     Dwarf_Small *dw_obj_pointersize,
09768     Dwarf_Bool *dw_obj_is_big_endian,
09769     Dwarf_Unsigned *dw_obj_machine, /*architecture*/
09770     Dwarf_Unsigned *dw_obj_flags,
09771     Dwarf_Small *dw_path_source,
09772     Dwarf_Unsigned *dw_ub_offset,
09773     Dwarf_Unsigned *dw_ub_count,
09774     Dwarf_Unsigned *dw_ub_index,
09775     Dwarf_Unsigned *dw_comdat_groupnumber);
09776
09788 DW_API Dwarf_Unsigned dwarf_get_section_count(Dwarf_Debug dw_dbg);
09789
09808 DW_API int dwarf_get_section_max_offsets_d(Dwarf_Debug dw_dbg,
09809     Dwarf_Unsigned * dw_debug_info_size,
09810     Dwarf_Unsigned * dw_debug_abbrev_size,
09811     Dwarf_Unsigned * dw_debug_line_size,
09812     Dwarf_Unsigned * dw_debug_loc_size,
09813     Dwarf_Unsigned * dw_debug_aranges_size,
09814
09815     Dwarf_Unsigned * dw_debug_macinfo_size,
09816     Dwarf_Unsigned * dw_debug_pubnames_size,
09817     Dwarf_Unsigned * dw_debug_str_size,
09818     Dwarf_Unsigned * dw_debug_frame_size,
09819     Dwarf_Unsigned * dw_debug_ranges_size,
09820
09821     Dwarf_Unsigned * dw_debug_pubtypes_size,
09822     Dwarf_Unsigned * dw_debug_types_size,
09823     Dwarf_Unsigned * dw_debug_macro_size,
09824     Dwarf_Unsigned * dw_debug_str_offsets_size,
09825     Dwarf_Unsigned * dw_debug_sup_size,
09826
09827     Dwarf_Unsigned * dw_debug_cu_index_size,
09828     Dwarf_Unsigned * dw_debug_tu_index_size,
09829     Dwarf_Unsigned * dw_debug_names_size,
09830     Dwarf_Unsigned * dw_debug_loclists_size,
```



```

09831     Dwarf_Unsigned * dw_debug_rnglists_size);
09880 DW_API int dwarf_sec_group_sizes(Dwarf_Debug dw_dbg,
09881     Dwarf_Unsigned *dw_section_count_out,
09882     Dwarf_Unsigned *dw_group_count_out,
09883     Dwarf_Unsigned *dw_selected_group_out,
09884     Dwarf_Unsigned *dw_map_entry_count_out,
09885     Dwarf_Error *dw_error);
09886
09917 DW_API int dwarf_sec_group_map(Dwarf_Debug dw_dbg,
09918     Dwarf_Unsigned dw_map_entry_count,
09919     Dwarf_Unsigned *dw_group_numbers_array,
09920     Dwarf_Unsigned *dw_sec_numbers_array,
09921     const char **dw_sec_names_array,
09922     Dwarf_Error *dw_error);
09937 DW_API int dwarf_encode_leb128(Dwarf_Unsigned dw_val,
09938     int *dw_nbytes,
09939     char *dw_space,
09940     int dw_splen);
09941 DW_API int dwarf_encode_signed_leb128(Dwarf_Signed dw_val,
09942     int *dw_nbytes,
09943     char *dw_space,
09944     int dw_splen);
09945 /* Same for LEB decoding routines.
09946 caller sets endptr to an address one past the last valid
09947 address the library should be allowed to
09948 access. */
09949 DW_API int dwarf_decode_leb128(char *dw_leb,
09950     Dwarf_Unsigned *dw_leblen,
09951     Dwarf_Unsigned *dw_outval,
09952     char *dw_endptr);
09953 DW_API int dwarf_decode_signed_leb128(char *dw_leb,
09954     Dwarf_Unsigned *dw_leblen,
09955     Dwarf_Signed *dw_outval,
09956     char *dw_endptr);
09973 DW_API const char * dwarf_package_version(void);
09974
09990 DW_API int dwarf_set_stringcheck(int dw_stringcheck);
09991
10013 DW_API int dwarf_set_reloc_application(int dw_apply);
10014
10039 DW_API void (*dwarf_get_endian_copy_function(Dwarf_Debug dw_dbg))
10040     (void *, const void *, unsigned long);
10041
10042 /* A global flag in libdwarf. Applies to all Dwarf_Debug */
10043 DW_API extern Dwarf_Cmdline_Options dwarf_cmdline_options;
10044
10059 DW_API void dwarf_record_cmdline_options(
10060     Dwarf_Cmdline_Options dw_dd_options);
10061
10080 DW_API int dwarf_set_de_alloc_flag(int dw_v);
10081
10110 DW_API int dwarf_library_allow_dup_attr(int dw_v);
10111
10133 DW_API Dwarf_Small dwarf_set_default_address_size(
10134     Dwarf_Debug dw_dbg,
10135     Dwarf_Small dw_value);
10136
10162 DW_API int dwarf_get_universalbinary_count(
10163     Dwarf_Debug dw_dbg,
10164     Dwarf_Unsigned *dw_current_index,
10165     Dwarf_Unsigned *dw_available_count);
10166
10188 DW_API int dwarf_object_detector_path_b(const char * dw_path,
10189     char *dw_outpath_buffer,
10190     unsigned long dw_outpathlen,
10191     char ** dw_gl_pathnames,
10192     unsigned int dw_gl_pathcount,
10193     unsigned int *dw_ftype,
10194     unsigned int *dw_endian,
10195     unsigned int *dw_offsetsize,
10196     Dwarf_Unsigned *dw_filesize,
10197     unsigned char *dw_pathsource,
10198     int * dw_errcode);
10199
10200 /* Solely looks for dSYM */
10201 DW_API int dwarf_object_detector_path_dSYM(const char * dw_path,
10202     char * dw_outpath,
10203     unsigned long dw_outpath_len,
10204     char ** dw_gl_pathnames,
10205     unsigned int dw_gl_pathcount,
10206     unsigned int *dw_ftype,
10207     unsigned int *dw_endian,
10208     unsigned int *dw_offsetsize,
10209     Dwarf_Unsigned *dw_filesize,
10210     unsigned char *dw_pathsource,
10211     int * dw_errcode);
10212

```

```
10213 DW_API int dwarf_object_detector_fd(int dw_fd,
10214     unsigned int    *dw_ftype,
10215     unsigned int    *dw_endian,
10216     unsigned int    *dw_offsetsz,
10217     Dwarf_Unsigned  *dw_filesize,
10218     int             *dw_errcode);
10281 DW_API enum Dwarf_Sec_Alloc_Pref dwarf_set_load_preference(
10282     enum Dwarf_Sec_Alloc_Pref dw_load_preference);
10283
10323 DW_API int dwarf_get_mmap_count(Dwarf_Debug dw_dbg,
10324     Dwarf_Unsigned *dw_mmap_count,
10325     Dwarf_Unsigned *dw_mmap_size,
10326     Dwarf_Unsigned *dw_malloc_count,
10327     Dwarf_Unsigned *dw_malloc_size);
10330 #ifdef __cplusplus
10331 }
10332 #endif /* __cplusplus */
10333 #endif /* _LIBDWARF_H */
```

Index

- [.debug_addr access: DWARF5, 143](#)
 - [dwarf_dealloc_debug_addr_table, 144](#)
 - [dwarf_debug_addr_by_index, 144](#)
 - [dwarf_debug_addr_table, 145](#)
- [/home/davea/dwarf/code/src/bin/dwarfexample/jitreader.c, 311](#)
- [/home/davea/dwarf/code/src/bin/dwarfexample/showsectiongroups.c, 311](#)
- [/home/davea/dwarf/code/src/lib/libdwarf/dwarf.h, 313](#)
- [/home/davea/dwarf/code/src/lib/libdwarf/libdwarf.h, 333](#)
- [A Consumer Library Interface to DWARF, 1](#)
- [A simple report on section groups., 297](#)
- [Abbreviations Section Details, 173](#)
 - [dwarf_get_abbrev, 174](#)
 - [dwarf_get_abbrev_children_flag, 175](#)
 - [dwarf_get_abbrev_code, 175](#)
 - [dwarf_get_abbrev_entry_b, 175](#)
 - [dwarf_get_abbrev_tag, 176](#)
- [Access GNU .gnu_debuglink, build-id., 220](#)
 - [dwarf_add_debuglink_global_path, 221](#)
 - [dwarf_basic_crc32, 221](#)
 - [dwarf_crc32, 222](#)
 - [dwarf_gnu_debuglink, 222](#)
 - [dwarf_suppress_debuglink_crc, 223](#)
- [Access to Section .debug_sup, 184](#)
 - [dwarf_get_debug_sup, 184](#)
- [Accessing accessing raw rnglist, 290](#)
- [Accessing rnglists section, 291](#)
- [Attaching a tied dbg, 252](#)
- [Basic Library Datatypes Group, 41](#)
 - [Dwarf_Addr, 41](#)
 - [Dwarf_Bool, 41](#)
 - [Dwarf_Half, 41](#)
 - [Dwarf_Off, 42](#)
 - [Dwarf_Ptr, 42](#)
 - [Dwarf_Signed, 42](#)
 - [Dwarf_Small, 42](#)
 - [Dwarf_Unsigned, 42](#)
- [checkexamples.c, 33, 311](#)
- [Compilation Unit \(CU\) Access, 71](#)
 - [dwarf_child, 72](#)
 - [dwarf_cu_header_basics, 72](#)
 - [dwarf_dealloc_die, 73](#)
 - [dwarf_die_from_hash_signature, 73](#)
 - [dwarf_find_die_given_sig8, 74](#)
 - [dwarf_get_die_infotypes_flag, 74](#)
 - [dwarf_next_cu_header_d, 74](#)
 - [dwarf_next_cu_header_e, 75](#)
 - [dwarf_offdie_b, 76](#)
 - [dwarf_siblingof_b, 77](#)
 - [dwarf_siblingof_c, 78](#)
- [Debugging Information Entry \(DIE\) content, 78](#)
 - [dwarf_addr_form_is_indexed, 80](#)
 - [dwarf_arrayorder, 80](#)
 - [dwarf_attr, 81](#)
 - [dwarf_bitoffset, 81](#)
 - [dwarf_bitsize, 82](#)
 - [dwarf_bytesize, 82](#)
 - [dwarf_CU_dieoffset_given_die, 83](#)
 - [dwarf_debug_addr_index_to_addr, 83](#)
 - [dwarf_die_abbrev_children_flag, 84](#)
 - [dwarf_die_abbrev_code, 84](#)
 - [dwarf_die_abbrev_global_offset, 84](#)
 - [dwarf_die_CU_offset, 85](#)
 - [dwarf_die_CU_offset_range, 85](#)
 - [dwarf_die_offsets, 86](#)
 - [dwarf_die_text, 86](#)
 - [dwarf_diename, 87](#)
 - [dwarf_dieoffset, 87](#)
 - [dwarf_dietype_offset, 88](#)
 - [dwarf_get_cu_die_offset_given_cu_header_offset_b, 88](#)
 - [dwarf_get_die_address_size, 89](#)
 - [dwarf_get_version_of_die, 89](#)
 - [dwarf_hasattr, 90](#)
 - [dwarf_highpc_b, 90](#)
 - [dwarf_language_version_data, 91](#)
 - [dwarf_language_version_string, 91](#)
 - [dwarf_lowpc, 92](#)
 - [dwarf_lvn_name, 92](#)
 - [dwarf_lvn_name_direct, 92](#)
 - [dwarf_lvn_table_entry, 93](#)
 - [dwarf_offset_list, 94](#)
 - [dwarf_srclang, 94](#)
 - [dwarf_srclanglname, 95](#)
 - [dwarf_srclanglname_version, 95](#)
 - [dwarf_tag, 96](#)
 - [dwarf_validate_die_sibling, 96](#)
- [Default stack frame macros, 52](#)
- [Defined and Opaque Structs, 43](#)
 - [Dwarf_Abbrev, 45](#)
 - [Dwarf_Arange, 45](#)
 - [Dwarf_Attribute, 45](#)
 - [Dwarf_Block, 45](#)
 - [Dwarf_Cie, 45](#)
 - [Dwarf_Debug, 45](#)

- Dwarf_Debug_Addr_Table, [45](#)
- Dwarf_Debug_Fission_Per_CU, [46](#)
- Dwarf_Die, [46](#)
- Dwarf_Dnames_Head, [46](#)
- Dwarf_Dsc_Head, [46](#)
- Dwarf_Error, [46](#)
- Dwarf_Fde, [46](#)
- Dwarf_Form_Data16, [46](#)
- Dwarf_Frame_Instr_Head, [47](#)
- Dwarf_Func, [47](#)
- Dwarf_Gdbindex, [47](#)
- Dwarf_Global, [47](#)
- Dwarf_Gnu_Index_Head, [47](#)
- Dwarf_Handler, [47](#)
- Dwarf_Line, [47](#)
- Dwarf_Line_Context, [48](#)
- Dwarf_Loc_Head_c, [48](#)
- Dwarf_Locdesc_c, [48](#)
- Dwarf_Macro_Context, [48](#)
- Dwarf_Macro_Details, [48](#)
- Dwarf_Obj_Access_Interface_a, [48](#)
- Dwarf_Obj_Access_Methods_a, [48](#)
- Dwarf_Obj_Access_Section_a, [48](#)
- dwarf_printf_callback_function_type, [49](#)
- Dwarf_Ranges, [49](#)
- Dwarf_Regtable3, [49](#)
- Dwarf_Regtable_Entry3, [49](#)
- Dwarf_Rnglists_Head, [51](#)
- Dwarf_Sec_Alloc_Pref, [52](#)
- Dwarf_Section, [51](#)
- Dwarf_Sig8, [51](#)
- Dwarf_Str_Offsets_Table, [51](#)
- Dwarf_Type, [51](#)
- Dwarf_Var, [51](#)
- Dwarf_Weak, [52](#)
- Dwarf_Xu_Index_Header, [52](#)
- Demonstrating reading DWARF without a file., [292](#)
- Detaching a tied dbg, [253](#)
- Determine Object Type of a File, [247](#)
- DIE Attribute and Attribute-Form Details, [97](#)
 - dwarf_attr_offset, [99](#)
 - dwarf_attrlist, [99](#)
 - dwarf_convert_to_global_offset, [100](#)
 - dwarf_dealloc_attribute, [100](#)
 - dwarf_dealloc_uncompressed_block, [100](#)
 - dwarf_discr_entry_s, [101](#)
 - dwarf_discr_entry_u, [101](#)
 - dwarf_discr_list, [102](#)
 - dwarf_formaddr, [102](#)
 - dwarf_formblock, [103](#)
 - dwarf_formdata16, [103](#)
 - dwarf_formexprloc, [104](#)
 - dwarf_formflag, [104](#)
 - dwarf_formref, [105](#)
 - dwarf_formstdata, [105](#)
 - dwarf_formsig8, [106](#)
 - dwarf_formsig8_const, [106](#)
 - dwarf_formstring, [106](#)
 - dwarf_formudata, [107](#)
 - dwarf_get_debug_addr_index, [107](#)
 - dwarf_get_debug_str_index, [108](#)
 - dwarf_get_form_class, [108](#)
 - dwarf_global_formref, [109](#)
 - dwarf_global_formref_b, [109](#)
 - dwarf_hasform, [109](#)
 - dwarf_uncompress_integer_block_a, [110](#)
 - dwarf_whatattr, [110](#)
 - dwarf_whatform, [111](#)
 - dwarf_whatform_direct, [111](#)
- Documenting Form_Block, [262](#)
- DW_DLA alloc/dealloc typename&number, [53](#)
- DW_DLE Dwarf_Error numbers, [54](#)
 - DW_DLE_LAST, [63](#)
- DW_DLE_LAST
 - DW_DLE Dwarf_Error numbers, [63](#)
- dwarf.h, [29](#), [313](#)
- Dwarf_Abbrev
 - Defined and Opaque Structs, [45](#)
- dwarf_add_debuglink_global_path
 - Access GNU .gnu_debuglink, build-id., [221](#)
- Dwarf_Addr
 - Basic Library Datatypes Group, [41](#)
- dwarf_addr_form_is_indexed
 - Debugging Information Entry (DIE) content, [80](#)
- Dwarf_Arange
 - Defined and Opaque Structs, [45](#)
- dwarf_arrayorder
 - Debugging Information Entry (DIE) content, [80](#)
- dwarf_attr
 - Debugging Information Entry (DIE) content, [81](#)
- dwarf_attr_offset
 - DIE Attribute and Attribute-Form Details, [99](#)
- Dwarf_Attribute
 - Defined and Opaque Structs, [45](#)
- dwarf_attrlist
 - DIE Attribute and Attribute-Form Details, [99](#)
- dwarf_basic_crc32
 - Access GNU .gnu_debuglink, build-id., [221](#)
- dwarf_bitoffset
 - Debugging Information Entry (DIE) content, [81](#)
- dwarf_bitsize
 - Debugging Information Entry (DIE) content, [82](#)
- Dwarf_Block
 - Defined and Opaque Structs, [45](#)
- Dwarf_Block_s, [301](#)
- Dwarf_Bool
 - Basic Library Datatypes Group, [41](#)
- dwarf_bytesize
 - Debugging Information Entry (DIE) content, [82](#)
- dwarf_check_lineheader_b
 - Line Table For a CU, [114](#)
- dwarf_child
 - Compilation Unit (CU) Access, [72](#)
- Dwarf_Cie
 - Defined and Opaque Structs, [45](#)
- dwarf_cie_section_offset

- Stack Frame Access, [157](#)
- `dwarf_close_str_offsets_table_access`
 - Str_Offsets section details, [178](#)
- `Dwarf_Cmdline_Options_s`, [301](#)
- `dwarf_convert_to_global_offset`
 - DIE Attribute and Attribute-Form Details, [100](#)
- `dwarf_crc32`
 - Access GNU .gnu_debuglink, build-id., [222](#)
- `dwarf_CU_dieoffset_given_die`
 - Debugging Information Entry (DIE) content, [83](#)
- `dwarf_cu_header_basics`
 - Compilation Unit (CU) Access, [72](#)
- `dwarf_dealloc`
 - Generic dwarf_dealloc Function, [184](#)
- `dwarf_dealloc_attribute`
 - DIE Attribute and Attribute-Form Details, [100](#)
- `dwarf_dealloc_debug_addr_table`
 - .debug_addr access: DWARF5, [144](#)
- `dwarf_dealloc_die`
 - Compilation Unit (CU) Access, [73](#)
- `dwarf_dealloc_dnames`
 - Fast Access to .debug_names DWARF5, [186](#)
- `dwarf_dealloc_error`
 - Dwarf_Error Functions, [181](#)
- `dwarf_dealloc_fde_cie_list`
 - Stack Frame Access, [157](#)
- `dwarf_dealloc_frame_instr_head`
 - Stack Frame Access, [158](#)
- `dwarf_dealloc_gdbindex`
 - Fast Access to Gdb Index, [207](#)
- `dwarf_dealloc_loc_head_c`
 - Locations of data: DWARF2-DWARF5, [137](#)
- `dwarf_dealloc_macro_context`
 - Macro Access: DWARF5, [147](#)
- `dwarf_dealloc_ranges`
 - Ranges: code addresses in DWARF3-4, [128](#)
- `dwarf_dealloc_rnglists_head`
 - Rnglists: code addresses in DWARF5, [131](#)
- `dwarf_dealloc_uncompressed_block`
 - DIE Attribute and Attribute-Form Details, [100](#)
- `dwarf_dealloc_xu_header`
 - Fast Access to Split Dwarf (Debug Fission), [215](#)
- `Dwarf_Debug`
 - Defined and Opaque Structs, [45](#)
- `dwarf_debug_addr_by_index`
 - .debug_addr access: DWARF5, [144](#)
- `dwarf_debug_addr_index_to_addr`
 - Debugging Information Entry (DIE) content, [83](#)
- `Dwarf_Debug_Addr_Table`
 - Defined and Opaque Structs, [45](#)
- `dwarf_debug_addr_table`
 - .debug_addr access: DWARF5, [145](#)
- `Dwarf_Debug_Fission_Per_CU`
 - Defined and Opaque Structs, [46](#)
- `Dwarf_Debug_Fission_Per_CU_s`, [302](#)
- `Dwarf_Die`
 - Defined and Opaque Structs, [46](#)
- `dwarf_die_abbrev_children_flag`
 - Debugging Information Entry (DIE) content, [84](#)
- `dwarf_die_abbrev_code`
 - Debugging Information Entry (DIE) content, [84](#)
- `dwarf_die_abbrev_global_offset`
 - Debugging Information Entry (DIE) content, [84](#)
- `dwarf_die_CU_offset`
 - Debugging Information Entry (DIE) content, [85](#)
- `dwarf_die_CU_offset_range`
 - Debugging Information Entry (DIE) content, [85](#)
- `dwarf_die_from_hash_signature`
 - Compilation Unit (CU) Access, [73](#)
- `dwarf_die_offsets`
 - Debugging Information Entry (DIE) content, [86](#)
- `dwarf_die_text`
 - Debugging Information Entry (DIE) content, [86](#)
- `dwarf_diename`
 - Debugging Information Entry (DIE) content, [87](#)
- `dwarf_dieoffset`
 - Debugging Information Entry (DIE) content, [87](#)
- `dwarf_dietype_offset`
 - Debugging Information Entry (DIE) content, [88](#)
- `dwarf_discr_entry_s`
 - DIE Attribute and Attribute-Form Details, [101](#)
- `dwarf_discr_entry_u`
 - DIE Attribute and Attribute-Form Details, [101](#)
- `dwarf_discr_list`
 - DIE Attribute and Attribute-Form Details, [102](#)
- `dwarf_dnames_abbrevtable`
 - Fast Access to .debug_names DWARF5, [186](#)
- `dwarf_dnames_bucket`
 - Fast Access to .debug_names DWARF5, [187](#)
- `dwarf_dnames_cu_table`
 - Fast Access to .debug_names DWARF5, [187](#)
- `dwarf_dnames_entrpool`
 - Fast Access to .debug_names DWARF5, [188](#)
- `dwarf_dnames_entrpool_values`
 - Fast Access to .debug_names DWARF5, [189](#)
- `Dwarf_Dnames_Head`
 - Defined and Opaque Structs, [46](#)
- `dwarf_dnames_header`
 - Fast Access to .debug_names DWARF5, [190](#)
- `dwarf_dnames_name`
 - Fast Access to .debug_names DWARF5, [190](#)
- `dwarf_dnames_offsets`
 - Fast Access to .debug_names DWARF5, [191](#)
- `dwarf_dnames_sizes`
 - Fast Access to .debug_names DWARF5, [191](#)
- `Dwarf_Dsc_Head`
 - Defined and Opaque Structs, [46](#)
- `dwarf_errmsg`
 - Dwarf_Error Functions, [182](#)
- `dwarf_errmsg_by_number`
 - Dwarf_Error Functions, [182](#)
- `dwarf_errno`
 - Dwarf_Error Functions, [182](#)
- `Dwarf_Error`
 - Defined and Opaque Structs, [46](#)
- `Dwarf_Error_Functions`, [181](#)

- dwarf_dealloc_error, [181](#)
- dwarf_errmsg, [182](#)
- dwarf_errmsg_by_number, [182](#)
- dwarf_errno, [182](#)
- dwarf_error_creation, [183](#)
- dwarf_error_creation
 - Dwarf_Error Functions, [183](#)
- dwarf_expand_frame_instructions
 - Stack Frame Access, [158](#)
- Dwarf_Fde
 - Defined and Opaque Structs, [46](#)
- dwarf_fde_section_offset
 - Stack Frame Access, [159](#)
- dwarf_find_die_given_sig8
 - Compilation Unit (CU) Access, [74](#)
- dwarf_find_macro_value_start
 - Macro Access: DWARF2-4, [153](#)
- dwarf_finish
 - Libdwarf Initialization Functions, [64](#)
- Dwarf_Form_Class
 - Enumerators with various purposes, [43](#)
- Dwarf_Form_Data16
 - Defined and Opaque Structs, [46](#)
- Dwarf_Form_Data16_s, [302](#)
- dwarf_formaddr
 - DIE Attribute and Attribute-Form Details, [102](#)
- dwarf_formblock
 - DIE Attribute and Attribute-Form Details, [103](#)
- dwarf_formdata16
 - DIE Attribute and Attribute-Form Details, [103](#)
- dwarf_formexploc
 - DIE Attribute and Attribute-Form Details, [104](#)
- dwarf_formflag
 - DIE Attribute and Attribute-Form Details, [104](#)
- dwarf_formref
 - DIE Attribute and Attribute-Form Details, [105](#)
- dwarf_formsdata
 - DIE Attribute and Attribute-Form Details, [105](#)
- dwarf_formsig8
 - DIE Attribute and Attribute-Form Details, [106](#)
- dwarf_formsig8_const
 - DIE Attribute and Attribute-Form Details, [106](#)
- dwarf_formstring
 - DIE Attribute and Attribute-Form Details, [106](#)
- dwarf_formudata
 - DIE Attribute and Attribute-Form Details, [107](#)
- Dwarf_Frame_Instr_Head
 - Defined and Opaque Structs, [47](#)
- Dwarf_Func
 - Defined and Opaque Structs, [47](#)
- Dwarf_Gdbindex
 - Defined and Opaque Structs, [47](#)
- dwarf_gdbindex_addressarea
 - Fast Access to Gdb Index, [207](#)
- dwarf_gdbindex_addressarea_entry
 - Fast Access to Gdb Index, [208](#)
- dwarf_gdbindex_culist_array
 - Fast Access to Gdb Index, [208](#)
- dwarf_gdbindex_culist_entry
 - Fast Access to Gdb Index, [209](#)
- dwarf_gdbindex_cuvector_inner_attributes
 - Fast Access to Gdb Index, [209](#)
- dwarf_gdbindex_cuvector_instance_expand_value
 - Fast Access to Gdb Index, [210](#)
- dwarf_gdbindex_cuvector_length
 - Fast Access to Gdb Index, [210](#)
- dwarf_gdbindex_header
 - Fast Access to Gdb Index, [211](#)
- dwarf_gdbindex_string_by_offset
 - Fast Access to Gdb Index, [212](#)
- dwarf_gdbindex_symboltable_array
 - Fast Access to Gdb Index, [212](#)
- dwarf_gdbindex_symboltable_entry
 - Fast Access to Gdb Index, [212](#)
- dwarf_gdbindex_types_culist_array
 - Fast Access to Gdb Index, [213](#)
- dwarf_gdbindex_types_culist_entry
 - Fast Access to Gdb Index, [213](#)
- dwarf_get_abbrev
 - Abbreviations Section Details, [174](#)
- dwarf_get_abbrev_children_flag
 - Abbreviations Section Details, [175](#)
- dwarf_get_abbrev_code
 - Abbreviations Section Details, [175](#)
- dwarf_get_abbrev_entry_b
 - Abbreviations Section Details, [175](#)
- dwarf_get_abbrev_tag
 - Abbreviations Section Details, [176](#)
- dwarf_get_address_size
 - Object Sections Data, [233](#)
- dwarf_get_arange
 - Fast Access to a CU given a code address, [193](#)
- dwarf_get_arange_cu_header_offset
 - Fast Access to a CU given a code address, [193](#)
- dwarf_get_arange_info_b
 - Fast Access to a CU given a code address, [193](#)
- dwarf_get_aranges
 - Fast Access to a CU given a code address, [194](#)
- dwarf_get_cie_augmentation_data
 - Stack Frame Access, [159](#)
- dwarf_get_cie_index
 - Stack Frame Access, [160](#)
- dwarf_get_cie_info_b
 - Stack Frame Access, [160](#)
- dwarf_get_cie_of_fde
 - Stack Frame Access, [161](#)
- dwarf_get_cu_die_offset
 - Fast Access to a CU given a code address, [195](#)
- dwarf_get_cu_die_offset_given_cu_header_offset_b
 - Debugging Information Entry (DIE) content, [88](#)
- dwarf_get_debug_addr_index
 - DIE Attribute and Attribute-Form Details, [107](#)
- dwarf_get_debug_str_index
 - DIE Attribute and Attribute-Form Details, [108](#)
- dwarf_get_debug_sup
 - Access to Section .debug_sup, [184](#)

- dwarf_get_debugfission_for_die
 - Fast Access to Split Dwarf (Debug Fission), [215](#)
- dwarf_get_debugfission_for_key
 - Fast Access to Split Dwarf (Debug Fission), [215](#)
- dwarf_get_die_address_size
 - Debugging Information Entry (DIE) content, [89](#)
- dwarf_get_die_infotypes_flag
 - Compilation Unit (CU) Access, [74](#)
- dwarf_get_die_section_name
 - Object Sections Data, [233](#)
- dwarf_get_die_section_name_b
 - Object Sections Data, [233](#)
- dwarf_get_EH_name
 - Names DW_TAG_member etc as strings, [229](#)
- dwarf_get_endian_copy_function
 - Miscellaneous Functions, [247](#)
- dwarf_get_fde_at_pc
 - Stack Frame Access, [161](#)
- dwarf_get_fde_augmentation_data
 - Stack Frame Access, [162](#)
- dwarf_get_fde_exception_info
 - Stack Frame Access, [163](#)
- dwarf_get_fde_for_die
 - Stack Frame Access, [163](#)
- dwarf_get_fde_info_for_all_regs3
 - Stack Frame Access, [163](#)
- dwarf_get_fde_info_for_all_regs3_b
 - Stack Frame Access, [163](#)
- dwarf_get_fde_info_for_cfa_reg3_b
 - Stack Frame Access, [164](#)
- dwarf_get_fde_info_for_cfa_reg3_c
 - Stack Frame Access, [165](#)
- dwarf_get_fde_info_for_reg3_b
 - Stack Frame Access, [165](#)
- dwarf_get_fde_info_for_reg3_c
 - Stack Frame Access, [165](#)
- dwarf_get_fde_instr_bytes
 - Stack Frame Access, [167](#)
- dwarf_get_fde_list
 - Stack Frame Access, [167](#)
- dwarf_get_fde_list_eh
 - Stack Frame Access, [168](#)
- dwarf_get_fde_n
 - Stack Frame Access, [168](#)
- dwarf_get_fde_range
 - Stack Frame Access, [168](#)
- dwarf_get_form_class
 - DIE Attribute and Attribute-Form Details, [108](#)
- dwarf_get_FORM_CLASS_name
 - Names DW_TAG_member etc as strings, [229](#)
- dwarf_get_frame_instruction
 - Stack Frame Access, [169](#)
- dwarf_get_frame_instruction_a
 - Stack Frame Access, [170](#)
- dwarf_get_FRAME_name
 - Names DW_TAG_member etc as strings, [229](#)
- dwarf_get_frame_section_name
 - Object Sections Data, [233](#)
- dwarf_get_frame_section_name_eh_gnu
 - Object Sections Data, [234](#)
- dwarf_get_globals
 - Fast Access to .debug_pubnames and more., [196](#)
- dwarf_get_globals_header
 - Fast Access to .debug_pubnames and more., [197](#)
- dwarf_get_gnu_index_block
 - Fast Access to GNU .debug_gnu_pubnames, [203](#)
- dwarf_get_gnu_index_block_entry
 - Fast Access to GNU .debug_gnu_pubnames, [203](#)
- dwarf_get_gnu_index_head
 - Fast Access to GNU .debug_gnu_pubnames, [205](#)
- dwarf_get_GNUKIND_name
 - Names DW_TAG_member etc as strings, [229](#)
- dwarf_get_GNUIVIS_name
 - Names DW_TAG_member etc as strings, [229](#)
- dwarf_get_harmless_error_list
 - Harmless Error recording, [225](#)
- dwarf_get_line_section_name_from_die
 - Object Sections Data, [234](#)
- dwarf_get_LLEX_name
 - Names DW_TAG_member etc as strings, [230](#)
- dwarf_get_location_op_value_c
 - Locations of data: DWARF2-DWARF5, [137](#)
- dwarf_get_locdesc_entry_d
 - Locations of data: DWARF2-DWARF5, [138](#)
- dwarf_get_locdesc_entry_e
 - Locations of data: DWARF2-DWARF5, [139](#)
- dwarf_get_loclist_c
 - Locations of data: DWARF2-DWARF5, [139](#)
- dwarf_get_loclist_context_basics
 - Locations of data: DWARF2-DWARF5, [140](#)
- dwarf_get_loclist_head_basics
 - Locations of data: DWARF2-DWARF5, [140](#)
- dwarf_get_loclist_head_kind
 - Locations of data: DWARF2-DWARF5, [141](#)
- dwarf_get_loclist_lle
 - Locations of data: DWARF2-DWARF5, [141](#)
- dwarf_get_loclist_offset_index_value
 - Locations of data: DWARF2-DWARF5, [141](#)
- dwarf_get_MACINFO_name
 - Names DW_TAG_member etc as strings, [230](#)
- dwarf_get_macro_context
 - Macro Access: DWARF5, [147](#)
- dwarf_get_macro_context_by_offset
 - Macro Access: DWARF5, [147](#)
- dwarf_get_macro_defundef
 - Macro Access: DWARF5, [148](#)
- dwarf_get_macro_details
 - Macro Access: DWARF2-4, [153](#)
- dwarf_get_macro_import
 - Macro Access: DWARF5, [149](#)
- dwarf_get_MACRO_name
 - Names DW_TAG_member etc as strings, [230](#)
- dwarf_get_macro_op
 - Macro Access: DWARF5, [149](#)
- dwarf_get_macro_startend_file
 - Macro Access: DWARF5, [151](#)

- dwarf_get_mmap_count
 - Section allocation: malloc or mmap, [248](#)
- dwarf_get_offset_size
 - Object Sections Data, [234](#)
- dwarf_get_pubtypes
 - Fast Access to .debug_pubnames and more., [197](#)
- dwarf_get_ranges_b
 - Ranges: code addresses in DWARF3-4, [129](#)
- dwarf_get_ranges_baseaddress
 - Ranges: code addresses in DWARF3-4, [129](#)
- dwarf_get_real_section_name
 - Object Sections Data, [235](#)
- dwarf_get_rnglist_context_basics
 - Rnglists: code addresses in DWARF5, [131](#)
- dwarf_get_rnglist_head_basics
 - Rnglists: code addresses in DWARF5, [132](#)
- dwarf_get_rnglist_offset_index_value
 - Rnglists: code addresses in DWARF5, [132](#)
- dwarf_get_rnglist_rle
 - Rnglists: code addresses in DWARF5, [133](#)
- dwarf_get_rnglists_entry_fields_a
 - Rnglists: code addresses in DWARF5, [133](#)
- dwarf_get_section_count
 - Object Sections Data, [235](#)
- dwarf_get_section_info_by_index
 - Object Sections Data, [236](#)
- dwarf_get_section_info_by_index_a
 - Object Sections Data, [236](#)
- dwarf_get_section_info_by_name
 - Object Sections Data, [237](#)
- dwarf_get_section_info_by_name_a
 - Object Sections Data, [237](#)
- dwarf_get_section_max_offsets_d
 - Object Sections Data, [238](#)
- dwarf_get_str
 - String Section .debug_str Details, [177](#)
- dwarf_get_tied_dbg
 - Libdwarf Initialization Functions, [65](#)
- dwarf_get_universalbinary_count
 - Miscellaneous Functions, [243](#)
- dwarf_get_version_of_die
 - Debugging Information Entry (DIE) content, [89](#)
- dwarf_get_xu_hash_entry
 - Fast Access to Split Dwarf (Debug Fission), [217](#)
- dwarf_get_xu_index_header
 - Fast Access to Split Dwarf (Debug Fission), [217](#)
- dwarf_get_xu_index_section_type
 - Fast Access to Split Dwarf (Debug Fission), [218](#)
- dwarf_get_xu_section_names
 - Fast Access to Split Dwarf (Debug Fission), [219](#)
- dwarf_get_xu_section_offset
 - Fast Access to Split Dwarf (Debug Fission), [219](#)
- Dwarf_Global
 - Defined and Opaque Structs, [47](#)
- dwarf_global_cu_offset
 - Fast Access to .debug_pubnames and more., [198](#)
- dwarf_global_die_offset
 - Fast Access to .debug_pubnames and more., [198](#)
- dwarf_global_formref
 - DIE Attribute and Attribute-Form Details, [109](#)
- dwarf_global_formref_b
 - DIE Attribute and Attribute-Form Details, [109](#)
- dwarf_global_name_offsets
 - Fast Access to .debug_pubnames and more., [198](#)
- dwarf_global_tag_number
 - Fast Access to .debug_pubnames and more., [199](#)
- dwarf_globals_by_type
 - Fast Access to .debug_pubnames and more., [199](#)
- dwarf_globals_dealloc
 - Fast Access to .debug_pubnames and more., [201](#)
- dwarf_globname
 - Fast Access to .debug_pubnames and more., [201](#)
- dwarf_gnu_debuglink
 - Access GNU .gnu_debuglink, build-id., [222](#)
- dwarf_gnu_index_dealloc
 - Fast Access to GNU .debug_gnu_pubnames, [205](#)
- Dwarf_Gnu_Index_Head
 - Defined and Opaque Structs, [47](#)
- Dwarf_Half
 - Basic Library Datatypes Group, [41](#)
- Dwarf_Handler
 - Defined and Opaque Structs, [47](#)
- dwarf_hasattr
 - Debugging Information Entry (DIE) content, [90](#)
- dwarf_hasform
 - DIE Attribute and Attribute-Form Details, [109](#)
- dwarf_highpc_b
 - Debugging Information Entry (DIE) content, [90](#)
- dwarf_init_b
 - Libdwarf Initialization Functions, [65](#)
- dwarf_init_path
 - Libdwarf Initialization Functions, [66](#)
- dwarf_init_path_a
 - Libdwarf Initialization Functions, [67](#)
- dwarf_init_path_dl
 - Libdwarf Initialization Functions, [67](#)
- dwarf_init_path_dl_a
 - Libdwarf Initialization Functions, [68](#)
- dwarf_insert_harmless_error
 - Harmless Error recording, [225](#)
- dwarf_iterate_fde_all_regs3
 - Stack Frame Access, [171](#)
- dwarf_iterate_fde_callback_function_type
 - Stack Frame Access, [157](#)
- dwarf_language_version_data
 - Debugging Information Entry (DIE) content, [91](#)
- dwarf_language_version_string
 - Debugging Information Entry (DIE) content, [91](#)
- dwarf_library_allow_dup_attr
 - Miscellaneous Functions, [244](#)
- Dwarf_Line
 - Defined and Opaque Structs, [47](#)
- Dwarf_Line_Context
 - Defined and Opaque Structs, [48](#)
- dwarf_line_is_addr_set
 - Line Table For a CU, [114](#)

- dwarf_line_srcfileno
 - Line Table For a CU, [115](#)
- dwarf_lineaddr
 - Line Table For a CU, [115](#)
- dwarf_linebeginstatement
 - Line Table For a CU, [115](#)
- dwarf_lineblock
 - Line Table For a CU, [116](#)
- dwarf_lineendsequence
 - Line Table For a CU, [116](#)
- dwarf_lineno
 - Line Table For a CU, [117](#)
- dwarf_lineoff_b
 - Line Table For a CU, [117](#)
- dwarf_linesrc
 - Line Table For a CU, [118](#)
- dwarf_load_loclists
 - Locations of data: DWARF2-DWARF5, [142](#)
- dwarf_load_rnglists
 - Rnglists: code addresses in DWARF5, [134](#)
- Dwarf_Loc_Head_c
 - Defined and Opaque Structs, [48](#)
- Dwarf_Locdesc_c
 - Defined and Opaque Structs, [48](#)
- dwarf_loclist_from_expr_c
 - Locations of data: DWARF2-DWARF5, [143](#)
- dwarf_lowpc
 - Debugging Information Entry (DIE) content, [92](#)
- dwarf_lvn_name
 - Debugging Information Entry (DIE) content, [92](#)
- dwarf_lvn_name_direct
 - Debugging Information Entry (DIE) content, [92](#)
- dwarf_lvn_table_entry
 - Debugging Information Entry (DIE) content, [93](#)
- dwarf_machine_architecture
 - Object Sections Data, [239](#)
- dwarf_machine_architecture_a
 - Object Sections Data, [239](#)
- Dwarf_Macro_Context
 - Defined and Opaque Structs, [48](#)
- dwarf_macro_context_head
 - Macro Access: DWARF5, [151](#)
- dwarf_macro_context_total_length
 - Macro Access: DWARF5, [152](#)
- Dwarf_Macro_Details
 - Defined and Opaque Structs, [48](#)
- Dwarf_Macro_Details_s, [302](#)
- dwarf_macro_operands_table
 - Macro Access: DWARF5, [152](#)
- dwarf_next_cu_header_d
 - Compilation Unit (CU) Access, [74](#)
- dwarf_next_cu_header_e
 - Compilation Unit (CU) Access, [75](#)
- dwarf_next_str_offsets_table
 - Str_Offsets section details, [179](#)
- Dwarf_Obj_Access_Interface_a
 - Defined and Opaque Structs, [48](#)
- Dwarf_Obj_Access_Interface_a_s, [303](#)
- Dwarf_Obj_Access_Methods_a
 - Defined and Opaque Structs, [48](#)
- Dwarf_Obj_Access_Methods_a_s, [303](#)
- Dwarf_Obj_Access_Sections_a
 - Defined and Opaque Structs, [48](#)
- Dwarf_Obj_Access_Sections_a_s, [304](#)
- dwarf_object_finish
 - Libdwarf Initialization Functions, [69](#)
- dwarf_object_init_b
 - Libdwarf Initialization Functions, [69](#)
- Dwarf_Off
 - Basic Library Datatypes Group, [42](#)
- dwarf_offdie_b
 - Compilation Unit (CU) Access, [76](#)
- dwarf_offset_list
 - Debugging Information Entry (DIE) content, [94](#)
- dwarf_open_str_offsets_table_access
 - Str_Offsets section details, [179](#)
- dwarf_package_version
 - Miscellaneous Functions, [244](#)
- dwarf_print_lines
 - Line Table For a CU, [118](#)
- dwarf_printf_callback_function_type
 - Defined and Opaque Structs, [49](#)
- Dwarf_Printf_Callback_Info_s, [305](#)
- dwarf_prologue_end_etc
 - Line Table For a CU, [119](#)
- Dwarf_Ptr
 - Basic Library Datatypes Group, [42](#)
- Dwarf_Ranges
 - Defined and Opaque Structs, [49](#)
- Dwarf_Ranges_Entry_Type
 - Enumerators with various purposes, [43](#)
- Dwarf_Ranges_s, [305](#)
- dwarf_record_cmdline_options
 - Miscellaneous Functions, [245](#)
- dwarf_register_printf_callback
 - Line Table For a CU, [119](#)
- Dwarf_Regtable3
 - Defined and Opaque Structs, [49](#)
- Dwarf_Regtable3_s, [306](#)
- Dwarf_Regtable_Entry3
 - Defined and Opaque Structs, [49](#)
- Dwarf_Regtable_Entry3_s, [306](#)
- dwarf_return_empty_pubnames
 - Fast Access to .debug_pubnames and more., [201](#)
- dwarf_rnglists_get_rle_head
 - Rnglists: code addresses in DWARF5, [135](#)
- Dwarf_Rnglists_Head
 - Defined and Opaque Structs, [51](#)
- Dwarf_Sec_Alloc_Pref
 - Defined and Opaque Structs, [52](#)
- dwarf_sec_group_map
 - Section Groups Objectfile Data, [241](#)
- dwarf_sec_group_sizes
 - Section Groups Objectfile Data, [241](#)
- Dwarf_Section
 - Defined and Opaque Structs, [51](#)

- dwarf_set_de_alloc_flag
 - Miscellaneous Functions, [245](#)
- dwarf_set_default_address_size
 - Miscellaneous Functions, [245](#)
- dwarf_set_frame_cfa_value
 - Stack Frame Access, [171](#)
- dwarf_set_frame_rule_initial_value
 - Stack Frame Access, [172](#)
- dwarf_set_frame_rule_table_size
 - Stack Frame Access, [172](#)
- dwarf_set_frame_same_value
 - Stack Frame Access, [173](#)
- dwarf_set_frame_undefined_value
 - Stack Frame Access, [173](#)
- dwarf_set_harmless_error_list_size
 - Harmless Error recording, [226](#)
- dwarf_set_harmless_errors_enabled
 - Harmless Error recording, [226](#)
- dwarf_set_load_preference
 - Section allocation: malloc or mmap, [249](#)
- dwarf_set_reloc_application
 - Miscellaneous Functions, [246](#)
- dwarf_set_stringcheck
 - Miscellaneous Functions, [246](#)
- dwarf_set_tied_dbg
 - Libdwarf Initialization Functions, [70](#)
- dwarf_siblingof_b
 - Compilation Unit (CU) Access, [77](#)
- dwarf_siblingof_c
 - Compilation Unit (CU) Access, [78](#)
- Dwarf_Sig8
 - Defined and Opaque Structs, [51](#)
- Dwarf_Sig8_s, [307](#)
- Dwarf_Signed
 - Basic Library Datatypes Group, [42](#)
- Dwarf_Small
 - Basic Library Datatypes Group, [42](#)
- dwarf_srcfiles
 - Line Table For a CU, [120](#)
- dwarf_srclang
 - Debugging Information Entry (DIE) content, [94](#)
- dwarf_srclangname
 - Debugging Information Entry (DIE) content, [95](#)
- dwarf_srclangname_version
 - Debugging Information Entry (DIE) content, [95](#)
- dwarf_srclines_b
 - Line Table For a CU, [121](#)
- dwarf_srclines_comp_dir
 - Line Table For a CU, [121](#)
- dwarf_srclines_dealloc_b
 - Line Table For a CU, [122](#)
- dwarf_srclines_files_data_b
 - Line Table For a CU, [122](#)
- dwarf_srclines_files_indexes
 - Line Table For a CU, [123](#)
- dwarf_srclines_from_linecontext
 - Line Table For a CU, [124](#)
- dwarf_srclines_include_dir_count
 - Line Table For a CU, [124](#)
- dwarf_srclines_include_dir_data
 - Line Table For a CU, [125](#)
- dwarf_srclines_subprog_count
 - Line Table For a CU, [125](#)
- dwarf_srclines_subprog_data
 - Line Table For a CU, [126](#)
- dwarf_srclines_table_offset
 - Line Table For a CU, [126](#)
- dwarf_srclines_two_level_from_linecontext
 - Line Table For a CU, [127](#)
- dwarf_srclines_version
 - Line Table For a CU, [127](#)
- dwarf_str_offsets_statistics
 - Str_Offsets section details, [180](#)
- Dwarf_Str_Offsets_Table
 - Defined and Opaque Structs, [51](#)
- dwarf_str_offsets_value_by_index
 - Str_Offsets section details, [180](#)
- dwarf_suppress_debuglink_crc
 - Access GNU .gnu_debuglink, build-id., [223](#)
- dwarf_tag
 - Debugging Information Entry (DIE) content, [96](#)
- Dwarf_Type
 - Defined and Opaque Structs, [51](#)
- dwarf_uncompress_integer_block_a
 - DIE Attribute and Attribute-Form Details, [110](#)
- Dwarf_Unsigned
 - Basic Library Datatypes Group, [42](#)
- dwarf_validate_die_sibling
 - Debugging Information Entry (DIE) content, [96](#)
- Dwarf_Var
 - Defined and Opaque Structs, [51](#)
- Dwarf_Weak
 - Defined and Opaque Structs, [52](#)
- dwarf_whatattr
 - DIE Attribute and Attribute-Form Details, [110](#)
- dwarf_whatform
 - DIE Attribute and Attribute-Form Details, [111](#)
- dwarf_whatform_direct
 - DIE Attribute and Attribute-Form Details, [111](#)
- Dwarf_Xu_Index_Header
 - Defined and Opaque Structs, [52](#)
- Enumerators with various purposes, [42](#)
 - Dwarf_Form_Class, [43](#)
 - Dwarf_Ranges_Entry_Type, [43](#)
- Examining Section Group data, [253](#)
- Example getting .debug_ranges data, [282](#)
- Example walking CUs(d), [258](#)
- Example walking CUs(e), [257](#)
- Extracting fde, cie lists., [278](#)
- Fast Access to .debug_names DWARF5, [185](#)
 - dwarf_dealloc_dnames, [186](#)
 - dwarf_dnames_abbrevtable, [186](#)
 - dwarf_dnames_bucket, [187](#)
 - dwarf_dnames_cu_table, [187](#)
 - dwarf_dnames_entrypool, [188](#)

- dwarf_dnames_entrypool_values, 189
- dwarf_dnames_header, 190
- dwarf_dnames_name, 190
- dwarf_dnames_offsets, 191
- dwarf_dnames_sizes, 191
- Fast Access to .debug_pubnames and more., 195
 - dwarf_get_globals, 196
 - dwarf_get_globals_header, 197
 - dwarf_get_pubtypes, 197
 - dwarf_global_cu_offset, 198
 - dwarf_global_die_offset, 198
 - dwarf_global_name_offsets, 198
 - dwarf_global_tag_number, 199
 - dwarf_globals_by_type, 199
 - dwarf_globals_dealloc, 201
 - dwarf_globname, 201
 - dwarf_return_empty_pubnames, 201
- Fast Access to a CU given a code address, 192
 - dwarf_get_arange, 193
 - dwarf_get_arange_cu_header_offset, 193
 - dwarf_get_arange_info_b, 193
 - dwarf_get_aranges, 194
 - dwarf_get_cu_die_offset, 195
- Fast Access to Gdb Index, 206
 - dwarf_dealloc_gdbindex, 207
 - dwarf_gdbindex_addressarea, 207
 - dwarf_gdbindex_addressarea_entry, 208
 - dwarf_gdbindex_culist_array, 208
 - dwarf_gdbindex_culist_entry, 209
 - dwarf_gdbindex_cuvector_inner_attributes, 209
 - dwarf_gdbindex_cuvector_instance_expand_value, 210
 - dwarf_gdbindex_cuvector_length, 210
 - dwarf_gdbindex_header, 211
 - dwarf_gdbindex_string_by_offset, 212
 - dwarf_gdbindex_symboltable_array, 212
 - dwarf_gdbindex_symboltable_entry, 212
 - dwarf_gdbindex_types_culist_array, 213
 - dwarf_gdbindex_types_culist_entry, 213
- Fast Access to GNU .debug_gnu_pubnames, 202
 - dwarf_get_gnu_index_block, 203
 - dwarf_get_gnu_index_block_entry, 203
 - dwarf_get_gnu_index_head, 205
 - dwarf_gnu_index_dealloc, 205
- Fast Access to Split Dwarf (Debug Fission), 214
 - dwarf_dealloc_xu_header, 215
 - dwarf_get_debugfission_for_die, 215
 - dwarf_get_debugfission_for_key, 215
 - dwarf_get_xu_hash_entry, 217
 - dwarf_get_xu_index_header, 217
 - dwarf_get_xu_index_section_type, 218
 - dwarf_get_xu_section_names, 219
 - dwarf_get_xu_section_offset, 219
- Generic dwarf_dealloc Function, 183
 - dwarf_dealloc, 184
- Harmless Error recording, 224
 - dwarf_get_harmless_error_list, 225
 - dwarf_insert_harmless_error, 225
 - dwarf_set_harmless_error_list_size, 226
 - dwarf_set_harmless_errors_enabled, 226
- JIT and special case DWARF, 23
- LEB Encode and Decode, 242
- Libdwarf Initialization Functions, 64
 - dwarf_finish, 64
 - dwarf_get_tied_dbg, 65
 - dwarf_init_b, 65
 - dwarf_init_path, 66
 - dwarf_init_path_a, 67
 - dwarf_init_path_dl, 67
 - dwarf_init_path_dl_a, 68
 - dwarf_object_finish, 69
 - dwarf_object_init_b, 69
 - dwarf_set_tied_dbg, 70
- libdwarf.h, 31, 333
- Line Table For a CU, 112
 - dwarf_check_lineheader_b, 114
 - dwarf_line_is_addr_set, 114
 - dwarf_line_srcfileno, 115
 - dwarf_lineaddr, 115
 - dwarf_linebeginstatement, 115
 - dwarf_lineblock, 116
 - dwarf_lineendsequence, 116
 - dwarf_lineno, 117
 - dwarf_lineoff_b, 117
 - dwarf_linesrc, 118
 - dwarf_print_lines, 118
 - dwarf_prologue_end_etc, 119
 - dwarf_register_printf_callback, 119
 - dwarf_srcfiles, 120
 - dwarf_srclines_b, 121
 - dwarf_srclines_comp_dir, 121
 - dwarf_srclines_dealloc_b, 122
 - dwarf_srclines_files_data_b, 122
 - dwarf_srclines_files_indexes, 123
 - dwarf_srclines_from_linecontext, 124
 - dwarf_srclines_include_dir_count, 124
 - dwarf_srclines_include_dir_data, 125
 - dwarf_srclines_subprog_count, 125
 - dwarf_srclines_subprog_data, 126
 - dwarf_srclines_table_offset, 126
 - dwarf_srclines_two_level_from_linecontext, 127
 - dwarf_srclines_version, 127
- Location/expression access, 264
- Locations of data: DWARF2-DWARF5, 135
 - dwarf_dealloc_loc_head_c, 137
 - dwarf_get_location_op_value_c, 137
 - dwarf_get_locdesc_entry_d, 138
 - dwarf_get_locdesc_entry_e, 139
 - dwarf_get_loclist_c, 139
 - dwarf_get_loclist_context_basics, 140
 - dwarf_get_loclist_head_basics, 140
 - dwarf_get_loclist_head_kind, 141
 - dwarf_get_loclist_lle, 141
 - dwarf_get_loclist_offset_index_value, 141

- dwarf_load_loclists, 142
- dwarf_loclist_from_expr_c, 143
- Macro Access: DWARF2-4, 153
 - dwarf_find_macro_value_start, 153
 - dwarf_get_macro_details, 153
- Macro Access: DWARF5, 146
 - dwarf_dealloc_macro_context, 147
 - dwarf_get_macro_context, 147
 - dwarf_get_macro_context_by_offset, 147
 - dwarf_get_macro_defundef, 148
 - dwarf_get_macro_import, 149
 - dwarf_get_macro_op, 149
 - dwarf_get_macro_startend_file, 151
 - dwarf_macro_context_head, 151
 - dwarf_macro_context_total_length, 152
 - dwarf_macro_operands_table, 152
- Miscellaneous Functions, 243
 - dwarf_get_endian_copy_function, 247
 - dwarf_get_universalbinary_count, 243
 - dwarf_library_allow_dup_attr, 244
 - dwarf_package_version, 244
 - dwarf_record_cmdline_options, 245
 - dwarf_set_de_alloc_flag, 245
 - dwarf_set_default_address_size, 245
 - dwarf_set_reloc_application, 246
 - dwarf_set_stringcheck, 246
- Names DW_TAG_member etc as strings, 227
 - dwarf_get_EH_name, 229
 - dwarf_get_FORM_CLASS_name, 229
 - dwarf_get_FRAME_name, 229
 - dwarf_get_GNUIKIND_name, 229
 - dwarf_get_GNUIVIS_name, 229
 - dwarf_get_LLEX_name, 230
 - dwarf_get_MACINFO_name, 230
 - dwarf_get_MACRO_name, 230
- Object Sections Data, 231
 - dwarf_get_address_size, 233
 - dwarf_get_die_section_name, 233
 - dwarf_get_die_section_name_b, 233
 - dwarf_get_frame_section_name, 233
 - dwarf_get_frame_section_name_eh_gnu, 234
 - dwarf_get_line_section_name_from_die, 234
 - dwarf_get_offset_size, 234
 - dwarf_get_real_section_name, 235
 - dwarf_get_section_count, 235
 - dwarf_get_section_info_by_index, 236
 - dwarf_get_section_info_by_index_a, 236
 - dwarf_get_section_info_by_name, 237
 - dwarf_get_section_info_by_name_a, 237
 - dwarf_get_section_max_offsets_d, 238
 - dwarf_machine_architecture, 239
 - dwarf_machine_architecture_a, 239
- Ranges: code addresses in DWARF3-4, 128
 - dwarf_dealloc_ranges, 128
 - dwarf_get_ranges_b, 129
 - dwarf_get_ranges_baseaddress, 129
- Reading gdbindex addressarea, 285
- Reading .debug_funcnames (nonstandard), 271
- Reading .debug_macinfo (DWARF2-4), 278
- Reading .debug_macro data (DWARF5), 275
- Reading .debug_names data, 273
- Reading .debug_types (nonstandard), 272
- Reading .debug_varnames data (nonstandard), 272
- Reading .debug_weaknames (nonstandard), 271
- Reading a location expression, 266
- Reading an aranges section, 281
- Reading cu and tu Debug Fission data, 286
- Reading gdbindex data, 284
- Reading high pc from a DIE., 287
- Reading Split Dwarf (Debug Fission) data, 288
- Reading Split Dwarf (Debug Fission) hash slots, 287
- Reading string offsets section data, 280
- Reading the .eh_frame section, 279
- Reading the gdbindex symbol table, 285
- Retrieving tag,attribute,etc names, 288
- Rnglists: code addresses in DWARF5, 130
 - dwarf_dealloc_rnglists_head, 131
 - dwarf_get_rnglist_context_basics, 131
 - dwarf_get_rnglist_head_basics, 132
 - dwarf_get_rnglist_offset_index_value, 132
 - dwarf_get_rnglist_rle, 133
 - dwarf_get_rnglists_entry_fields_a, 133
 - dwarf_load_rnglists, 134
 - dwarf_rnglists_get_rle_head, 135
- Section allocation: malloc or mmap, 248
 - dwarf_get_mmap_count, 248
 - dwarf_set_load_preference, 249
- Section Groups Objectfile Data, 240
 - dwarf_sec_group_map, 241
 - dwarf_sec_group_sizes, 241
- Stack Frame Access, 154
 - dwarf_cie_section_offset, 157
 - dwarf_dealloc_fde_cie_list, 157
 - dwarf_dealloc_frame_instr_head, 158
 - dwarf_expand_frame_instructions, 158
 - dwarf_fde_section_offset, 159
 - dwarf_get_cie_augmentation_data, 159
 - dwarf_get_cie_index, 160
 - dwarf_get_cie_info_b, 160
 - dwarf_get_cie_of_fde, 161
 - dwarf_get_fde_at_pc, 161
 - dwarf_get_fde_augmentation_data, 162
 - dwarf_get_fde_exception_info, 163
 - dwarf_get_fde_for_die, 163
 - dwarf_get_fde_info_for_all_regs3, 163
 - dwarf_get_fde_info_for_all_regs3_b, 163
 - dwarf_get_fde_info_for_cfa_reg3_b, 164
 - dwarf_get_fde_info_for_cfa_reg3_c, 165
 - dwarf_get_fde_info_for_reg3_b, 165
 - dwarf_get_fde_info_for_reg3_c, 165
 - dwarf_get_fde_instr_bytes, 167
 - dwarf_get_fde_list, 167
 - dwarf_get_fde_list_eh, 168

- dwarf_get_fde_n, [168](#)
- dwarf_get_fde_range, [168](#)
- dwarf_get_frame_instruction, [169](#)
- dwarf_get_frame_instruction_a, [170](#)
- dwarf_iterate_fde_all_regs3, [171](#)
- dwarf_iterate_fde_callback_function_type, [157](#)
- dwarf_set_frame_cfa_value, [171](#)
- dwarf_set_frame_rule_initial_value, [172](#)
- dwarf_set_frame_rule_table_size, [172](#)
- dwarf_set_frame_same_value, [173](#)
- dwarf_set_frame_undefined_value, [173](#)
- Str_Offsets section details, [178](#)
 - dwarf_close_str_offsets_table_access, [178](#)
 - dwarf_next_str_offsets_table, [179](#)
 - dwarf_open_str_offsets_table_access, [179](#)
 - dwarf_str_offsets_statistics, [180](#)
 - dwarf_str_offsets_value_by_index, [180](#)
- String Section .debug_str Details, [177](#)
 - dwarf_get_str, [177](#)
- Using dwarf_attrlist(), [261](#)
- Using dwarf_expand_frame_instructions, [279](#)
- Using dwarf_attrlist(), [251](#)
- Using dwarf_child(), [255](#)
- Using dwarf_discr_list(), [263](#)
- Using dwarf_get_globals(), [270](#)
- Using dwarf_globals_by_type(), [271](#)
- Using dwarf_init_path(), [249](#)
- Using dwarf_init_path_dl(), [250](#)
- Using dwarf_offdie_b(), [260](#)
- Using dwarf_offset_given_die(), [261](#)
- Using dwarf_offset_list(), [261](#)
- Using dwarf_siblingof_b(), [254](#)
- Using dwarf_siblingof_c(), [254](#)
- Using dwarf_srcfiles(), [269](#)
- Using dwarf_srclines_b(), [267](#)
- Using dwarf_srclines_b() and linecontext, [269](#)
- using dwarf_validate_die_sibling, [255](#)
- Using GNU debuglink data, [289](#)